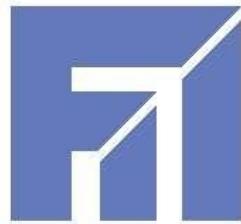


UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Clasificarea automată a melodiiilor în genuri muzicale

propusă de

Dorin - Andrei - Beniamin Miron

Sesiunea: iulie, 2017

Coordonator științific

Conf. Dr. Liviu Ciortuz

UNIVERSITATEA ALEXANDRU IOAN CUZA IAŞI

FACULTATEA DE INFORMATICĂ

Clasificarea automată a melodiilor în genuri muzicale

Dorin - Andrei - Benjamin Miron

Sesiunea: iulie, 2017

Coordonator științific

Conf. Dr. Liviu Ciortuz

DECLARAȚIE PRIVIND ORIGINALITATE ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul „*Clasificarea automată a melodiilor în genuri muzicale*” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imaginile etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, *data*

Dorin – Andrei - Beniamin Miron

DECLARAȚIE DE CONSUMĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Clasificarea automată a melodiilor în genuri muzicale*”, codul sursă al programelor și celealte conținuturi (grafice, multimedia, date de test etc.) care însotesc 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” 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, *data*

Dorin – Andrei - Beniamin Miron

Rezumat

Genurile muzicale sunt etichete create de oameni pentru a caracteriza piesele muzicale și sunt folosite, în mod special, pentru a le organiza pe categorii. Un gen muzical este caracterizat de caracteristicile comune care sunt promovate de membri comunităților ce le susțin. Aceste trăsături, în mod normal, se referă la instrumentele muzicale folosite, la structura ritmului sau la armonia melodiei.

Clasificarea automată a melodiilor poate asista sau poate chiar înlătări modul în care se face în prezent acest proces: manual, de către oameni. Clasificarea automată pe baza genului poate oferi diverse tehnici pentru dezvoltarea și evaluarea atributelor pentru orice fel de analiză a fișierelor audio pe baza conținutului lor.

Această lucrare prezintă un sistem de clasificare automată a melodiilor în 10 genuri, în funcție de variația energiei din jurul mai multor frecvențe. Setul de date folosit este unul ce de-a lungul timpului s-a consacrat în probleme de acest gen. El este public și disponibil sub numele GTZAN. Clasificarea se face folosind mai mulți algoritmi din domeniul învățării automate. Se aplică, pe rând, 6 clasificatori, iar apoi, pe baza rezultatelor obținute este creat un clasificator expert care va fi antrenat pentru a acorda diferite ponderi clasificatorilor deja creați.

Cuprins

Rezumat.....	1
Introducere	5
1 Definirea problemei.....	7
2 Cercetări din domeniu	8
2.1 “Clasificarea semnalelor audio în genuri muzicale”	8
2.2 “Recunoașterea genurilor muzicale”	9
2.3 “Clasificarea genurilor muzicale”	9
2.4 “Studiu pentru înțelegerea fișierelor audio”	10
3 Setul de date si trăsăturile folosite.....	11
3.1 Setul de date	11
3.2 Trăsăturile MFCC	11
3.2.1 Împărțirea în cadre (frame-uri) scurte	11
3.2.2 Estimarea periodogramei spectrului de energie	12
3.2.3 Logaritmarea energiilor.....	13
3.2.4 Transformarea cosinusului discret.....	13
4 Algoritmi utilizați.....	14
4.1 Random Forest	14
4.1.1 Arbori de decizie	14
4.1.2 Bagging	15
4.1.3 Îmbunătățirea algoritmului Bagging	16
4.1.4 Pseudocod Random Forest	17
4.2 Rețele neuronale artificiale.....	18
4.2.2 Neuroni artificiali	18
4.2.3 Structura rețelelor neuronale	19
4.3 K-nearest neighbors (KNN)	21
5.3.1 Pre-procesarea datelor pentru KNN	22
5.3.2 Pseudocod KNN.....	22
4.4 Regresia Logistică	23
4.4.1 Calcularea probabilităților folosind regresia logistică.....	23
4.4.2 Învățarea modelului de regresie logistică	24
4.4.3 Clasificarea cu regresia logistică	25
5 Gradient Boosting	26
5.1 AdaBoost.....	26
5.2 Gradient Boosting	27
5.2.1 Funcția de eroare	27
5.2.2 Clasificatorii slabii	27

5.2.3 Îmbunătățirea modelului	28
6 Detalii de implementare	29
6.1 Limbajul de programare	29
6.2 Biblioteca scikit-learn	29
6.2.1 Documentația	30
6.2.2 Expertiza contribuitorilor	30
6.2.3 Variația domeniilor acoperite	30
6.2.4 Simplitate	31
7 Metodologia de clasificare a melodilor.....	32
7.1 Modele individuale.....	33
7.1.1 Regresie Logistică	33
7.1.2 Rețele Neuronale Artificiale.....	35
7.1.3 Gradient Boosting	36
7.1.4 Random Forest	36
7.1.5 KNN	36
7.2 Asamblarea modelelor individuale.....	36
8.Rezultate obținute.....	38
Concluzii și direcții de continuare a acestui studiu	39
Anexa 1.....	40
Anexa 2	42
Blues VS. Rest	42
Clasic VS. Rest.....	42
Country VS. Rest	43
Disco VS. Rest	43
Jazz VS. Rest.....	44
Hiphop VS. Rest.....	44
Metal VS. Rest	45
Pop VS. Rest	45
Raggae VS. Rest.....	46
Rock VS. Rest	46
Anexa 3.....	47
Anexa 4.....	49
Bibliografie și Webografie	51

Introducere

Dimensiunea mare de informații multimedia de pe internet face necesară construirea unor noi tipuri de utilitare automate capabile să clasifice fișierele audio în genuri muzicale, pe baza conținutului lor. Importanța clasificării melodilor în genuri muzicale a fost subliniată de către A.C.North și David J Hargreaves în studiul lor¹. Aceștia au făcut un experiment prin care au vrut să observe cât de mult îi influentează pe ascultători genul muzical în care este susținută o piesă muzicală. Rezultatul acestui studiu a fost că ascultătorii asociază mai ușor melodii diferite care aparțin aceluiași gen, decât aceeași melodie cântată în genuri diferite.

Necesitatea unui mecanism eficient de clasificare automată a melodilor crește direct proporțional cu numărul de melodii disponibile pe internet. Existența unor algoritmi capabili să clasifice automat melodile în diferite genuri sunt de un real folos celor ce lucrează în domeniul muzical, în special în cazul administratorilor unor colecții mari de melodii. Aceste colecții, în momentul de față, sunt grupate manual. Ca orice alt proces manual și acesta este greoi și anevoieios. Mai mult decât atât în cazul clasificării manuale poate apărea și subiectivitatea și se poate ajunge la situația în care diferiți oameni clasifică aceeași melodie în două genuri diferite, în funcție de preferințele lor. Acest lucru poate duce la multe inconsistențe.

Chiar și muzicologii și-au arătat interesul față de aplicații practice care să clasifice fișierele audio după gen. În momentul de față există o cunoaștere limitată a mecanismului pe care noi, ca oameni, îl folosim, în mod natural, pentru a grupa melodile în genuri muzicale. Nu știm exact atributele ce sunt utilizate precum nici cât de importante sunt. Un sistem care să poată clasifica, în mod automat fișierele audio, ar putea oferi aceste informații.

Lucrarea de față prezintă un sistem de clasificare a melodilor în 10 genuri, în funcție de coeficientii MFCC. Acești coeficienți au fost calculați pentru prima dată de Davis și Mermelstein în 1980² și de atunci sunt un punct de referință în domeniul procesării fișierelor audio. Atunci când au fost creați acești coeficienți s-a încercat să se înțeleagă modul în care urechea umană modeleză sunetele și să se reproducă acest proces.

Setul de date folosit pentru antrenarea modelului este GTZAN. Acesta a fost folosit pentru prima dată în domeniul analizei pieselor audio de către G.Tzanetakis și P.Cook³. Fișierele sunt colectate în anul 2000-2001 din mai multe surse inclusiv CD-uri personale, radio și chiar înregistrări cu microfonul pentru a reprezenta o varietate de condiții de înregistrare a unei piese audio. Setul de date constă în 1000 de melodii de lungime de 30 de secunde. Aceste melodii sunt împărțite în 10 genuri; câte 100 pentru fiecare gen muzical: blues, clasic, country, disco, hip-hop, jazz, metal, pop, reggae și rock.

Prima abordare a fost aceea de a găsi un singur clasificator, cel mai bun, care să obțină cea mai mare acuratețe. S-au încercat, pe rând, următorii clasificatori: Regresie Logistică, Rețele Neuronale, Mașini pe Vector Suport (SVM), Gradient Boosting, Random Forest și KNN, fiecare cu acuratețea lui proprie. Unii clasificatori s-au pliat pe această problemă, având o acuratețe destul de bună, alții nu s-au potrivit la fel de bine. Cea mai bună acuratețe a fost cea a algoritmului Random Forest. Rata sa de succes a fost de 77.975%.

Apoi, în încearcarea de a obține o acuratețe mai bună, s-a mai folosit un clasificator care are rolul de a acorda diferite ponderi celorlalți algoritmi amintiți deja. Această tehnică este cunoscută ca folosirea unui clasificator de experți. S-a dovedit că această abordare a dus la obținerea unei acurații de 88.275%.

Cercetarea în acest domeniu poate avea implicații mai mari decât doar clasificarea după genuri muzicale. Tehnicile dezvoltate pentru crearea sistemelor de clasificare a melodii în diferite genuri pot fi adaptate și pentru alte tipuri de clasificare, spre exemplu clasificarea stilului sau a perioadei de timp când a apărut o melodie.

1 Definirea problemei

Tehnologia actuală permite memorarea semnalelor sonore în format digital și pot fi reprezentate sub forma unei secvențe $S = \langle s_1 s_2 \dots s_N \rangle$, unde s_i reprezintă semnalul sonor de la momentul i , iar N reprezintă dimensiunea fișierului audio sau numărul de momente memorate. Această secvență conține foarte multe informații acustice, trăsături ale ritmului, precum și armonia melodiei ce pot fi extrase și interpretate. Inițial se extrag anumite trăsături la nivel de cadre scurte ale semnalului sonor, apoi aceste trăsături se agregă pentru a obține o imagine de ansamblu despre întreaga secvență ce formează fișierul audio. Așadar, din secvența inițială se poate extrae un vector de trăsături $\bar{X} = \langle x_1, x_2 \dots x_d \rangle$ în care fiecare atribut x_j este extras din S printr-o anumită procedură. În lucrarea de față vectorul \bar{X} de trăsături reprezintă coeficienții MFCC.

Având un vector ce conține anumite trăsături ale unei piese muzicale putem să definim problema de clasificare automată a melodiei după gen ca o problemă clasică de clasificare, folosind attributele obținute drept valori de intrare. Din mulțimea finită de genuri muzicale $G = \{\text{blues, clasic, country, disco, hip-hop, jazz, metal, pop, reggae și rock}\}$, trebuie aleasă o singură categorie \hat{g} care reprezintă cel mai bine genul muzical al melodiei asociate secvenței de semnal digital S .

Dintr-o perspectivă statistică modalitatea de a găsi cel mai probabil gen $\hat{g} \in G$, dându-se vectorul de caracteristici \bar{X} , este definită în felul urmator:

$$\hat{g} = \arg \max P(g | \bar{X}), \quad g \in G$$

unde $P(g | \bar{X})$ reprezintă probabilitatea ca să dândeze vectorul de attribute X , acesta să descrie o piesă muzicală care să aparțină clasei g , $P(g)$ reprezintă probabilitatea apriori ca o nouă melodie să aparțină unui anumit gen (această probabilitate poate fi estimată din frecvența genului g din setul de date), iar $P(\bar{X})$ reprezintă probabilitatea de apariție a vectorului \bar{X} . Ultima probabilitate este necunoscută, de obicei, dar se poate calcula știind că $\sum_{g \in G} P(g | \bar{X}) = 1$. Deci probabilitatea căutată poate fi calculată, din punct de vedere statistic, pentru fiecare $\hat{g} \in G$ cu ajutorul următoarei formule

$$P(g | \bar{X}) = \frac{P(\bar{X} | g) * P(g)}{\sum_{g \in G} P(\bar{X} | g) * P(g)}.$$

2 Cercetări din domeniu

Din perspectiva recunoasterii și creării de modele, clasificarea melodii în funcție de gen este o problemă interesantă de cercetare, din moment ce melodii au o dinamică a semnalului audio foarte variată. Un alt aspect interesant este că clasificarea genurilor este o problemă cu mai mult de două categorii. Pentru a putea face clasificări cu mai mult de două clase există cel puțin două abordări:

1. Folosirea tehniciilor și a algoritmilor care pot clasifica instanțele în mai multe clase, în mod natural, producând o separare complexă a spațiului variabilelor de intrare. Ca exemplu astfel de algoritmi pot fi amintiți arborii de decizie, KNN sau rețelele neuronale.
2. Folosirea unei strategii pentru descopunerea problemei de clasificare în mai multe clase într-o serie de subprobleme de clasificare binară, folosind algoritmi ce vor clasifica spațiul de soluții în mod binar. Algoritmi de acest fel sunt SVM sau clasificare logistică.

De câteva ori s-a încercat o abordare în manieră asamblistă. Aceste abordări s-au dovedit a fi un succes. Ele constau în aplicarea a mai multor clasificatori, nu doar a unuia singur, fiecare specializat pe o anumită parte a problemei. În acest fel, fiecare clasificator este antrenat pe câte un subset diferit de atribute, iar rezultatul final va fi combinația acestor clasificatori de către un alt clasificator ce este antrenat în îmbinarea acestor modele în mod dinamic. Această abordare poate fi privită și ca o mixtură de experți aplicată la problema clasificării. De asemenea, poate fi privită și ca o variație a algoritmului boosting.

O posibilă explicație a succesului acestei metode este că aplicarea clasificatorilor pe doar un subset de atribute face ca suprafața de decizie să fie mult mai simplă, iar clasificatorii pot să producă rezultate cu acuratețe mai mari.

2.1 “Clasificarea semnalelor audio în genuri muzicale”³

Tzanetakis și Cook au fost primii care au abordat și definit această problemă. În studiu³ lor de cercetare ei au propus un set cuprinzător de trăsături ce pot reprezenta o piesă muzicală. Aceste trăsături sunt obținute în urma procesării semnalelor sonore și includ printre altele și atribute ale timbrului vocal, ale ritmului și ale notelor muzicale folosite. Ca algoritmi de clasificare ei au folosit clasificarea gaussiană, mixtura gaussiană și clasificatorul KNN (K Nearest-Neighbors).

Experimentul s-a făcut pe setul de date GTZAN, același set pe care l-am folosit și noi, în această lucrare. Setul de date constă în 1000 de instanțe ce împărțite în 10 genuri muzicale,

cu trăsături extrase din primele 30 de secunde ale fiecărei melodii. Rezultatele obținute de ei indică o acuratețe în jur de 50% folosind toate cele 10 genuri.

2.2 “Recunoașterea genurilor muzicale”⁴

K. Kosina a dezvoltat MUGRAT (**M**usic **G**enre **R**ecognition by **A**nalysis of **T**exture) în lucrarea sa de dizertație. Acesta este un prototip capabil să recunoască genul muzical al unei melodii bazat pe subsetul de date oferit de cei de la MARSYAS. În acest caz atributele au fost selectate doar din segmente de lungime de 3 secunde alese în mod aleator din întregul fișier audio.

Experimentele au fost făcute într-o bază de date alcătuită din 186 de melodii din 3 genuri diferite. Folosind algoritmul de clasificare 3-NN, Kosina a obținut un model ce are acuratețea de aproximativ 88%. În această cercetare, autoarea studiului confirmă faptul că încercarea manuală de a clasifica piesele muzicale după gen duce la rezultate inconsistentе.

2.3 “Clasificarea genurilor muzicale”⁵

Li, Ogihara și Li au prezentat un studiu comparativ între atributele incluse în setul de date MARSYAS și un alt set de atribute bazate pe DWCH (**D**aubechies **W**avelet **C**oefficient **H**istograms), folosind alți algoritmi de clasificare precum SVM (**S**upport **V**ector **M**achines) și LDA (**L**inear **D**iscriminant **A**nalysis). Pentru comparare ei au luat două seturi de date: (a) setul de date folosit inițial de Tzanetakis și Cook (GTZAN), cu atributele extrase de la inceputul melodiei și (b) un set de date compus de 755 de piese muzicale împărțite în 5 genuri, cu atributele extrase din fiecare piesă muzicală de la secunda 31 până la secunda 61. Aceste experimente au arătat faptul că algoritmul de clasificare SVM este mai bun decât celelalte: în cazul (a) având o acuratețe de 72% folosind atributele originale și o acuratețe de 78% folosind atributele DWCH. În cazul (b) acuratețea a fost de 71% pe atributele originale și 74% pe atributele generate de DWCH.

Autorii studiului au mai făcut încă o comparație a strategiilor de descompunere a problemei de clasificare a mai multor categorii în mai multe subprobleme de clasificare binară. Problema inițială de a împărți setul de date în 5 genuri a fost descompusă într-o serie de probleme de clasificare binară: atât folosind strategia OAA (**O**ne-**A**gainst-**A**ll) cât și folosind RR (**R**ound-**R**obin). Rezultatele cele mai bune au fost obținute folosind algoritmul SVM și metoda OAA de descompunere a problemei.

2.4 “Studiu pentru înțelegerea fișierelor audio”⁶⁷

Grimaldi, Cunningham și Kokaram au încercat diverse strategii de descopunere a acestei probleme folosind algoritmi de clasificare specializați doar pe o parte din problemă și apoi combinarea rezultatelor cu un clasificator expert. Autorii au descompus problema originală folosind OAA (**One-Against-All**), RR (**Round-Robin**) și o tehnică de selectarea aleatoare în subspații.

Ei au mai comparat și diferite metode de selecție a trăsăturilor ordonându-le după importanța lor folosind mai multe tehnici de ordonare a lor precum IG (**Information Gain**) sau GR (**Gain Ratio**) sau folosind algoritmi de combinare a atributelor precum PCA (**Principal Component Analysis**).

Cercetarea lor s-a bazat pe un set de date ce conține 200 de piese muzicale ce sunt grupate în 5 genuri și au folosit KNN ca algoritm de clasificare. Trăsăturile au fost obținute de pe întregul fișier audio folosindu-se DWT (Discrete Wavelet Transform). Pentru clasificatorul KNN transformarea PCA a fost cea mai eficientă tehnică de selecție a trăsăturilor ajungându-se la o acuratețe de 79%. Metoda RR de asamblare a reușit să obțină o acuratețe de 81% și când s-a folosit IG dar și atunci când s-a folosit GR.

3 Setul de date si trăsăturile folosite

3.1 Setul de date

Setul de date folosit pentru antrenarea modelului este GTZAN. Acesta a fost folosit pentru prima dată în domeniul analizei pieselor audio de către G.Tzanetakis și P.Cook3.

Fișierele sunt colectate în anul 2000-2001 din mai multe surse inclusiv CD-uri personale, radio și chiar înregistrări cu microfonul pentru a reprezenta o varietate de condiții de înregistrare a unei piese audio. Setul de date constă în 1000 de melodii de lungime de 30 de secunde. Aceste melodii sunt împărțite în 10 genuri; câte 100 pentru fiecare gen muzical: blues, clasic, country, disco, hip-hop, jazz, metal, pop, reggae și rock.

3.2 Trăsăturile MFCC

Primul pas ce trebuie făcut pentru clasificarea melodiei este acela de a extrage anumite trăsături din ele. Partea cea mai importantă este încerarea de a înțelege modul în care urechea umană modelează sunetele și cât de bine acest proces este reproducus.

Mel Frequency Coefficients (MFCC) sunt caracteristici folosite foarte frecvent în vorbirea automată și în recunoașterea vocală. Acești coeficienți au fost calculați prima dată de Davis și Mermelstein⁸ în 1980 și de atunci au rămas un punct de referință în domeniul procesării fișierelor audio.

Atunci când vrem să calculăm acești coeficienți trebuie să trecem prin mai multe etape:

1. Împărțirea în cadre (frame-uri) scurte;
2. Pentru fiecare frame se va face o estimare a periodogramei spectrului de energie;
3. Se logaritmează fiecare energie;
4. Se calculează DCT din valorile logaritmice la pasul anterior;
5. Se păstrează primii 13 coeficienți.

3.2.1 Împărțirea în cadre (frame-uri) scurte

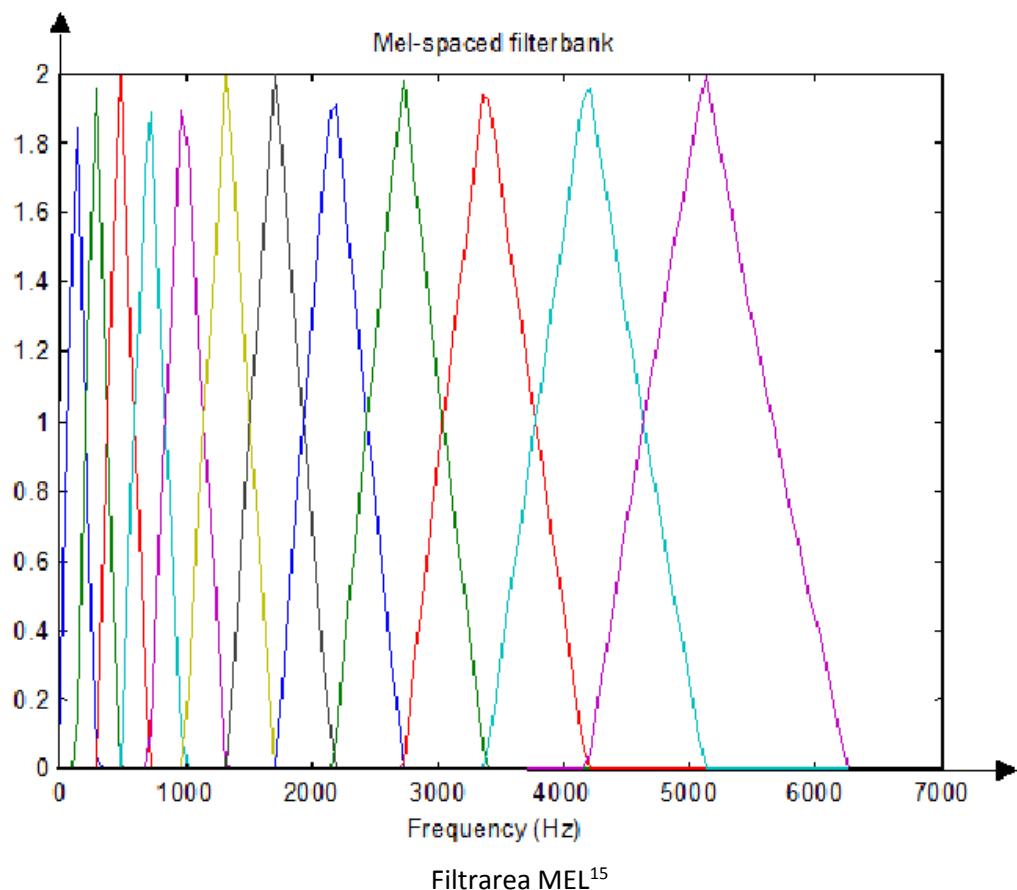
Un semnal audio se schimbă în continuu, dar pentru simplitate putem presupune că pentru perioade scurte de timp, semnalul audio nu se schimbă foarte mult (această presupunere este făcută din punct de vedere statistic; este evident faptul că se schimbă chiar și în intervale foarte mici de timp). De aceea vom împărți semnalul audio în frame-uri de 20-30 milisecunde. Dacă frame-ul ar fi mai mic atunci nu ar fi existat destule date pentru a realiza o estimare

rezolabilă a variației energiei, iar dacă frame-ul ar fi mai mare atunci diferențele dintre frameuri nu ar putea fi atribuite rezonabile pentru problema identificării genului unei melodii.

3.2.2 Estimarea periodogramei spectrului de energie

Următorul pas este să calculăm energia din jurul mai multor frecvențe pentru fiecare frame. Acest pas simulează funcția cohleei, denumită și melcul membranos. Aceasta este un organ din urechea internă umană care vibrează în diferite locuri în funcție de frecvența pe care o are sunetul. În funcție de locul în care vibrează cohleea, diferenți nervi se activează și primesc informații pe care le vor transporta către creier. Periodograma face cam același lucru, identificând frecvența din fiecare frame.

Estimarea făcută de periodogramă încă mai conține multe informații nefolositoare. În special cohleea nu poate face diferența dintre două frecvențe apropiate. Acest efect devine din ce în ce mai pronunțat pe măsură ce frecvența crește. Din acest motiv, vom grupa mai multe estimări și le vom aduna pentru a avea o idee despre cât de mult variază energia în diferite regiuni ale melodiei. Acest lucru este făcut de filtrarea Mel. Filtrarea Mel cuprinde o grupare de filtre: primul filtru ne va spune cât de multă energie există aproape de 0 Hertz. Cu cât frecvența crește, filtrele se adaptează astfel încât diferența de energie să nu mai fie o problemă.



3.2.3 Logaritmarea energiilor

După ce s-a filtrat energia din sunetul audio, pe mai multe frecvențe, trebuie să le logaritmăm. Această logaritmare se face pentru a ne apropiă cât mai mult de modul în care funcționează urechea umană: noi nu auzim intensitatea sonoră pe o scală liniară. În general, pentru a dubla volumul unui sunet trebuie să fie consumată de până la 8 ori mai multă energie. Acest lucru înseamnă că variațiile mari de energie pot să nu sună la fel de diferit dacă sunetul era mult mai intens la început. Această operație face ca atributele noastre să fie mai asemănătoare cu ceea ce auzim.

3.2.4 Transformarea cosinusului discret

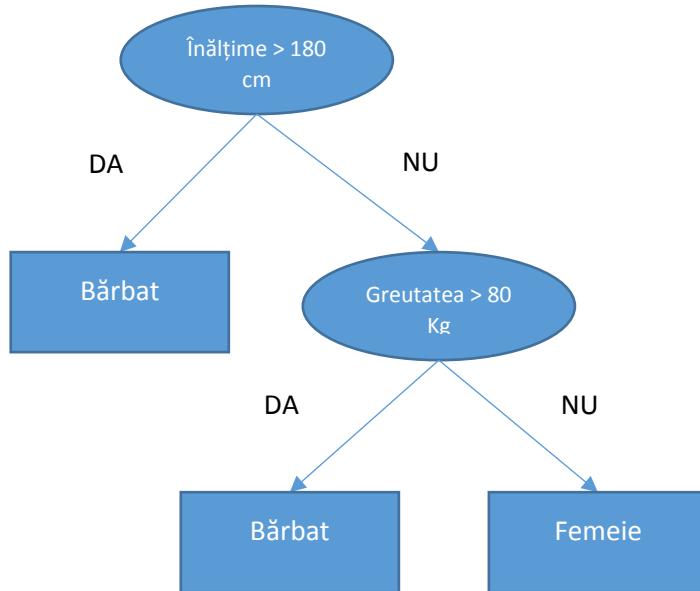
Ultimul pas este o transformare numită DCT (transformarea discretă a cosinusului). Ea este aplicată peste valorile obținute la pasul anterior. Există cel puțin 2 motive mari pentru care se execută această operație. În primul rând, filtrele pe care le-am aplicat se suprapun și sunt destul de corelate între ele. DCT reușește să decoreleze energiile. În al doilea rând, coeficienții MFCC conțin doar 13 valori, nu 26, câți rezultă după filtrarea MEL. Acest lucru se întâmplă deoarece ultimii 13 coeficienți conțin schimbări foarte brăzde de energie și acest lucru afectează procesarea ulterioară.

4 Algoritmi utilizați

4.1 Random Forest

4.1.1 Arbori de decizie

Arborii de decizie constituie o metodă foarte folosită în practică. Ei pot fi folosiți atât pentru clasificare, cât și pentru regresie. Arborii de decizie antrenați pot fi reprezentați și ca un set de reguli de tip if-else, pentru a putea fi citiți mai usor. Iar etapa de antrenare nu este altceva decât învățarea ordinii atributelor ce trebuie testate și a condițiilor ce trebuie validate.



Arborii de decizie clasifică instanțele sortându-le de-a lungul drumului de la rădăcină la frunze. Fiecare nod din arbore este, de fapt, un test asupra unui atribut al instanței ce urmează să fie clasificat. Fiecare ramură ce descinde dintr-un nod corespunde unei posibile valori a atributului ce tocmai a fost testat.

Clasificarea unei noi instanțe pornește de la rădăcina arborelui, apoi se testează, pe rând, atributele specificate în nodul curent, pentru ca, mai apoi, să se treacă la subarborele ce corespunde valorii atributului instanței în cauză. Acest proces se repetă până când se ajunge la o frunză a arborelui. Valoarea din frunza la care s-a ajuns este categoria în care va fi încadrată noua instanță.

Algoritmul învață arboriele de decizie construindu-i de sus (de la rădăcină) în jos (spre frunze). Cea mai importantă alegere pe care o face acest algoritm este selectarea trăsăturii de test pentru fiecare nod din arbore. La fiecare pas se alege acel atribut care este cel mai folositor pentru clasificare. Pentru a măsura care atribut este cel mai potrivit pentru un anumit pas s-a

introdus o proprietate statistică numită “informația câștigată” (**Information Gain**), ce măsoară cât de bine separă datele de antrenare atributul ales.

Pentru a putea vorbi de IG (**Information Gain**) trebuie definită o modalitate de măsurare a (im)puritații unei colecții de exemple (de instanțe), pe care o vom numi **entropie**. O colecție este mai ușor de clasificat dacă ea este mai pură. Când se alege un atribut pentru un nod, acesta trebuie să împartă colecția de exemple în subcolecții a căror sumă de impuritate să fie cat mai mică decât impuritatea colecției inițiale. Altfel, acea divizare nu a adus nici un câștig de informație.

Dându-se o colecție de instanțe S , cu exemple atât pozitive cât și negative, entropia colecției S relativă la clasificarea booleană este: $\text{entropie}(S) \equiv -p_{\oplus} * \log_2 p_{\oplus} - p_{\ominus} * \log_2 p_{\ominus}$, unde p_{\oplus} este proporția exemplelor pozitive din S , iar p_{\ominus} este proporția exemplelor negative din S . În calcule se va considera că $0 * \log 0$ este egal cu 0.

Având o modalitate de a măsura impuritatea datelor de antrenare, se poate defini castigul informational, adică putem măsura cât de bun este un atribut pentru clasificarea datelor. IG-ul (**Information Gain**) nu face altceva decât să spună dacă atunci când se împart datele de antrenare după un atribut entropia va fi mai mică. Cu alte cuvinte, avem un câștig de informație atunci când, împărțind colecția de date după un anumit atribut, suma entropiilor subcolecțiilor este mai mică decât entropia colecției inițiale. Formal putem scrie acest lucru în felul urmator:

$$IG(P, h) \equiv \text{Entropie}(P) - (\text{Entropie}(P \text{ cu } h > 180cm) + \text{Entropie}(P \text{ cu } h < 180cm))$$

Unde P este o colecție de persoane, iar h este un atribut al persoanelor și anume înălțimea.

4.1.2 Bagging

Bootstrap Aggregation (**Bagging**) este o metodă foarte simplă dar și foarte puternică de asamblare. O metodă de asamblare este o tehnică ce combină predicția făcută de mai mulți clasificatori pentru a obține o predicție ce are acuratețea mai mare decât orice alt model luat individual.

Arborii de decizie sunt sensibili la datele pe care sunt antrenați. Cu alte cuvinte, dacă datele de antrenare sunt schimbate, sau dacă arborele e antrenat pe un subset din datele de intrare, atunci modelul rezultat poate fi destul de diferit și poate chiar să facă predicții diferite.

Să presupunem că avem un set de date de 1000 de instanțe și vrem să creem arbori de decizie. Metoda Bagging funcționează în felul următor: în primul rând se crează mai multe sub-seturi de date, împărțind setul initial (ex. se vor crea 100 sub-seturi). Apoi pentru fiecare subset

de date se va antrena câte un arbore de decizie. Ultima fază este cea de predicție: pentru a reuși să facem o predicție pentru o nouă instanță, o vom clasifica cu fiecare arbore de decizie creat, iar predicția finală va fi categoria în care a fost încadrată de cele mai multe ori noua instanță.

Spre exemplu să presupunem ca au fost antrenați 5 arbori de decizie care fac următoarele predicții pentru o nouă instanță: femeie, femeie, barbat, femeie, barbat. Vom considera că instanța aparține categoriei care a apărut de cele mai multe ori. În acest exemplu vom face predicția “femeie”.

4.1.3 Îmbunătățirea algoritmului Bagging

Random Forest este unul dintre cei mai cunoscuți, dar în același timp și unul din cei mai puternici algoritmi din învățarea automată. Este o îmbunătățire a algoritmului de învățare automată descris mai sus, Bootstrap Aggregation sau Bagging.

Problema arborilor de decizie este că sunt construși în manieră greedy. Adică, la fiecare pas trebuie să aleagă o variabilă după care să facă divizarea spațiului datelor de intrare astfel încât să minimizeze eroarea de antrenare (și, în mod implicit și eroarea de testare/validare). Chiar dacă aplicăm metoda Bagging, arborii de decizie rezultați pot avea o structură similară, iar predicția lor va fi corelată.

Combinarea predicțiilor de la multiple modele are rezultate mai bune atunci când predicțiile sub-modelelor sunt necorelate sau cât mai puțin corelate. Random Forest schimbă algoritmul de creare al arborilor într-o manieră în care predicția lor va fi mai puțin corelată.

Când se construiesc, în modul clasic, arborii de decizie, atunci când algoritmul trebuie să selecteze o trăsătură pentru a face divizarea, el are la dispoziție toate atributele datelor de intrare și toate valorile posibile pentru a selecta divizarea optimă. Algoritmul de Random Forest modifică această procedură în aşa fel încât algoritmul de învățare să fie limitat la un eșantion aleator de atrbute din care să facă alegerea.

Numărul de atrbute din care se poate alege la fiecare divizare poate fi specificat ca parametru pentru algoritm. Pentru a găsi cel mai bun numar de trăsături se folosesc diferite valori și se compară rezultatele de la cross-validation. Valoarea standard pentru clasificare este radical din numărul total de atrbute, iar pentru regresie se va alege doar o treime din totalul de atrbute.

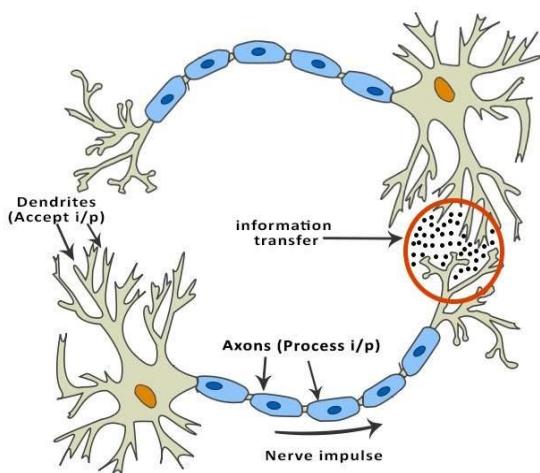
4.1.4 Pseudocod Random Forest

Parametri: Un set de date de antrenare $S = (x_1, y_1), \dots, (x_n, y_n)$, atributele F și numărul de arbori folosiți B

```
1     function RandomForest( $S, F$ )
2          $H \leftarrow \emptyset$ 
3         for  $i \in 1, \dots, B$  do
4              $S^{(i)} \leftarrow$  Se seleactează un eșantion din  $S$ 
5              $h_i \leftarrow$  CreazaArboreNou( $S^{(i)}, F$ )
6              $H \leftarrow H \cup \{h_i\}$ 
7         end for
8         return  $H$ 
9     end function
10    function CreazaArboreNou( $S, F$ )
11        Pentru fiecare nod:
12             $f \leftarrow$  o submulțime foarte mică a lui  $F$ 
13            Împarte folosind cea mai bună trăsătură din  $f$ 
14            return Arborele creat
15    end function
```

4.2 Rețele neuronale artificiale

Ideea din spatele rețelelor neuronale artificiale se bazează pe ipoteza că ceea ce se întâmplă în creierul nostru atunci când luam deciziile corecte, poate fi imitat folosind calculatorul, simulând în acest fel neuronii și dendritele.



Exemplu de funcționare a neuronului¹⁴

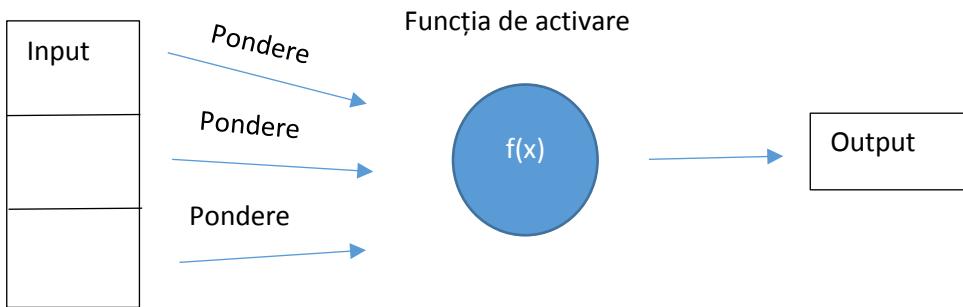
Corpul uman este compus din 100 de miliarde de celule nervoase. Ele sunt conectate unele cu altele prin axoni. Stimulii din mediu extern sau datele de intrare de la organele senzoriale sunt acceptate de dendrite. Acești stimuli crează impulsuri electrice care traversează rețeaua neuronală. Un neuron poate trimite mesajul la un alt neuron pentru a se ocupa de problemă sau poate decide să nu-l trimită mai departe, caz în care stimulul extern este ignorat.

Rețelele Neuronale Artificiale (Artificial Neural Networks - ANN) sunt compuse din noduri multiple, care încearcă să imite funcționalitatea neuronilor biologici din creierul uman. Neuronii sunt conectați și interacționează unii cu alții prin legăturile dintre ei. Nodurile primesc valori de intrare și aplică operații simple asupra acestor date. Rezultatul acestor operații este pasat altor neuroni.

Puterea rețelelor neuronale vine din abilitatea lor de a învăța, pe de o parte, reprezentarea datelor de antrenare și, pe de alta parte, relația dintre aceste date cu variabila de ieșire, care trebuie prezisă. În acest sens, rețelele neuronale învață o mapare. Din punct de vedere matematic, ele sunt capabile să învețe orice funcție și a fost demonstrat că pot fi privite ca un algoritm universal de aproximare.

4.2.2 Neuroni artificiali

Unitatea de bază dintr-o rețea neuronală sunt neuronii. Aceștia sunt unități simple de calcul ce au câte o pondere pentru fiecare valoare de intrare și se calculează valoarea de ieșire folosind o anumită funcție de activare.



Ponderile sunt de obicei inițializate cu valori aleatoare mici, cuprinse în intervalul $[0,0.3]$, dar există și modalități mai complexe de inițializare a lor, dar pe care nu le vom discuta aici.

Funcția de activare transmite o valoare mai departe în funcție de suma produselor dintre ponderi și atributele de intrare. Această valoare reprezintă rezultatul neuronului. Dacă suma este mai mare decât un anumit prag (sa spunem mai mare de 0.5), atunci rezultatul neuronului va fi 1, altfel va fi 0.

Cel mai mult des folosite funcții de activare sunt cele neliniare. Aceste funcții permit rețelelor neuronale să combine mai multe valori de intrare în moduri mult mai complexe, îmbunătățind astfel capabilitatea modelului creat de întreaga rețea. Un exemplu de funcție neliniară este funcția logistică (sau sigmoid). Rezultatul este o valoare reală între 0 și 1.

4.2.3 Structura rețelelor neuronale

Neuronii sunt grupați și formează, împreună, rețelele neuronale. Un rând de neuroni formează un strat din rețeaua neuronală. În practică, rețelele neuronale sunt construite din mai multe strate.

4.2.3.1 Nivel de intrare

Primul strat se mai numește stratul de intrare (“Input Layer”) sau stratul vizibil deoarece cu acest strat se interacționează, fiind partea expusa din rețeaua neuronală care intermediază trecerea de la datele de intrare la restul rețelei. De obicei, acest strat are câte un neuron pentru fiecare atribut pe care îl au instanțele cu care se lucrează. Acest strat nu conține același tip de neuroni ca cei pe care i-am descris mai sus, ci sunt pur și simplu niște intermediari pentru a face legătura dintre celelalte straturi și datele de intrare.

4.2.3.2 Nivelurile ascunse

Următoarele straturi, după cel de intrare sunt numite “straturi ascunse” deoarece ele nu interacționează cu datele de intrare ci, ele, primesc ca valori de intrare, rezultatul stratului

anterior. Cea mai simplă structură de rețea neuronala este cea care are un singur neuron în stratul ascuns. Rezultatul acestui neuron va constitui și rezultatul final al rețelei.

Având în vedere puterea de calcul disponibilă și ușor de accesat⁹ dar și bibliotecile eficiente ce există, se pot construi așa numitele “deep neural networks”. Acestea sunt speciale, deoarece conțin multe straturi ascunse. În trecut, dura foarte mult ca acestea să fie antrenate, dar acum acest timp a fost redus la câteva minute folosind noi tehnici și hardware modern.

4.2.3.3 Nivelul de ieșire

Ultimul strat se numește stratul de ieșire și este responsabil pentru a întoarce o valoare sau un vector de valori ce corespund formatului cerut de problemă. La ultimul strat se pune problema a cât de mulți neuroni trebuie să existe aici și ce funcție de activare este mai bună având în vedere problema care trebuie rezolvată.

Pentru o problemă de regresie acest strat este construit dintr-un singur neuron și nu este necesară nici o funcție de activare.

Pentru o problemă de clasificare binară e recomandat să fie tot un singur neuron pe acest strat și să se folosească funcția sigmoid (numită și funcția logistică ce este folosită la regresie logistică), funcție a cărui rezultat este o valoare între 0 și 1 reprezentând probabilitatea ca instanța curentă să aparțină primei categorii. Iar apoi clasificarea se va face folosind un prag cu valoarea de 0.5. Dacă probabilitatea este mai mare de 0.5 atunci instanța va fi clasificată în prima categorie, altfel va fi clasificată în a doua categorie.

Pentru o problemă de clasificare, cu mai mult de două categorii, este bine să avem mai mulți neuroni pe ultimul strat. În practică se folosește câte un neuron pentru fiecare categorie. În acest caz funcția de activare recomandată este softmax a cărui rezultat este probabilitatea prezisă de rețea ca instanța testată să aparțină fiecărei categorii. Clasificarea constă în a alege categoria cu cea mai mare probabilitate din aceste valori.

4.3 K-nearest neighbors (KNN)

KNN nu construiește alt model decât acela de a memora întregul set de date de antrenare, deci nu este nevoie de o etapă de învățare. Deoarece setul de date este memorat, aceste date trebuie să fie consistente. În practică, se recomandă ca setul de date să fie preprocesat. Actualizarea modelului cu date noi se poate face foarte ușor, fără să fie nevoie de procesări complexe.

Predicția pentru o nouă instanță (X) se bazează pe căutarea în întregul set de date a celor mai asemănătoare K instanțe (a celor mai apropiate K vecini). În cazul regresiei rezultatul va fi media valorilor vecinilor, iar în cazul clasificării rezultatul va fi categoria cu cel mai mare număr de vecini.

Pentru a determina care sunt cele mai apropiate K instanțe cu instanța ce trebuie clasificată trebuie să se folosească o măsură de distanță. Pentru valorile reale ale variabilelor de intrare, cea mai cunoscută măsură de distanță este distanța euclidiană. Aceasta se definește ca fiind rădăcina pătrată a sumei diferențelor valorilor de pe fiecare dimensiune la pătrat dintre punctul nou (X_{nou}) și punctul existent (X_i). Aceasta este distanța clasică dintre două puncte.

$$d(X_{nou}, X_i) = d(X_i, X_{nou}) = \sqrt{(X_{i_1} - X_{nou_1})^2 + (X_{i_2} - X_{nou_2})^2}$$

Alte măsuri de distanțe populare sunt:

- Distanța Hamming: Calculează distanța dintre doi vectori binari;
- Distanța Manhattan: Calculează distanța dintre doi vectori reali, sumând valoarea absolută a diferențelor;
- Distanța Minkowski: este o generalizare a distanței euclidiene și a distanței Manhattan.

Pentru fiecare problemă trebuie căutată masura de distanță care se potrivește cel mai bine în funcție de proprietățile datelor cu care se lucrează. Atunci când nu se cunoaște cea mai bună măsură de distanță, se pot experimenta mai multe metriki de distanță cu diferite valori pentru K și se va alege modelul care va avea cea mai bună acuratețe.

Spre exemplu, distanța Euclidiană este o metrică bună atunci când instanța de intrare are attribute numerice la o scală asemănătoare (ex.lungimea și lățimea dreptunghiurilor). Distanța Manhattan este o metrică bună pentru cazul în care variabila de intrare nu are attribute de același tip (ex.vârstă, sexul, greutatea unei persoane).

Valoarea cea mai bună pentru K poate fi găsită prin încercarea mai multor valori și selectarea celei mai bune. Este o bună idee să se folosească valori mai mici decât 21.

Complexitatea algoritmului KNN crește pe măsură ce cresc și datele de antrenare. Pentru seturi de date foarte mari, KNN va alege random un eșantion din care va calcula cei mai apropiati K vecini.

Când se folosește KNN pentru clasificare, predicția va fi calculată în funcție de clasa care are cele mai multe apariții în cei mai apropiati K vecini. Fiecare instanță, în esență, votează pentru clasa căreia îi aparține și clasa cu cele mai multe voturi câștigă.

Se poate calcula și probabilitatea cu care o instanță aparține unei clase. Spre exemplu, pentru o problemă de clasificare binară: $P(\text{bărbat}) = \frac{\text{Numărul de instanțe apropiate de tipul bărbat}}{\text{Numărul total de vecini, } K}$.

Dacă există un număr par de clase, atunci este o bună idee ca valoarea lui K să fie impară, pentru a evita situațiile în care două clase pot avea același număr de voturi, situație în care apare un caracter random în algoritm. Iar dacă avem un număr impar de clase, este o bună idee ca numărul de vecini pe care-i vom lua în considerare (K) să fie un număr par.

5.3.1 Pre-procesarea datelor pentru KNN

- Re-scalarea datelor: KNN funcționează bine dacă toate datele sunt la aceeași scală. Se recomandă ca datele să se normalizeze într-un interval cuprins între 0 și 1.
- Ștergerea instanțelor care nu au toate atributele: dacă lipsesc atrbute, înseamnă că nu se poate calcula similaritatea cu noile instanțe. Acest lucru înseamnă că aceste date oricum nu pot fi folosite și este natural, să le scoatem din setul de date.
- Numarul de dimensiuni: KNN se pliază pentru problemele în care datele sunt dispuse într-un spațiu cu dimensiuni cât mai puține. Se poate aplica și pe mai multe dimensiuni (de ordinul sutelor de atrbute), dar nu va avea aceeași acuratețe ca alte tehnici de clasificare. Pentru KNN se poate aplica o reducere de dimensiuni asupra atrbutelor de intrare.

5.3.2 Pseudocod KNN

Parametri: Un set de date de antrenare $S = (x_1, y_1), \dots, (x_n, y_n)$, o instanță de testare I

```
function kNN(S, I)
    1. Iterarea prin fiecare item din setul de date și calcularea
       distantei de la acesta la instanța de testare
    2. Clasifica noua instanță cu clasa majoritară dintre cei mai
       apropiati vecini ai săi din setul de date
end function
```

4.4 Regresia Logistică

Regresia logistică este o tehnică împrumutată de învățarea automată din statistică. Această metodă se aplică, în special, în cazul problemelor de clasificare binară. Numele vine de la funcția nucleu care stă în centrul acestei metode: funcția logistică.

Funcția logistică, numită și funcția sigmoid, a fost creată de statisticieni pentru a descrie proprietățile creșterii populației într-un anumit mediu. Această funcție poate lua ca argument orice număr real și îi atribuie o valoarea între 0 și 1, dar niciodată exact aceste limite.

$$\frac{1}{1 + e^{-x}}$$

Regresia logistică folosește o ecuație ca model al datelor din acest motiv se aseamană foarte mult cu regresia liniară. Pentru fiecare valoare de intrare (X) se aplică o combinație liniară folosind diferite ponderi (coeficienți) pentru a se prezice o valoare de ieșire (y). Diferența față de regresie liniară este că valoarea de ieșire este binară, luând valori din mulțimea {0, 1} și nu valori numerice.

Următoarea ecuație reprezintă un exemplu pentru acest mod de clasificare:

$$y = \frac{1}{1 + e^{-(b_0 + b_1 * x)}}$$

Unde y este rezultatul, iar b_0 și b_1 sunt coeficienții pentru atrbutele valorii de intrare (X). Dacă datele de intrare sunt formate din mai multe atrbute (coloane), atunci pentru fiecare atrbut vom avea asociat câte un coeficient b (o valoare reală, constantă) care trebuie învățat din datele de antrenare.

Reprezentarea modelului pe care trebuie să o memorăm sunt doar coeficienții ecuației, adică valorile corespunzătoare pentru fiecare b .

4.4.1 Calcularea probabilităților folosind regresia logistică

Regresia logistică modelează probabilitatea unei instanțe de a apartine unei categorii (ex. Prima categorie). De exemplu, dacă am modela sexul oamenilor (bărbați/ femei), în funcție de înălțimea lor, atunci prima categorie poate fi *bărbat*, iar modelul regresiei logistice poate fi scris sub forma de probabilitate: probabilitatea ca dându-se înălțimea unei persoane, aceasta să fie fie bărbat. Mai formal:

$$P(\text{sex} = \text{bărbat} | \text{înălțime}) .$$

Generalizând putem spune că regresia logistică modelează probabilitatea ca dându-se o instanță de intrare (X), aceasta să aparțină unei clase prestabilite (Y=1). Formal putem scrie în felul următor:

$$P(X) = P(Y = 1 | X)$$

Regresia logistică este o metodă liniară, dar predicția este transformată de funcția logistică. Din acest motiv nu mai putem să privim rezultatul predicției ca o combinație liniară a datelor de intrare, aşa cum se întâmplă în cazul regresiei liniare. Aplicând funcția logistică pe probabilitatea anterioară, putem privi modelul obținut în felul următor:

$$P(X) = P(Y = 1 | X) = \frac{1}{1 + e^{-(b_0 + b_1 * x)}} = \frac{1}{1 + \frac{1}{e^{(b_0 + b_1 * x)}}} = \frac{e^{(b_0 + b_1 * x)}}{e^{(b_0 + b_1 * x)} + 1}$$

În urma unor transformări imediate, se poate observa că:

$$P(X) = \frac{e^{(b_0 + b_1 * x)}}{e^{(b_0 + b_1 * x)} + 1} \leftrightarrow \frac{P(X)}{(1 - P(X))} = e^{(b_0 + b_1 * x)}$$

Logaritmând ajungem la:

$$\ln\left(\frac{P(X)}{1 - P(X)}\right) = b_0 + b_1 * x$$

Această formulă este folositoare, deoarece se pot face calculele din partea dreaptă în mod liniar (în același mod ca la regresia liniară), iar rezultatul, din partea stângă, poate fi privit ca logaritmul din probabilitatea ca instanta X să apartină primei categorii.

4.4.2 Învățarea modelului de regresie logistică

Coeficienții algoritmului de regresie logistică trebuie să fie estimați din datele de antrenare. Această estimare se realizează folosind o metodă numită “estimarea probabilității maxime” (Maximum Likelihood Estimation) - MLE.

Această tehnică de estimare a coeficienților este comună în algoritmii de învățare automată, chiar dacă face anumite presupuneri despre distribuția datelor. Cei mai buni coeficienți vor determina un model care va face o predicție foarte aproape de 1 (persoana este bărbat) pentru categoria de bază și o valoare foarte aproape de 0 (persoana este femeie) pentru cealaltă categorie. În mod intuitiv această tehnică de estimare, pentru cazul regresiei logistice poate fi privită ca o metodă de a căuta cei mai buni coeficienți astfel încât eroarea (discordanța dintre categoria prezisă și categoria reală) să fie cât mai mică.

4.4.3 Clasificarea cu regresia logistică

Predicția categoriei cu regresia logistică este simplă. Să presupunem că avem un model care face predicția dacă o persoană este bărbat sau femeie, pe baza înălțimii.

Dându-se o persoană de 150 de cm să se determine dacă aceasta este bărbat sau femeie. Să presupunem ca am învățat deja coeficienții și aceștia sunt $b_0 = -100$ și $b_1 = 0.6$. Folosind ecuația de mai sus putem calcula probabilitatea ca o persoană de 150 de cm să fie bărbat:

$$P(\text{sex} = \text{barbat} | \text{înălțime} = 150) = \frac{e^{(b_0 + b_1 * x)}}{e^{(b_0 + b_1 * x)} + 1} = \frac{e^{(-100 + 0.6 * 150)}}{e^{(-100 + 0.6 * 150)} + 1} \approx 0.00004$$

Deci probabilitatea este aproape 0 ca această persoană să fie bărbat. În practică, putem folosi direct probabilitatile și vom face clasificarea în felul următor: Dacă probabilitatea este mai mică decât 0.5, atunci acea persoană este catalogată ca fiind bărbat, altfel va fi catalogată femeie.

5 Gradient Boosting

Gradient Boosting face parte din categoria celor mai puternici tehnici de construire a modelelor din învățarea automată. Ideea unui algoritm de tip boosting (de îmbunătățire) este că un clasificator slab poate fi modificat pentru a deveni mai bun.

Un clasificator slab este definit ca fiind un clasificator a cărui performanță este puțin mai bună decât a unei clasificări aleatoare. Ideea din spatele acestui tip de algoritm este de a filtra instanțele de intrare, lăsându-le doar pe cele pe care clasificatorul slab să se concentreze și să se antreneze pentru a le clasifica corect și astfel să devină un clasificator mai bun.

“Ideeia este de a folosi o metodă de învățare slabă de câteva ori pentru a reuși să satisfacem ipoteza, de fiecare dată concentrându-ne atenția pe acele instanțe pe care în etapa anterioară le-am clasificat greșit.”¹⁰

5.1 AdaBoost

Prima implementare a unui algoritm de tip boosting care a avut succes în aplicații practice este Boostingul Adaptiv sau AdaBoost pe scurt.

“Boosting se referă la problema generală de a produce un model cu acuratețe foarte mare combinând mai multe modele cu acuratețe mai mici.”¹¹

Clasificatorii slabii folosiți de AdaBoost sunt arbori de decizie cu o singură divizare. AdaBoost acordă diferite ponderi instanțelor de intrare. Instanțele care sunt mai greu de clasificat vor avea o pondere mai mare, iar cele care sunt deja clasificate corect vor avea o pondere mai mică. Clasificatorii noi sunt antrenați să se adapteze concentrându-se, în mod special, în clasificarea corectă a acelora instanțe care au ponderi mai mari, adică a acelor instanțe care nu au fost clasificate corect până acum.

“Acest lucru înseamnă că instanțele care sunt greu de clasificat corect vor primi o pondere mai mare până când modelul creat va reuși să le clasifice corect.”¹²

Predicția finală se face pe baza majorității voturilor a acestor clasificatori slabii, fiecare având câte o pondere individuală direct proporțională cu acuratețea lor individuală. Varianta de implementare a algoritmului AdaBoost cu cel mai mare succes pentru problema clasificării binare a fost numit AdaBoost.M1.

5.2 Gradient Boosting

AdaBoost și ceilalți algoritmi au fost inspirați dintr-o abordare statistică numită inițial ARCing.

“ARCing este un acronim pentru Reponderarea Adapтивă și Combinatorială. Fiecare iterație constă în a minimiza ponderile urmată de o nouă recalculare atât a clasificatorilor cât și a ponderilor.”¹³

Această tehnică a fost dezvoltată mai departe de către Friedman și a fost numită inițial de acesta “Gradient Boosting Machines”. Astfel a apărut o nouă soluție de optimizare numerică ce minimizează eroarea modelului prin adăugarea clasificatorilor slabii, pe care va încerca apoi să-i îmbunătățească. Generalizarea făcută de Friedman a permis utilizarea diferitelor funcții de calculare a eroarei și extinderea acestei tehnici și la alte probleme pe lângă clasificarea binară: problema regresiei, problema clasificării în mai multe categorii, dar și în multe alte domenii.

Atunci când vorbim de algoritmul de Gradient Boosting trebuie să vorbim de cel puțin 3 elemente:

3. O funcție ce calculează eroarea;
4. Un clasificator slab care să facă predicții;
5. O modalitate de a îmbunătăți modelul prin antrenarea clasificatorului să minimizeze eroarea;

5.2.1 Funcția de eroare

Această funcție depinde foarte mult de ce fel de problemă trebuie să fie rezolvată. În primul rând trebuie să fie derivabilă, pentru a o putea optimiza mai ușor. Există foarte multe astfel de funcții considerate standard, dar există și posibilitatea de a defini unele noi, atunci când problemele întâlnte în practică impun acest lucru. Spre exemplu în problema regresiei cea mai folosită funcție este pătratul erorii, iar în problema clasificării se folosește funcția logaritm.

5.2.2 Clasificatorii slabii

Arborii de decizie sunt folosiți ca și clasificatori slabii de către gradient boosting. Inițial, în cazul algoritmului AdaBoost, erau construiți arbori de decizie având o adâncime foarte mică de cele mai multe ori fiind cu doar o singură divizare. Gradient Boosting folosește, în general de la 4 până la 8 niveluri de adâncime.

În mod uzual, pentru a constrânge arborii de decizie să rămână slabii, se poate specifica un număr maxim de niveluri, un număr maxim de noduri de decizie sau un număr maxim de frunze. Dar aceștia încă sunt construiți în manieră greedy.

5.2.3 Îmbunătățirea modelului

Arborii sunt adăugați câte unul la fiecare iterație. O procedură asemănătoare cu cea a gradientului descendente folosește pentru a minimiza eroarea atunci când se adaugă noul arbore. În mod tradițional, gradientul descendente este folosit pentru a minimiza un set de parametri precum coeficienții unei ecuații de regresie sau ponderile într-o rețea neuronală. După calcularea erorii ponderile sunt actualizate, iar eroarea se minimizează.

Noul arbore creat se adaugă la secvența de arbori deja existenți într-un efort de a corecta sau de a îmbunătăți rezultatul final al modelului.

Algoritmul se oprește fie atunci când a fost atins un anumit număr de arbori, fie atunci când eroarea a atins un punct acceptabil sau atunci când nu se mai pot face îmbunătățiri.

6 Detalii de implementare

6.1 Limbajul de programare

Dintotdeauna a fost o provocare alegerea unui limbaj de programare care să fie capabil să le surclaseze pe celelalte, mai ales, când vine vorba de *Data Science*. Dificultatea este și mai accentuată atunci când o persoană nouă dorește să se specializeze în acest domeniu și nu știe ce limbaj de programare să aprofundeze.

Limbajele de programare precum *R*, *Python*, *Octave*, *Matlab*, *Julia*, etc. oferă câte o serie unică de capabilități ce au drept scop reducerea complexității de implementare a operațiilor de analiză a datelor comparativ cu limbajele de programare tradiționale ca *Java*, *C++*, *C* etc.

Siva Prasad Katru, architect la institutul de tehnologie din India, a creat o clasificare a limbajelor de programare în funcție de diferite metriki, scopul fiind acela de a-l identifica pe cel care s-a impus în domeniul învățării automate.

Pentru a ajunge la o concluzie obiectivă, a creat o scală de la 0 la 5, notând limbajele cu câte o notă pentru fiecare caracteristică din cele alese:

- Viteza de execuție;
- Dificultatea învățării;
- Diversitatea uneltelor oferite pentru analiza datelor;
- Metode de vizualizare;
- Unelte de dezvoltare (IDE etc.);
- Dificultatea integrării unei noi aplicații noi cu una deja existentă;
- Oportunitățile locurilor de muncă de pe piață;

Adunând notele fiecărei caracteristici a obținut o notă după care a făcut clasificarea finală.

Analizând rezultatele obținute din anexa 1, el trage fără reținere următoarea concluzie “*Python* conduce dețasat, dar *R* vine din urmă destul de puternic”.

6.2 Biblioteca scikit-learn

Din multitudinea de utilitare și biblioteci de analiză avansată a datelor (*Spark* -cu biblioteca *MLLib*-, *R*, *Scikit-learn*, *GraphLab* etc.), am optat pentru biblioteca Scikit-learn deoarece este ușor de folosit, este foarte bine documentată și conține implementări a algoritmilor populari din învățarea automată.

6.2.1 Documentația

Principalul motiv pentru care am ales să folosesc biblioteca scikit-learn a fost datorită documentației foarte bine organizată. Persoanele care doresc să contribuie la acest proiect open-source sunt nevoite să scrie și documentația aferentă însotită obligatoriu și de exemple de scripturi care să ruleze pe seturi de date mici, ca exemplu de funcționare. Pe lângă documentație, comunitatea de persoane care s-a format este consecventă, iar contribuțiile noi sunt folositoare și de calitate. De asemenea, ei sunt încurajați să dezvolte, în permanență, noi teste pentru asigurarea calității tuturor funcționalităților disponibile.

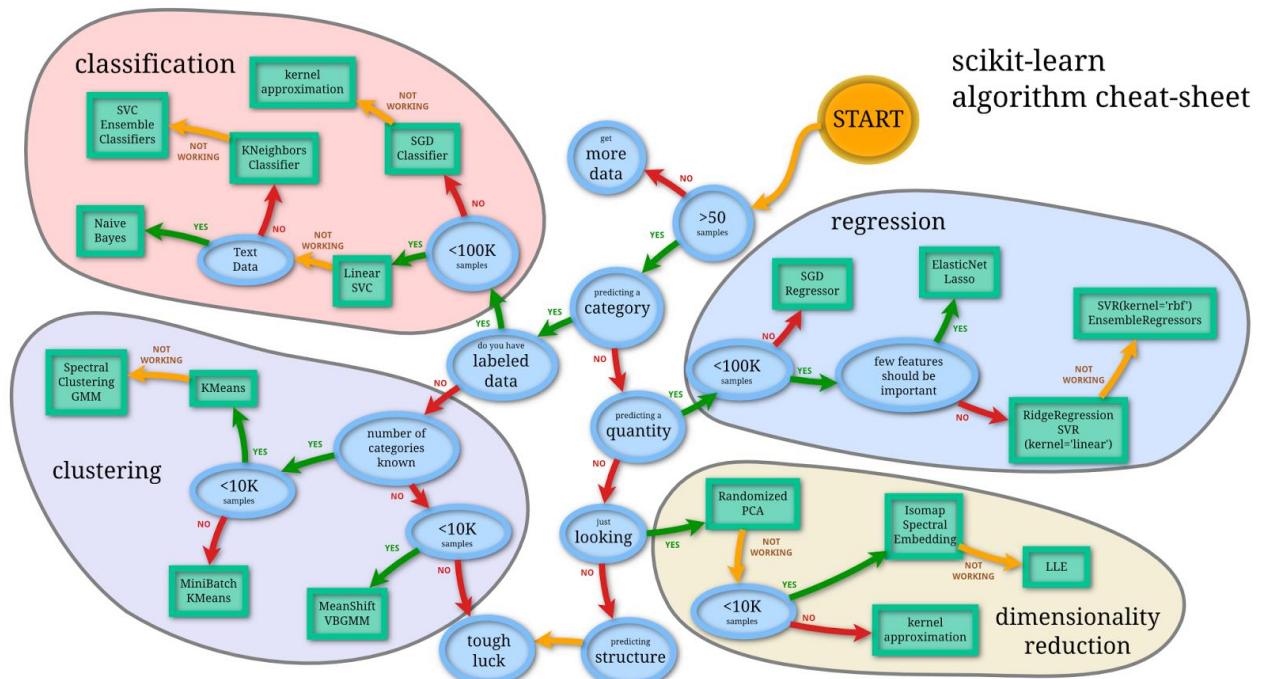
6.2.2 Expertiza contribuitorilor

Comunitatea de persoane care contribuie la scikit-learn include și experți din mai multe domenii, printre care și cel al învățării automate și cel al dezvoltării de software. Un mic grup dintre ei dedică o parte din timpul lor profesional pentru a ajuta la dezvoltarea acestui proiect.

6.2.3 Variația domeniilor acoperite

Lista cu uneltele disponibile în scikit-learn cuprinde majoritatea domeniilor din învățarea automată (precum clasterizare, clasificare, regresie, etc). Iar din moment ce scikit-learn este dezvoltat de o comunitate numeroasă, este foarte posibil ca tehnicele oferite să se înmulțească în urmatoarea perioadă.

Utilizatorii nu sunt nevoiți să aleagă dintre multiple implementări ale aceluiași algoritm (o problemă cu care se confruntă utilizatorii limbajului R). Pentru a-i ajuta pe utilizatori să găsească modelul care se potrivește cel mai bine cu problema lor, Andreas Muller a creat următoarea diagramă:



Imaginea este preluată de pe al doilea diapozitiv de pe prezentarea pe care a susținut-o Andreas Müller la "SciPy 2016 Conference talk"

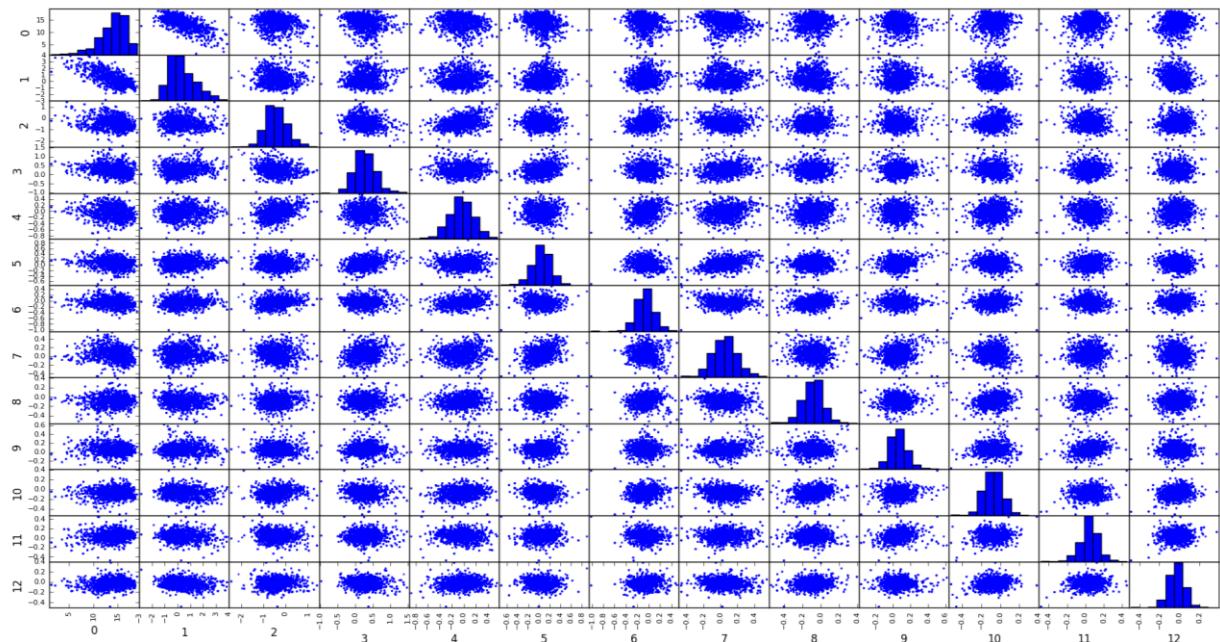
6.2.4 Simplitate

Scikit-learn este o bibliotecă de învățare automată. Scopul ei este de a oferi un set de algoritmi obișnuiti utilizatorilor de python, printr-o interfață consistentă. Acest lucru aduce cu sine alegeri dificile care trebuie făcute pentru a discerne ce noi funcționalități pot fi adăugate. De exemplu, comunitatea a observat că algoritmii de Deep - Learning au cateva dependente speciale care ar fi trebuit incluse. Dar pentru a nu complica lucrurile s-a decis renunțarea la includerea lor și implementarea, în schimb, a unui algoritm de tip Multilayer Perceptron.

7 Metodologia de clasificare a melodiilor

Problema clasificării automate a melodiilor în genuri diferite poate fi descrisă ca sarcina de a atribui pentru fiecare melodie câte un gen muzical. În acest proces de clasificare se crează un model care să aproximeze cât mai bine datele de antrenare.

În abordarea pe care o vom prezenta, crearea modelului cuprinde mai mulți pași. În primul rând, se vor extrage attributele din melodie conform pașilor explicați la capitolul corespunzător. În învățarea automată este importantă studierea datelor de intrare, în special este necesar ca acestea să nu fie corelate. Sunt algoritmi care sunt sensibili la covariația a două sau a mai multor attribute de aceea trebuie să ne asigurăm că attributele extrase sunt reprezentative pentru piesele muzicale. Pentru acest lucru am generat mai multe grafice pe care le-am unit în imaginea de mai jos.



În această imagine putem observa pe diagonală histogramele pentru fiecare din cele 13 attribute. Observăm cum aceste attribute sunt normal distribuite și au valori variate. De asemenea, în restul pozițiilor din imagine avem o serie de puncte ce reprezintă attributele de intrare. Spre exemplu pe prima coloană avem o serie de grafice în care fiecare punct are ca valoare pe abscisă primul atribut, iar pe ordonată, pe rând, fiecare din celelalte attribute.

Analizând aceste grafice putem concluziona că nu există două attribute care să fie dependente. Dacă ar fi existat cel puțin două attribute care să fie covariate atunci graficele ar fi

avut forme alungite, eventual în direcția fie a diagonalei principale, fie în direcția diagonalei secundare.

După acest pas vom avea setul de date pregătit pentru procesările ulterioare. Acesta constă în 1000 de instanțe de intrare, fiecare având câte 13 atribute, reprezentând media energiilor ce apar pe durata melodiiilor grupate în 13 grupe, în funcție de frecvența lor. Prima categorie, cea de antrenare va fi folosită pentru antrenarea simultană și independentă a 6 clasificatori:

1. Regresie Logistică,
2. Rețele Neuronale,
3. Mașini pe Vector Suport (SVM),
4. Gradient Boosting,
5. Random Forest,
6. KNN.

7.1 Modele individuale

Prima abordare a fost aceea de a găsi un singur clasificator, cel mai bun, care să obțină cea mai mare acuratețe. S-au încercat, pe rând, mai mulți clasificatori, fiecare cu câte o acuratețe proprie. Unii clasificatori s-au pliat pe această problemă, având o acuratețe destul de bună, alții nu s-au potrivit la fel de bine.

În anexa 2 am generat pentru fiecare clasificator, câte 10 grafice ROC. Fiecare grafic reprezintă acuratețea a câte un model individual în a clasifica un anumit gen în comparație cu celelalte. Aceste grafice sunt grupate pe genuri pentru a se face mai ușor comparația dintre aceste modele.

7.1.1 Regresie Logistică

Primul clasificator încercat a fost Regresia Logistică. Această tehnică de clasificare este folosită, în mod special, în cazul problemelor de clasificare binară. În cazul de față, problema cere o clasificare în 10 categorii diferite. De aceea s-a aplicat o tehnică numită în învățarea automată One - Against -All (Unul – împotriva - Tuturor).

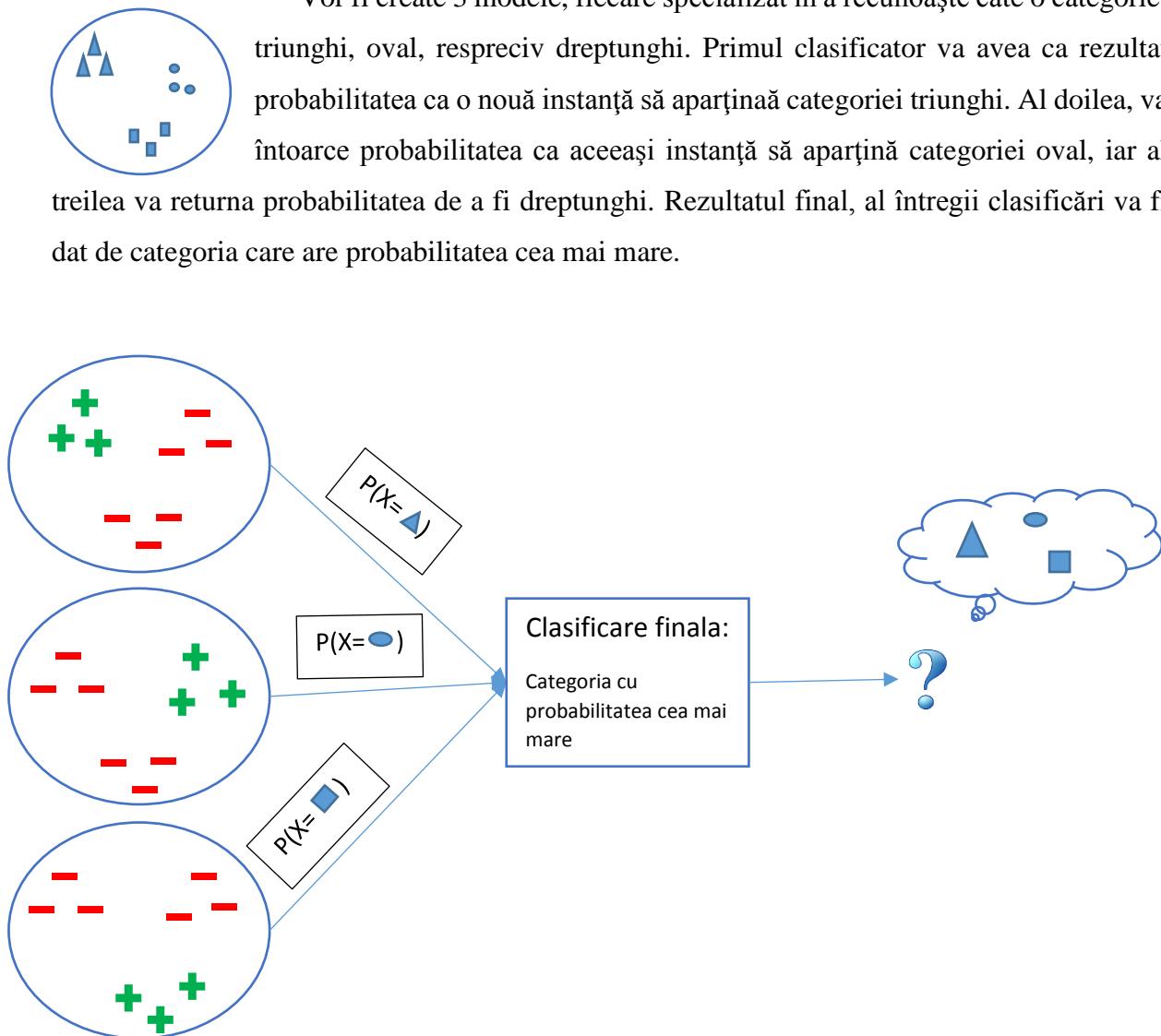
7.1.1.1 One-Against-All (OAA)

Această strategie constă în a antrena câte un clasificator pentru fiecare categorie, specializându-se să facă distincția între instanțele care aparțin unei singure categorii și cele care nu aparțin acestei categorii. Fiecare clasificator va returna probabilitatea ca noua instanță să

aparțină clasei pe care s-a specializat el. Iar rezultatul final, va fi genul muzical cu probabilitatea cea mai mare ca noua instanță să-i aparțină, conform tuturor modelelor.

Spre exemplu dacă am presupune că am vrea să facem o clasificare în 3 categorii de obiecte și că avem urmatoarele instanțe: triunghiuri, ovale și dreptunghiuri.

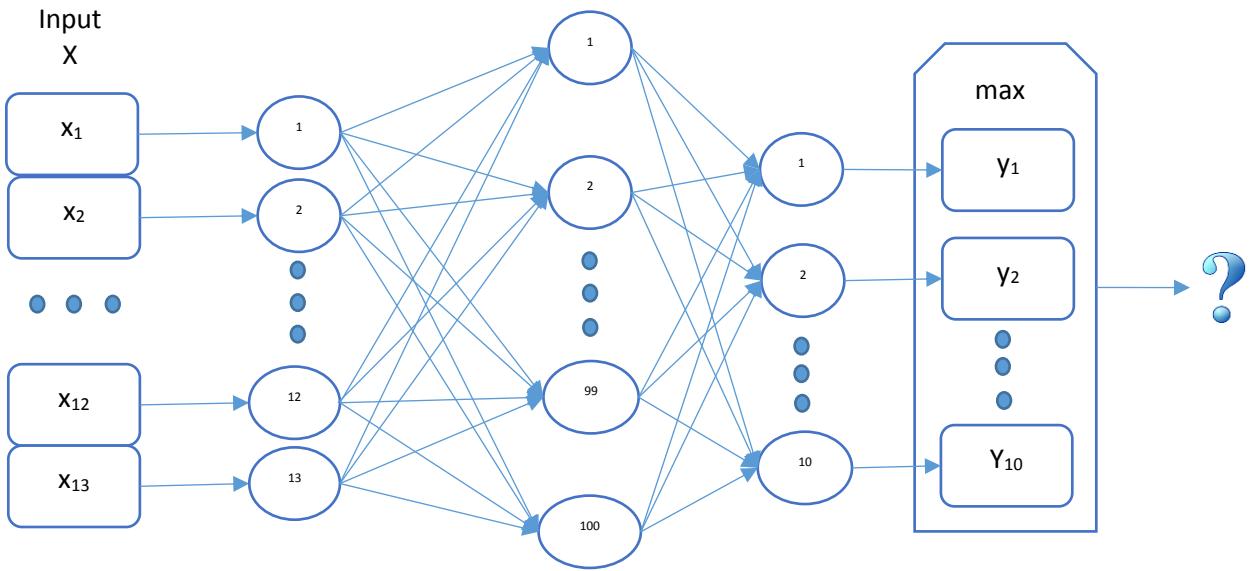
Vor fi create 3 modele, fiecare specializat în a recunoaște câte o categorie: triunghi, oval, respectiv dreptunghi. Primul clasificator va avea ca rezultat probabilitatea ca o nouă instanță să aparțină categoriei triunghi. Al doilea, va întoarce probabilitatea ca aceeași instanță să aparțină categoriei oval, iar al treilea va returna probabilitatea de a fi dreptunghi. Rezultatul final, al întregii clasificări va fi dat de categoria care are probabilitatea cea mai mare.



Deoarece, în problema clasificării melodiilor avem 10 genuri muzicale, vom avea 10 modele create cu ajutorul regresiei logistice. Dar rezultatul final va fi, un singur gen muzical. Această abordare nu modelează datele nici foarte bine dar nici extraordinar de rău, având o acuratețe de 46%. Spunem că această acuratețe nu este foarte rea în condițiile în care, probabilitatea de a alege în mod aleator genul unei noi instanțe și ca aceasta să fie varianta corectă este doar de 10%. Deci regresia logistică aplicată pe problema clasificării melodiilor în genuri muzicale, este de 4.6 ori mai bună decât un clasificator aleator.

7.1.2 Rețele Neuronale Artificiale

Al doilea clasificator încercat este un model creat cu ajutorul rețelelor neuronale. Primul strat din rețea creată are 13 neuroni, deoarece datele cu care lucrăm au câte 13 atribute. Reteaua va avea un singur strat ascuns, ce va fi format din 100 de neuroni. Iar ultimul strat este format din 10 neuroni, fiecare având câte o probabilitate asociată pentru fiecare gen muzical.



Ultimul strat are ca funcție de activare funcția softmax. Aceasta este o generalizare a funcției logistice și se folosește de obicei în cazul problemelor de clasificare ce au mai mult de două categorii. Această funcție va avea ca valoare de intrare un vector z 10-dimensional și ca ieșire va returna tot un vector cu 10 valori y cu valori între 0 și 1. Vectorul de ieșire este construit în așa fel încât suma valorilor lui să fie 1, adică valorile din vectorul y reprezintă, de fapt, probabilitățile de aparteneță a unei instanțe la o anumita categorie.

Funcția softmax se poate defini în felul următor:

$$y_c = \frac{e^{z_c}}{\sum_d^{10} e^{z_c}} \text{ pentru fiecare } c = 1 \dots 10$$

Numitorul $\sum_d^{10} e^{z_c}$ este termenul ce se asigură că suma tuturor valorilor din vectorul de ieșire sunt 1 ($\sum_{c=1}^1 y_c = 1$). Din punct de vedere grafic funcția softmax poate fi reprezentată ca un strat cu 10 neuroni. Putem scrie probabilitățile categoriilor $t=c$ pentru $c = 1 \dots 10$ dându-se valoarea de input z în felul următor:

$$\begin{bmatrix} P(t = 1|z) \\ \dots \\ P(t = 10|z) \end{bmatrix} = \frac{1}{\sum_d^{10} e^{z_c}} \begin{bmatrix} e^{z_1} \\ \dots \\ e^{z_{10}} \end{bmatrix}$$

Unde $P(t = c|z)$ este probabilitatea ca dându-se valoarea de intrare z ea să reprezinte o instanță din clasa c .

Rezultatul final al rețelei neuronale va fi categoria care are probabilitatea cea mai mare din cele 10.

7.1.3 Gradient Boosting

Acest clasificator este unul de tip asamblist ce crează mai mulți clasificatori slabii, iar apoi încearcă prin diverse tehnici să-i asambleze și să creeze alții noi pentru a rezulta un model cât mai veridic. Ca și clasificator este ales modelul creat de arborii de decizie. Cel mai bun rezultat a fost obținut folosind 100 de astfel de clasificatori. Chiar dacă inițial nu s-a luat în calcul, a trebuit să revenim asupra acestor clasificatori pentru a-i forța să rămână slabii. Pentru acest lucru am pus două condiții suplimentare și anume: fiecare arbore de decizie poate avea maxim 8 frunze și poate fi divizat de maxim 5 ori. Acuratețea acestui clasificator este una din cele mai bune, alături de cea obținută de Random Forest, și anume de 76.3%.

7.1.4 Random Forest

Următorul clasificator folosit, pe care îl vom prezenta este Random Forest. Aceasta s-a potrivit cel mai bine pe această problemă și a oferit cele mai bune rezultate individuale. El a asamblat 100 de arbori de decizie. Pentru acest algoritm, arborii de decizie utilizati nu au fost constrânsi cu nimic, spre deosebire de cei folosiți de Gradient Boosting. Acuratețea acestuia a fost de 77.975% .

7.1.5 KNN

KNN este unul din puținii algoritmi care pot să clasifice, prin natura algoritmului toate cele 10 genuri simultan. Ceilalți algoritmi recurg la diverse metode de descompunere a problemei în subprobleme și apoi la asamblarea lor. Prin încercări repetate, s-a ajuns la concluzia că cele mai bune rezultate pentru acest clasificator sunt obținute atunci când luăm în considerare cei mai apropiati 5 vecini. Acuratețea este comparabilă cu cea a regresiei logistice sau a rețelelor neuronale și anume de 53.225%.

7.2 Asamblarea modelelor individuale

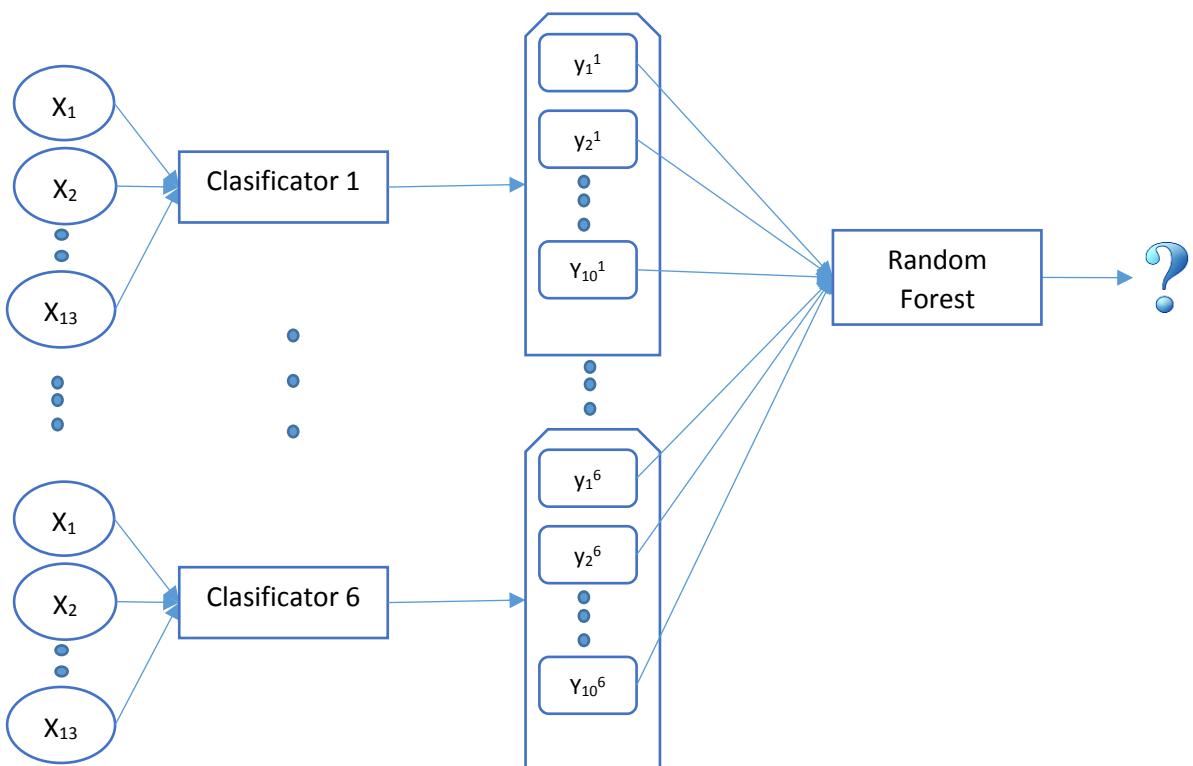
Setul de date folosit constă în 1000 de instanțe de intrare, fiecare având câte 13 atribute, reprezentând suma energiilor ce apar pe durata melodiilor grupate în 13 grupe, în funcție de frecvența lor. După ce avem setul de date pregătit, acesta este împărțit, pentru început, în 2 subcategorii: date de antrenare (A1) și date de testare (T1), în proporție de 60% respectiv 40%. Prima categorie (A1), cea de antrenare va fi folosită pentru antrenarea simultană a cei 6 clasificatori.

Fiecare din acești clasificatori vor fi antrenați pe 600 de instanțe, fiecare având câte 13 atribute. După faza de antrenare, fiecare clasificator urmează să fie testat. Testarea se face folosind (T1).

În această fază de testare se generează câte o matrice de confuzie pentru fiecare clasificator. Aceste matrici pot fi găsite în Anexa 3. Pentru o matrice de confuzie abscisa va reprezenta rata clasificărilor false făcute, adică câte melodii au fost încadrate într-un anumit gen, cu toate că ele aparțineau altuia. Iar pe ordonată, matricea va avea reprezentată rata clasificărilor corecte.

După cum se poate observa, din subcapitolul anterior fiecare clasificator individual, produce câte 10 probabilități, câte una pentru fiecare gen. Din aceste 10 probabilități clasificatorul o va lua pe cea mai mare și prediciția sa va fi categoria care corespunde acestei probabilități.

Pe lângă această fază de testare, pentru fiecare din aceleasi 240 de instanțe se memorează probabilitățile fiecărei instanțe de a aparține la cele 10 genuri. Rezultatul întors de fiecare din acești clasificatori este reprezentat de un vector de 10 valori reale, ce se vor concatena formând într-un final un vector de 60 de valori. Aceste 60 de valori vor fi valorile de intrare pentru un alt clasificator de tip Random Forest. Acest clasificator se consideră a fi un clasificator expert pentru că el va atribui diferite ponderi clasificărilor individuale. Pentru a reuși să-l atrenăm și să-l testăm corect din punct de vedere statistic vom împărți datele de testare (T1) încă o dată în două, în aceeași proporție: 60% (TA2) și 40% (TT2). Doar că aceste date nu vor conține cele 13 atribute inițiale, ci vor conține 60 de atribute: câte 10 probabilități de la fiecare clasificator. Cu ajutorul setului de date TA2, care conține 240 de melodii, se antrenează clasificatorul expert, iar cu setul de date TT2 se va testa clasificatorul expert. Acuratețea acestui clasificator s-a dovedit a fi cu mult mai bună decât a celorlalți clasificatori. Aceasta este de 88%.



8. Rezultate obținute

În ciuda naturii foarte complexe a problemei, melodiile pot fi clasificate în mod automat. Iar rezultatele acestui proces automat sunt cu mult mai bune decât clasificarea făcută manual de oameni. Trăsăturile folosite sunt cele propuse de G. Tzanetakis și P. Coo și poartă numele de MFCC. Antrenând câte un model pentru fiecare din cei 6 clasificatori (Regresie Logistică, Rețele Neuronale, Mașini pe Vector Suport (SVM), Gradient Boosting, Random Forest, KNN) și analizând rezultatele fiecărui s-a ajuns la o acuratețe individuală de 77.975%. Acest rezultat aparține modelului creat cu algoritmul Random Forest.

Apoi, în încearcarea de a obține o acuratețe mai bună, s-a mai folosit un clasificator care are rolul de a acorda diferite ponderi celorlalți algoritmi amintiți deja. Această tehnică este cunoscută ca folosirea unui clasificator a experților. S-a dovedit că această abordare a dus la obținerea unei acurați de 88.275%.

Pentru a vizualiza modul în care se face clasificarea melodiilor am încercat combinarea a catorva tehnici din învățarea automată. În primul rând, pentru a reprezenta datele de intrare este nevoie de un spațiu în 13 dimensiuni. Cum o astfel de reprezentare nu se poate pune în practică am aplicat, pe datele de intrare, un algoritm de reducere a dimensiunilor și anume PCA (Principal Component Analysis). Astfel s-a ajuns de la 13 dimensiuni, la doar două. Dar pentru a putea vedea concret fiecare instanță cum a fost clasificată trebuie să se folosească toate cele 13 atrbuite, altfel clasificarea nu este validă. Acestea au fost clasificate folosindu-se toate cele 13 atrbute, iar apoi coordonatelor bidimensionale le-a fost atribuit genul cu care a fost clasificată o anumită melodie. Aceste imagini pot fi vizualizate în Anexa 4.

Concluzii și direcții de continuare a acestui studiu

În această lucrare am implementat un sistem software capabil să identifice genul muzical al pieselor muzicale. Acest sistem poate fi îmbunătățit prin adăugarea de noi atribută cu ajutorul cărora să se facă diferență dintre genuri mai bine. Cel mai bun model individual a fost Random Forest, de aceea el a fost ales mai apoi ca și clasificator expert.

De asemenea, pentru obținerea unor rezultate care să reflecte cât mai bine realitatea, ar trebui să se facă un studiu comparativ între mai multe seturi de date deoarece în practică, o piesă muzicală, un fișier audio sau un conținut multimedia audio poate avea o multitudine de forme, cu multe variante și pentru a vedea cât de bun este un model în practică, ar trebui studiate mai multe seturi de date.

Iar după aceste studii, cred că ar fi foarte benefică oferirea acestei capabilități ca și un serviciu web, accesibil de oricine are nevoie. O astfel de clasificare în timp real ar putea fi folosită atât pentru recunoașterea genului muzical al unei melodii cât și pentru recomandarea de melodii pentru utilizatori. Așa cum au concluzionat A.C.North și David J Hargreaves în studiul lor¹ oamenii asociază mai repede melodiile care aparțin aceluiași gen muzical, deci poate fi o îmbunătățire a modelelor actuale de recomandare a melodiilor.

Anexa 1

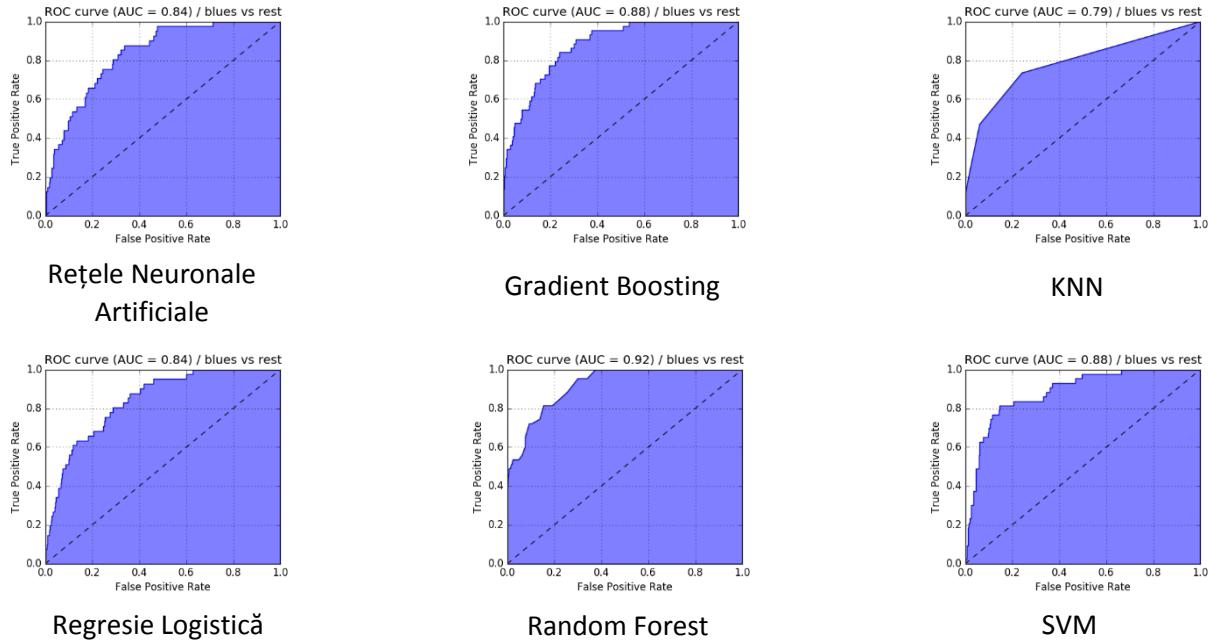
Limbajul de programare	Avantaje	Dezavantaje	Biblioteci importante
<i>R</i>	1. Open source 2. Bun pentru analiză statistică și procesarea datelor 3. O colecție impresionante de algoritmi	1. Destul de greu de învățat 2. Comenzi neobișnuite	1. Gbm 2. RTextTools 3. Dplyr, zoo, 4. Ggplot2, 5. Carot
<i>Python</i>	1. Open source 2. Ușor de învățat 3. Păstrează avantajele unui limbaj de programare general 4. Poate fi folosit pentru BigData	1. Viteza de execuție 2. Trebuie să se țină cont dacă bibliotecile sunt portate de la versiunea 2.x la 3.x	1. Scikit-learn 2. Pandas, 3. Matplotlib 4. Numpy 5. Scipy 6. Theano, 7. Nltk
<i>MATLAB</i>	Potrivit pentru: 1. proceze complexe matematice precum operații cu matrici 2. învățare automată 3. procesarea semnalului audio 4. procesare de imagini	1. Lipsește un ecosistem open source 2. Există dificultăți atunci când datele nu pot fi reprezentate sub forma de matrice	1. Statistică și învățare automată 2. Procesare de imagini 3. Optimizare
<i>OCTAVE</i>	1. Open source 2. Potrivit pentru operații numerice 3. Este compatibil cu MATLAB 3. Bun pentru a construi modele preliminare	1. Inteoperabilitatea cu date externe (baze de date, fișiere csv etc) este destul de slabă	1. Libsvm, 2. Shogun, 3. Liblinear, 4. Ltfat, 5. Vlfeat
<i>Julia</i>	1. Open source 2. Proiectat pentru calcule numerice și științifice 3. Performanță foarte bună 4. Poate apela funcții Python și C	1. Este un limbaj nou de programare 2. Nu există foarte multe biblioteci externe	MLBase MLUtils MLKernels, Clustering MachineLEarning

Acstea tabele sunt o traducere a unei analize făcute de Siva Prasad Katru. Preluată de pe <https://www.linkedin.com/pulse/r-vs-python-matlab-octave-julia-who-winner-siva-prasad-katru>

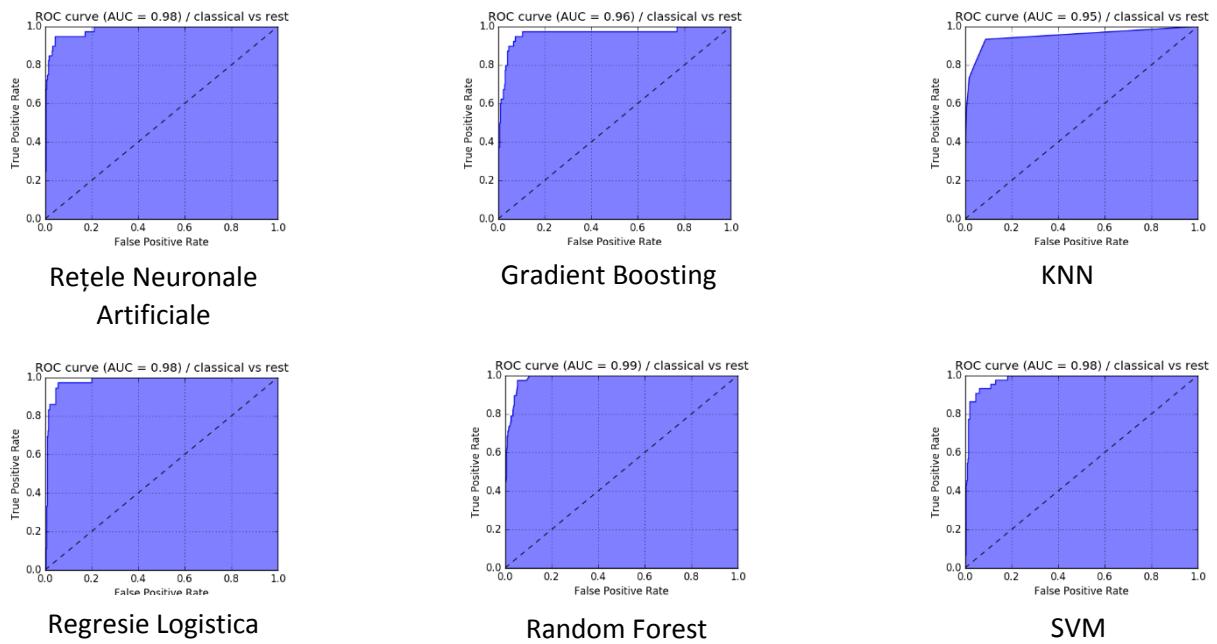
<i>Limbajul de programare</i>	Viteza de executie	Dificultatea invatarii	Capabilitati de analiza a datelor	Capabilitati grafice	Utilitati (IDE, plugins)	Suportul comunitatii	Integrarea cu aplicatiile existente	Joburi disponibile	Scor total
<i>R</i>	3	1	5	4	4	4	3	4	28
<i>Python</i>	4	4	3	3	3	5	5	5	32
<i>MATLAB</i>	2	3	4	4	5	3	2	2	25
<i>OCTAVE</i>	2	2	3	2	2	3	2	1	17
<i>Julia</i>	5	3	4	2	2	2	3	1	22

Anexa 2

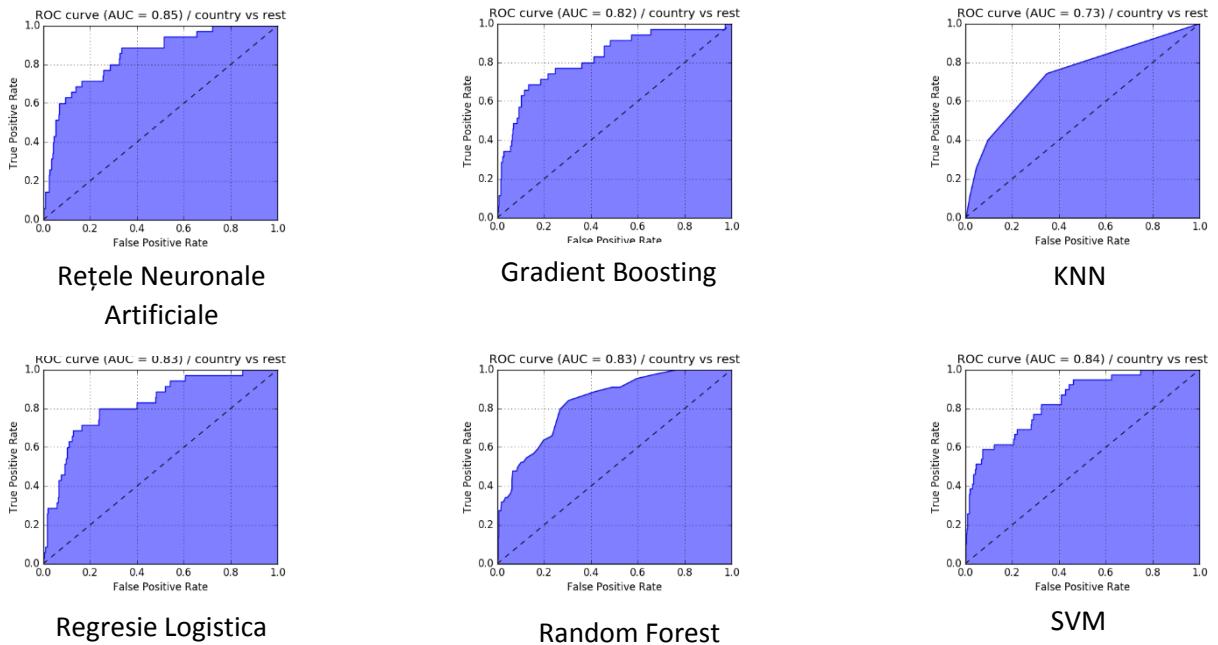
Blues VS. Rest



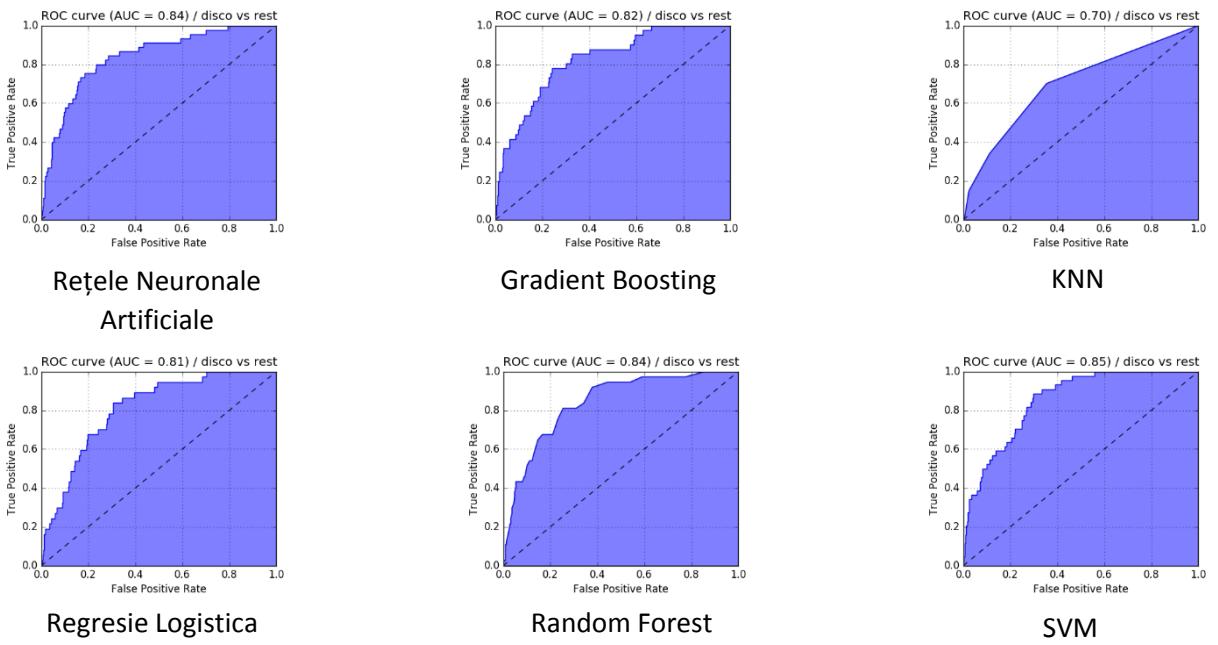
Clasic VS. Rest



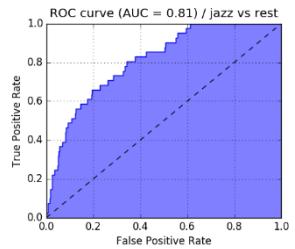
Country VS. Rest



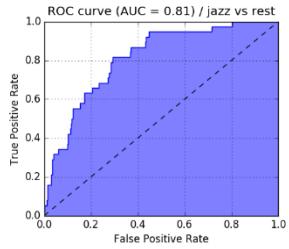
Disco VS. Rest



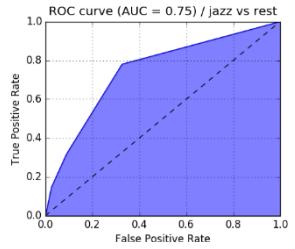
Jazz VS. Rest



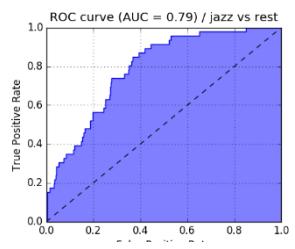
**Rețele Neuronale
Artificiale**



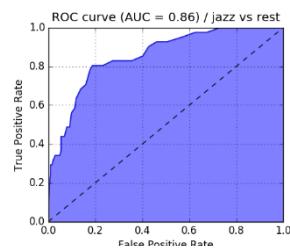
Gradient Boosting



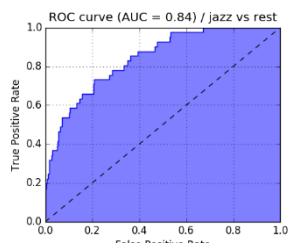
KNN



Regresie Logistica

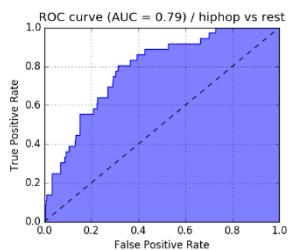


Random Forest

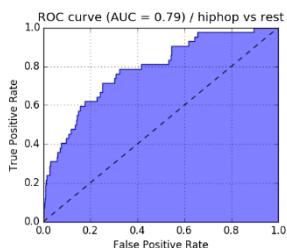


SVM

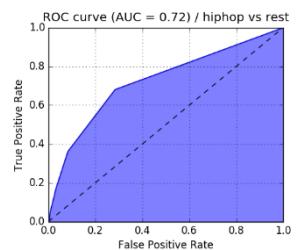
Hiphop VS. Rest



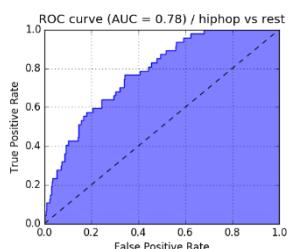
**Rețele Neuronale
Artificiale**



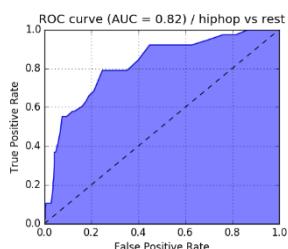
Gradient Boosting



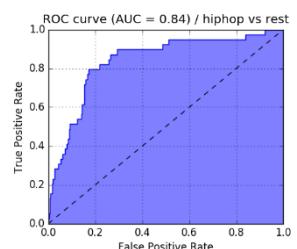
KNN



Regresie Logistica

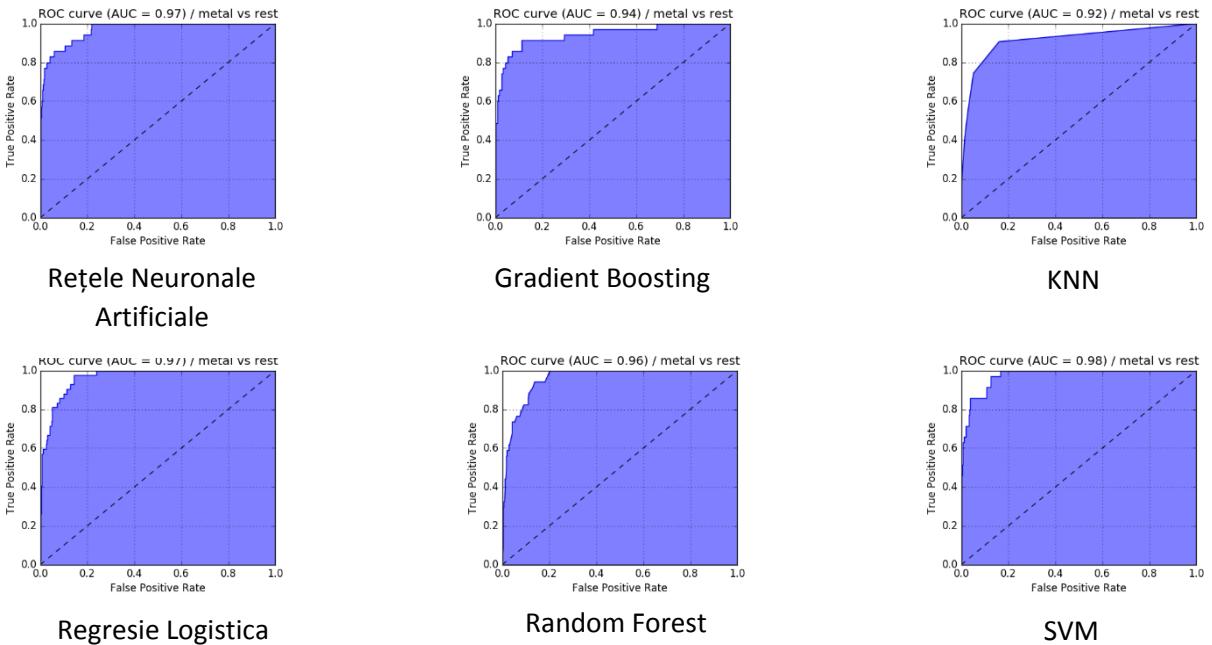


Random Forest

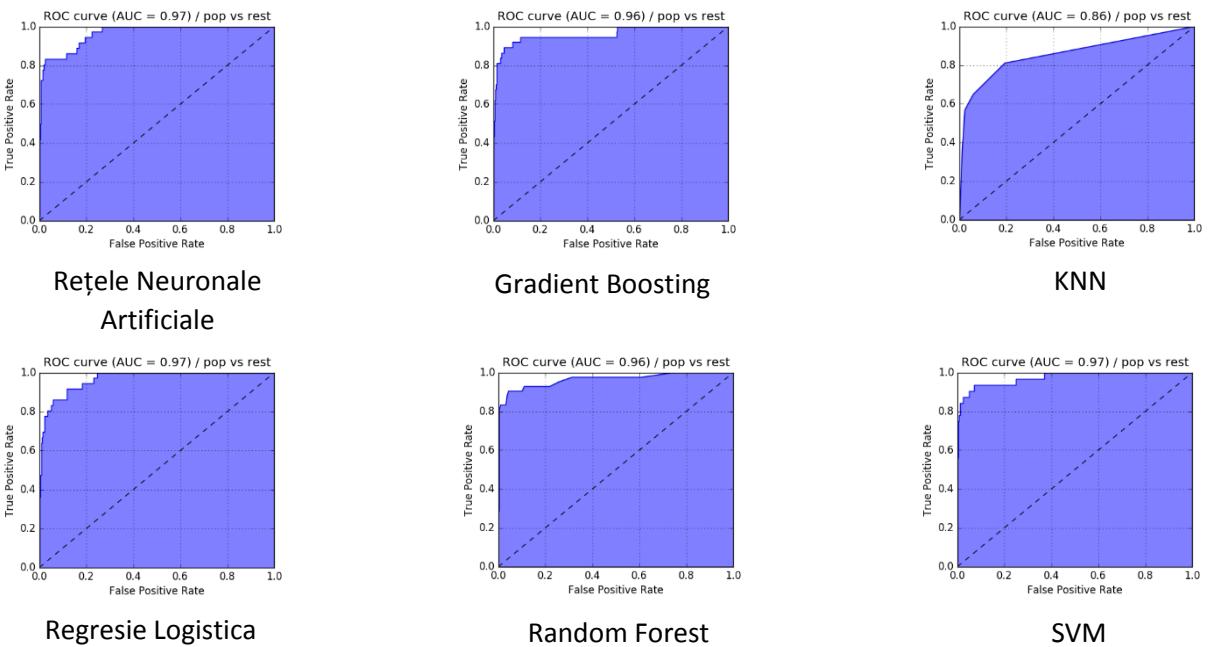


SVM

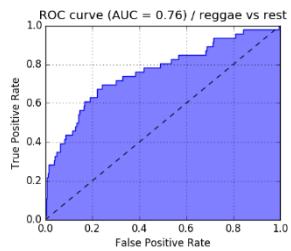
Metal VS. Rest



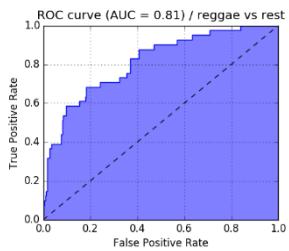
Pop VS. Rest



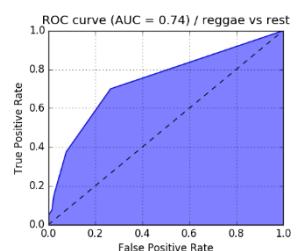
Ragga VS. Rest



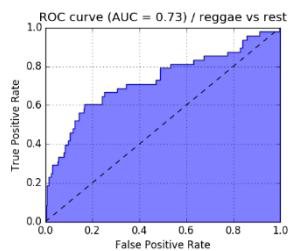
Rețele Neuronale
Artificiale



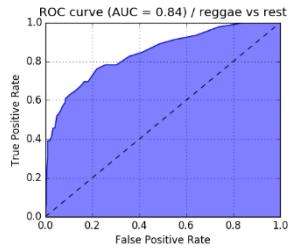
Gradient Boosting



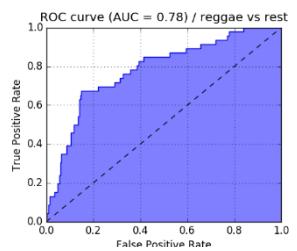
KNN



Regresie Logistica

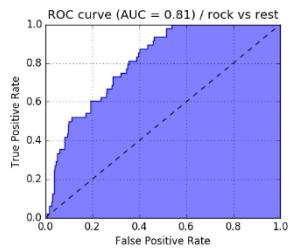


Random Forest

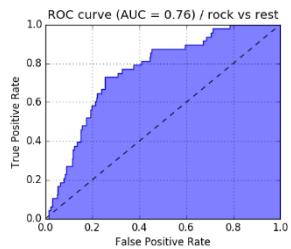


SVM

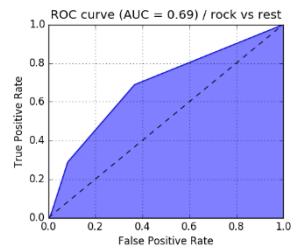
Rock VS. Rest



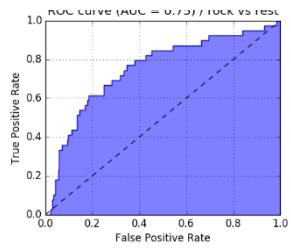
Rețele Neuronale
Artificiale



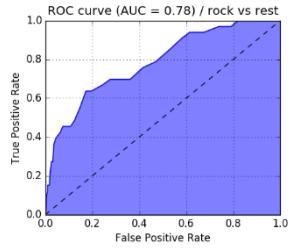
Gradient Boosting



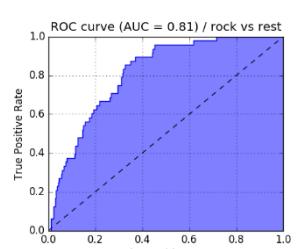
KNN



Regresie Logistica

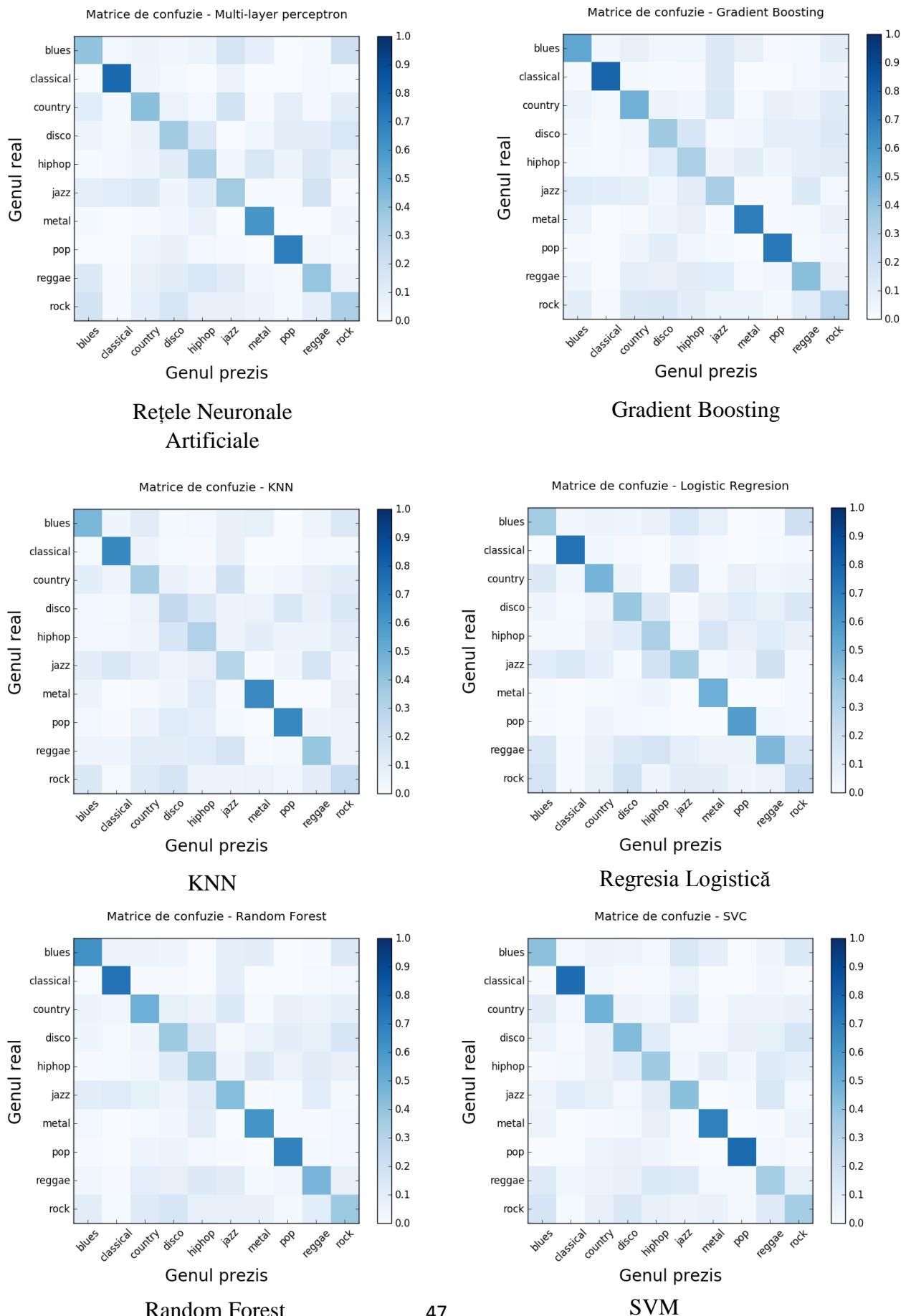


Random Forest

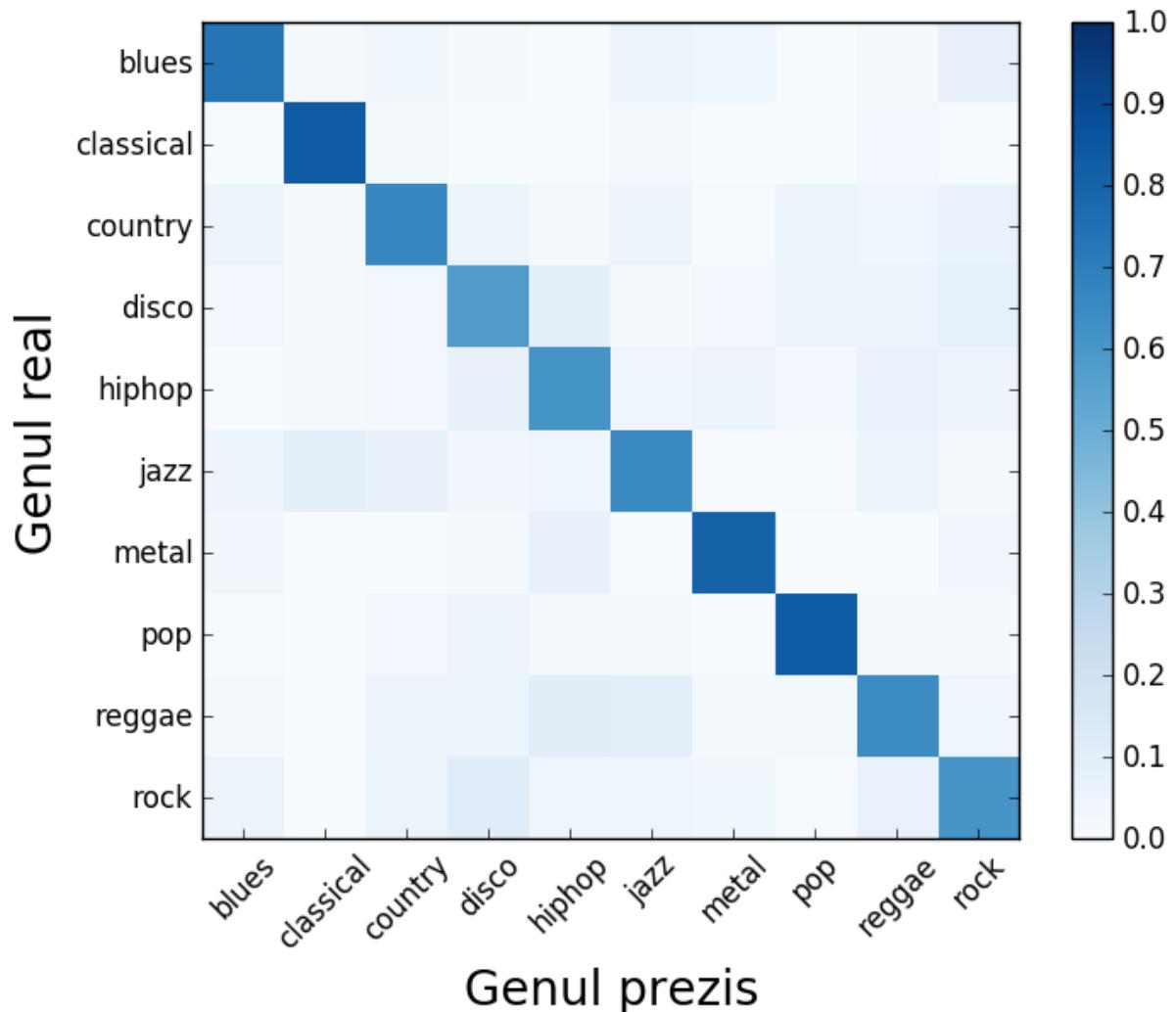


SVM

Anexa 3

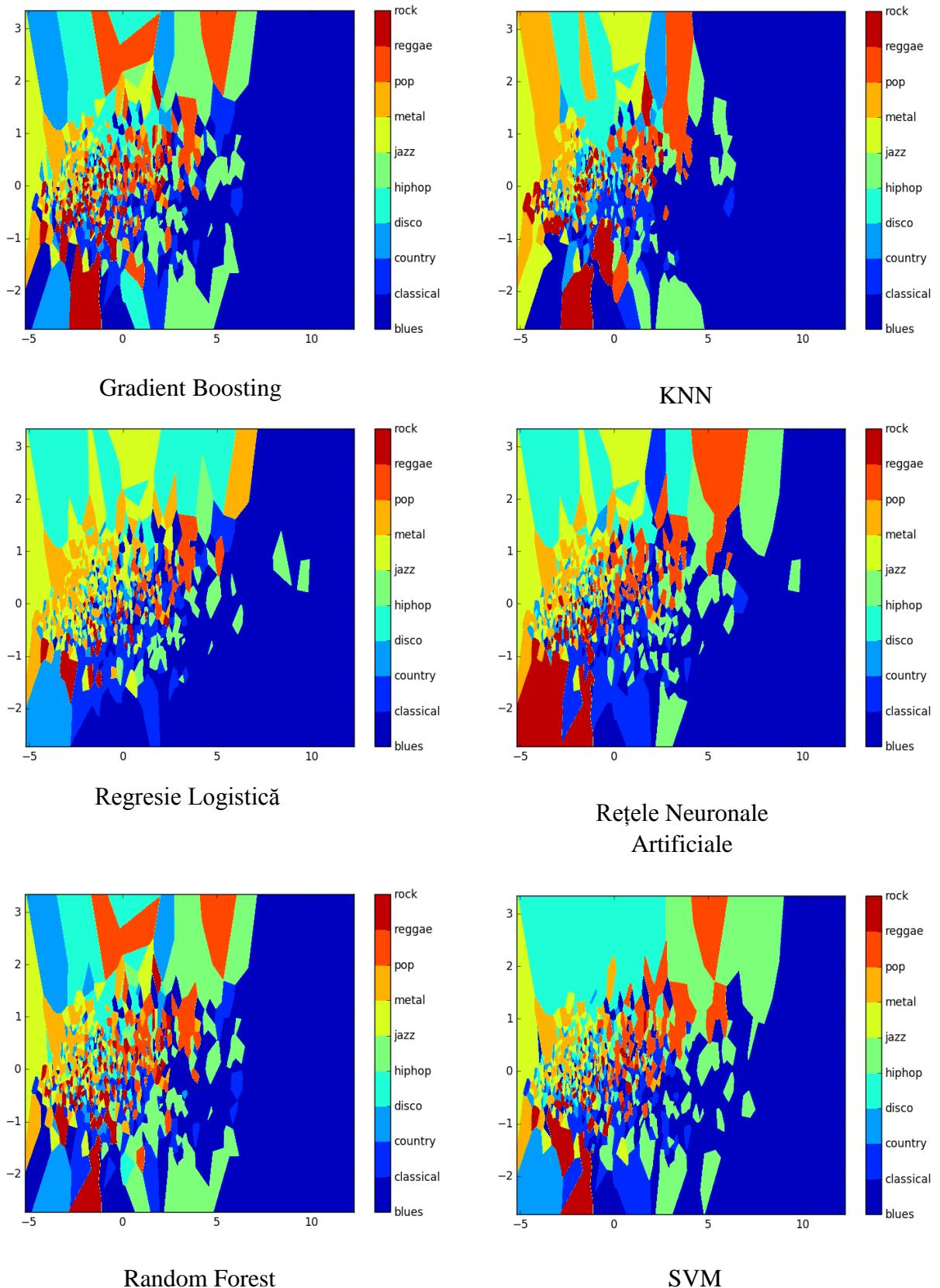


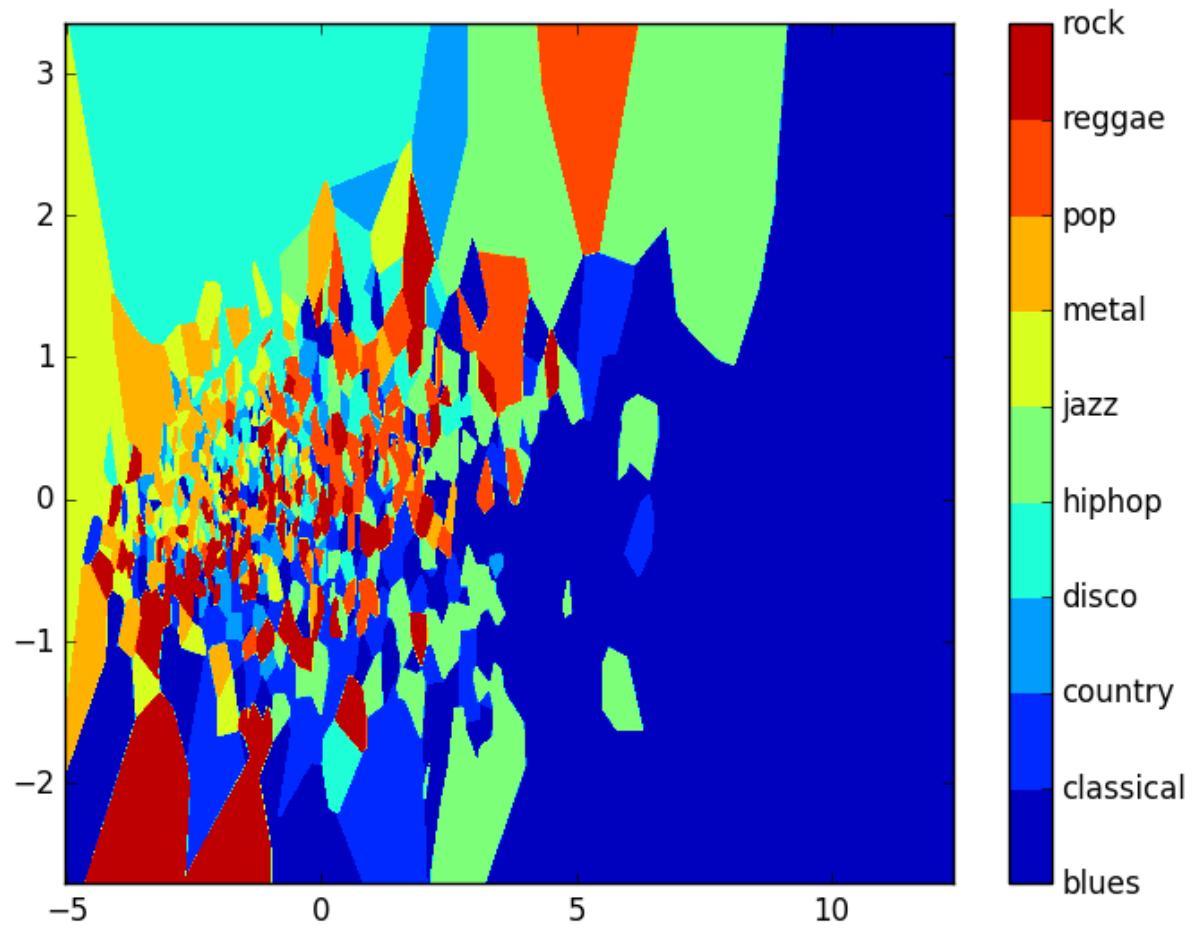
CEPS classifier - Confusion matrix



Clasificatorul expert

Anexa 4





Bibliografie și Webografie

¹ North, A. C., and D. J. Hargreaves. 1997. "Liking for musical styles". *Music Scientae* 1: 109–28.

[https://www.researchgate.net/publication/239066308_Liking_for_musical_styles]

² S. B. Davis and P. Mermelstein: "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences", *IEEE Transactions on Acoustic, Speech, and Signal Processing*, Vol. 28, No. 4, August 1980, pp. 357–366.

³ Tzanetakis, G., and P. Cook. 2002. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing* 10 (5): 293–302.
[https://www.researchgate.net/publication/220656193_Musical_Genre_Classification_of_Audio_Signals]

⁴ K. Kosina. Music Genre Recognition. MSc. Dissertation, Fachschule Hagenberg, June 2002.
[<http://kyrah.net/mugrat/mugrat.pdf>]

⁵ T. Li; M. Ogiara; Q. Li. A Comparative study on content-based Music Genre Classification. Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Toronto, ACM Press, pages 282-289, 2003.

⁶ M. Grimaldi; P. Cunningham; A. Kokaram. A wavelet packet representation of audio signals for music genre classification using different ensemble and feature selection techniques. *Proceedings of the 5th ACM SIGMM International Workshop on Multimedia Information Retrieval*, ACM Press, pages 102-108, 2003.

⁷ M. Grimaldi; P. Cunningham; A. Kokaram. An evaluation of alternative feature selection strategies and ensemble techniques for classifying music. *Workshop on Multimedia Discovery and Mining*, 14th European Conference on Machine Learning, 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Dubrovnik, Croatia, 2003

⁸ S.B. Davis, and P. Mermelstein (1980), "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4), pp. 357–366
[<https://books.google.at/books?id=yjzCra5eW3AC&pg=PA65#v=onepage&q&f=false>]

⁹ <https://cloud.google.com/products/machine-learning/>

¹⁰ Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World, page 152, 2013

¹¹ A decision-theoretic generalization of on-line learning and an application to boosting, 1995

¹² Applied Predictive Modeling, 2013

¹³ Prediction Games and Arching Algorithms [PDF], 1997

¹⁴https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm

¹⁵<http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>

Liviu Ciortuz, Alina Munteanu, Elena Bădărău – “Exerciții de învățare automată”, Editura Universității “Alexandru Ioan Cuza” Iași

https://www.researchgate.net/figure/220942923_fig1_Fig-1-An-example-illustrating-the-problem-space-decomposition-strategy

https://www.researchgate.net/publication/3333877_Musical_genre_classification_of_audio_signals_IEEE_Trans_Speech_Audio_Process

<http://machinelearningmastery.com/logistic-regression-for-machine-learning/>

https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm

<https://hal.archives-ouvertes.fr/inria-00103955/document>

<http://amueller.github.io/>

<http://machinelearningmastery.com/logistic-regression-for-machine-learning/>

https://en.wikipedia.org/wiki/Music_genre

<https://www.linkedin.com/pulse/r-vs-python-matlab-octave-julia-who-winner-siva-prasad-katru>

<https://www.oreilly.com/ideas/six-reasons-why-i-recommend-scikit-learn>