# 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.

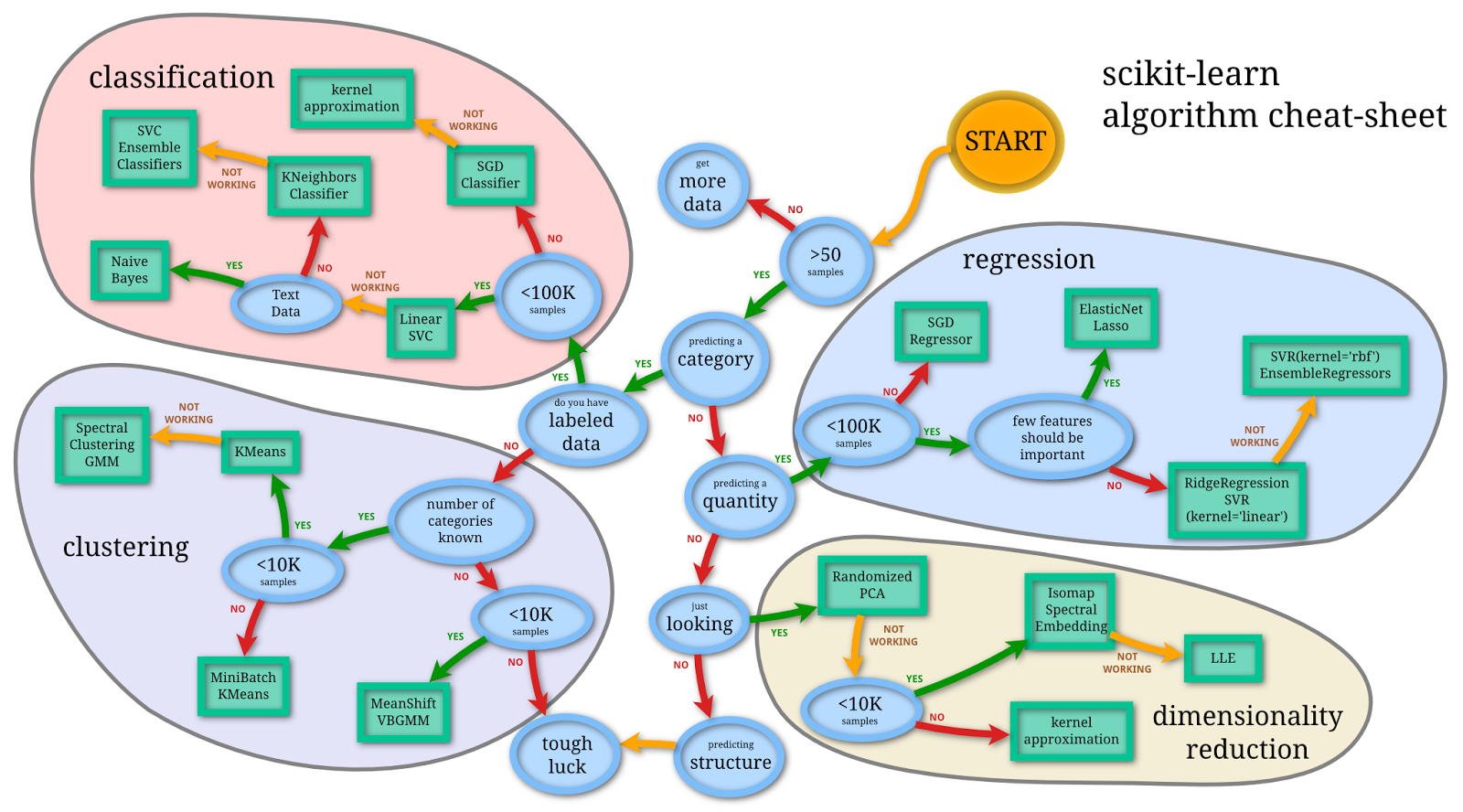
### Contribuitorii sunt experți

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.

### Acoperă majoritatea ariilor din domeniul învățării 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.

Bărbat

Bărbat

Femeie

DA

NU

DA

NU

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.

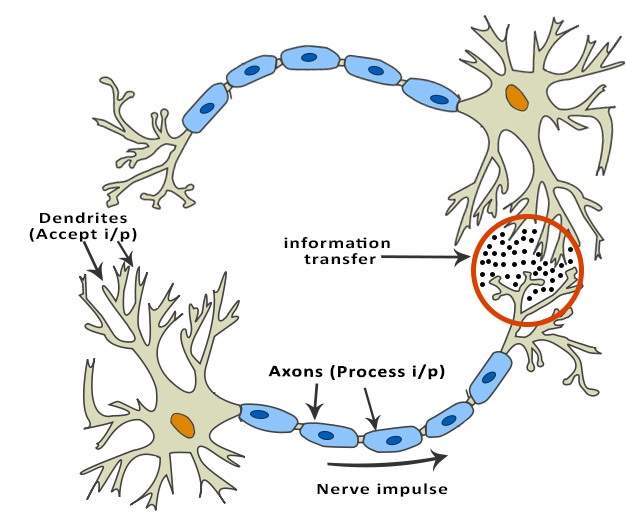
Combinarea predicțiilor de la multiple modele are rezultate mai bune atunci când predicțiile sub-modelelor sunt necorelate sau cât mai puțin corelate. Random Forest schimbă algoritmul de creare al arborilor într-o manieră în care predicția lor va fi mai puțin corelată.

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.

# 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.

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 in care stimulul extern este ignorat.

Rețelele Neuronale Artificiale (**A**rtificial **N**eural **N**etworks - 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.

### Multilayer Perceptron

În domeniul rețelelor neuronale artificiale (de cele mai multe ori numite simplu rețele neuronale) modelul “multilayer perceptron” este, cel mai probabil, cel mai folosit tip de rețea reuronală. Un perceptron este un model format dintr-un singur neuron și a fost precursorul rețelelor neuronale largi care există astăzi.

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 reuronale 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.

### 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.

Funcția de activare

Output

Input

Pondere

Pondere

Pondere

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. 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 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.

### Rețele neuronale

Neuronii sunt grupați și formează, împreună, rețele neuronale.

Un rând de neuroni formează un strat din rețeua neuronală. În practică, rețelele neuronale sunt construite din mai multe strate.

#### 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 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.

#### Niveluri 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 reuronală 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 accesat [ **LINK -** https://cloud.google.com/products/machine-learning/ ]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.

#### Nivelul de iesire

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 buna 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.

# K-nearest neighbors (KNN)

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 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 in 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 şi îi atribuie o valoarea între 0 și 1, dar niciodată exact aceste limite.

Regresia logistică foloseşte o ecuaţie ca model al datelor din acest motiv se aseamană foarte mult cu regresia liniară. Pentru fiecare valoare de intrare (X) se aplică o combinaţie liniară folosind diferite ponderi (coeficienţi) pentru a se prezice o valoare de ieşire (y). Diferenţa faţă de regresie liniară este că valoarea de ieşire este binară, luând valori din mulţimea {0, 1} şi nu valori numerice.

Următoarea ecuaţie reprezintă un exemplu pentru acest mod de clasificare:

Unde y este rezultatul, iar sunt coeficienţii pentru atributele valorii de intrare (X). Dacă datele de intrare sunt formate din mai multe atribute (coloane), atunci pentru fiecare atribut vom avea asociat câte un coeficient b( o valoare reală, constantă) care trebuie învăţat din datele de antrenare.

Reprezentarea modelului pe care trebuie să o memorăm sunt doar coeficienţii ecuaţiei, adică valorile corespunzăoare pentru fiecare b.

### Regresia logistică prezice probabilități

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

.

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

Regresia logistică este o metodă liniară, dar predicţia este transformată de functţa logistică. Din acest motiv nu mai putem să privim rezultatul predicţiei ca o combinaţie liniară a datelor de intrare, aşa cum se întâmplă în cazul regresiei liniare. Aplicând funcţia logistică pe probabilitatea anterioară, putem privi modelul obţinut în felul următor:

În urma unor transformări imediate, se poate observa că:

Logaritmând ajungem la:

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

### Învăţarea modelului de regresie logistică

Coeficienţii algoritmului de regresie logistică trebuie să fie estimaţi din datele de antrenare. Această estimare se realizează folosind o metodă numită “estimarea probabilităţii maxime” (**M**aximum **L**ikelihood **E**stimation) - MLE.

Această tehnică de estimare a coeficienţilor este comună în algoritmii de învăţare automată, chiar dacă face anumite presupuneri despre distribuţia datelor. Cei mai buni coeficienţi vor determina un model care va face o predicţie foarte aproape de 1 (persoana este bărbat) pentru categoria de bază şi 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 privit ca o metodă de a căuta cei mai buni coeficienţi astfel încât eroarea (discordanţa dintre categoria prezisă şi categoria reală) să fie cât mai mică.

### Clasificarea cu regresia logistică

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

Dându-se o persoană de 150 de cm să se determine dacă aceasta este bărbat sau femeie.  
Să presupuem ca am învăţat deja coeficienţii şi aceştia sunt şi . Folosind ecuaţia de mai sus putem calcula probabilitatea ca o persoană de 150 de cm să fie bărbat:

Deci probabilitatea este aproape 0 ca această persoană să fie bărbat. În practică, putem folosi direct probabilitatilea şi 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.

# 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 diferenţele dintre frame-uri nu ar putea fi comparate .

## Estimarea periodogramei spectrului de energie

Următorul pas este să calculăm energia din jurul mai multor 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.

Introducere

Dimensiunea mare de informatii multimedia de pe internet face necesara construirea unor noi tipuri de utilitare automate, capabile sa clasifice melodii dupa gen, pe baza continutului. Genul unei melodii este unul din cele mai importante caracteristici ce descriu o melodie si este folosita pentru organizarea colectiilor mari de muzica digitala.

Din perspectiva recunoasterii de modele, clasificarea melodiilor in functie de gen este o problema interesanta de cercetare, din moment ce melodiile au o variatie a semnalului audio foarte variata. Un alt aspect interesat este ca clasificarea genurilor este o problema cu mai multe clase. Pentru a putea face clasificari cu mai mult de doua clase putem avea doua abordari:

1. Folosirea tehnicilor care pot clasifica instantele in mai multe clase, in mod natural, producand o separare complexa a spatiului instantelor. Exemple de tehnici pot fi arbori de decizie, K nearest neighbors, retele neuronale etc.

2. Folosirea unei strategii pentru descopunerea problemei de clasificare in mai multe clase intr-o serie de probleme de clasificare binara, folosind clasificatori care vor clasifica spatiul de solutii in mod binar. Exemple de tehnici SVM , clasificare logistica, etc.

De cateva ori s-a incercat o abordare in maniera asamblista. Aceste abordari s-au dovedit a fi un succes. Ele constau in aplicarea a mai multor clasificatori, nu doar a unuia singur, fiecare specializat pe o anumita parte a problemei. In acest fel, fiecare clasificator este antrenat pe atribute diferite, iar rezultatul final va fi combinatia acestor clasificatori de catre un alt clasificator ce combina aceste modele in mod dinamic. Aceasta abordare poate fi privita ca o mixtura de experti aplicata la problema clasificarii. De asemenea, poate fi privita si ca o variatie a algoritmului bossting. O posibila explicatie a succesului acestei metode este ca aplicarea clasificatorilor pe doar un subset de atribute face ca suprafata de decizie sa mult mai simpla, iar clasificatorii pot sa produca rezultate cu acuratete mai mare.

**sa continui ...**

# Clasificarea melodiilor

Problema clasificatrii automate a melodiilor dupa gen poate fi descrisa ca sarcina de a atribui pentru fiecare melodie cate un gen muzical. In acest proces de clasificare se incearca crearea unui model cunoscut care sa aproximeze cat mai bine datele de antrenare.

In abordarea pe care o vom prezenta, crearea modelului cuprinde mai multi pasi. In primul rand, se vor extrage atributele din melodie conform pasilor **explicati la capitolul MFCCs .** Dupa ce acest pas vom avea setul de date pregatit pentru procesarile ulterioare. Acesta consta in 1000 de instante de intrare, fiecare avand cate 13 atribute, reprezentand suma energiilor ce apar pe durata melodiilor grupate in 13 grupe, in functie de frecventa.

Dupa ce avem setul de date pregatit, acesta este impartit initial in 2 sub-categorii: date de antrenare si date de testare, in proportie de 60% resprectiv 40%. Prima categorie, cea de antrenare va fi folosita pnetru antrenarea simultana si independenta a 6 clasificatori:

1. Regresie Logistica,
2. Retele Neuronale,
3. Masini pe Vector Suport (SVM),
4. Gradient Boosting,
5. Random Forest,
6. KNN

Fiecare din cei 6 clasificatori vor fi antrenati pe 600 de instante, fiecare avand cate 13 atribute. Dupa faza de antrenare, fiecare clasificator urmeaza sa fie testat. Pntru asta, datele de testare au mai fost impartie, inca o data in 2, in aceeasi proportie: 60% si 40%. Din cele 400 de instante nefolosite la antrenare, se aleg 240 cu ajutorul carora se testeaza modelul obtinut si se calculeaza acuratetea pentru fiecare clasificator.

Prima abordare a fost aceea de a gasi un singur clasificator, cel mai bun, care sa obtina cea mai mare acuratete. S-au incercat, pe rand, mai multi clasificatori, fiecare cu cate o acuratete diferita. Unii clasificatori s-au pliat pe aceasta problema, avand o acuratete destul de buna, altii nu s-au potrivit la fel de bine.

## Regresie Logistica

Primul clasificator incercat a fost Regresia Logistica. Aceasta tehnica de clasificare este folosita, in mod special, in cazul problemelor de de clasifice binara. In cazul de fata, problema cere o clasificare in 10 categorii diferite. De aceea s-a aplicat o tehnica numita in invatarea automata One-vs-All (Unul-vs-Toti).

### One-vs-All

Aceasta strategie consta in a antrena cate un clasificator pentru fiecare categorie, specializandu-se sa faca distinctia intre instantele care apartin unei singure categorii si cele care nu apartin acestei categorii. Fiecare clasificator va returna probabilitatea ca noua instanta sa apartina clasei pe care s-a specializat el. Iar rezultatul final, va fi categoria cu probabilitatea cea mai mare ca noua instanta sa-i apartina, conform tuturor modelelor.

Spre exemplu sa presupun ca vrem sa clasificam 3 categorii de obiecte si avem urmatoarele instante:

Vor fi create 3 modele fiecare specializat in a recunoaste cate o categorie: triunghi, oval, respreciv dreptunghi. Primul clasificator va avea ca rezultat probabilitatea ca o noua instanta sa apartina categoriei triunghi. Al doilea, va intoarce probabilitatea ca aceeasi instanta sa apartina categoriei oval, respectiv dreptunghi. Rezultatul final, al intregii clasificari va fi dat de categoria care are probabilitatea cea mai mare.

P(X= )

P(X= )

P(X= )

Clasificare finala:

Categoria cu probabilitatea cea mai mare



Deoarece, in problema clasificarii melodiilor avem 10 genuri muzicale, vom avea 10 modele create cu ajutorul regresiei logistice. Dar rezultatul final va fi, un singur gen muzical. Acesta abordare modeleaza datele nici foarte bine dar nici extraordinar de rau, avand o acuratete de 46.32%. Spunem ca aceasta acuratete nu este foarte rea in conditiile in care, probabiliattea pentru a alege in mod aleator, genul unei noi instante si a nimeni varianta corecta este de doar 10%. Deci regresia logistica aplicata pe problema clasificarii melodiilor dupa gen, este de 4.6 ori mai buna decat un clasificator aleator.

## Rețele Neuronale

Al doilea clasificator incercat este un model creat cu ajutorul retelelor neuronale. Primul strat din reteaua creata are 13 neuroni, deoarece datele cu care lucram au cate 13 atribute. Reteaua va area un singur strat ascus, ce va fi format din 100 de neuroni. Iar ultimul strat este format din 10 neuroni, fiecare cu cate o probabilitate asociata pentru fiecare gen muzical.

x1

x2

x12

x13

Input

X

y1

y2

Y10

max



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

## Gradient Boosting

## Random Forest

## KNN

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Y1 | Y2 | Y3 | Y4 | Y5 | Y6 | Y7 | Y8 | Y9 | Y10 |

Clasificator

Pe langa aceasta faza de testare, pentru fiecare aceleasi 240 de instante se memoreaza probabilitatile fiecarei instante de a apartine la fiecare

Rezultaul intors de fiecare din acesti clasificatori este reprezentat de un vector de 10 valori reale, ce reprezinta probabilitatea ca o

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 (one versul all)

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

AMGC problem

https://www.researchgate.net/figure/220942923\_fig1\_Fig-1-An-example-illustrating-the-problem-space-decomposition-strategy

Sa ma uit aici:

https://www.researchgate.net/publication/3333877\_Musical\_genre\_classification\_of\_audio\_signals\_IEEE\_Trans\_Speech\_Audio\_Process

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