

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAŞI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**Tehnici de clasificare a textului în
domeniul culinar**

propusă de

Mircea Rareş - 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 a textului în
domeniul culinar**

propusă de

Mircea Rareș - Gabriel

Sesiunea: iulie, 2019

Coordonator științific

Conf. Dr. Răschip Mădălina

Avizat,

Îndrumător Lucrare

Titlul, Numele și prenumele: Conferențiar Dr. Răschip Mădălina

Data: 27 iunie 2019 Semnătura: _____

DECLARAȚIE
privind originalitatea conținutului lucrării de licență

Subsemnatul **Mircea G. N. Rareș - Gabriel** cu domiciliul în **Strada Smirodava, bl. 58, ap. 56, municipiul Roman, județul Neamț** născut la data de **24.09.1997**, identificat prin CNP **1970924270861**, absolvent al Universității „Alexandru Ioan Cuza” din Iași, Facultatea de **Informatică**, specializarea **Informatică**, promovată **2016 - 2019**, declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea cu titlul:

Tehnici de clasificare a textului în domeniul culinar

elaborată sub îndrumarea **Conferențiar Dr. Răschip Mădălina**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consumând inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

**Dată azi,
27 iunie 2019**

**Semnătură student,
Mircea Rareș - Gabriel**

DECLARAȚIE DE CONSUMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „**Tehnici de clasificare a textului în domeniul culinar**”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași,
27 iunie 2019

Semnătură student,
Mircea Rareș - Gabriel

Cuprins

1 Descrierea problemei	3
1.1 Metode existente	3
1.2 Setul de date	4
2 Extragerea trăsăturilor	6
2.1 Bag-of-Words	6
2.2 Tf-Idf	7
2.3 Word-2-Vec	7
3 Algoritmii de clasificare	9
3.1 SVC Liniar	9
3.2 Regresia Logistică	12
3.3 Random Forest	14
4 Descrierea metodei utilizate	16
4.1 Setul de date	16
4.2 Procesarea datelor	18
4.2.1 Tf-Idf	18
4.2.2 Bag-Of-Words	19
4.2.3 Word-2-Vec	19
4.3 Clasificarea rețetelor culinare	20
4.3.1 Linear SVC	20
4.3.2 Regresie Logistică	21
4.3.3 Random Forest	21
4.4 Vizualizarea setului de date	23
4.4.1 UMAP	23
4.4.2 T-SNE	23
4.5 Tehnologii folosite	24
4.5.1 Python	24

4.5.2	SciKit-Learn	25
4.5.3	BeautifulSoup	25
4.5.4	Gensim	26
4.5.5	Pandas	26
4.5.6	Matplotlib	27
4.6	Rezultate experimentale	27
5	Concluzii	29
	Bibliografie	30

Capitolul 1

Descrierea problemei

Mâncarea stă la baza civilizației umane, face parte din fundația piramidei nevoilor a lui *Maslow* fiind una dintre nevoie de bază a omului. Încă din faza incipientă apariției omului, hrana a fost una dintre ţintele de bază a societății, împreună cu adăpostul și procrearea speciei. De-a lungul timpului, noi am evoluat, am migrat pe toate cele 7 continente, de 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 bucătăriei proprii, 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 a reușit să exporte influențe culinare.

Având în vedere multitudinea de rețete la nivel global, ne-am propus cu ajutorul algoritmilor de Învățare Automată, să realizăm o clasificare a rețetelor în funcție de bucătăria din care provin. Pentru a realiza acest obiectiv, va fi nevoie de o investigație asupra seturilor de date de care ne putem folosi, mai apoi despre procesele de extragere de trăsături iar într-un final clasificarea datelor.

Pentru a realiza aceasta, ne vom folosi de Python, cu toată suita de biblioteci care vine în ajutorul domeniului de Învățare Automată. Vom folosi *pandas* pentru a putea prelucra datele mai ușor cu ajutorul structurii *DataFrame*, *sklearn* pentru implementarea algoritmilor, *matplotlib* pentru reprezentările grafice. A fost utilizat un program crawler¹, cu ajutorul *BeautifulSoup* pentru a putea genera setul de date.

1.1 Metode existente

În domeniul clasificării textelor din domeniul culinar există lucrări de specialitate care propun fie clasificarea rețetelor în funcție de ingrediente pe care le conțin, fie propun o clasificare a rețetelor în funcție de imagini.

În lucrarea [1] a fost creat set de date, cu ajutorul căruia este realizată o clasificare a datelor de tip text în funcție de ingrediente a bucătăriilor considerate. A fost realizată o reprezentare vizuală a setului de date, cu ajutorul algoritmului T-SNE de reducere a dimensionalității.

¹crawler = program care parurge site-uri web, în căutarea un informații dorite de noi; informațiile le caută după *div – uri* din *html*(în cazul nostru)

În lucrarea [2] a fost propus un sistem care este capabil să identifice o rețetă în funcție de imaginile primite și să selecteze o imagine reprezentativa pentru o rețetă primită la intrare. Pentru a realiza aceasta, s-a recurs mai întâi la generarea unui set de date, format din cel puțin 10 milioane de imagini, însumând peste 100 GB de imagini. Pentru implementare s-au utilizat rețele neuronale convoluționale.

Alți cercetători au propus investigarea preferințelor culinare din lumea întreagă [3]. Aceștia au prezentat cum se organizează bucătăriile în funcție de popoare și care sunt tiparele acestora. Aceștia au descoperit că ingredientele nu sunt alese în mod aleator și că după ani de „mutații” ale bucătăriilor, gustul mâncării a rămas relativ constant.

1.2 Setul de date

După etapa de studiere a metodelor existente, am dorit să experimentez diferite metode pe seturile de date folosite și am descoperit că toate trei veneau cu un set de date total diferit față de celelalte la nivel de reprezentare și de date utilizate. De exemplu, în lucrarea [2] se folosea de un set de date gigant alcătuit din 13 milioane de poze cu mâncare, care ajungea la peste 150 GB, era pur și simplu mult prea mare pentru a putea fi folosit pe un computer personal și nici nu erau îndeplinite specificațiile cerute.

Celelalte două lucrări [1, 3] foloseau seturi de date mai apropriate de ceea ce căutam. Cel propus în lucrarea [3] utiliza o reprezentare bazată pe tabele cu structuri de unu-la-unu, fiind dificil de lucrat cu o asemenea reprezentare în Python.

Setul de date din lucrarea [1] continea un fisier de tip json, care putea fi transformat cu o foarte mare ușurință într-un *DataFrame*, iar în acesta se găsea denumirea rețetei, ingredientele și bucătăria din care făcea parte. Mai lipsea un element pe care îl căutam, fiind modul de preparare.

Sursa acestul set de date este site-ul www.yummly.com. Acesta oferea un API la oricare se putea abona și putea face requesturi GET având acces la toate rețetele de pe această pagină. Din păcate, API-ul a fost închis, ceea ce ne-a făcut să considerăm varianta unui crawler cu ajutorul căruia să putem crea setul de date dorit.

După o studiere a site-urilor de rețete, cel mai potrivit pentru sarcina de extragere de rețete s-a dovedit a fi www.allrecipes.com, deoarece avea rețetele împărțite în funcție de bucătărie și extragerea acestora era mai puțin complicată.

Cu ajutorul crawler-ului, am reușit să creăm un set de date de aproximativ 5000 rețete, memorate într-un fisier json în care vom avea o listă de rețete, acestea fiind strucțurate în tipul *cheie-valoare* unde sunt stocate numele rețetei, o listă de ingrediente și instrucțiunile acesteia.

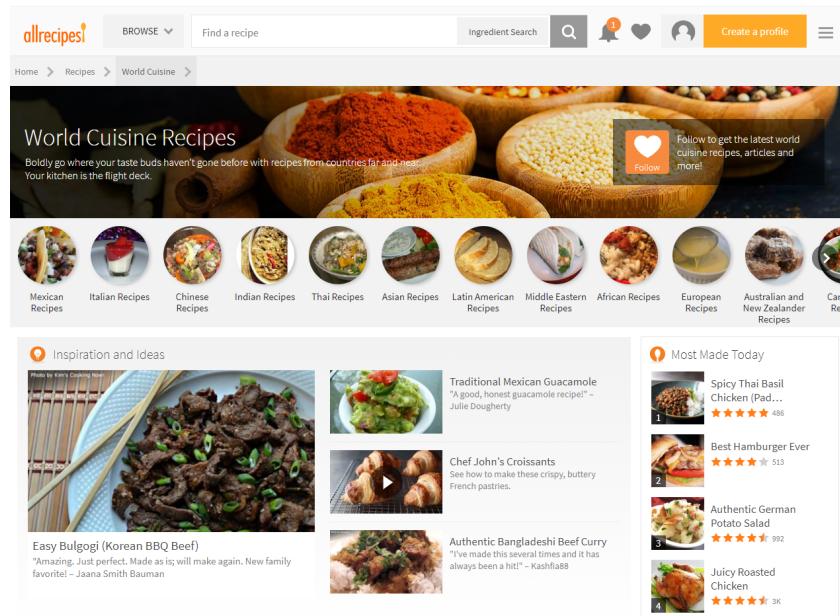


Figura 1.1: Pagina principală a rețetelor grupate pe bucătării

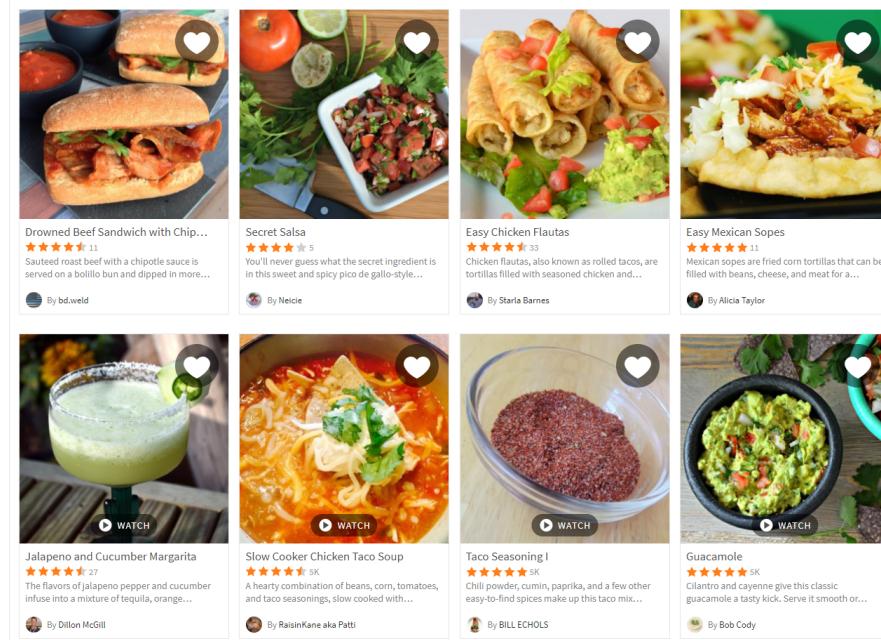


Figura 1.2: Fragment din prima pagină cu rețete mexicane

Ingredients

- + 12 ounces chipotle cooking sauce (such as Knorr®)
- + 1 (14 ounce) can reduced-sodium beef broth
- + 1/4 cup chopped fresh cilantro (optional)
- + 2 tablespoons vegetable oil
- + 1 onion, thinly sliced
- + 3 cloves garlic, minced
- + 1 pound thinly sliced deli roast beef
- + 4 bolillo rolls, halved and lightly toasted
- + 4 sprigs fresh cilantro, or to taste (optional)
- + Add all ingredients to list

Directions

🕒	Prep 15 m	Cook 20 m	Ready In 35 m
1 Combine chipotle cooking sauce, beef broth, and 1/4 cup chopped cilantro in a saucepan; bring to a boil. Reduce heat to medium-low and simmer, stirring occasionally, for 10 minutes.			
2 Heat oil in a skillet over medium-high heat; sauté onion until softened, about 5 minutes. Stir garlic into onion and cook for 1 minute. Add roast beef and 1/4 cup chipotle sauce mixture and cook, stirring constantly, until heated through, about 2 minutes.			
3 Ladle remaining chipotle sauce mixture into 4 bowls for dipping. Spoon roast beef mixture onto the bottom half of each bun, top with a cilantro sprig, and place the top on each bun. Dip sandwiches into sauce.			

Figura 1.3: Lista ingredientelor

Figura 1.4: Lista instrucțiunilor

Capitolul 2

Extragerea trăsăturilor

Pentru reprezentare, am folosit de algoritmi din categoria de extragere de trăsături cum ar fi Tf-Idf, Bag-of-Words și Word-2-Veс. Acest proces constă în crearea unui corpus, de obicei un ansamblu de texte, în cazul nostru pașii rețetei sau chiar ingrediente, care sunt trimise spre algoritmii menționați mai sus pentru a fi procesat.

Etapele prin care trec toți cei trei sunt etapa de tokenizare, unde fiecare cuvânt este considerat a fi un token și atribuit un identificator(de obicei număr natural).

Spațiile și semnele de punctuație sunt considerate separatori iar în funcție de caz, putem și cuvintele de tip *stop – words*.¹

După etapa de tokenizare, se trece la etapa de transformare a corpusului într-o reprezentare vectorială, unde fiecare algoritm aplica metoda proprie de vectorizare, de exemplu, Bag-of-Words numără efectiv de câte ori se găsește cuvântul în corpus, Tf-idf calculează raportul între frecvența cuvântului în text și numărul de texte din corpus care conțin acel cuvânt. Word2Vec contexte utilizate pentru a extrage similaritățile cuvintelor.

2.1 Bag-of-Words

Denumirea acestui algoritm provine de la faptul că fiecare text, cum ar fi o propoziție sau frază, este reprezentată de un multi-set(*bag*) de cuvinte, indiferent de gramatică sau ordine, în care se păstrează valorile într-o structură de tip dictionar(cheie - valoare), unde cheia este cuvântul în sine, iar valoarea este frecvența acestui cuvânt în textul respectiv [5].

Exemplu: *Ana are mere și George are pere.*

Modelul Bag-of-Words va conține:

$$\{\text{Ana: 1, are: 2, mere: 1, și: 1, George: 1, pere: 1}\}$$

Se pot utiliza și n-grame pentru a construi reprezentarea de tip Bag-of-Words. Termenul n-grame este utilizat pentru a grupa cuvintele în perechi de cuvinte succesive

¹stop-word = cuvânt care este ignorat la începutul procesului de procesare a limbajului natural; de asemenea, sunt luate în considerare cele mai comune cuvinte din limba respectivă, exemplu: "this", "that"

de căte n și în cadrul reprezentării Bag-of-Words utilizând n-grame se va identifica frecvența acestora.

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

$\{ \text{Ana are},$
 $\text{are mere},$
 $\text{mere și},$
 $\text{și George},$
 $\text{George are},$
 $\text{are pere} \}$

2.2 Tf-Idf

Algoritmul tf-idf[6] ("term frequency ²-inverse document frequency ³") semnifică cât de important este un cuvânt într-un corpus, proporțional cu numărul de texte din corpus care conțin acel cuvânt. Astfel încât putem spune că, valoarea tf-idf a unui cuvânt, 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, detaliem formulele matematice a celor două componente care îl alcătuiesc.

Termenul "term-frequency":

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

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

Termenul "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 . Termenul $1+$ este necesar deoarece se dorește evitarea situației în care tokenul nu se va regăsi în niciun text, caz în care vom avea o împărțire la 0.

Utilizând termenii detaliați mai sus, formula pentru Tf-Idf este:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D).$$

2.3 Word-2-Vec

Word-2-Vec [7] este un algoritm relativ recent apărut în domeniul Învățării Automate și presupune transformarea unui corpus de texte (*Word*) într-un spațiu vectorial (*2Vec*). Scopul său principal este acela de a grupa cuvintele care sunt similare în acel spațiu vectorial.

²term frequency = frecvența termenului

³inverse document frequency = frecvența inversă a documentului

Acest algoritm de extragere de trăsături are la bază o rețea neuronală cu două straturi și se împarte în două categorii: *Continuous Bag of Words(CBOW)* și *Skip Gram*.

Continous Bag of Words este asemănător cu *Bag of Words*, și se folosește de cuvintele alăturate pentru a putea face predicția cuvântului țintă. De exemplu, dacă avem 5 cuvinte și e nevoie să-l prezicem pe cel din mijloc (al treilea), modelul CBOW se va folosi de primele 2 și ultimele 2 cuvinte pentru a realiza această predicție.

Dacă presupunem că pe poziția i avem cuvântul pe care vrem să-l prezicem, atunci vom lua în considerare $i - 2, i - 1, i + 1$ și $i + 2$.

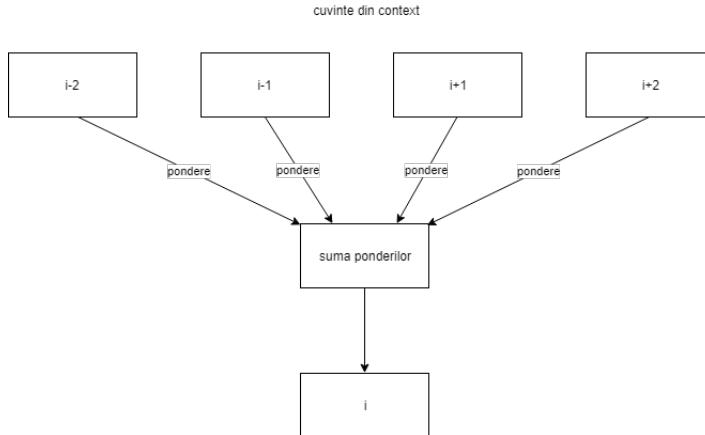


Figura 2.1: Continous Bag of Words

În cazul în care folosim modelul *Skip Gram*, parcursul este fix invers față de cel de la CBOG, adică va fi trimis un cuvânt iar pentru această intrare, vor rezulta cuvintele care fac parte din contextul cuvântului dat ca intrare.

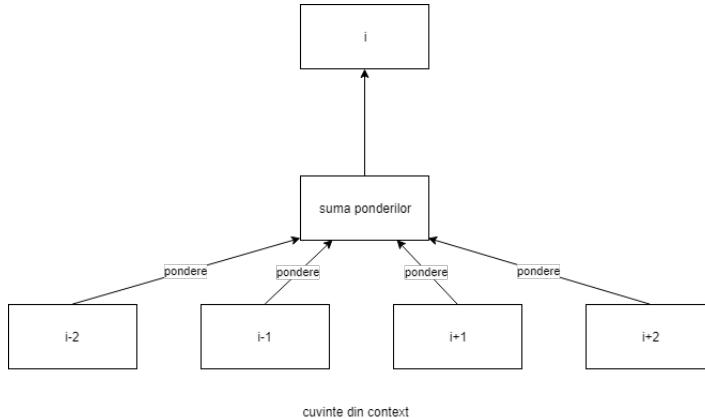


Figura 2.2: Skip Gram

În urma studiilor efectuate asupra acestor două modele, s-a observat ca *Bag of Words Continuu* are o acuratețe mult mai bună decât *Skip Gram*, singurele dezavantaje fiind că pentru CBOW este de nevoie de un set de date mai mare și de mai mult timp de execuție, față de *Skip Gram*, care se descurcă pe seturile de date de dimensiune mai mică.

Capitolul 3

Algoritmii de clasificare

Următoarea etapă, de după cea de extragere de trăsături, este cea de utilizare a algoritmilor de clasificare. Acești algoritmi vor primi ca input vectorii de trăsături și vor folosi vectorii ca atare, fără a reducere dimensionalitatea acestora, axându-ne pe maximizarea acurateții prin intermediul ajustării de hiper-parametri.

Algoritmii de clasificare selectați sunt: SVC liniar, Regresie logistica și Random Forest. Algoritmul SVC liniar separă instanțele cu ajutorul hiper-planurilor care maximizează marginea dintre două clase. Algoritmul de Regresie Logistică modelează probabilități cu ajutorul funcției softmax. Random Forest este un algoritm de tip ansamblu care utilizează o multitudine de arbori de decizie pentru a îndeplini sarcina de clasificare.

3.1 SVC Liniar

Algoritmul care stă la baza Linear SVC (*Support Vector Classifier*) este SVM (*Support Vector Machines* [8]). Acesta este un algoritm de învățare supervizată folosit pentru a separa instanțele aparținând unor clase diferite într-un spațiu multidimensional.

SVM este unul dintre cei mai folosiți algoritmi de clasificare din domeniul Învățării Automate. Sarcina acestuia este ca într-un spațiu n-dimensional, unde n este numărul de trăsături, să găsească un hiper-plan care să clasifice cel puțin două puncte din acel spațiu.

Majoritatea SVM-urilor folosesc să zisa margină maximă în alegerea hiper-planului, deoarece aceasta maximizează distanța dintre cele două puncte. Utilizarea maximului distanței dintre două puncte ne va asigura că punctele care se vor afla pe o parte sau alta a graniței de decizie, vor fi corect clasificate.

Pentru a crea această margină maximă, SVM utilizează de niște vectori suport care se află în proximitatea hiper-planului și pot influența poziția acestuia.

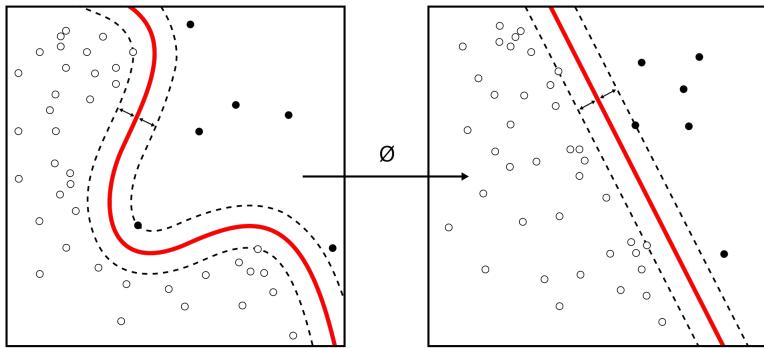


Figura 3.1: Exemplu de hiper-plan cu margine maximă¹

În situațiile în care nu avem un set de date care este liniar separabil, SVM vine cu un artificiu numit în engleză *Kernel Trick*. Acest artificiu mapează setul de date într-un spațiu multi-dimensional unde va încerca să găsească un hiper-plan care să separe instanțele de antrenament.

Acest *Kernel Trick* este realizat cu ajutorul unor funcții kernel, care reduc complexitatea identificării a funcției de mapare; aşadar, o funcție kernel definește un produs intern într-un spațiu multi-dimensional.

Exemple de funcții kernel:

- kernel liniar: $K(x, y) = x^T y, x, y \in R^d$
- kernel polinomial: $K(x, y) = (x^T y + r)^n, x, y \in R^d, r \geq 0$
- kernel gausian(sau radial basis kernel/RBF): $K(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}}, x, y \in R, \sigma > 0$

Dintre acestea, noi vom folosi funcția kernel liniară, deoarece aceasta este funcția folosită în implementarea algoritmului SVC Liniar. Funcția respectivă are rolul de a aduce valorile aflate într-un spațiu multi-dimensional într-unul bidimensional, în care problema să fie liniar separabilă.

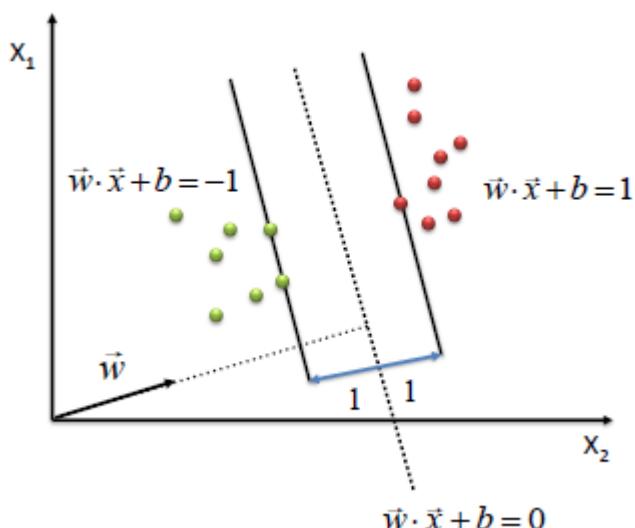


Figura 3.2: Hiper-plan însotit de formule matematice²

¹https://upload.wikimedia.org/wikipedia/commons/f/fe/Kernel_Machine.svg

În prezentul grafic se regăsește vectorul w perpendicular pe hiper-plan și vectorul x reprezentat de punctele de pe margine. Din graficul de mai sus, se observă funcțiile celor două linii suport și a hiper-planului:

$$\begin{aligned} w \cdot x + b &= 1 \text{ și } w \cdot x + b = -1 \\ \Rightarrow w \cdot x + b &= 0 \end{aligned}$$

Considerăm un vector cu etichete y , $+1$ în cazul punctelor roșii și -1 în cazul celor verzi, astfel încât:

$$\begin{aligned} w \cdot x + b &> +1, \forall y_i = +1 \\ w \cdot x + b &< -1, \forall y_i = -1 \end{aligned}$$

Putem aduce formulele de mai sus la forma:

$$y_i(w \cdot x_i + b) \geq 1, \forall i(1)$$

Pentru a calcula distanța dintre cele două linii suport, ne vom folosi de distanța euclidiană dintre două puncte de pe cele două linii. Vom nota x_r punctul roșu și x_v cel verde.

$$x_r \cdot w = 1 - b \text{ și } x_v \cdot w = -1 - b$$

De aici, calculul distanței este:

$$\frac{(x_r - x_v) \cdot w}{\|w\|} = \frac{1-b - (-1-b)}{\|w\|} = \frac{2}{\|w\|} (2)$$

Această funcție a distanței stă la baza problemei de optimizare a granitei de decizie. Optimizarea se poate realiza prin maximizarea (2) însă, pentru simplitate, vom folosi inversa funcției, astfel încât vom urmări să minimizăm $\frac{1}{2}\|w\|^2$ cu ajutorul formulei (1).

Pentru a rezolva această problemă, vom utiliza funcția Lagrange de transformare:

$$\begin{aligned} L(w, b, \alpha) &= \frac{1}{2}\|w\|^2 - \sum_{i=1}^l \alpha_i(y_i((w \cdot x_i) + b) - 1) \quad (3), \\ \text{unde } \alpha &= (\alpha_1, \dots, \alpha_l)^T \text{ este vectorul de multiplicare Lagrange} \end{aligned}$$

Pentru a putea continua, vom alege să derivăm după w de unde va rezulta următoarea ecuație:

$$\begin{aligned} \nabla_w L(w, b, \alpha) &= w - \sum_{i=1}^l y_i x_i \alpha_i = 0 \\ \text{de unde rezultă că} \\ w &= \sum_{i=1}^l y_i x_i \alpha_i \quad (4) \end{aligned}$$

Substituind (4) în (3) rezultă,

²https://www.saedsayad.com/support_vector_machine.htm

$$L(w, b, \alpha) = -\frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) + \sum_{j=1}^l \alpha_j - b(\sum_{i=1}^l y_i \alpha_i)$$

Pentru a rezolva problema de minimizare, putem considera $\sum_{i=1}^l y_i \alpha_i = 0$, astfel vom ajunge la un rezultat final:

$$L(w, b, \alpha) = -\frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) + \sum_{j=1}^l \alpha_j$$

[14]

3.2 Regresia Logistică

Regresia logistică [9] este un model care se bazează pe funcția logistică, sau sigmoid:

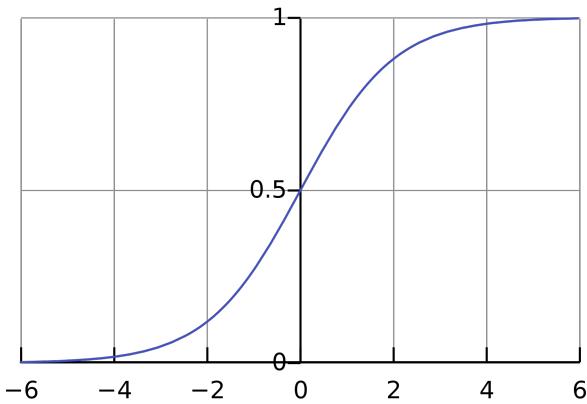


Figura 3.3: Graficul funcției sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$

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

Având în vedere faptul că setul nostru de date este împărțit în 15 clase, putem discuta despre o problemă de *multi-clasificare*, astfel încât vom folosi o generalizare a acestui algoritm, numit și regresie logistică multi-nominală (cunoscută și sub numele de *MaxEnt* [10]).

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.

Calculul este:

$$\ln P(Y_i = 1) = \beta_1 \cdot X_i - \ln Z$$

$$\ln P(Y_i = 2) = \beta_2 \cdot X_i - \ln Z$$

.....

$$\ln P(Y_i = K) = \beta_k \cdot X_i - \ln Z,$$

unde β_i este factorul de regresie, atribuit fiecărei valori de intrare X_i este valoarea input iar $-LnZ$ 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 anterioară prin eliminarea logaritmului:

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

Putem extrage valoarea lui Z din suma probabilităților, astfel:

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

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:

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

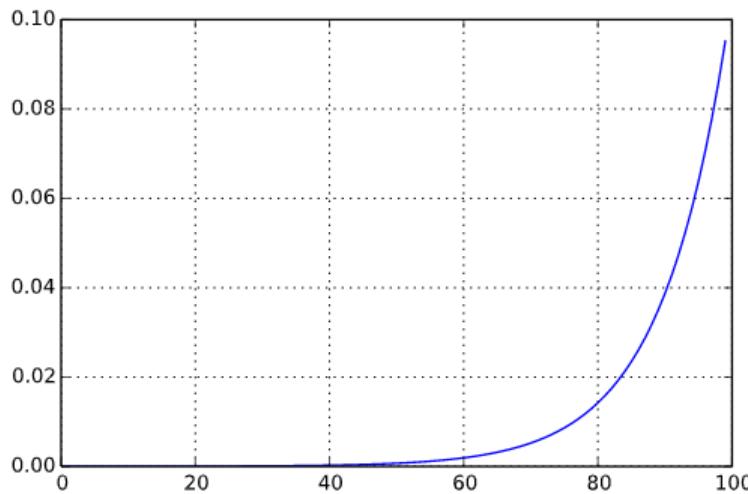


Figura 3.4: Graficul funcției softmax

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

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

Concluzionând, Regresia Logistică se poate dovedi un clasificator bun pentru setul nostru de date, având în vedere că poate fi folosit și pentru probleme de clasificare unde sunt mai mult de două clase. Regresia logistică vine cu propunere nouă de clasificare, mai precis, unei instanțe de test îi se asignează o probabilitate pentru fiecare etichetă.

3.3 Random Forest

Random Forest este un algoritm de învățare automată de tip ansamblu. Acest concept de ansamblu face referire la faptul că această "pădure" este formată dintr-un ansamblu de arbori de decizie, de unde vine și termenul de *Forest*, singura diferență este că aceștia sunt generați aleator, *Random*, din atributele primite ca intrare. Există algoritmi care dau rezultate mai slabe dacă sunt folosiți singuri, iar dacă sunt puși la un loc (într-un ansamblu), pot da rezultate foarte bune.

Pentru a înțelege cum funcționează Random Forest, trebuie să considerăm mai întâi arborii de decizie. Acești arbori sunt formati din atributele care formează datele de intrare; de exemplu, dacă am vrea să vedem dacă va ploua mâine, putem considera umiditatea, dacă a fost înnorat și dacă a fost frig a ultimelor zile. Pe baza arborelui creat putem afla cum va fi vremea în ziua următoare.

Arborii de decizie sunt un model de reprezentare arborescentă a deciziilor care se iau asupra unui set de date. Nodurile sunt reprezentate de un atribut, iar copiii săi pot fi un alt nod, sau o frunză, semnificând o decizie. De obicei, este ales atributul care împarte cel mai uniform setul de date dacă ar sta la baza unei decizii; această uniformitate se mai numește și entropie. Apoi, se repetă acest procedeu de alegere a nodurilor de decizie se repetă până se utilizează toate atributele, rezultând un arbore de decizie.

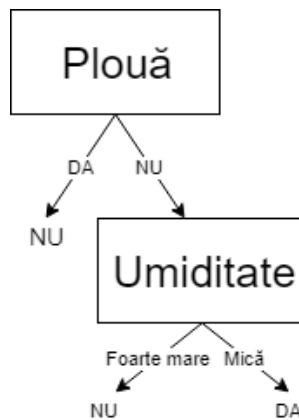


Figura 3.5: Exemplu de arbore de decizie, creat în situația în care o persoană vrea să iașă la alergat

Random Forest este alcătuit dintr-o multitudine de astfel de arbori, numai că de data aceasta, arborii nu vor mai fi creați neapărat din toate atributele de intrare, ci se va

face ceea ce se numește *bagging* sau *bootstraping*. Aceasta metodă presupune utilizarea a aceluiași număr de attribute(cu tot cu deciziile aferente); acestea sunt luate aleator și pot exista duplicate în acest set nou de attribute, numit și *bootstrap*. S-a observat ca după *bootstraping*, mai rămân aproximativ o treime din attribute într-un set de date care se va numi *out – of – bag*.

Algoritmul va folosi mai departe valorile din *bootstrap* pentru a genera un arbore și până se ajunge la numărul dorit de arbori, va repeta acesti doi pași: generează *bootstrap* și din acesta creează un arbore. Este preferat să se folosească un număr cât mai mare de arbori, deoarece cu cât este pădurea mai deasă cu atât modelul este mai robust și se poate ajunge la o acuratețe mai bună. Acești arbori vor avea nodurile de decizie cât mai diferite între ei, putându-se respecta caracteristicile *Random*.

După generarea "pădurii", se începe procedeul de clasificare, unde fiecare moștră, va trece prin toti arborii. Acești arbori vor oferi o ieșire sau vot, acesta se va centraliza și se va alege eticheta cu cele mai multe voturi, procedura de alegere numindu-se vot majoritar.

Principalul avantaj pentru care am considerat utilizarea algoritmului Random Forest a fost că șansele ca să facă overfitting pe setul de date sunt foarte mici și pentru că folosind parametrii optimi, putem obține o acuratețe ridicată.

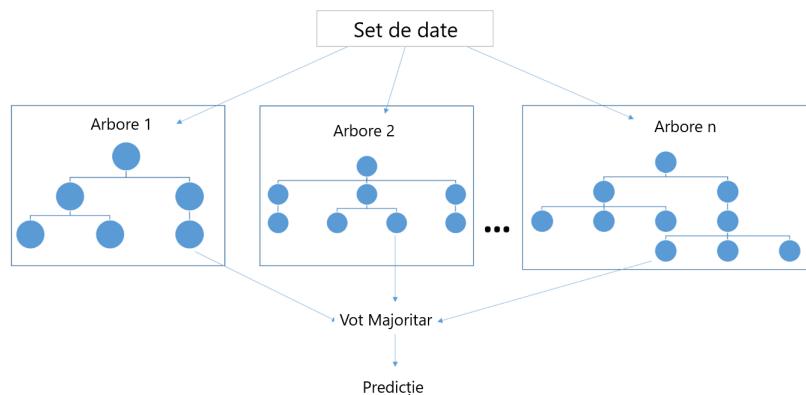


Figura 3.6: Exemplu de clasificare cu Random Forest

Capitolul 4

Descrierea metodei utilizate

În următoarele subcapitole voi discuta despre cum am ajuns la o soluție cu ajutorul algoritmilor prezentați mai sus, cei de clasificare și cei de reprezentare. Fiecare etapă este prezentată în cadrul subcapitolelor care urmează.

4.1 Setul de date

Setul de date a fost generat, de către mine, cu ajutorul unei biblioteci din Python, BeautifulSoup. Aceasta constă într-un număr de 13 bucătării și aproximativ 5000 rețete culese pe de site-ul www.allrecipes.com.

Bucătăriile care fac parte din acest set de date sunt:

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

Acest set de date conține înregistrări ce dețin un id, nume, tipul bucătăriei, o listă de ingrediente și metoda de preparare.

Exemplu:

```
"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 sauce)"  
],  
"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 2 minutes. Drain  
the corn in a colander set in the sink. Mix together the  

```

Următoarea diagramă prezintă repartizarea rețetelor pe bucătării:

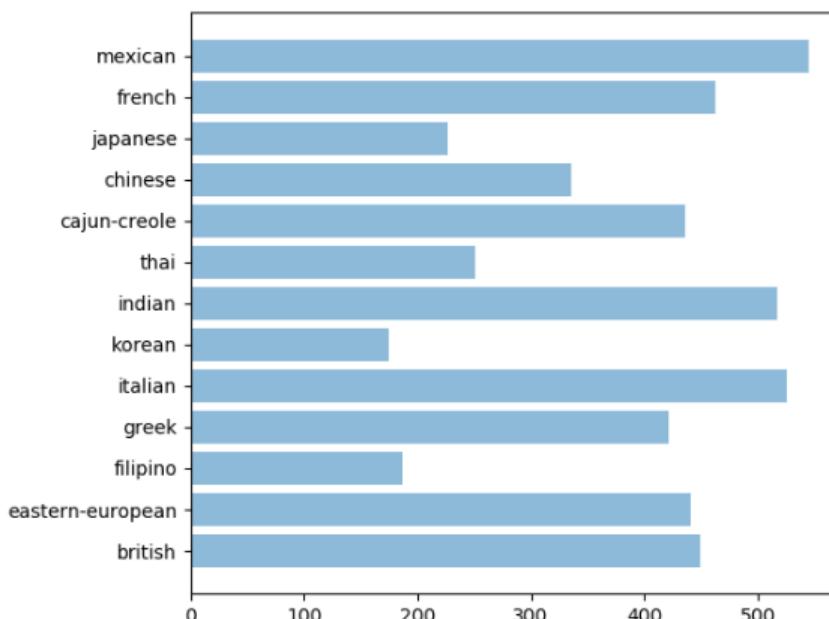


Figura 4.1: Repartizarea rețetelor pe bucătării

4.2 Procesarea datelor

În această etapă ne vom folosi setul de date creat anterior și îl vom trimite mai departe algoritmilor care vor procesa acest corpus și vor oferi vectorii de trăsături caracteristici fiecărui. Înainte de a începe procesarea propriu-zisă, mai întâi voi salva datele din fișierul JSON utilizând biblioteca *pandas* într-o structură de tip *DataFrame* care îmi va permite să lucrez mai ușor cu aceste rețete.

Având în vedere faptul că există aproximativ aproape 5000 de rețete în setul de date voi folosi în jur de 90% din date drept set de antrenare, iar restul de 10% pentru testare, deci undeva la 4500 și 500.

După ce am realizat împărțirea setului mare de date în unul de antrenare și unul de testare, acestea vor fi trimise mai departe la etapa de extragere de trăsături (algoritmi descriși anterior la capitolul 2).

Pentru fiecare din acești algoritmi, am realizat o ajustare a hiper-parametrilor pentru a fi siguri că extragerea de trăsături este cât mai precisă, astfel încât să ne putem axa pe maximizarea acurateței algoritmilor de clasificare.

Pașii realizați pentru crearea reprezentării:

1. Se instantiază algoritmul cu parametrii de rigoare
2. Trimitem ca intrare întregul corpus (format din setul de antrenament și testare) pentru a genera vocabularul
`corpus = train['directions'].append(test['directions'])`
3. La final, avem etapa de transformare a seturilor de date după care le vom trimite mai departe la clasificatori

4.2.1 Tf-Idf

```
vectorizer = TfidfVectorizer(  
    stop_words='english',  
    ngram_range=(1, 1),  
    min_df=4)  
tfidf_vect = vectorizer.fit(corpus)
```

În cazul Tf-Idf, inițial ne-am asigurat că am scos cuvintele *stop – words*, deoarece, acestea nu sunt relevante în a determina o rețetă, ele făcând pur și simplu legătura dintre două propozitii dintr-o frază sau a unor părți de vorbire.

După aceea, nu am folosit modelul de n-grame ((1, 1) semnifică faptul că un cuvânt, reprezintă o 1-gramă), deoarece am observat că acest algoritm tende să dea rezultate mult mai bune în felul acesta; parametru *min_df* care face referire la frecvența minimă a unui token într-un document.

Având algoritmul instantiat, urmează etapa propriu zisă de extragere de trăsături, unde vom crea pe rând vectorii de trăsături pentru instanțele de antrenare și de testare.

```
corpus_train = train[ 'directions' ]  
tfidf_train = tfidf_vect.transform(corpus_train)
```

```

corpus_test = test['directions']
tfidf_test = tfidf_vect.transform(corpus_test)

```

4.2.2 Bag-Of-Words

```

vectorizer = CountVectorizer(
    stop_words='english',
    ngram_range=(1, 2),
    min_df=5)
bow_vect = vectorizer.fit(corpus)

```

Urmărind tiparul de la $Tf - Idf$, am eliminat cuvintele stop-words. După, prin încercări succesive, am reușit să ajung la hiper-parametrii ideali, din punctul de vedere al setului nostru de date.

De data aceasta, algoritmul nostru de extragere de trăsături va considera unigramele și bi-gramele și pe lângă acestea, numărul minim de apariții în document al tokenurilor este de 5.

```

corpus_train = train['directions']
bow_train = bow_vect.transform(corpus_train)

corpus_test = test['directions']
bow_test = bow_vect.transform(corpus_test)

```

4.2.3 Word-2-Vec

Pentru utilizarea algoritmului Word2Vec, a fost nevoie ca mai întâi să tokenizez instrucțiunile rețetelor, conform cerințelor bibliotecii *gensim* pentru a folosi modelul Word2Vec. Pentru aceasta, am folosit biblioteca *nltk* cu ajutorul căreia mi-am tokenizat instrucțiunile din rețete și le-am trimis mai departe sub numele de *sentences*.

```

w2vec = gensim.models.Word2Vec(
    sentences,
    size=250,
    window=5,
    min_count=1)

```

În frântura de cod de mai sus se poate vedea instanțierea algoritmului, cu vectorul de instrucțiuni tokenizate, *size* = 250, care reprezintă dimensiunea vectorilor de cuvinte, următorul parametru *window* = 5 semnifică dimensiunea contextului în care se va încerca precizarea cuvântului. Ultimul parametru este destul de asemănător cu cel de *Bag - of - Words* și *Tf - Idf*, fiind numărul minim de apariții în text.

```

def tokenize_corpus(corpus):
    transformed_corpus = []
    for document in corpus:
        tokenized_doc = []
        for sent in nltk.sent_tokenize(document):
            tokenized_doc += nltk.word_tokenize(sent)
        transformed_corpus.append(np.array(tokenized_doc))
    return np.array(transformed_corpus)

```

Functia `tokenize_corpus` produce liste de tokeni din corpusul primit prin intermediul variabilei `corpus` pentru a fi folosita la generarea unui vector de medii.

```
tokenized_corpus = tokenize_corpus(corpus)

return np.array([
    np.mean([self.word2vec.wv[w] for w in words
             if w in self.word2vec.wv]
            or [np.zeros(self.dim)], axis=0)
    for words in tokenized_corpus
])
```

Dupa tokenizarea corpusului, vom crea un nou vector care contine o medie pentru fiecare cuvant in parte din toti vectorii de cuvinte din text, astfel incat la clasificare sa utilizam un singur vector de trasaturi. In cazul in care intr-un text din corpus nu este gasit un cuvant la calculul mediei, acesta va fi considerat ca un vector de dimensiune egală cu numărul de cuvinte din vocabular, cu elementul 0 pe fiecare pozitie.

```
mean_embedding_vectorizer = MeanEmbeddingVectorizer(w2vec)

corpus_train = train['directions']
w2v_train = mean_embedding_vectorizer.transform(corpus_train)

corpus_test = test['directions']
w2v_test = mean_embedding_vectorizer.transform(corpus_test)
```

4.3 Clasificarea rețetelor culinare

Dupa procesarea de la etapa anterioara, avand vectorii de trasaturi creati, putem trece mai departe la etapa de propriu-zisa de clasificare. Aici, pentru fiecare algoritm de clasificare in parte, vom ajusta hiper-parametrii pentru a obtine o clasificare cat mai buna din punct de vedere al acuratetei.

Mai jos voi explica modul de utilizare a algoritmilor împreună cu hiper-parametrii folosiți de către aceștia.

Hiper-parametrii au fost reglați pentru fiecare tip de reprezentare in parte, pentru a maximiza acuratețea acestora.

4.3.1 Linear SVC

Parametrul C transmite algoritmului care să fie rația corecției la clasificare, în sensul în care cu cât este mai mare această valoare, cu cât hiper-planul clasifică mai bine punctele de antrenare, cu atât mai mult va fi aleasă o margine mai mică a hiper-planului. Invers, cu cât valoarea lui C este mai mică, cu atât marginea hiper-planului va fi mai mare, deși valorile de antrenare sunt linear separabile, chit că acuratețea va scădea.

```
classifier_svc = LinearSVC(C)
classifier_svc.fit(train, classes)
prediction_svc = classifier_svc.predict(test)
```

Pentru fiecare reprezentare, am folosit următoarele valori ale lui C:

Ajustări			
Parametri	Tf-Idf	Bag-of-Words	Word-2-Vec
C	0.5	0.1	25

Figura 4.2: Alegerea parametrilor în funcție de algoritmul de extragere de trăsături

4.3.2 Regresie Logistică

Parametrul C este mai numit și parametru de regularizare și are același efect asupra algoritmului ca și la Linear SVC.

```
classifier_regr = LogisticRegression(C)
classifier_regr.fit(train, classes)
prediction_regr = classifier_regr.predict(test)
```

Pentru fiecare reprezentare, am folosit următoarele valori ale lui C:

Ajustări			
Parametri	Tf-Idf	Bag-of-Words	Word-2-Vec
C	10	1	1000

Figura 4.3: Alegerea parametrilor în funcție de algoritmul de extragere de trăsături

4.3.3 Random Forest

La o primă vedere, algoritmul Random Forest este mai bogat în hiper-parametri (și în optimizările acestora). Pentru început, am limitat numărul maxim de arbori din pădure, prin parametrul *n_estimators*, apoi am setat parametrul *bootstrap*, care indică utilizarea întregului set de date pentru a construi fiecare arbore (acesta este comportamentul în cazul *False*, valoarea *True* fiind implicită). Pentru a păstra o oarecare eficiență timp a execuției clasificării, am limitat adâncimea maximă a arborilor, cu ajutorul parametrului *max_depth*.

Având o bună parte a parametrilor fixați, am trecut la următoarea etapă, de a restricționa situațiile de împărțire a nodurilor, unde am specificat *min_samples_split* reprezentând numărul minim de intrări necesare într-un nod. Apoi, Am setat numărul minim de intrări într-un nod pentru a fi un frunză cu *min_samples_leaf*.

În final, am ales numărul maxim de trăsături pentru a putea face o împărțire a unui nod, *max_features*

```
classifier_rf = RandomForestClassifier(
    min_samples_split,
    n_estimators,
    max_depth,
    max_features,
    bootstrap,
    min_samples_leaf)
```

```

classifier_rf.fit(train, classes)
prediction_nb = classifier_rf.predict(test)

```

Parametri	Ajustări		
	Tf-Idf	Bag-of-Words	Word-2-Vec
min_samples_split	5	2	2
n_estimators	1400	1400	1000
max_depth	80	40	50
max_features	'sqrt'	'auto'	'auto'
bootstrap	False	False	False
min_samples_leaf	1	1	1

Figura 4.4: Alegerea parametrilor în funcție de algoritmul de extragere de trăsături

4.4 Vizualizarea setului de date

Vizualizările grafice ne permit o perspectivă diferită față de cea oferită de clasificatori, în sensul în care aceștia ne pot oferi doar clasele în urma predicției.

Algoritmii folosiți în reprezentarea grafică sunt noi apărute în domeniul Învățării Automate, sunt eficienți și vin cu o abordare diferită față de ceilalți, utilizând diverse procedee matematice pentru a putea realiza o afișare cât mai corectă a datelor.

4.4.1 UMAP

Numele acestei funcții provine de la *Uniform Manifold Approximation and Projection for Dimension Reduction*, de aici prescurtarea UMAP, și este un algoritm de reducere a dimensionalității bazat pe tehnici de învățare a distribuțiilor. În etapa de reprezentare grafică, intervin mai mulți factori, precum k - vecini apropiati, simplexuri și metriki a distanțelor dintre puncte.

```
umap = UMAP(  
    n_neighbors=5,  
    min_dist=0.3,  
    metric='correlation'  
)
```

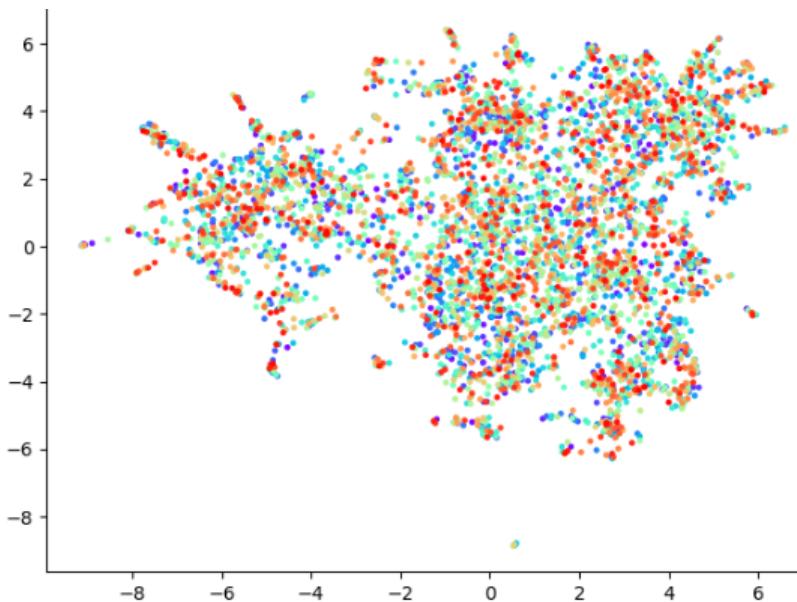


Figura 4.5: Exemplu de reprezentare UMAP

4.4.2 T-SNE

Numele de T-SNE provine de *t-Distributed Stochastic Neighbor Embedding* sau Încorporare t-distribuită stocastică a vecinilor, este un algoritm de vizualizare a setului de date. Aceasta construiește o distribuție probabilistă asupra obiectelor din spațiile n-dimensionale (n are o valoare mare) astfel încât alegerea instanțelor care aparțin aceleiași clase să aibă o probabilitate mare de a fi grupate. Apoi, în spațiul n-dimensional

(unde n are o valoare mică) va realiza o distribuție similară ca în spațiul de mari dimensiuni în același timp reducând divergența punctelor.

```
tsne = TSNE(  
    perplexity=50,  
    n_components=2,  
    n_iter=2000,  
    verbose=1  
)
```

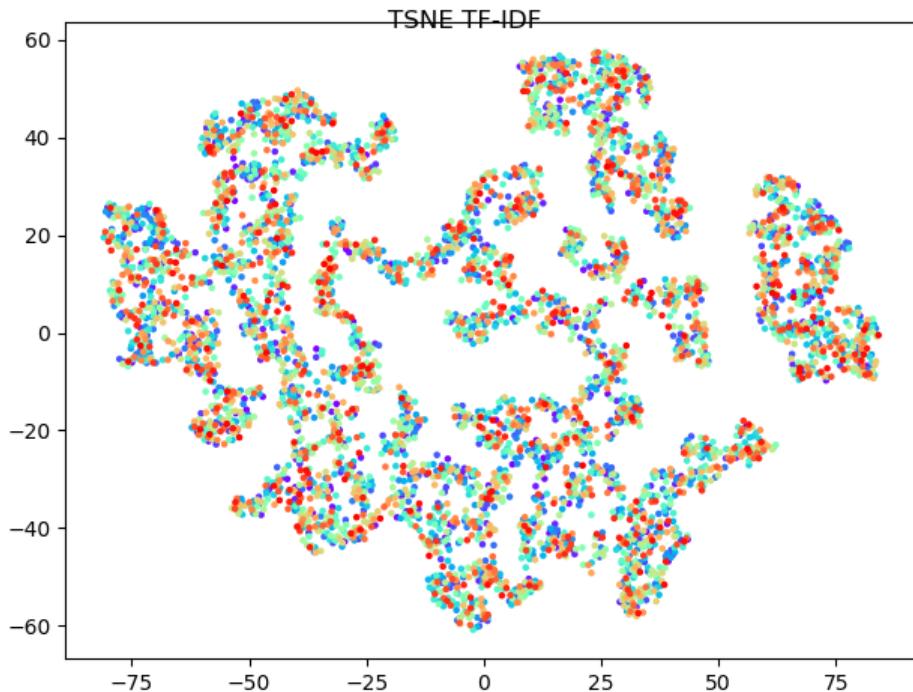


Figura 4.6: Exemplu de reprezentare TSNE

4.5 Tehnologii folosite

4.5.1 Python

Python este un limbaj pe programare interpretat, high-level și utilizat într-o mulțime de domenii, printre care cel de Învățare Automată.

Popularitatea limbajului a crescut constant în ultimii ani, datorită faptului că este ușor de scris, de folosit și de citit, utilizatorii arătând o preferință spre expresivitate în detrimentul eficienței.

Este relativ ușor de învățat, dynamically-typed, adică nu folosește tipuri, și luând în calcul expresivitatea, acesta încurajează productivitatea, de aceea este un limbaj de programare accesibil majorității în ziua de astăzi. La baza dezvoltării Python, stau următoarele principii

- Frumos e mai bun decât urât
- Explicit e mai bun decât implicit



Figura 4.7: Logo-ul Python

- Simplu e mai bun decât complex
- Complex e mai bun decât complicat
- Lizibilitatea contează

Un alt beneficiu al limbajului de programare Python este conținerea unei colecții de biblioteci, în mod special cele de Învățare Automată dezvoltate până acum, cum ar fi *SciKit-Learn*, *TensorFlow + Keras* și pe lângă acestea, conține și biblioteci care să vin în ajutorul procesului, cum ar fi *Pandas* pentru putea manevra setul de date mai ușor și *matplotlib* pentru o vizualizare a setului de date.

Toate acestea adunate, fac Python-ul o unealtă, poate cea mai puternică la momentul actual, în domeniul Învățării Automate și este prima alegere pentru mulți entuziaști sau pentru cei care profesează în acest domeniu. [11, 12]

4.5.2 SciKit-Learn

Această bibliotecă a fost creată cu scopul de a veni în ajutorul procesului de Învățare Automată, punând la dispoziție multiple implementări ale majorității algoritmilor existenți, lăsând utilizatorii să poată modifica hiper-parametrii acestor algoritmi.



Figura 4.8: Logo SciKit-Learn

4.5.3 BeautifulSoup

BeautifulSoup este biblioteca care a ajutat la extragerea rețetelor de pe internet, aceasta punând la dispoziție un crawler cu care se parcurge pagini web în căutarea de informații.

Primul pas în începerea procesului de "crawl" prin site-uri web, este realizarea unui request de încărcare a linkului primit dintr-o listă de bucătării, de exemplu, cea mexicană.

```
webpage = requests.get(link)
```

După realizarea requestului cu succes, instantiem crawler-ul, cu codul html al paginii și specificăm atributului *features* că acesta va parsea un html.

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

Crawler-ul fiind pregătit, va începe să parseze fiecare articol, reprezentat de un chenar în care se va găsi o scurtă descriere a rețetei, o poză, bucătarul și o notă de la 1 la 5. Dintre acestea, noi căutam referința la pagina rețetei, pe care o vom stoca într-o listă de link-uri.

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

Pentru fiecare link din listă, va executa un request de tip GET pentru a-și lua informațiile de pe link-ul respectiv și va începe căutarea a unor noi informații, cum ar fi titlul, instrucțiunile și ingredientele, pe care le va salva într-o structură de tip cheie valoare, exportată într-un fișier exportată drept în format JSON.

```
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"})
```

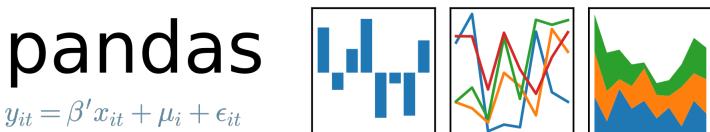
4.5.4 Gensim

Numele provine de la "generate similar" deoarece scopul inițial al bibliotecii era de a genera o listă scurtă cu cele mai similare articole cu un articol dat. În timp, scopul s-a mutat spre claritate, eficiență și scalabilitate iar acum este una dintre cele mai performante biblioteci în domeniul modelării nesupervizate semantice.[13]

În această bibliotecă se regăsește o implementare a algoritmului Word2Vec, fiind inițial dezvoltat de către Google și pus la dispoziție prin intermediul TensorFlow. Am preferat utilizarea modelului Word2Vec din gensim datorită faptului că acesta este destul de ușor de folosit și utilizarea sa nu necesită alte biblioteci auxiliare.

4.5.5 Pandas

Pandas este o bibliotecă cu scopul de a ușura manipularea seturilor de date care pot fi stocate în format json sau csv, venind cu structuri de date și operații proprii, fiind mult mai ușor de utilizat decât cu metoda clasică. De exemplu, dacă pentru json este nevoie de mai multe for-uri pentru a-l putea parcurge, în pandas există o structură de date specializată numită *DataFrame* care simplifică enorm de mult manipularea setului de date.



În realizarea lucrării biblioteca *pandas* a venit în ajutor la generarea seturilor de antrenare/testare, deoarece s-a putut face operații de separare mult mai simplu cu ajutorul *DataFrame*.

4.5.6 Matplotlib

Matplotlib este biblioteca utilizată pentru reprezentările grafice, din cadrul căruia am folosit *scatterplot* pentru reprezentările TSNE/UMAP iar *barchart* pentru diverse statistici pe setul de date.

4.6 Rezultate experimentale

Cu ajutorul ajustării hiper-parametrilor, s-a observat o creștere constantă a acurateței în general, având mai jos date care să susțină această îmbunătățire. La o primă vedere, algoritmul Random Forrest a fost cel mai slab clasat la utilizarea algoritmilor fără niciun fel de ajustare.

Pentru a calcula acuratețea, a fost utilizată metoda de calcul clasică, adică:

$$\text{Acuratețe} = \frac{\text{Numărul de instanțe clasificate corect}}{\text{Numărul total de instanțe}} \cdot 100$$

Rezultate	Tf-Idf	Bag-Of-Words	Word-2-Vec
SVC Liniar	71.13%	65.52%	60.05%
Regresie Logistică	68.34%	63.65%	58.32%
Random Forrest	58.87%	55.32%	48.86%

Figura 4.9: Rezultate inițiale, fără ajustări

Cu ajustarea hiper-parametrilor, se observă o creștere a acurateței în general, unde toate combinațiile de algoritm extragere de trăsături - clasificator, variază peste 70%. Cea mai mare creștere dintre acestea se poate observa la Random Forrest, unde se observă o creștere de la o medie de 54% la 73%.

Rezultate	Tf-Idf	Bag-Of-Words	Word-2-Vec
SVC Liniar	81.41%	76.58%	71.54%
Regresie Logistică	78.29%	76.58%	73.87%
Random Forrest	77.95%	76.6%	65.32%

Figura 4.10: Rezultatele clasificării pe instrucțiuni

Rezultate	Tf-Idf	Bag-Of-Words	Word-2-Vec
SVC Liniar	80.33%	76.58%	56.52%
Regresie Logistică	77.43%	75.29%	61.39%
Random Forrest	77.55%	74.61%	48.72%

Figura 4.11: Rezultatele clasificării pe ingrediente

Putem observa din cele două figuri că rezultatele sunt aproximativ identice pentru cele două moduri de reprezentări, instrucțiuni și ingrediente.

Diferențe majore în clasificare se observă în cazul în care a fost folosit Word-2-Vec pe ingrediente, deoarece după cum se poate observa în figura de mai jos, ingredientele sunt destul de specifice pentru fiecare rețetă în parte. Astfel, orice variație în cantitate, a modului de sătire a carnii, conduce la o reducere substanțială a acurateții.

```
"ingredients": [
    "1 pound flank steak, thinly sliced",
    "5 tablespoons soy sauce",
    "2 1/2 tablespoons white sugar",
    "1/4 cup chopped green onion",
    "2 tablespoons minced garlic",
    "2 tablespoons sesame seeds",
    "2 tablespoons sesame oil",
    "1/2 teaspoon ground black pepper"
],
```

Figura 4.12: Exemplu de listă de ingrediente

Făcând o comparație cu rezultatele din lucrarea [1] și rezultatele obținute în prezența lucrarei, observăm că acestea sunt destul de apropiate ca valoare, ceea ce semnifică faptul că rezultatele pe care le-am obținut sunt destul de bune.

Capitolul 5

Concluzii

În concluzie, după aplicarea algoritmilor de extragere de trăsături și de clasificare putem spune că este posibilă utilizarea metodelor de Învățare Automată în domeniul culinar. Astfel, prin experimentul derulat în prezenta lucrare, am demonstrat că putem clasifica cu succes anumite bucătării folosindu-ne și de ingrediente și de modul de preparare.

Ca direcție de viitor, putem studia pentru fiecare bucătărie ingredientele de bază pentru a urmări evoluția rețetelor în jurul lor sau am putea studia ce efecte poate avea alegerea unor ingrediente în clasificarea la o bucătărie sau alta.

Bibliografie

- [1] Rishikesh Ghewari, Sunil Raiyani *Predicting Cuisine from Ingredients*
<http://jmcauley.ucsd.edu/cse255/projects/fa15/029.pdf>
- [2] Javier Marín, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, Antonio Torralba
Recipe1M: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images
Tue, 30 Oct 2018
<http://arxiv.org/abs/1810.06553>
- [3] Navjot Singh, Ganesh Bagler *Data-driven investigations of culinary patterns in traditional recipes across the world*
<https://arxiv.org/ftp/arxiv/papers/1803/1803.04343.pdf>
- [4] https://en.wikipedia.org/wiki/Information_retrieval
- [5] https://en.wikipedia.org/wiki/Bag-of-words_model
- [6] <https://en.wikipedia.org/wiki/Tfidf>
- [7] <https://www.adityathakker.com/introduction-to-word2vec-how-it-works/>
- [8] https://en.wikipedia.org/wiki/Support-vector_machine
- [9] https://en.wikipedia.org/wiki/Logistic_regression
- [10] https://en.wikipedia.org/wiki/Multinomial_logistic_regression
- [11] <https://www.quora.com/Why-is-Python-so-popular-in-machine-learning>
- [12] [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
- [13] Radim Řehůřek and Petr Sojka *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*
Valletta, Malta, 22 May 2010
<http://is.muni.cz/publication/884893/en>
- [14] Deng, Naiyang and Tian, Yingjie and Zhang, Chunhua *Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions*, 2012
143985792X, 9781439857922
1st, Chapman & Hall/CRC