Theory:

1. Working Principle: The main pipeline of the system works on a singular directional system flow; this refers to the algorithm getting input from the users in form of an array of ingredients and receiving top 3 recipes they can cook using them. The matching and ranking of recipes with respect to ingredients displayed a use case for natural language processing during the training process for the algorithm. A large amount of data was to be collected for the process of training manually from various food blogging websites. To make the API integrate with the algorithm for outputting back to the client, we also required a web server that can be integrated easily into the system.

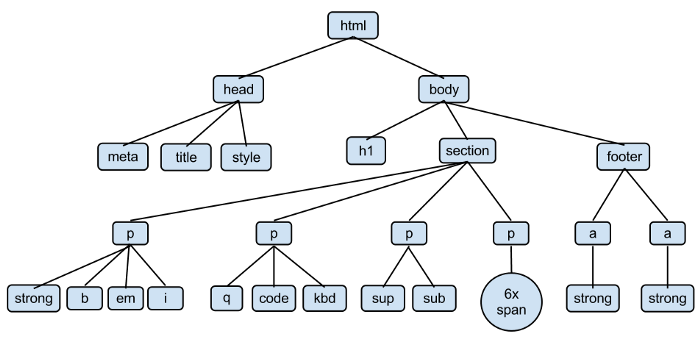
The working process of the whole project is divided into four steps.

* 1. Data mining and web scraping
  2. Data cleaning, modelling, and training
  3. Backend server for API routing to the algorithm
  4. Front end web application to host a prototype of the project

1. Data Mining and Web Scrapping:

Collection of data is a crucial element of the whole backbone of the system. Without an adequate quantity, the system will not be able to generate useful patterns and will go under the case of underfitting. At the same time collection of data manually requires large amount of human input, computational resources, and time. TO fix this problem, S.A.R.A integrates a web scrapping stage using “BeautifulSoup” which is a python-based web scrapping library able to automate the process of extracting information from the internet efficiently out of XML and HTML files.

BeautifulSoup works on a parse tree methodology that is very similar to the tree data structure. It allows the code to browse multiple pages concurrently and in a hierarchical format due to this structure. The tree is presented in the following format.



In this tree, the HTML tag is the node tree as it is having no parents of his own, and all other tags are nodes of the tree. Each node contains all the information of held inside of it, which can be extracted using a structured python code.

Each HTML element present inside the website’s source code is broken down and traversed using various attributes, this can be class names, IDs or even tag structure. To extract information from any node, the code requires a path it must look under which should be definitive and as precise as possible. As modern websites tend to have a well-documented and structured codebase, it makes it easier to use browser developer tools (example: Chrome dev tools) to investigate the source code of the website and find this very path.

For S.A.R.A, the three websites that were short listed for extraction were:

1. <http://www.foodnetwork.com>
2. <http://allrecipes.com>
3. <http://www.epicurious.com>

The selection of these websites over others was done based on cleaner source code and ease of access to the elements containing the recipes.

The next step was to extract hyperlinks to all these recipes’ page to extract information, to achieve this the code went through a loop of all the recipes on the search page and stored the links giving inside an <a> (‘anchor tag’) into a list data structure. BeautifulSoup allow provides methods to iterate through pages which helped in storing all theses links spread across multiple pages.

Once the list of hyperlinks is completed, the code then loops through this entire list and scrapes every page for tags containing this four information:

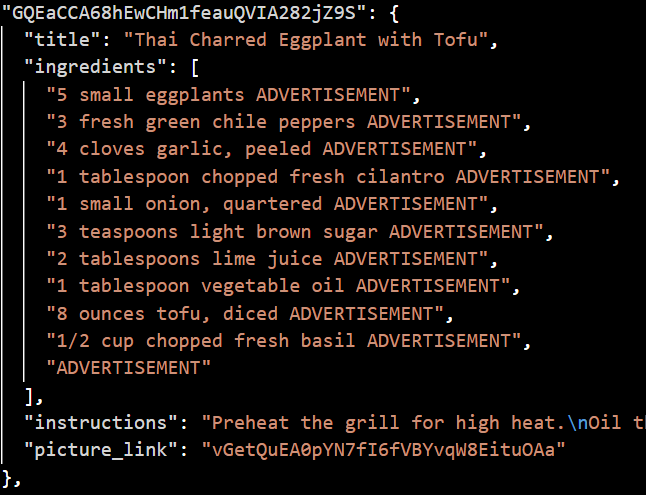
1. Ingredients
2. Instructions
3. Title

These tags are once again found using a well defined ‘Schema’ that refers to the path the algorithm should take to extract inside the source code.

This schema was generated using an opensource library, “recipe\_scrapers” which can be downloaded from pip. To make sure the process of scrapping thousands of webpages was done in a quicker speed, multiprocessing was incorporated with 12 core CPU using the python multiprocessing library.

Finally, the data extracted from these pages is stored inside a JSON file which is later utilized by the machine learning algorithm to train upon.

An example of how the recipes is stored inside the json object is displayed here.



The data extracted was saved inside the objects completely raw, thus a large amount of strictly ruled cleaning was required to make it stable and usable.

1. Data Cleaning and Recipe Recommendation:

**Data Cleaning And Exploration**

The JSON files consists of raw data scrapped directly from the websites, this data accompanies many anomalies compared to what training data should look like. Cleaning of this data was done using python powered by Jupyter Notebook. The libraries used for this includes pandas, NumPy, spacy and regex.

Creation of a combined data frame consisting of all the recipes from the three recipes was to be made, this concatenation was done with pandas. A sample of the combined table is displayed below.

Graphical user interface, text, application

Description automatically generated

With pandas it becomes easy to study the entire data frame for missing values and removing all those rows consisting of the same, these rows were dropped as it can make the whole algorithm go wrong.

Pandas’ inbuilt functions provides a well insight into how many missing values we have obtained in our database.

Text

Description automatically generated

Along with that the picture column was also dropped as it did not offer any useful insight to the algorithm at current stage.

After the removal of all the rows of recipes containing null (or NaN) values, we were left with a total of 122922 recipes with 3 columns each.

This takes care of the rows which are completely missing. The recipes which do not have instructions with proper length are also not much useful to us, this refers to all the recipes with length less than 20 characters. Removing these using python lambda function, we drop down to 122911 recipes.

Graphical user interface, application, Word

Description automatically generated

The next part of cleaning goes into pruning stage, that refers to removal of digits, spacing and punctuations. Also, the term ‘ADVERTISEMENT’ was copied in various lines of ingredients due to the ad sections of the webpages, to remove all theses regex along with python.

Graphical user interface, text, application, email

Description automatically generated

The next stages include parsing the entire textual data together, removal of new lines, tabs and spacings in between, removal of digits and symbols. This is important for the tokenization stage of the project.

As our entire algorithm works on matching documents with appropriate ingredients, its best to combine all three columns of title, ingredients, and instruction into singular textual components.

After all the stages of cleaning the final clean text for ‘Brown Sugar Meatloaf’ is presented as:

A picture containing diagram

Description automatically generated

**Tokenization using spacy:**

Tokenization is defined as: “Tokenization breaks the raw text into words, sentences called tokens.”.

Extracting information from text is difficult, thus the system should break the sentences down into smaller chunks that can be worked upon. In example of ‘Brown Sugar Meatloaf’ we will have to split the entire raw cleaned text into words that can be corelated to one another. This corelation helps interpreting the meaning of the text by overviewing the entire sequence of words. There are various NLP techniques to obtain tokenized result, S.A.R.A incorporates the spacy tokenizer method.

Spacy is faster than many other techniques and provides an ease of customization. Spacy also allows us to use tokens of other well-defined models, one of which is ‘en\_core\_web\_sm”, an English pipeline optimized for CPU. Tokenization is a heavy process and usage of multi processors is necessary to make it faster, for this python’s multiprocessing library was used. This is highly useful for indexing and high search results.

Graphical user interface, text, application, email

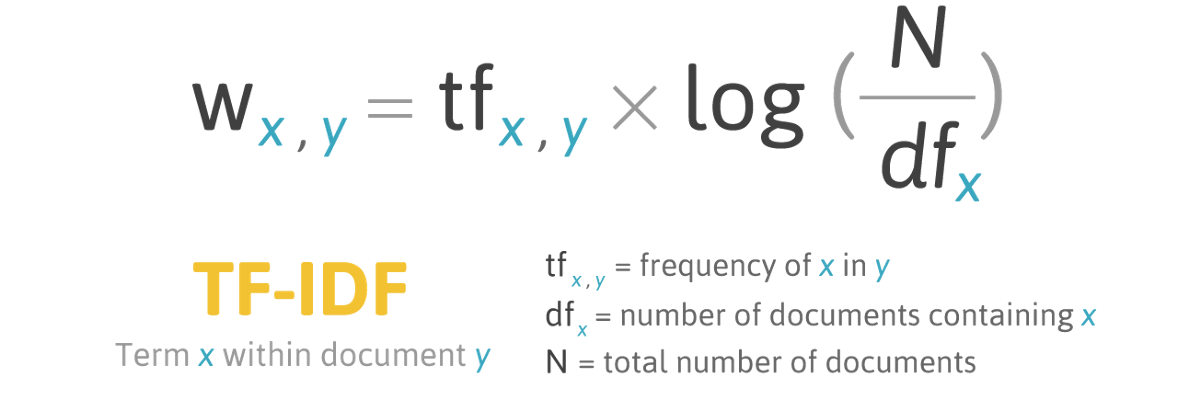
Description automatically generated

To avoid re-computation of the tokenized text, pickle was used to dump it into the fire system and load it whenever needed again.

**Word Embedding:**

Conversion of these blocks of text into something quantitative is required to prepare prediction out of it, word embeddings are used just for the same. Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation. This is useful to understand relations between different words falling under similar meanings.

In the case of recipes, it is required to find the relevance of each word inside its sentence/corpus. This will make it possible for us to match ingredients to certain recipes. The algorithm we have used for the same is Term Frequency-Inverse Document Frequency (TF-IDF). It is the calculation of the relevancy of a word in a series of text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the data-set.



Sklearn provides a method ‘TfidfVectorizer’ that was used to convert the entire tokenized texts into vector form. The vector form is used to describe the text in the bag of term models.

Graphical user interface, text, application, chat or text message

Description automatically generated

The vectorizer used here provides the generic weights to all the topics, and that will relate to the frequency of these topics appearing inside the entire document set.

Text\_tfidf contains the vector form of the entire document and tfidf\_words contain the features that have been obtained via unsupervised learning done with TF-IDF. These vector forms are carried forward for further processing.

**Topic Modelling:**

Topic Modelling refers to the process to automatically identify topics present in a text object and derive hidden patterns exhibited by text corpus. This allows better decision making, search engines and NLP applications to use word embeddings.

Topic Modelling is unsupervised in nature, the patterns that occur out of the corpus is referred as ‘Topics’ in large cluster of texts. In the recipes corpus, the topics comes out as “bakery”,” oil”, “bread” and many more. These topics will be used to map to certain documents working like an indexed database engine. There are two algorithms that can be used for topic modelling which are well received by the community:

1. LDA: Latent Dirichlet Allocation
2. NNMF: Non-negative matrix factorization

Table

Description automatically generated with medium confidenceWorking of LDA: LDA, or Latent Derelicht Analysis is a probabilistic model, and to obtain cluster assignments, it uses two probability values: P( word | topics) and P( topics | documents). These values are calculated based on an initial random assignment, after which they are repeated for each word in each document, to decide their topic assignment. In an iterative procedure, these probabilities are calculated multiple times, until the convergence of the algorithm.

The topics extracted with their probability from some of the recipes are listed down below:

Working of NMF: Non-negative Matrix Factorization is a Linear-algebraic model, that factors high-dimensional vectors into a low-dimensionality representation. NMF takes advantage of the fact that the vectors are non-negative. By factoring them into the lower-dimensional form, NMF forces the coefficients to also be non-negative. NNMF works based on matrices.

If the original matrix is denoted as **A,** we can obtain two matrices **W** and **H**, such that A= W.H

A: input that contains which words appear in which documents

W: the topics discovered from the document

H: the weights for the topics in each document

The calculation of W and H is done over an optimizing process inside an objective function, updating both W and H iteratively until convergence:

Inside this function, we measure the error of reconstruction between A and product of its factors W and H, based on Euclidean distance. For our case, A will be the entire set of recipe text, W will be the topics extracted from it and H will be the weights given to these topics.

Table

Description automatically generated with medium confidenceSet of topics with score for NNMF process is given as

Choosing between LDA and NNMF can be done based on score and stability of convergence. To obtain better insight on the score and performance, graphs for the three topics were displayed next to each other.

The first set of graphs compare the number of documents to be chosen for training purposes per step, we test it on the three topics taken for example above. This will be taken as a cut-off of documents for a singular topic:

Document Score refers to the number of accurate prediction one topic can do for x number of documents.

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Observing both NNMF and LDA’s graphs for Document Score it is visible that in 2000 falls in middle of the document number where a spike upwards (uptick) or level off can be seen. It can be seen as the document score in increases, the range of document being covered by the documents starts falling. To find a more precise number we compare each of these document score with their respective topics.

This is done in descending order to find the convergence and distinction in predictions.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

As per the graphs and outputs of the three test topics, NMF was chosen as the algorithm. The key reasons for the same were as follows:

1. NNMF required lesser computational power on the workbench when training and provided ability to train for higher epochs.
2. NNMF was used as it provided a good convergence and had more distinction on topics.
3. NNMF showcases a levelling out on 200 documents, therefore as per the objective function 200 was chosen as the document cut off.

**Modelling:**

Once all the data is ready for modelling, we set the parameters for it with respect to the studies and analysis done.

* The number of topics per corpora **50**
* The number of top documents within each topic to extract keyword **200**
* The number of keywords to extract from each topic **25**
* The number of top documents within each topic to tag **2000**
* The Length of word-radius that defines the neighbourhood for each word in the text rank adjacency table **4**

Now the system must distribute weight between title, text, and categories to make the searching adequate.

The chosen weights for each of them are as follows:

* Weight for title: **0.2**
* Weight for text: **0.3**
* Weight for categories: **0.5**

A list of stop words were defined that can be utilized by the model to understand where to finish sentences, these can be manually altered.

Using these parameters, the algorithms generate tags and distributes them to their respective documents for all the receives and gets index for them in the original pandas data frame, the topics are first transformed into matrix followed by generation of keyword index. This is done for all the recipes and key words are extracted, scored, and stored for querying.

These tags are stored for each recipe with index, for reduction of re-computation this data is saved inside a csv file. A sample head of this table is shown below



**Querying algorithm:**

For Querying these tags, the algorithm takes in input from the user in form of string of ingredients separated by spaces. These ingredients are then sent into the same TF-IDF vectorizer that was utilized to train the algorithm in the beginning, this is done to map the relationship between the queried ingredients with the recipes’ keywords.

The output of the vectorized query is sent into a function to calculate the overall score of the keywords, using the weights for title, text, and categories.

Graphical user interface, text, application

Description automatically generatedThese relations and scores are sent to the pandas series containing all the recipe scores that matched with the queried keywords. The top three from this series is returned and displayed to the user.

1. Deployment:

**Flask Backend**

To make this query function accessible to a simple function for the client side a microservice was designed in python using the flask framework. Flask has tools, libraries and technologies that allows functions to run independently on a server.

The sytem’s query algorithm was separated from the training code and put together inside of a flask routing function, this converts the function to an application program interface (API).

Text

Description automatically generatedAn API is a set of routines, protocols and tools that are used for building software applications and making them highly interactive and easy to integrate.

This flask server can be made accessible anywhere in the internet by the power of hosting. Amazon Web Service’s Lambda function will properly suffice the role of hosting this function without any troubles.

The flask Server loads the csv file containing the tokenized and tagged CVSs of the recipes and re-runs the query function on the parameters sent to the route.

**Next.js Front-End Prototype:**

For the prototype and rendering of recipes on a web application Vercel’s Next.js was used.

Next.js is a JavaScript framework built over React as a production ready framework, covered with features like:

1. SEO Optimization
2. Automatic Code Splitting
3. Server Side Rendering and much more

Axios, a library made in JavaScript to interact with web protocols was used to send a get request to the flask server.

A React hook was used to obtain the query parameters, in this case the ingredients, and the returning json objects containing recipes were dynamicly displayed on the website.

Google Search Engine API was utilized to obtain images from the recipes name directly from Google Images.