

Hidden Markov Based Mathematical Model dedicated to Extract Ingredients from Recipe Text

Zied Baklouti

▶ **To cite this version:**

Zied Baklouti. Hidden Markov Based Mathematical Model dedicated to Extract Ingredients from Recipe Text. 2021. hal-03355929

HAL Id: hal-03355929
https://hal.science/hal-03355929

Preprint submitted on 28 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Hidden Markov Based Mathematical Model dedicated to Extract Ingredients from Recipe Text

Zied Baklouti*†

March 2021

Abstract

Natural Language Processing (NLP) is a branch of artificial intelligence that gives machines the ability to decode human languages. Part-of-speech tagging (POS tagging) is a pre-processing task that requires an annotated corpus. Rule-based and stochastic methods showed remarkable results for POS tag prediction. On this work, I performed a mathematical model based on Hidden Markov structures and I obtained a high-level accuracy of ingredients extracted from text recipe with performances greater than what traditional methods could make without unknown words consideration.

1 Introduction

Artificial intelligence had shown a great progress in the recent years especially the deep learning branch where learning techniques have been improved very quickly. The combination of representation learning and deep learning have allowed the emerging of a new AI class called deep reinforcement learning.

Deep Reinforcement learning tend to estimate value functions from experiments and simulations and using dynamic programming through Deep Reinforcement learning is an efficient way to build reactive strategies acting on instantaneous control. An algorithm which approves its performance by experience is an algorithm capable of avoiding his own mistakes through a combination of a strong memory fed by fresh helpful data and the ability to keep winning predictions after a long-term performance (Barto, Bradtke, & Singh, 1995) (Mnih et al., 2015).

Neural Network can be considered as a dynamic Reinforcement Learning scheme where the layers are putted in a parallel way to have a cascaded transmission of the treated signal (Fukushima & Miyake, 1982) (LeCun et al., 1989)

*Corresponding author: ziedbaklouti@outlook.com
†MSc computer science and mathematics, Paris University

and where a prior knowledge is important to predict the output state of new observations.

Sequential modeling is a way to process data in natural language processing by maximizing awards after manipulating situation and producing resulting actions (Vithayathil Varghese & Mahmoud, 2020) (LeCun, Bengio, & Hinton, 2015). A sequential model representation is influenced by its data representation and how tensors are trained to produce an optimal control (Bengio, Courville, & Vincent, 2013)

To improve the target learning task, transfer learning is used as a powerful technique to increase the value of the most probable cases inside a state matrix (Boutsioukis, Partalas, & Vlahavas, 2011). Transferring the knowledge helps us to reduce the amount of data consumed and rely on feature engineering to reduce the noise caused by annotation errors and other tag-set anomalies in a context of multi-agent system.

Extracting ingredients automatically from a recipe text is an extremely useful activity especially when we want to analyze a massive data of text recipes. Rule-Based methods were implemented to extract information from unstructured recipe data (Silva, Ribeiro, & Ferreira, 2019) Ingredients is not the only useful information we want to extract; in this work we are going to use Hidden Markov Models especially Viterbi algorithm with some modification to make it receiving two unique features: POS-tags and tokens, to predict ingredient states.

2 Previous works

Many previous works were interested in analyzing cuisine recipes, for example Sina Sajadmanesh (Sajadmanesh et al., 2017) presented an analysis of the ingredients diversity around the word using an ingredient-based classifier to differentiate between recipes around the word based on its geographical identity. Sina Sajadmanesh (Sajadmanesh et al., 2017) studied the diversity of ingredients in dishes with introduction of global diversity (the ability to have diversified ingredients between recipes) and local diversity (the ability to have diversified ingredients within a recipe).

Other related work for culinary habits is Yong-Yeol Ahn paper (Ahn, Ahnert, Bagrow, & Barabási, 2011) who introduced the notion of Flavor Network and tried to verify the Food Paring hypothesis introduced on the 90's by Heston Blumenthal and Francois Benzi. Flavor network as described by Ahn is a graph where the nodes are the ingredients extracted from recipes and weights are shared flavors between nodes. Food paring hypothesis is an indicator calculated after forming the Flavor Network to show if in a country or in a geographical part of the word we have tasty recipes or the ingredients do not have similar molecules. Tiago Simas (Simas, Ficek, Diaz-Guilera, Obrador, & Rodriguez, 2017) introduced the notion of food bridging formed with semi-metric distances.

A group of scientists in a recent publication (Van Erp et al., 2021) developed a state of the art of the use of artificial intelligence and natural language

processing in analyzing food recipes. In this article we can found collected references talking about the challenging part in collecting food and recipe data. For example, Ahnert (Ahnert, 2013) presented the emergence of computational gastronomy in food science and its effect on culinary practices. Aiello and al (Aiello, Schifanella, Quercia, & Del Prete, 2019) discovered what are the most important predictors in food responsible of three diseases in a population situated in London. (Amato & Cozzolino, 2020) extracted ingredients from food text to alert readers from allergens presence in a recipe. I agree with (Van Erp et al., 2021) concerning how challenging to use IA in food domain and how it will resolve issues concerning the creation of a data driven analysis of nutrition. In our paper data extracted can be used in a phone application or a recommended system for people who want to take care of their health.

All previously cited researches on cuisine recipes need information extraction from text recipe to use it on graphical visualization and statistical analysis. Information extraction can be used manually by extracting ingredients indicated on recipes or automatically. The problem in automatic extraction is that information should be precise to have also precise analysis, for example some ingredients take only one word and others can take two or three words. Another problem on automatic information extraction is that some ingredients that take one word have in common some words with other ingredients that have more than one word which make automatic information extraction more difficult. I tried to develop a mathematical model dedicated to extract ingredients from text recipe written in Arabic language with precision higher than what traditional methods could make. According to Cutting (Cutting, Kupiec, Pedersen, & Sibun, 1992), a Tagger must be robust that should deal with unknown words, efficient that can deal with large corpora, accurate that can tag with high accuracy, tunable that can deal with different corpora and reusable that take small efforts to re-target a new corpus. There are three types of POS Tagger: Taggers based on stochastic models, Taggers based on rules and Taggers Based on neural networks. On this work we will use Taggers based on HMM models. The use of POS tags as external features to solve NER problems was experimented by Zhou (Zhou & Su, 2002) but it was discarded because it showed bad results but our methodology and experiments demonstrate that using POS tags as external features is not a bad idea. This could be explained by the difference between our tokens and Zhou's tokens: tokens as defined by Zhou is a pair of word-feature and in our model, token is only a word from our corpus.

3 Training the dataset

Hidden Markov Models are used in previous works to resolve named entity recognition (NER) problems. In our case we have a NER problem with one Named Entity to extract (Ingredients) and with various boundaries (one word, two words, three words).

Our sentences belong to only one category of phrases on Arabic language: The noun phrase "الجمل الاسمية". It is a kind of phrase that don't begin with

verbs (phrase that don't describe an action).

Noun phrases are constituted with 2 parts: The beginning of a sentence "المبتدا" contains the object of the information, and the end of a sentence "الخبر" containing the core information.

Table1 shows how the dataset is trained:

English translation	Tokens in arabic	Label	detected ingredients
a little of	رشة	C	0
salt	ملح	D	1
and	و	J	0
black (اسود)	فلفل	E	1
pepper (فلفل)	اسود	F	2
.	.	.	0

Table 1: Example of trained sentence in our trained corpus

When an ingredient is found on a corpus, we attribute 1 to the concerned token. If the information we want to extract takes more than one word we attribute 2 to the extra word completing the information. $0, 1, 2, 3$ is the new list of tags used on the HMM tagger in the second phase of our ingredient extractor system. Our word dictionary is constituted with 807 words. The number of trained sentences is 1973. The number of POS tags appearing on the first layer of our ingredient extractor is 14 different POS tags. The sentences are describing ingredients constituting a cuisine recipe. Table 1 shows an example of a trained sentence. The entire trained dataset and the code used for this paper is available in this gethub link*.

4 Methodology

Our Ingredient extractor is constituted with two parts. In the first part we are going to predict part of speech tags of our sequences. On the second part we are going to use predicted Part of Speech as a grammatical feature to predict ingredients state.

4.1 Grammatical Feature Collection

In our Model we have observation sequences which are sentences containing the information we want to extract (Ingredients). The first part of our Ingredient Extractor will try to collect grammatical features by resolving the POS tagging problem. At the end of this part, we want to obtain the grammatical role for each token on each sequence. We experimented both first order HMM and second order HMM to predict ingredients state. We obtained strangely better results with first order HMM. This could be explained by the fact that in Arabic

*https://github.com/Zied130/Hidden-Markov-Based-Mathematical-Model

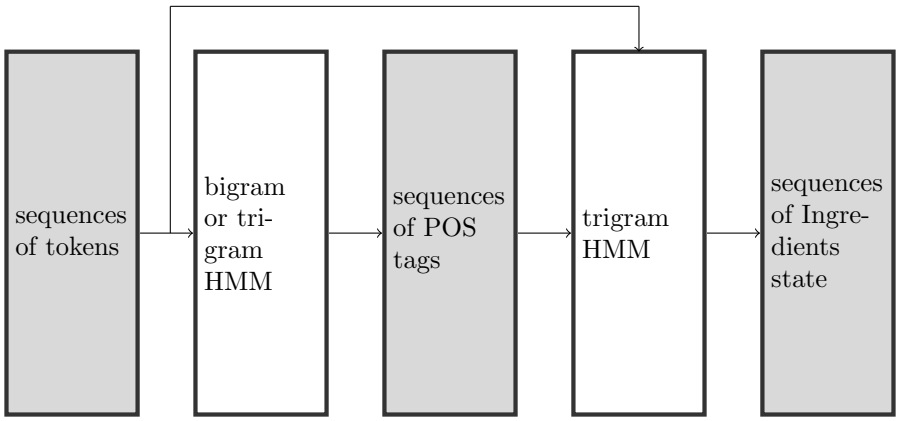


Figure 1: Methodology Diagram of our Ingredient Extractor

language a POS tag is organized in bi-gram manner. For example, if we have an E tag **منعوت** we are certain to have an F tag **نعت** because in Arabic a word tagged as an E tag **منعوت** could be tagged as a noun but if the intention of the writer is to describe that noun with an adjective coming after it the exact tag to be used is an E tag **منعوت**.

POS Tag (arabic)	POS Tag (english)	Label
إسم	Noun	A
رقم	Number	B
إسم معرف	Known Noun	C
إضافة	Noun after Known Noun	D
منعوت	Noun before the Adjective	E
نعت	Adjective	F
حرف جر	Preposition	G
إسم مجرور	Noun after preposition	H
وحدة قيس	unit of measure	I
واو العطف	AND	J
فعل مبني للمجهول	Passive Verb	K
المفعول المطلق	Superlative	L
أداةُ عَطْفٌ غير واو العطف	OR	M
.	.	.

Table 2: POS Tag-set used to annotate our corpus

The elements needed to define a first order HMM in order to perform a POS tag task are:

- N=14, the number of POS tags each token can take as shown in Table 2 we have 14 POS tag. We represent the sequence of POS tags by the

sequence $T = \{t_1, t_2..., t_{14}\}$.

- M=807, is the sequence of symbols generated by the stochastic procedure of the HMM model. In our case it is the list of tokens (words) constituting our lexicon.
- Matrix A containing the state transition probability distribution. In our case it is the probability to move from one POS tag to another POS tag.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

$$a_{ij} = P(T_r = t_j / T_{r-1} = t_i)$$

- Matrix B containing the observation symbol probability distribution. In our case it is the probability to emit a word v_k given that the system is emitting one of the state of the sequence $T = \{t_1, t_2..., t_{14}\}$.

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{bmatrix}$$

$$b_{jk} = P(V_r = v_k / T_r = t_j)$$

- π a vector containing the probability distributions that a sentence begins with one of the elements of T

$$\pi = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \dots & \pi_{14} \end{bmatrix}$$

4.2 Predicting Sequences of Ingredients State

4.2.1 Prediction Using First Order HMM

Suppose that we want to use a first order HMM to predict ingredients state on the second part of our Ingredient Extractor without using grammatical features estimated on previous section, in this case the elements needed to define this HMM according to Rabiner(Rabiner, 1989):

- N=4, the number of distinct states each token can take. In our case it is one element of the set $\Gamma = 0, 1, 2, 3$
- M=807, is the sequence of symbols generated by the stochastic procedure of the HMM model. In our case it is the list of tokens (words) constituting our lexicon.

- Matrix A containing the state transition probability distribution. In our case it is the probability to move from one of the states on the set Γ to another state on the same set.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

$$a_{ij} = P(\Gamma_r = \gamma_j / \Gamma_{r-1} = \gamma_i)$$

- Matrix B containing the observation symbol probability distribution. In our case it is the probability to emit a word v_k given that the system is emitting one of the states of the sequence Γ.

$$B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{bmatrix}$$

$$b_{jv_k} = P(V_r = v_k / \Gamma_r = \gamma_j)$$

- π a vector containing the probability distributions that a sentence begins with one of the elements of Γ

$$\pi = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \pi_4 \end{bmatrix}$$

This model described above is a first-order hidden Markov model, we can call it also a bi-gram model to predict ingredients state.

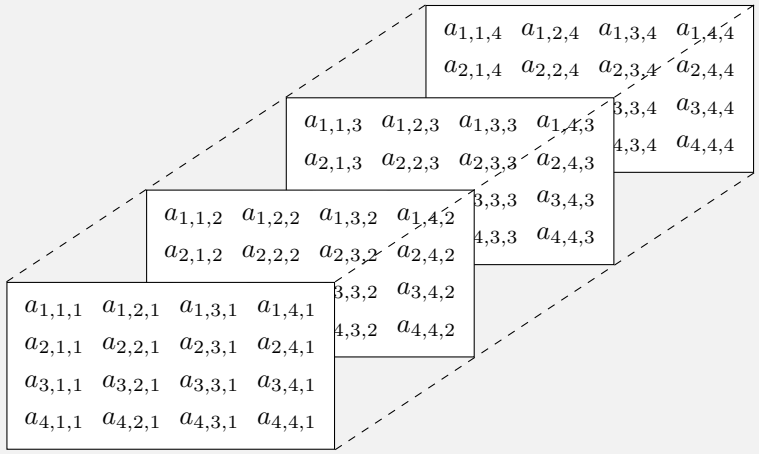
4.2.2 Prediction Using Full Second Order HMM

Scott M.Thede(Thede & Harper, 1999) used a full second-order Hidden Markov Model for his POS tagger. To constitute this trigram tagger from the previous bigram tagger, Scott M.Thede(Thede & Harper, 1999) replaced bi-gram transition probability a_{ij} with a trigram transition probability a_{ijk} and replaced bi-gram lexical probability b_{jv_k} with a trigram lexical probability b_{ijv_k}. If we want to use a full second-order hidden Markov model as described by Scott M.Thede(Thede & Harper, 1999) the element needed to define this HMM are:

- N=4, the number of distinct states each token can take.
- M=807, number of tokens.
- Matrix A shown on figure 2 containing the state transition probability distribution. For a trigram model this matrix is of dimensions $N \times N \times N$.

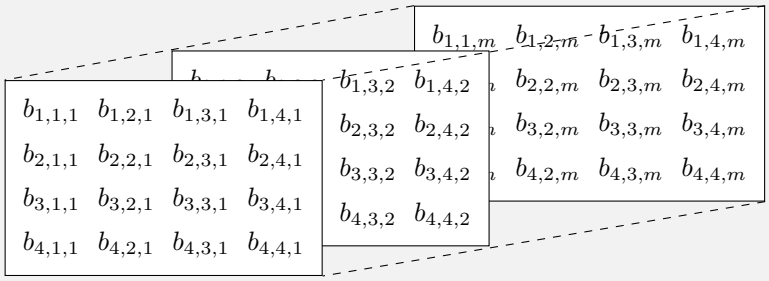
- Matrix B shown on figure 3 containing the observation symbol probability distribution. For trigram model this matrix is of dimensions $N \times N \times M$
- π a vector containing the probability distributions that a sentence begins with one of the elements of Γ

$$\pi = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \pi_4 \end{bmatrix}$$



$$a_{ijk} = P(\Gamma_r = \gamma_k / \Gamma_{r-1} = \gamma_j, \Gamma_{r-2} = \gamma_i)$$

Figure 2: Transition matrix used in full second order HMM



$$b_{ij}(v_k) = P(V_r = v_k/\Gamma_r = \gamma_j, \Gamma_{r-1} = \gamma_i)$$

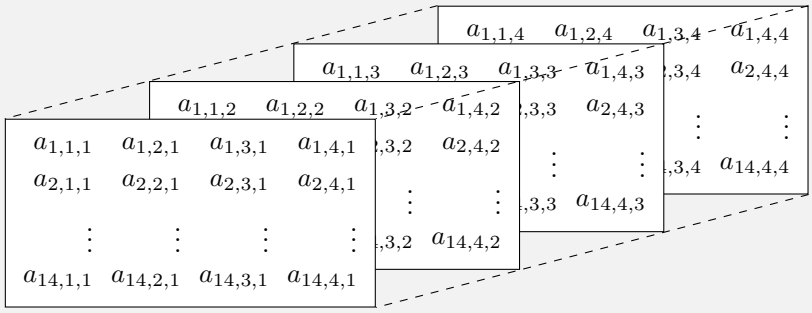
Figure 3: Lexical matrix used in full second order HMM

4.2.3 Prediction with the introduction of collected Grammatical Features

At this point of the research, grammatical features are still not introduced on the model. When we observed the elements constituting the full second-order Hidden Markov Model contextual probabilities and lexical probabilities are formed to include second order information. What we tried to do on this stage is to keep the trigram property of the model but modifying the parameters of the contextual and lexical probabilities to introduce collected grammatical features. The changes made are: the trigram transition probability $a_{ijk} = P(\Gamma_r = \gamma_k/\Gamma_{r-1} = \gamma_j, \Gamma_{r-2} = \gamma_i)$ is replaced by the probability $a_{ijk} = P(\Gamma_r = \gamma_k/\Gamma_{r-1} = \gamma_j, T_{r-1} = \tau_i)$ and the trigram lexical probability $b_{ij}(v_k) = P(V_r = v_k/\Gamma_r = \gamma_j, \Gamma_{r-1} = \gamma_i)$ is replaced by the probability $b_{ij}(v_k) = P(V_r = v_k/\Gamma_r = \gamma_j, T_r = \tau_i)$.The elements of the HMM becomes:

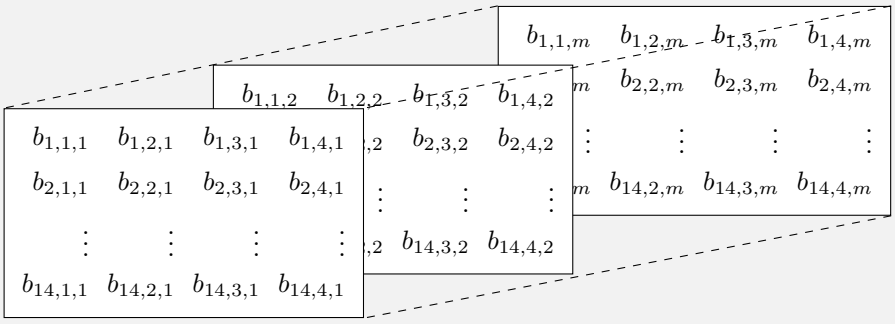
- N=4, the number of distinct elements of the set Γ.
- M=809, number of tokens.
- K=14, the number of distinct POS tags.
- Matrix A containing the state transition probability distribution. This matrix is of dimensions $N \times N \times K$ as shown on figure 4.
- Matrix B containing the observation symbol probability distribution. This matrix is of dimensions $M \times N \times K$ as shown on figure 5.
- π a vector containing the probability distributions that a sentence begins with one of the elements of Γ

$$\pi = \begin{bmatrix} \pi_1 & \pi_2 & \pi_3 & \pi_4 \end{bmatrix}$$



$$a_{ijk} = P(\Gamma_r = \gamma_k / \Gamma_{r-1} = \gamma_j, T_{r-1} = \tau_i)$$

Figure 4: Transition Matrix used in our Ingredient Extractor



$$b_{ij}(v_k) = P(V_r = v_k / \Gamma_r = \gamma_j, T_r = \tau_i)$$

Figure 5: Lexical Matrix used in our Ingredient Extractor

5 Calculating Probabilities for Unknown Words

One common problem in part of speech tagging processing is to predict the hidden state of an unknown word.

Andrei Mikheev(Mikheev, 1996) constructed an automatic technique for learning English part-of-speech guessing rules.

Scott M.Thede(Thede & Harper, 1999) used a Second-Order Hidden Markov

Model for POS tagging by estimating the probability P(word has suffix s_k/tag is t_j)

when he tried to predict the hidden state of an unknown word.

Scott M.Thede(Thede & Harper, 1999) used a trigram tagger respecting the elements that constitute an HMM explained in details by Lawrence R. Rabiner (Rabiner, 1989)

On this work we tried to create a first-order Hidden Markov Model POS Arabic Tagger and we choose the prefix of the word as a word feature rather than word suffix as described by Thede.

After training the corpus we can estimate the probability $P(v_i/t_i)$ by counting the number of times the hidden state t_i was attributed to the word v_i.

For example, we had a corpus constituted of 4 words as represented on the Table 3. We trained the corpus with 3 hidden states and we obtain the discrete state-dependent word probabilities table:

	2	البرتقال	الاكليل	أو
B	1	0	0	0
C	0	0.5	0.5	0
M	0	0	0	1

Table 3: The discrete state-dependent word probabilities table for a mini-corpus

For example, the probability $P(w_i =$**البرتقال**$/t_i = C)$ is estimated by counting the number of times the word **البرتقال** appears with the tag C divided by the number of times C appears on the corpus.

In Arabic as in English, words are constituted morphologically based on its grammatical role on a sentence. This is why suffix and prefix of a word can help us to predict its tag.

Andrei Mikheev (Mikheev, 1996) presented a list of morphological rules for English to predict unknown words. For example, the word we obtain by adding the prefix "un" to a word that was tagged with the tags (VBD) and (VBN) can play the role of an adjective on a sentence.

Such morphological rule can be also applied to Arabic Language. For example, adding "**ال**" to a word that played the role of a noun "**إسم**" or a noun that comes before the adjective "**منعوت**" can be estimated as a Known Noun "**إسم معرف**".

Scott M.Thede (Thede,1998) was inspired by morphological rules described by Mikheev to construct his unknown word predictor. He first created a lexicon to estimate $P(w_i/t_i)$, then he used prefix and suffix of words on this lexicon to predict possible tags. This method can be used to our corpus as in Arabic word feature extraction can be done by the prefix of the words.

For example, the discrete state-dependent word probabilities table for the previous mini-corpus can be represented depending on word prefixes:

	2	ال	ال	أو
B	1	0	0	0
C	0	0.5	0.5	0
M	0	0	0	1

Table 4: The discrete state-dependent word probabilities table for a mini-corpus (prefix-based approach)

We considered the two first characters of a word on this work if a word counts more than two characters and the first character of a word if the word is constituted with only one character. After that a probability distribution for each affix is created by adding the probabilities $P(v_i/t_i)$ for words with the same tag. For example, $P(prefix =$"ال"$/t_i = C) = P(v_i =$البر تقال$/t_i = C) + P(v_i =$الاكليل$/t_i = C)$ the Table 4 becomes:

	2	ال	أو
B	1	0	0
C	0	1	0
M	0	0	1

Table 5: The discrete state-dependent word probabilities table for a mini-corpus after adding the probabilities $P(v_i/t_i)$ for words with the same tag (prefix-based approach)

Table 5 constitute the matrix C used on the Hidden Markov Model algorithm to replace the matrix B when a word is unknown.

	Accuracy	F1 score
First order Hidden Markov Model applied to tokens to predict ingredients	96.64 %	70.07 %
Second order Hidden Markov Model applied to tokens to predict ingredients	95.85 %	68.22 %
First Order Hidden Markov Model applied to tags to predict ingredients	79.35 %	54.36 %
Second Order Hidden Markov Model applied to tags to predict ingredients	87.74 %	58.53 %
Our ingredient extractor with 100 % accuracy on the first layer with $\lambda = 4$	98.44 %	81.44 %
Our ingredient extractor with 90.33 % accuracy on the first layer with 82.03 % F1 score on the first layer with $\lambda = 4$	97.08 %	74.36 %

Table 6: Comparison of performances between our ingredient extractor and other HMM taggers using the trained corpus as a testing dataset to avoid unknown words

6 Results and Interpretation

First, we tested the Modified Viterbi algorithm used in our Ingredient Extractor on the same trained corpus to avoid using matrix C described in Table (5) . Our Ingredient Extractor shows the highest accuracy and F1 score even when accuracy on the first layer don't overcome 91% Table (6). These first results before dividing the dataset into two separate training and testing sides are encouraging and shows that including POS tags inside the calculation of a state matrix isn't a bad idea because the state of an ingredient depend also from the state of the POS tag.

Second, we tested the Modified Viterbi algorithm used in our Ingredient Extractor on a testing dataset different from the trained dataset and we compare the results obtained with the results for a 1^{st} order HMM and a 2^{nd} order HMM using 10-fold cross-validation. We obtain in Table (11) an f1 score of 67.31 % for our IE when the first layer has a 100 % accuracy against 78.92 % F1 score for a 1^{st} order HMM and 76.58 % F1 score for a 2^{nd} order HMM, our F1 score decreased with a percentage of 14.13 % when we split the dataset into a training and a testing dataset for our IE. This decrease on F1 score could be explained by

tag	accuracy	f1 score	accuracy for unknown words	accuracy for known words	number of unknown words	unknown words percentage
0	94.77 %	67.67 %	67.78 %	96.33 %	78	9.67 %
1	93.02 %	65.56 %	63.93 %	95.2 %	107	13.26 %
2	94.15 %	67.44 %	69.9 %	95.81 %	88	10.9 %
3	94.01 %	87.89 %	62.65 %	95.7 %	75	9.29 %
4	93.14 %	87.59 %	67.96 %	94.73 %	92	11.4 %
5	95.78 %	69.61 %	76.4 %	96.83 %	83	10.29 %
6	94.38 %	67.53 %	54.65 %	96.49 %	80	9.91 %
7	95.08 %	91.17 %	61.22 %	97.17 %	90	11.15 %
8	96.83 %	94.04 %	79.76 %	97.7 %	77	9.54 %
9	94.92 %	90.7 %	64.95 %	96.72 %	88	10.9 %
Avg	94.61 %	78.92 %	66.92 %	96.27 %	86	10.63 %

Table 7: Average performances of first order HMM used to predict ingredients state using 10 fold cross-validation and a 80% training dataset and 20 % testing dataset separated

tag	accuracy	f1 score	accuracy for unknown words	accuracy for known words	number of unknown words	unknown words percentage
0	93.72 %	64.13 %	70.0 %	94.81 %	78	9.67 %
1	92.53 %	63 %	64.75 %	94.19 %	107	13.26 %
2	94.14 %	65.6 %	70.87 %	95.39 %	88	10.9 %
3	93.94 %	86.92 %	68.67 %	95.02%	75	9.29 %
4	93.42 %	86.61 %	66.02 %	94.81 %	92	11.4 %
5	93.83 %	65.85 %	62.92 %	95.18 %	83	10.29 %
6	93.76 %	65.63 %	60.47 %	95.19 %	80	9.91 %
7	94.04 %	88.56 %	55.1 %	95.96 %	90	11.15 %
8	95.34 %	90.99 %	60.71 %	96.77 %	77	9.54 %
9	94.34 %	88.56 %	61.86 %	95.91 %	88	10.9 %
Avg	93.91 %	76.58 %	64.14 %	95.32 %	86	10.63 %

Table 8: Average performances of second order HMM used to predict ingredients state using 10 fold cross-validation and a 80% training dataset and 20 % testing dataset separated

tag	accuracy	f1 score	accuracy for unknown words	accuracy for known words	number of unknown words	unknown words percentage
0	95.78 %	67.91 %	61.11 %	97.38 %	78	9.67 %
1	94.38 %	66.54 %	68.85 %	95.92 %	107	13.26 %
2	95.08 %	67.39 %	67.96 %	96.54 %	88	10.9 %
3	95.83 %	67.71 %	62.65 %	97.25 %	75	9.29 %
4	94.83 %	67.72 %	71.84 %	96 %	92	11.4 %
5	95.9 %	69.57 %	70.79 %	97.0 %	83	10.29 %
6	94.43 %	65.92 %	55.81 %	96.08 %	80	9.91 %
7	94.13 %	65.61 %	52.04 %	96.22 %	90	11.15 %
8	95.63 %	68.82 %	60.71 %	97.06 %	77	9.54 %
9	94.39 %	65.95 %	56.7 %	96.21 %	88	10.9%
Avg	95.04 %	67.31 %	62.85 %	96.57 %	86	10.63%

Table 9: Average performances in predicting ingredients states with $\lambda = 4$ when accuracy is 100 % on first layer using 10 fold cross-validation and a 80% training dataset and 20% testing dataset separated

	First Layer first order HMM				Second Layer second order HMM			
Fold	accuracy	f1 score	accuracy unknown words	accuracy known words	accuracy	f1 score	accuracy unknown words	accuracy known words
0	90.33 %	70.61 %	50.55 %	92.14 %	95.43 %	66.98 %	61.54 %	96.97 %
1	87.7 %	73.71 %	52.03 %	89.81 %	93.96 %	65.96 %	70.73 %	95.32 %
2	89.07 %	75.34 %	66.99 %	90.26 %	94.54 %	66.97 %	66.02 %	96.07 %
3	87.87 %	72.36 %	67.47 %	88.75 %	94.93 %	66.93 %	63.86 %	96.27 %
4	87.31 %	73.21 %	63.11 %	88.54 %	93.65 %	66.46 %	72.82 %	94.71 %
5	91.23 %	76.51 %	62.92 %	92.47 %	94.82 %	68.17 %	69.66 %	95.92 %
6	90.53 %	73.87 %	59.3 %	91.86 %	93.86 %	65.16 %	54.65 %	95.92 %
7	88.13 %	78.5 %	61.22 %	90.51 %	93.8 %	65.12 %	55.1 %	95.71 %
8	89.28 %	77.76 %	67.86 %	90.16 %	95.01 %	67.84 %	64.29 %	96.28 %
9	89.4 %	77.29 %	65.98 %	90.53 %	93.96 %	64.98 %	60.82 %	95.57 %
Avg	89.19 %	74.92 %	61.74 %	90.5 %	94.4 %	66.46 %	63.95 %	95.83 %

Table 10: Average performances in predicting ingredients states with various accuracies in the first layer with $\lambda = 4$ when accuracy is 100 % on first layer using 10 fold cross-validation and a 80% training dataset and 20% testing dataset separated t

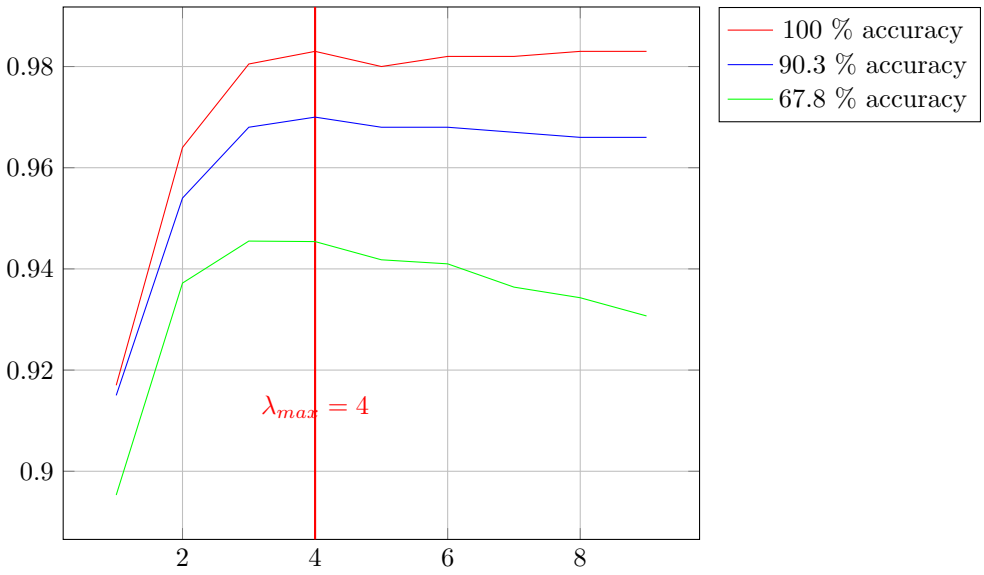


Figure 6: Variations of accuracy of our Ingredient Extractor depending on λ and accuracy score on the first layer

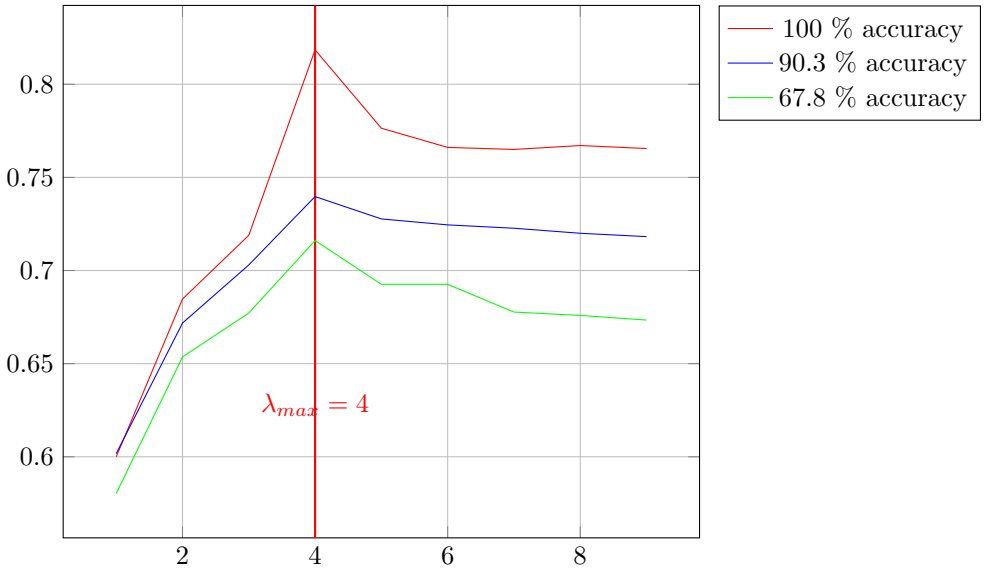


Figure 7: Variations of F1 score of our Ingredient Extractor depending on λ and accuracy score on the first layer

	known	unknown	overall	F1 score
First order Hidden Markov Model applied to tokens	96.27 %	66.92 %	94.61 %	78.92 %
Second order Hidden Markov Model applied to tokens	95.32 %	64.14 %	93.91 %	76.58 %
Our ingredient extractor with 100 % accuracy on the first layer with $\lambda = 4$	96.57 %	62.85 %	95.02 %	67.31 %
Our ingredient extractor with 89.19 % average accuracy on the first layer with 74.92 % average F1 score on the first layer with $\lambda = 4$	95.83 %	63.95 %	94.4 %	66.46 %

Table 11: Comparison of performances between the 3 methods

the change made in the lexical probability when it is confronted with unknown words making its contribution in the state matrix not as efficient as when the entire word is considered in the lexical probability. The hyper parameter λ is eliminated from our IE when the word to be estimated is unknown for a better accuracy and F1 score.

7 Conclusion

Our Ingredient Extractor algorithm showed great results. It is based on HMM methods. We realized it by training two layers: first we trained tokens by tagging POS tags and second, we trained tokens by extracting the ingredients. Our HMM model needed modifications in iteration step because we didn't get a square transition probability matrix or a square lexical probability matrix after the training step. A detailed iterations of our method is illustrated in appendix A. We can ameliorate our model by calculating the probabilities in it differently not as simple as we deed. We can make our model more interesting by adding two layers, one for extracting quantities and the other for extracting unities.

A Detailed Iterations of Modified Second Order Viterbi Algorithm Used in our Ingredient Extractor

1. Predicting Tags for the first Layer:

   We use for predicting tags a first order hidden Markov model as this method have a better accuracy than second order hidden Markov model.

2. Predicting Ingredients state for the second Layer:

   The variables:

   - $\delta_l(i,j) = max_{\tau_1 \ldots \tau_{l-1}} P(\tau_1 ... \tau_l = [t_i, \gamma_j] / v_1 ... v_l, t_1 ... t_l), l = 1...L$
   - $\psi_l(i,j) = argmax_{\tau_1 \ldots \tau_{l-1}} P(\tau_1 ... \tau_l = [t_i, \gamma_j] / v_1 ... v_l, t_1 ... t_l), l = 1...L$

   The procedure:

   (1) Initialization step:

$$\delta_1(i,j) = \begin{cases} \pi_j b_{ij}(v_1) & v_1 \text{ is known i=1..14 j=1..4} \\ \\ \pi_j c_{ij}(v_1) & v_1 \text{ is unknown i=1..14 j=1..4} \end{cases}$$

$$\psi_1(i,j) = 0 \text{ i=1..14 j=1..4}$$

   (2) Iteration step:

$$\delta_l(i,j) = \begin{cases} max_j[\delta_{l-1}(i,j) a_{ijk}] b_{ij}(v_l) & v_l \text{ is known , l=2...L} \\ & \text{length j=1...4, i=1..14} \\ & \text{k=1...4} \\ \\ max_j[\delta_{l-1}(i,j) a_{ijk}] c_{ij}(v_l) & v_l \text{ is unknown , l=2...L} \\ & \text{j=1...4, i=1..14} \\ & \text{k=1...4} \end{cases}$$

$$\psi_l(i,j) = argmax_j[\delta_{l-1}(i,j) a_{ijk}] \text{j=1..4,i=1..14,k=1..4}$$

   (3) Termination:

$$\tau_L^* = argmax_{i=y(L), j=1..4} \delta_L(i,j)$$

   where y(L) is the tag calculated at previous layer in position L

(4) Backtracking:

$$\tau_l^* = \psi_l(y(l), \tau_{l+1}^*) \text{ l=L-1..2,1}$$

where y(l) is the tag calculated at previous layer in position l

3. Predicting Ingredients state for the second Layer in Log-space and introduction of the hyper-parameter λ

The variables:

- $\delta_l(i,j) = max_{\tau_1 \ldots \tau_{l-1}} P(\tau_1 ... \tau_l = [t_i, \gamma_j]/v_1 ... v_l, t_1 ... t_l), l = 1...L$
- $\psi_l(i,j) = argmax_{\tau_1 \ldots \tau_{l-1}} P(\tau_1 ... \tau_l = [t_i, \gamma_j]/v_1 ... v_l, t_1 ... t_l), l = 1...L$

The procedure:

(1) Initialization step:

$$\delta_1(i,j) = \begin{cases} \log(\pi_j) + \log(b_{ij}(v_1)) & v_1 \text{ is known i=1..14 j=1..4} \\ \log(\pi_j) + \log(c_{ij}(v_1)) & v_1 \text{ is unknown i=1..14 j=1..4} \end{cases}$$

$$\psi_1(i,j) = 0 \text{ i=1..14 j=1..4}$$

(2) Iteration step:

$$\delta_l(i,j) = \begin{cases} max_j[\delta_{l-1}(i,j) + \log(a_{ijk})] + \lambda_{max} \log(b_{ij}(v_l)) & v_l \text{ is known l=2..L} \\ & \text{j=1...4, i=1..14} \\ & \text{k=1...4} \\ max_j[\delta_{l-1}(i,j) + \log(a_{ijk})] + \lambda_{max} \log(c_{ij}(v_l)) & v_l \text{ is unknown} \\ & \text{l=2...L j=1..4} \\ & \text{i=1..14 k=1..4} \end{cases}$$

$$\psi_l(i,j) = argmax_j[\delta_{l-1}(i,j) + \log(a_{ijk})] \text{ j=1..4,i=1..14,k=1..4}$$

(3) Termination:

$$\tau_L^* = argmax_{i=y(L), j=1..4} \delta_L(i,j)$$

where y(L) is the tag calculated at previous layer in position L

(4) Backtracking:

$$\tau_l^* = \psi_l(y(l), \tau_{l+1}^*) \text{ l=L-1..2,1}$$

where y(l) is the tag calculated at previous layer in position l

List of Figures

1 Methodology Diagram of our Ingredient Extractor 5
2 Transition matrix used in full second order HMM 8
3 Lexical matrix used in full second order HMM 9
4 Transition Matrix used in our Ingredient Extractor 10
5 Lexical Matrix used in our Ingredient Extractor 10
6 Variations of accuracy of our Ingredient Extractor depending on λ and accuracy score on the first layer 16
7 Variations of F1 score of our Ingredient Extractor depending on λ and accuracy score on the first layer 16

List of Tables

1 Example of trained sentence in our trained corpus 4
2 POS Tag-set used to annotate our corpus 5
3 The discrete state-dependent word probabilities table for a mini-corpus 11
4 The discrete state-dependent word probabilities table for a mini-corpus (prefix-based approach) 12
5 The discrete state-dependent word probabilities table for a mini-corpus after adding the probabilities $P(v_i/t_i)$ for words with the same tag (prefix-based approach) 12
6 Comparison of performances between our ingredient extractor and other HMM taggers using the trained corpus as a testing dataset to avoid unknown words 13
7 Average performances of first order HMM used to predict ingredients state using 10 fold cross-validation and a 80% training dataset and 20 % testing dataset separated 14
8 Average performances of second order HMM used to predict ingredients state using 10 fold cross-validation and a 80% training dataset and 20 % testing dataset separated 14
9 Average performances in predicting ingredients states with $\lambda = 4$ when accuracy is 100 % on first layer using 10 fold cross-validation and a 80% training dataset and 20% testing dataset separated . 15
10 Average performances in predicting ingredients states with various accuracies in the first layer with $\lambda = 4$ when accuracy is 100 % on first layer using 10 fold cross-validation and a 80% training dataset and 20% testing dataset separated t 15
11 Comparison of performances between the 3 methods 17

References

Ahn, Y.-Y., Ahnert, S. E., Bagrow, J. P., & Barabási, A.-L. (2011). Flavor network and the principles of food pairing. *Scientific reports*, *1*, 196.

Ahnert, S. E. (2013). Network analysis and data mining in food science: the emergence of computational gastronomy. *Flavour*, *2*(1), 1–3.

Aiello, L. M., Schifanella, R., Quercia, D., & Del Prete, L. (2019). Large-scale and high-resolution analysis of food purchases and health outcomes. *EPJ Data Science*, *8*(1), 14.

Amato, A., & Cozzolino, G. (2020). Safeeat: Extraction of information about the presence of food allergens in recipes. In *International conference on intelligent networking and collaborative systems* (pp. 194–203).

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial intelligence*, *72*(1-2), 81–138.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, *35*(8), 1798–1828.

Boutsioukis, G., Partalas, I., & Vlahavas, I. (2011). Transfer learning in multi-agent reinforcement learning domains. In *European workshop on reinforcement learning* (pp. 249–260).

Cutting, D., Kupiec, J., Pedersen, J., & Sibun, P. (1992). A practical part-of-speech tagger. In *Proceedings of the third conference on applied natural language processing* (pp. 133–140).

Fukushima, K., & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets* (pp. 267–285). Springer.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541–551.

Mikheev, A. (1996). Learning part-of-speech guessing rules from lexicon: Extension to non-concatenative operations. In *Proceedings of the 16th conference on computational linguistics-volume 2* (pp. 770–775).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286.

Sajadmanesh, S., Jafarzadeh, S., Ossia, S. A., Rabiee, H. R., Haddadi, H., Mejova, Y., ... Stringhini, G. (2017). Kissing cuisines: Exploring worldwide culinary habits on the web. In *Proceedings of the 26th international conference on world wide web companion* (pp. 1013–1021).

Silva, N., Ribeiro, D., & Ferreira, L. (2019). Information extraction from unstructured recipe data. In *Proceedings of the 2019 5th international conference on computer and technology applications* (pp. 165–168).

Simas, T., Ficek, M., Diaz-Guilera, A., Obrador, P., & Rodriguez, P. R. (2017). Food-bridging: a new network construction to unveil the principles of cooking. *Frontiers in ICT*, *4*, 14.

Thede, S. M., & Harper, M. P. (1999). A second-order hidden markov model for part-of-speech tagging. In *Proceedings of the 37th annual meeting of the association for computational linguistics on computational linguistics* (pp. 175–182).

Van Erp, M., Reynolds, C., Maynard, D., Starke, A., Ibáñez Martín, R., Andres, F., ... others (2021). Using natural language processing and artificial intelligence to explore the nutrition and sustainability of recipes and food. *Frontiers in Artificial Intelligence*, *3*(621577).

Vithayathil Varghese, N., & Mahmoud, Q. H. (2020). A survey of multi-task deep reinforcement learning. *Electronics*, *9*(9), 1363.

Zhou, G., & Su, J. (2002). Named entity recognition using an hmm-based chunk tagger. In *proceedings of the 40th annual meeting on association for computational linguistics* (pp. 473–480).