# Prezentarea Problemei

# Alegeri tehnice ?????

## 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*, chief architect la institutul de tehnologie din India, a creat o clasificare a limbajelor de programare în funcție de diferite metrici, 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 integrarii unei noi aplicatii cu una deja existenta;
* oportunitatile locurilor de munca pe piata;

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

Analizând rezultatele obținute din anexa 1, se poate spune fără reținere că *Python* conduce detașat, dar *R* vine din urmă destul de puternic”.

## 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, foarte bine documentată și conține implementări a algoritmilor populari din învățare automată.

### Documentație organizată

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 însoțită obligatoriu și de exemple de script-uri 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.

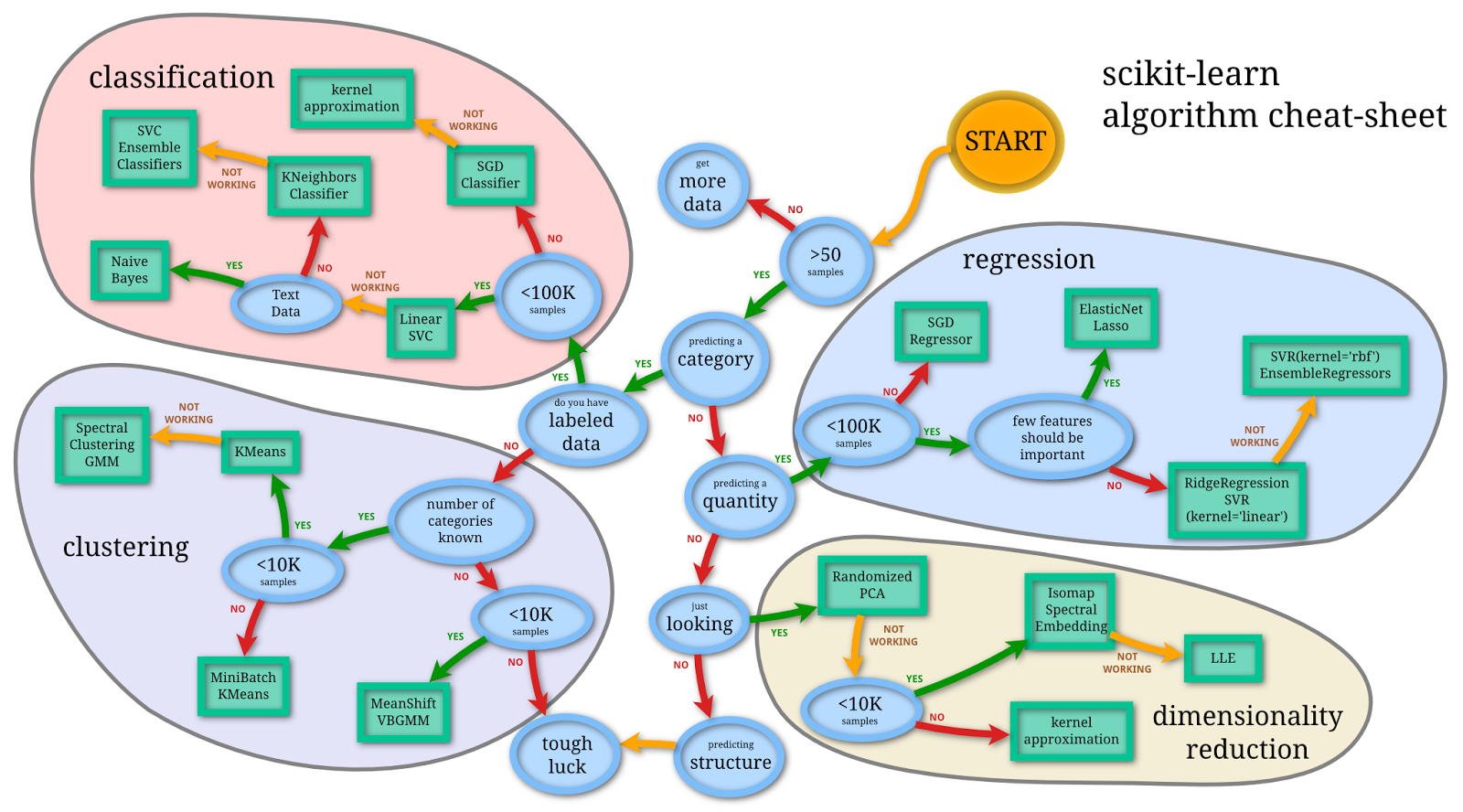
### Contributorii sunt experți

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 profesionl pentru a ajuta la dezvoltarea acestui proiect.

### Acopera majoritatea ariilor din domeniul invatarii automate

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 tehnicile 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 Muller la “SciPy 2016 Conference talk”

### Este simplu de folosit

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 aunui algortim de tip Multilayer Perceptron.

# Random Forest

## 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.

Barbat

Barbat

Femeie

NU

DA

NU

DA

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 dicizie 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” (**I**nformation **G**ain), ce măsoară cât de bine separa datele de antrenare atributul ales.

Pentru a putea vorbi de IG (**I**nformation **G**ain) 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 impurități sa fie cat mai mică decât impuritatea colecției inițiale. Altfel, acea divizare nu a adus nici un caștig de informație.

Dându-se o colecție de instanțe S, cu exemple atât pozitive cât și negative, entropia colecției S relativă la clasificarea booleană este: , unde este proporția exemplelor pozitive din S, iar 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 mica. 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:

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

## 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 rezultatat 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 inițial (ex. se vor crea 100 subseturi). 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”.

## Random Forest

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

Problema arborilor de decizie este că sunt construiți 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.

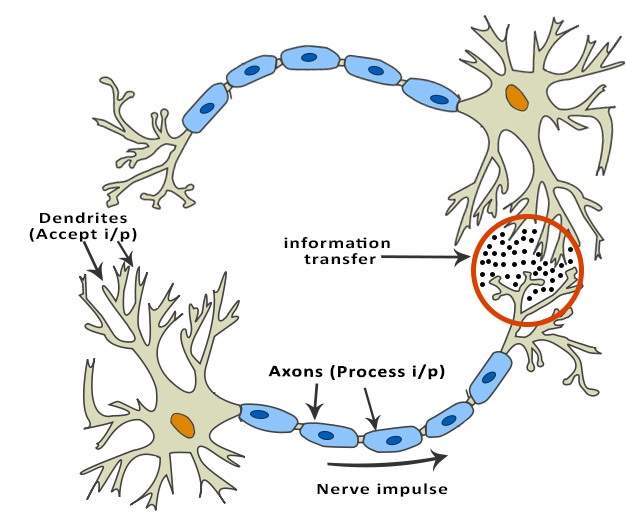
Combinarea predicțiilor de la multiple modele are rezultate mai bune atunci când predicțiile sub-modelelor sunt necorelate sau cât mai puțin corelate. Random Forest schimbă algoritmul de creare al arborilor într-o manieră în care predicția lor va fi mai puțin corelată.

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 optima. Algoritmul de Random Forest modifică această procedură în așa fel încât algoritmul de învățare să fie limitat la un eșantion aleator de atribute din care să facă alegerea.

Numărul de atribute 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-validare. Valoarea standard pentru clasificare este radical din numărul total de atribute, iar pentru regresie se va alege doar o treime din totalul de atribute.

# Retele neuronale artificiale

Ideea din spatele redelelor neuronaleartificiale se bazeaza pe presupunerea ca ceea ce se intampla in creierul nostru atunci cand luam deciziile corescte, poate fi imitat folosind calculatorul imitand in acest fel neuronii si dendritele.

Corpul uman este compus de 100 de miliarde de celule nervoase. Ele sunt conectat unele cu altele prin axoni. Stimulii din mediu extern sau datele de intrare de la organele de senzoriale sunt accptate de dendrite. Acesti stimuli creaza impulsuri electrice care traverseaza reteaua neuronala. Un neuron poate trimite mesajul la un alt neurona pentru a se ocupa de problema sau poate decide sa nu-l mai trimita mai departe.

Retelele neuronale artificiale sunt compuse din noduri multiple, care imita neuronii biologii din creierul uman. Neuronii sunt conectati prin legaturi si ei ineractioneaza unii cu altii. Nodurile primesc valori de intrare si aplica din operatii simple asupra acestor date. Rezultatul acestor operatii este pasat altor neuroni. Rezultatul unui neuron se numeste valoarea nodului sau activarea lui.

### Multilayer perceptron

In domeniul retelelor neuronale artificiale de celee mai mlti ori numite simplu retele neuronale si cel mai probabil sunt cele mai folosite tipuri de retele reuronale. Un perceptron este un model format dintr-un singur neuron si a fost precursorul retelelor neuronale largi care exista astazi.

Puterea retelelor neuronale vin din abilitatea lor de a invata reprezentarea datelor de antrenare si cum aceste date se relationeaza cu variabila de iesire, care trebuie prezisa. In acest sens, retelele reuronale invata o mapare. Din punct de vedere matematic, ele sunt capabile sa invete orice functie de mapare si a fost demonstrat ca pot fi privite ca un algoritm universal de aproximare.

### Neuronii

Unitatea de baza dintr-o retea neuronala sunt neuronii. Acestia sunt doar unitati simple de calcul care au propriile lor ponderi pe care le aplica valorii de intrare si produce o valoare de iesire folosind o anumita functie de activare.

Functia de activare

Output

Input

Pondere

Pondere

Pondere

Ponderile sunt de obicei initializate random cu valori mici, cuprinse in intervalul [0, 0.3], dar sunt si modalitati mai complexe de initializare a lor.

Functia de activare transmite o valoare mai departe in functie de suma produselor dintre ponderi si atributele de intrare ca si rezultat al neuronului. Daca suma este mai mare decat un anumit prag (sa spunem mai mare de 0.5), atunci rezultatul neuronului va fi 1, altfel va fi 0.

Cel mai mult se folosesc functii de activare neliniare. Aceste functii permit retelelor neuronale sa combine mai multe inputuri in moduri mult mai complexe, imbunatatind astfel capabilitatea modelului creat de intreaga retea. Un exemplu de functie logistica neliniara este functia logistica (sau sigmoid). Rezultatul este o valoare reala intre 0 si 1.

### Retele neuronale

SA COMPLETEZ

# K-nearest neighbors (K-NN)

KNN nu construieşte alt model decât acela de a memora întregul set de date de antrenare, deci nu este nevoie de o etapă de învăţare. Deoarece setul de date este memorat, aceste date trebuie să fie consistente. În practică, se recomandă ca setul de date sa fie pre-procesat. Actualizarea modelului cu date noi se poate face foarte uşor, fără să fie nevoie de procesări complexe.

Predicţia pentru o nouă instanţă (X) se bazează pe căutarea în întregul set de date a celor mai asemănătoare K instanţe (a celor mai apropiaţi 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 instanţe cu instanţa ce trebuie clasificată trebuie să se folosească o măsură de distanţă. Pentru valorile reale ale variabilelor de intrare, cea mai cunoscută măsură de distanţă este distanţa euclidiană. Aceasta se defineşte ca fiind rădăcina pătrată a sumei diferenţelor valorilor de pe fiecare dimensiune la pătrat dintre punctul nou (Xnou) si punctul existent (Xi). Aceasta este distanţa clasică dintre două puncte.

Alte măsuri de distanţe populare sunt:

* Distanţa Hamming: Calculează distanţa dintre doi vectori binari;
* Distanta Manhattan: Calculează distanţa dintre doi vectori reali, sumând valoarea absolută a diferenţelor;
* Distanta Minkowski: este o generalizare a distanţei euclidiene şi a distanţei Manhattan.

Pentru fiecare problemă trebuie căutată masura de distanţă care se potriveşte cel mai bine în funcţie de proprietăţile datelor cu care se lucrează. Atunci când nu se cunoaşte cea mai bună măsură de distanţă, se pot experimenta mai multe metrici de distanţă cu diferite valori pentru K şi se va alege modelul care va avea cea mai bună acurateţe.

Spre exemplu, distanţa Euclidiană este o metrică bună atunci când instanţa de intrare are atribute numerice la o scală asemănătoare (ex. lungimea şi lăţimea dreptunghiurilor). Distanţa Manhattan este o metrică bună pentru cazul în care variabila de intrare nu are atribute de acelaşi tip (ex. vârsta, sexul, greutatea unei persoane).

Valoarea cea mai bună pentru K poate fi găsită prin încercarea mai multor valori şi selectarea celei mai bune. Este o buna idee sa se foloseasca valori mai mici decat 21.

Complexitatea algoritmului KNN creşte pe măsură ce cresc şi datele de antrenare. Pentru seturi de date foarte mari, KNN va alege random un eşantion din care va calcula cei mai apropiaţi K vecini.

Când se foloseşte KNN pentru clasificare, predicţia va fi calculată în funcţie de clasa care are cele mai multe apariţii în cei mai apropiaţi K vecini. Fiecare instanţă, în esenţă, votează pentru clasa căreia îi aparţine şi clasa cu cele mai multe voturi câştigă.

Se poate calcula şi probabilitatea cu care o instanţă aparţine unei clase. Spre exemplu, pentru o problemă de clasificare binară: P(barbat) = Numărul de instanţe aproapiate de tipul bărbat raportat la numarul total de vecini, K.

Daca există un număr par de clase, atunci este o bună idee ca valoarea lui K să fie impară, pentru a evita situaţiile în care două clase pot avea acelaşi număr de voturi, situaţie în care apare un caracter random în algoritm. Iar dacă avem un număr impar de clase, este o buna idee ca numărul de vecini pe care-i vom lua in considerare (K) sa fie un numar par.

PRE-PROCESAREA DATELOR PENTRU KNN

* Re-scalarea datelor: KNN funcţionează bine dacă toate datele sunt la aceeaşi scală. Se recomandă ca datele să se normalizeze într-un interval cuprins între 0 şi 1.
* Ştergerea instanţelor care nu au toate atributele: dacă lipsesc atribute, înseamnă că nu se poate calcula similaritatea cu noile instanţe. Acest lucru înseamnă că aceste date oricum nu pot fi folosite şi este natural, să le scoatem din setul de date.
* Numarul de dimensiuni: KNN se pliază pentru problemele în care datele sunt dispunse într-un spaţiu cu dimensiuni cât mai puţine. Se poate aplica şi pe mai multe dimensiuni (de ordinul sutelor de atribute), dar nu va avea aceeaşi acurateţe ca alte tehnici de clasificare. Pentru KNN se poate aplica o reducere de dimensiuni asupra atributelor de intrare.

# Regresia Logistica

Regresia logistică este o tehnică împrumutată de învățarea automată din statistică. Această metodă se aplică, în special, în cazul problemelor de clasificare binara. Numele vine de la funcția nucleu pe care este construită această metodă: 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 si îi atribuie o valoarea între 0 și 1, dar niciodată exact aceste limite.

Regresia logistica foloseste o ecuatie ca model al datelor din acest motiv se aseamana foarte mult cu regresia liniara. Pentru fiecare valoarede intrare (x) se aplica o combitatie liniara folosind diferite ponderi (coeficienti) pentru a se prezice o valoare de iesire (y). Diferenta dintre regresie liniara este ca valoarea de iesire este binara, luand valori din multimea {0, 1} si nu valori numerice.

Urmatoarea ecuatie reprezinta un exemplu pentru acest mod de clasificare:

Unde y este rezultatul, iar fiecare sunt coeficientii valorii de intrare (x). Daca datele de intrare sunt formate din mai multe atribute (coloane), atunci pentru fiecare atribut vom avea asociat cate un coeficient b (o valoare reala, constanta) care trebuie invatat din datele de antrenare.

Reprezentarea modelului pe care trebuie sa o memoram sunt doar coeficientii ecuatiei, adica valorile corespunztoare pentru fiecare b.

### Regresia logistica prezice probabilitati

Regresia logistica modeleaza probabilitatea unei instante de a apartine unei categorii (ex. Prima categorie). De exeplu, daca am modela sexul oamenilor (barbati/femei), in functie de inaltimea lor, atunci prima categorie poate fi *barbat,* iar modelul regresiei logistice poate fi scris sub forma de probabilitate: probabilitatea ca dandu-se inaltimea unei persoane, aceasta sa fie fie barbat. Mai formal:

.

Generalizand putem spune ca regresia logistica modeleaza probabilitatea ca dandu-se o instanta de intratre (X), aceasta sa apartina unei clase prestabilite (Y=1). Formal putem scrie in felul urmator:

Regresia logistica este o metoda liniara, dar predictia este transformata de functia logistica. Din acest motiv nu mai putem sa privim rezultatul predictiei ca o combinatie liniara a datelor de intrare, asa cum se intampla in cazul regresiei liniare. Aplicand functia logistica pe probabilitatea anterioara, putem privi modelul obtinut in felul urmator:

In urma unor transformari imediate, se poate observa ca:

Logaritmand ajungem la:

.

Aceasta formula este folositoare, deoarece se poate face calculele din partea dreapta in mod liniar(in acelasi mod ca la regresia liniara), iar rezultatul, din partea stanga, poate fi privit ca logaritm din probabilitatea ca instanta X sa apartina primei categorii.

### Invatarea modelului de regresie logistica

Coeficientii algoritmului de regresie logistica trebuie sa fie estimati din datele de antrenare. Aceasta estimare se realizeaza cu folosind o metoda numita “estimarea probabilitatii maxime” (**M**aximum **L**ikelihood **E**stimation).

Aceasta tehnica de estimare a coeficientilor este comuna in algoritmi de invatare automata, chiar daca face anumite presupuneri despre distributia datelor. Cei mai buni coeficienti vor determina un model care va face o predictie foarte aproape de 1 (persoana este barbat) pentru categoria de baza si o valoare foarte aproape de 0 (persoana este femeie) pentru cealalta categorie. In mod intuitiv aceasta tehnica de estimare, pentru cazul regresiei logistice poate fi privit ca si o metoda de a cauta cei mai buni coeficienti astfel incat eroarea (discordanta dintre categoria prezisa si categoria reala) sa fie cat mai mica.

### Predictia categoriei cu regresia logistica

Predictia categoriei cu regresia logistica este simpla. Sa presupunem ca avem un model care face predictia daca o persoana este barbat sau femeie pe baza inaltimii.

Dandu-se o persoana de 150 de cm sa se determine daca aceasta este barbat sau femeie.  
Sa presupuem ca am invatat deja coeficientii si acestia sunt si . Folosind ecuatia de mai sus putem calcula probabilitatea ca o persoana de 150 de cm sa fie barbat:

Deci probabilitatea este aproape 0 ca aceasta persoana sa fie barbat. In practica, putem folosi direct probabilitatile si vom face clasificare in felul urmator: Daca probabilitatea este mai mica decat 0.5, atunci acea persoana este catalogata ca fiind barbat, altfel va fi catalogata femeie.

# Extragerea trăsăturilor MFCC

Primul pas ce trebuie făcut pentru clasificarea melodiilor este acela de a extrage trăsături din ele. Partea cea mai importantă este să încercăm să înţelegem modul în care urechea umană modelează sunetele şi să încercam să reproducem acest proces. Mel Frequency Coefficients( MFCC) sunt caracteristici folosite foarte frecvent în vorbirea automată şi în recunoaşterea vocală.

Aceşti coeficienţi au fost calculaţi prima dată de Davis şi Mermelstein în 1980 şi de atunci sunt un punct de referinţă în domeniul procesării fişiserelor audio.

Atunci când vrem să calculăm aceşti coeficienţi trebuie să trecem prin mai multe etape:

1. Împărţirea 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 logaritmate la pasul anterior
5. Se păstrează primii 13 coeficienţi

## Împărţirea 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 şi în intervale foarte mici de timp). De aceea vom împărţi 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 rezolabila a spectrului, iar dacă frame-ul ar fi mai mare atunci diferenţaele dintre frame-uri nu ar putea fi comparate .

## Estimarea periodogramei spectrului de energie

Următorul pas este să calculăm energia pe cele 13 frecvenţe pentru fiecare frame. Acest pas simulează funcţia cohleei, denumită şi melcul membranos. Acesta este un organ din urechea internă umană care vibrează în diferite locuri în funcţie de frecvenţa pe care o are sunetul. În funcţie de locul în care vibrează cohleea, diferiţi nervi se activează şi primesc informaţii pe care le vor transporta către creier. Periodograma face cam acelaşi lucru, identificând frecvenţa din fiecare frame.

Estimarea făcută de periodogramă încă mai conţine multe informaţii nefolositoare. În special cohleea nu poate face diferenţa dintre două frecvenţe apropiate. Acest efect devinte din ce în ce mai pronunţat pe măsură ce frecvenţa creşte. Din acest motiv, vom grupa mai multe estimări şi 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 exista aproape de 0 Hertz. Cu cât frecvenţa creşte, filtrele se adaptează astfel încât diferenţa de energie să nu mai fie o problemă.

## Se logaritmează fiecare energie

După ce s-a filtrat energia din sunetul audio, pe mai multe frecvențe, trebuie să le logaritmăm. Această logaritmare se face pentru a ne apropia 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 sune 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.

## Se calculează DCT

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 bruște de energie și acest lucru afectează procesarea ulterioară.

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

https://www.tutorialspoint.com/artificial\_intelligence/artificial\_intelligence\_neural\_networks.html

http://amueller.github.io/

|  |  |  |  |
| --- | --- | --- | --- |
| Limbajul de programare | Avantaje | Dezavantaje | Biblioteci importante |
| R | 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 |
| Python | 1. Open source  2. Ușor de învățat  3. Păstrează avantajele unui limbaje 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.Sciy  6.Theano,  7.Nltk |
| MATLAB | Potrivit pentru:  1. procese 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 |
| OCTAVE | 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 |
| Julia | 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 exista foarte multe biblioteci externe | MLBase  MLUtils  MLKernels,  Clustering  MAchineLEarning |

Anexa 1

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

Aceste tabele sunt o traducere a unei analize făcute de Siva Prasad Katru.

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