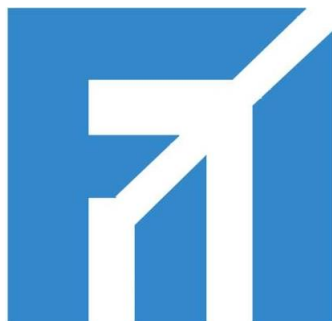


UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

# Tehnici de clasificare text în domeniul culinar

propusă de

*Mircea Rareș - Gabriel*

Sesiunea: *iulie, 2019*

Coordonator științific

**Conf. Dr. Răschip Mădălina**

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

**Tehnici de clasificare text în  
domeniul culinar**

propusă de

***Mircea Rareș - Gabriel***

Sesiunea: *iulie, 2019*

Coordonator științific

**Conf. Dr. Răschip Mădălina**

## Contents

<b>1</b>	<b>Descrierea problemei</b>	<b>2</b>
1.1	Metode existente . . . . .	2
1.2	Setul de date . . . . .	3
<b>2</b>	<b>Reprezentarea</b>	<b>4</b>
2.1	Bag-of-Words . . . . .	4
2.2	Tf-Idf . . . . .	5
2.3	Word-2-Vec . . . . .	5
<b>3</b>	<b>Algoritmii de clasificare</b>	<b>6</b>
3.1	Linear SVC . . . . .	6
3.2	Regresia Logistica . . . . .	6
3.3	Random Forrest . . . . .	9
<b>4</b>	<b>Descrierea metodei proprii</b>	<b>10</b>
4.1	Setul de date . . . . .	10
4.2	Procesarea datelor . . . . .	11
4.3	Utilizarea algoritmilor . . . . .	11
4.4	Reprezentări grafice . . . . .	11
4.4.1	UMAP . . . . .	11
4.4.2	T-SNE . . . . .	11
4.5	Limbajul folosit . . . . .	11
4.6	Biblioteci folosite . . . . .	11
4.6.1	Sklearn . . . . .	11
4.6.2	BeautifulSoup . . . . .	12
4.6.3	Gensim . . . . .	12
4.6.4	Pandas . . . . .	12
4.6.5	Plotlib . . . . .	12
4.7	Rezultatele . . . . .	12
<b>5</b>	<b>Concluzii</b>	<b>13</b>
<b>6</b>	<b>Bibliografie</b>	<b>14</b>

# 1 Descrierea problemei

Mâncarea stă la baza civilizației umane, face parte din fundația piramidei nevoilor a lui *Maslow* fiind una dintre nevoile de bază a omului. Încă din faza incipientă a omului, hrana a fost una dintre țintele de bază a societății, împreună cu adăpostul și reproducerea speciei. De-a lungul timpului, noi am evoluat, am migrat pe toate cele 7 continente, unde au apărut civilizații care mai de care diferite, acestea venind fiecare cu o anumită perspectivă asupra mâncării.

Perspectiva asta a rămas neschimbata odată cu trecerea timpului, fiecare civilizație rămânând fidelă ei, anii aducând mici îmbunătățiri la nivel culinar, sau chiar rețete noi. Dacă până la sfârșitul secolului *XX* umanitatea încă era separată de ziduri și cortine, odată cu intrarea în mileniul *III*, a început și procesul de globalizare, unde mai fiecare cultură mai bogată a reușit să exporteze niște influențe culinare celorlalte.

Având un număr generos de bucătării disponibile și pentru fiecare dintre acestea cel puțin 200 de rețete, m-am decis să realizez o clasificare pe text a acestor rețete și în funcție de modul de preparare a fiecăreia să o clasific cât mai corect.

Pentru a face asta, mă voi folosi de limbajul Python, care pune la dispoziție o suită de biblioteci pentru Machine Learning, cum ar fi *scikit*, de unde am folosit *sklearn* pentru algoritmii de extragere de trăsături și cei de clasificare, *matplotlib* pentru graficele care o să-mi vină în ajutor la descrierea problemei, *pandas* pentru a lucra mai ușor cu fișierele de tip *json* și în ultimul rând, *BeautifulSoup*, pe care l-am folosit la crearea setului de date, extragând date de pe internet.

Scopul lucrării este de a folosi acești algoritmi și biblioteci în a calcula acuratețea în domeniul clasificării text pe rețete, astfel încât nu ar mai trebui implementați de la 0, unde s-ar fi pus și problema corectitudinii și a eficienței.

## 1.1 Metode existente

După o căutare riguroasă în domeniul clasificării rețetelor, am găsit unele lucrări de specialitate dar care propuneau fie clasificarea rețetelor în funcție de ingredientele care le conțineau iar altele propuneau o clasificare a rețetelor în funcție de imagini, cu ajutorul rețelelor neuronale convoluționale. Au fost unele lucrări, care deși lucrau cu rețete, propuneau cu totul altă soluție, fapt ce nu mă ajuta prea mult.

Trebuie precizat că cele care făceau clasificare text pe ingredientele rețetelor, m-au ajutat să vizualizez și să înțeleg pașii necesari în realizarea acestei lucrări. Dat fiind faptul că aceste lucrări foloseau seturi de date cât mai diferite unul de celalalt, am fost nevoit să consider crearea propriului meu set de date, unde voi putea face clasificare text pe pașii rețetei, sau instrucțiunile sale.

## 1.2 Setul de date

Având în vedere că nu puteam să folosesc niciun dataset din lucrările enumerate mai sus, niciunul nefiind conform specificațiilor impuse de mine, am decis să utilizez un crawler pe un site de rețete, unde puteam parcurge paginile acestui site și să-mi extrag de pe fiecare rețetă în parte ce aveam nevoie.

O rețetă e era reprezentată de o pagină nouă, de unde îmi extrăgeam instrucțiunile de preparare, ingredientele și numele acesteia, le puneam într-o listă de dicționare, pe care la final o exportam într-un fișier JSON. Spre finalul acestui procedeu am ajuns la o sumă de aproximativ 5000 rețete de la 15 bucătării, având cel puțin vreo 150 de rețete pe fiecare bucătărie în parte

## 2 Reprezentarea

Pentru reprezentare, am folosit algoritmi din categoria de extragere de trăsături (*Information Retrieval*) cum ar fi Tf-Idf, Bag-of-Words și Word-2-Vec. Această extragere de trăsături constă într-o căutare de informații la nivelul unei propoziții, text, document și o suită de documente care se concretizează într-un așa zis *corpus*.

Având în vedere faptul că acești algoritmi au același scop, aceștia au și cam același parcurs în execuția lor. La început, după ce se face inițializarea lor, se trece la etapa de tokenizare, în care fiecare cuvânt este considerat un token și îi este atribuit un id (spațiile libere și semnele de punctuație sunt considerate separatori).

După etapa de tokenizare, se trece la etapa de numărare unde fiecare algoritm își aplică propria metodă de calcul, de exemplu, Bag-of-Words numără frecvența cuvintelor într-un text, după care se trece la etapa finală de vectorizare, unde se transformă aceasta colecție de texte într-un vector de trăsături a acestora.

### 2.1 Bag-of-Words

În primul rând, ar trebui explicată denumirea acestui algoritm; Ea provine de la faptul că fiecare text, cum ar o propoziție sau frază, este reprezentată de un multiset (*bag*) de cuvinte, indiferent de gramatică sau ordine, în care se păstrează valorile într-o structură de tip dicționar (cheie - valoare), unde cheia este cuvântul în sine, iar valoarea este frecvența acestui cuvânt în textul respectiv.

Exemplu: *Ana are mere și George are pere.*, output-ul *bag - of - words* va fi:

{Ana: 1, are: 2, mere: 1, si: 1, George: 1, pere: 1}

În cadrul bag of words, este introdus și termenul de *n - gram*, care încearcă grupeze cuvintele din text în perechi de cuvinte succesive de câte *n* și va număra frecvența acestora din *n - grame*. Pentru a înțelege mai ușor conceptul, voi exemplifica mai jos, folosind exemplul anterior:

Exemplu: *Ana are mere și George are pere.*, bi-gramele (2-gramele) vor fi:

{Ana are,  
are mere,  
mere și,  
și George,  
George are,  
are pere}

## 2.2 Tf-Idf

Algoritmul tf-idf provine în engleză de la "*term frequency-inverse document frequency*", adică acesta vrea să arate cât de important este un cuvânt într-un corpus, astfel încât valoarea tf-idf, este proporțională cu frecvența cuvântului în text și invers proporțională cu numărul de texte din corpus care conțin acel cuvânt.

Pentru a putea înțelege mai bine acest algoritm, ne uităm la formulele matematice a celor două componente care îl alcătuiesc.

Începem cu "*term-frequency*" :

$$tf(t, d) = f_{t,d}$$

unde  $f_{t,d}$  se traduce în frecvența cuvântului respectiv,  $t$ , în textul  $d$ .

Apoi avem "*inverse document frequency*":

$$idf(t, D) = \log \frac{N}{1 + |\{d \in D : t \in d\}|}$$

, unde  $N$  este numărul de texte în corpus și  $|\{d \in D : t \in d\}|$  este numărul de texte în care apare tokenul  $t$ , în cazul în care tokenul nu se va regăsi în niciun text, acea formula va da 0, astfel fiind nevoie de a adăuga 1+ pentru a nu avea nicio împărțire la 0.

Punând cap la cap formulele de mai jos, va rezulta formula finală la tf-idf, aceea fiind:  $tfidf(t, d, D) = tf(t, d) \times idf(t, D)$ .

## 2.3 Word-2-Vec

După cum spune și numele, acest algoritm preia un corpus de texte (*Word*) pe care îl transformă într-un spațiu vectorial (*2-Vec*), reprezentat de vectorii de trăsături a acelor cuvinte din corpus. Vectorii de trăsături care împart trăsături comune, sunt plasate de obicei cât mai aproape unul de celălalt.

În implementarea sa, sunt folosite rețelele neuronale cu două straturi

### 3 Algoritmii de clasificare

Următoarea etapă, de după cea de extragere de trăsături, este cea de clasificare. Aceasta clasificare se face în baza vectorilor de trăsături rezultați la etapa anterioară, unde aplicăm și setului de antrenare și setului de testare transformarea specifică fiecărui algoritm de extragere de trăsături.

#### 3.1 Linear SVC

Pentru a putea discuta despre Linear SVC (*Support Vector Classifier*), va trebui să intrăm puțin în detalii cu SVM (*Support Vector Machines*). Acesta este un algoritm de învățare supervizată folosit în separa instanțele aparținând unor clase diferite într-un spațiu multidimensional.

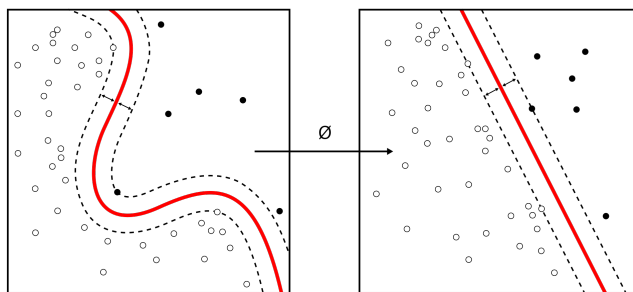
Mai sus am utilizat termenul separare, deoarece output-ul SVM-ului este reprezentat de un hiperplan, sau mai multe, cu ajutor cărora putem separa valori care ar părea imposibil de separat într-un spațiu bi-dimensional.

Find vorba de linearitate, SVM se folosește de hiperplan cu margine-maximă, adică acesta va căuta cele mai apropiate valori din două clase diferite pe care va construi hiperplanul, maximizându-se în așa fel distanța dintre cele două clase. Valorile folosite pentru a stabili acest hiperplan, sunt numite *vectori suport*.

Am ales acest algoritm datorită faptului că este eficient în spații cu multe dimensiuni, deoarece în timpul extragerii de trăsături, se poate ajunge și la valori mari (unde va pe la 3500).

Linear SVC este diferit de SVM prin două caracteristici, prima fiind cea că acest algoritm folosește o funcție kernel lineară și că poate primi și vectori de trăsături denși sau rari, fapt care ne va fi de ajutor mai încolo când vom trece la etapa de implementare.

Mai jos este un exemplu de hiperplan în SVM.

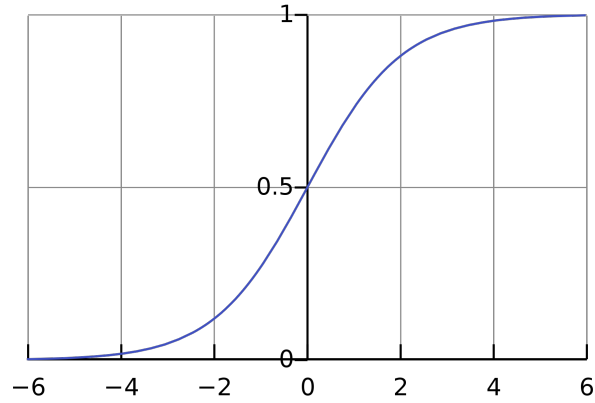


#### 3.2 Regresia Logistică

Regresia logistică este un model care se bazează pe funcția logistică, sau sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$





Acest algoritm măsoară relația dintre variabilele categoric dependente și una sau mai multe variabile independente estimând probabilitățile folosind o funcție logisitică. Variabilele depedente sunt reprezentate de clasa țintă pe care noi trebuie să o prezicem, așadar varibilele independente sunt trăsăturile de care ne vom folosi pe a prezice clasa țintă.

Având în vedere faptul că în setul nostru de date este împărțit în 15 clase, putem discuta despre o problemă de *multi-clasificare*, astfel încât vom folosi o particularitate a acestui algoritm, numit și regresie logistică multinominală(cunoscută și sub numele de *MaxEnt*.

Așadar, nu mai poate fi vorba de variabile dependente binare, ci mai degrabă categorice, deoarece se va calcula posibilitatea a  $n$  valori în loc de doar două.

Calculule pentru aceasta sunt următoarele:

$$\begin{aligned} \ln P(Y_i = 1) &= \beta_1 \cdot X_i - \ln Z \\ \ln P(Y_i = 2) &= \beta_2 \cdot X_i - \ln Z \\ &\dots\dots\dots \\ \ln P(Y_i = K) &= \beta_k \cdot X_i - \ln Z, \end{aligned}$$

unde  $\beta_i$  este factorul de regresie, atribuit fiecărei valori de input,  $X_i$  este valoarea input iar  $-\ln Z$  este factorul de normalizare, pentru a ne asigura că suma probabilităților de mai sus este egală cu 1:

$$\sum_{k=1}^K P(Y_i = k) = 1$$

Putem transforma ecuația probabilităților prin scoaterea logaritmului:

$$\begin{aligned} P(Y_i = 1) &= \frac{1}{Z} e^{\beta_1 \cdot X_i} \\ P(Y_i = 2) &= \frac{1}{Z} e^{\beta_2 \cdot X_i} \\ &\dots\dots\dots \\ P(Y_i = K) &= \frac{1}{Z} e^{\beta_K \cdot X_i} \end{aligned}$$

Putem extrage valoarea lui  $Z$  din suma probabilităților, astfel:

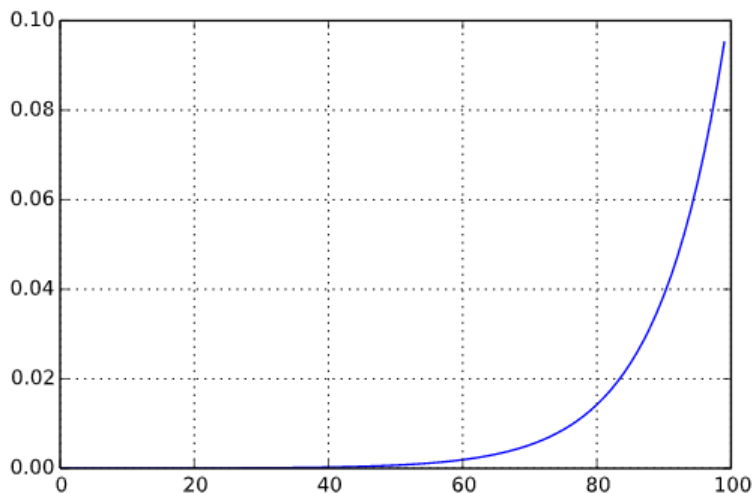
$$\begin{aligned} \sum_{k=1}^K P(Y_i = k) &= 1 \Rightarrow \sum_{k=1}^K \frac{1}{Z} e^{\beta_k \cdot X_i} \Rightarrow \frac{1}{Z} \sum_{k=1}^K e^{\beta_k \cdot X_i} \\ \Rightarrow Z &= \sum_{k=1}^K e^{\beta_k \cdot X_i} \end{aligned} \quad (2)$$

Din (1) și (2) putem calcula forma finală a probabilităților:

$$P(Y_i = c) = \frac{e^{\beta_c \cdot X_i}}{\sum_{k=1}^K e^{\beta_k \cdot X_i}}$$

Se observă că forma generală a probabilităților seamănă cu cea a funcției softmax:

$$softmax(k, x_i) = \frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}}$$



*graficul funcției softmax*

Softmax are rolul de a "exagera" diferențele dintre valorile de input, astfel încât  $softmax(k, x_i)$  va returna o valoare aproape de 0 atunci când  $x_i$  este destul de mic față de maximumul tuturor valorilor și va returna o valoare apropiată de 1  $x_i$  sunt destul de aproape de maximumul respectiv. Așadar, la final se va obține o aproximare a funcției indicator:

$$f(k) = \begin{cases} 1 & k = \operatorname{argmax}(x_1, \dots, x_k), \\ 0 & \text{altfel} \end{cases}$$

### **3.3 Random Forrest**

Ce face Random Forrest

## 4 Descrierea metodei proprii

În următoarele puncte voi discuta despre cum am ajuns la o soluție cu ajutorul algoritmilor explicați înainte, și cei de clasificare și cei de reprezentare. Aceasta a constat în mai mulți pași, fiecare etapă importantă, fiind explicată în cadrul unui subpunct.

### 4.1 Setul de date

Setul de date a fost generat de la 0, de către mine, cu ajutorul unei biblioteci din Python, BeautifulSoup și constă într-un număr de 15 bucătării și aproximativ vreo 5000 rețete culese pe de site-ul [www.allrecipes.com](http://www.allrecipes.com).

Bucătăriile care fac parte din acest set de date sunt:

- |            |                    |                |
|------------|--------------------|----------------|
| • French   | • Eastern European | • Italian      |
| • British  | • Indian           | • Chinese      |
| • Korean   | • Mexican          | • Cajun-Creole |
| • Thai     | • Greek            | • sss          |
| • Filipino | • Japanese         | • sss          |

Acest set de date constă într-un id, nume, tipul bucătăriei, o listă de ingrediente și metoda de preparare, după cum se poate vedea mai jos.

```
"id": 7456,
"title": "Spicy Creamy Cajun Ham and Black Eyed Peas Salad",
"cuisine": "cajun-creole",
"ingredients": [
    "2 cups fresh corn kernels",
    "2 (15 ounce) cans black-eyed peas, rinsed and drained",
    "1 cup cubed fully cooked ham",
    "3 stalks celery, finely chopped",
    "2 tablespoons chopped red onion",
    "2/3 cup sour cream",
    "1 tablespoon ketchup",
    "1 tablespoon dried cilantro",
    "1 teaspoon Cajun seasoning",
    "2 dashes hot pepper sauce (such as Tabasco\u00ae), or to taste"
],
"directions": "Place the corn into a saucepan, cover with water, and bring to a boil. Reduce heat and simmer until the corn is fully cooked, about 2minutes. Drain the corn in a colander set in the sink. Mix together the warmcorn, black-eyed peas, ham, celery, and onion in a salad bowl. Whisk together the sour cream, ketchup, cilantro, Cajun seasoning, and hot pepper sauce in a bowl until smooth. Stir the dressing lightly into the
```

black-eyed pea mixture until thoroughly mixed. Serve immediately.”

## 4.2 Procesarea datelor

Procesarea datelor se realizează prin utilizarea algoritmilor de la punctul 2, adică algoritmi de reprezentare. Aceștia ne ajută să normalizăm setul de date, în sensul în care fiecare algoritm lucrează într-un fel.

## 4.3 Utilizarea algoritmilor

După ce avem setul de date procesat de la etapa anterioară, putem trece mai departe la etapa următoare, unde vom face clasificarea propriu zisă

## 4.4 Reprezentări grafice

Avem setul de date trecut prin algoritmi de clasificare, astfel încât am putea genera o reprezentare grafică cu ajutorul următorilor algoritmi și a valorilor rezultate în urma procesării datelor.

Vom reduce la 2 dimensiuni, pentru a putea să îi dăm drept input K-Means-ului un set de date cât mai redus din punct de vedere dimensional.

### 4.4.1 UMAP

```
umap = UMAP()  
reduced_data = umap.fit_transform(data)
```

### 4.4.2 T-SNE

```
tsne = TSNE()  
reduced_data = tsne.fit_transform(data.toarray())
```

## 4.5 Limbajul folosit

Limbajul folosit este Python, un limbaj interpretat, care este destul de prietenos și care în ultima perioadă a căpătat destul de multă popularitate, tocmai datorită faptului că este destul de simplu de învățat și că are o multitudine de aplicații, cum ar fi domeniul de Data Science și Machine Learning cu vasta suită de biblioteci pe care pe rând le are la dispoziție (printre care sunt folosite și în realizarea acestei lucrări).

## 4.6 Biblioteci folosite

### 4.6.1 Sklearn

Algoritmi, tfidf, countvectorizer, svd, tsne

#### 4.6.2 BeautifulSoup

```
webpage = requests.get(link)
soup = BeautifulSoup(webpage.content, features="html.parser")

links = soup.find_all("article", {"class": "fixed-recipe-card"})

new_soup = BeautifulSoup(new_webpage.content, features="html.parser")

title = re.split(" Recipe - ", new_soup.title.text)[0]

directions = new_soup.find_all("span", {"class":
                                     "recipe-directions__list__item"})
ingredients = new_soup.find_all("span", {"class":
                                     "recipe-ingred-txt added"})

for direction in directions:
    direction = direction.text.replace("\n", "")

    if direction != "":
        directions_list.append(direction)

for ingredient in ingredients:
    ingredients_list.append(ingredient.next)

crawler, explicat cum am luat
```

#### 4.6.3 Gensim

folosit cu w2v

#### 4.6.4 Pandas

folosit pentru jsoane

#### 4.6.5 Plotlib

folosit sa fac reprezentarea grafica cu kmeans + svd/tsne

### 4.7 Rezultatele

Rezultatele sunt destul de variate, iar în funcție de unii parametri am obtinut anumite rezultate

SVC 71% - 79%

Regression 65% - 75%

RandomForrest mai slabut

## 5 Concluzii

## 6 Bibliografie

### References

- [1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The L<sup>A</sup>T<sub>E</sub>X Companion*. Addison-Wesley, Reading, Massachusetts, 1993.
- [2] Albert Einstein. *Zur Elektrodynamik bewegter Körper*. (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891–921, 1905.
- [3] Knuth: Computers and Typesetting,  
<http://www-cs-faculty.stanford.edu/~uno/abcde.html>