Cuprins

Introducere

Odată cu creșterea volumului de date, de informație din ultimii ani, învățarea automată a devenit o tehnică tot mai folosită în valorificarea acestora, iar valoarea pe care au dobândit-o datele în acești ani au făcut din învățarea automată o tehnologie tot mai de dorit de către cei care au acces la acestea. Astfel de tehnnici sunt favorizate și de continua creștere a puterii de procesare și a memoriei calculatoarelor, care acum permit antrenarea unor modele mult mai complexe și rapide față de deceniile precedente.

Pentru a înțelege importanța învățării automate, este necesară o prezentare a legăturii dintre aceasta și date. Învățarea automată este o tehnică analitică a datelor prin care se interpretează diferite tipare existente în cadrul acestora. Ea ne permite să-i dăm unui calculator posibilitatea de a învața din experiență: pe baza unor date, informații care i-au fost prezentate, acesta să poată oferi predicții, clasificări pe informații noi. Un algoritm de învățare automată își îmbunătățește astfel performanța cu cât cantitatea datelor este mai mare. De aceea, un aspect important îl reprezintă volumul de informație în continuă creștere.

Importanța acestui subiect în generația curentă și interesul personal pentru domeniul medical m-au convins în decizia de a trata în cadrul lucrării o problema curentă din medicină prin prisma învățării automate. În urma recomandărilor, am ales dintre aceste probleme curente tot mai prezente pe cea a cancerului de sân. Desigur, m-am axat strict pe rezultatele algoritmilor, pe îmbunătățirea acestora și prezentarea unor aspecte importante care intervin și pot fi îmbunătățite atunci când te confrunți cu o problema de învățare automată.

Am pornit de la simpla idee de a testa diferiți algoritmi de învățare automată pe un set de date reprezentând cazuri reale memorate de cancer de sân. Cu fiecare pas în schimb am descoperit profunzimea unei probleme de acest tip, cât de multe aspecte trebuie luate în calcul pentru orice mică îmbunătățire adusă rezultatelor și cât de multe opțiuni sunt existente pentru fiecare astfel de aspect în parte. Am simțit nevoia existenței unei lucrări, prezentări care să încorporeze toate aceste lucruri într-un singur pachet.

Pornind de la această dorință, am ajuns la concluzia că ar trebui să încerc realizarea unei lucrări care să atingă măcar la suprafață, câte puțin din fiecare dintre cele mai importante subiecte prezente în problemele de învățare automată: datele lipsa, selecția atributelor și ansamblurile. Astfel, structura lucrării va fi următoarea:

- în prima jumătate voi prezenta câteva aspecte des întâlnite în învățarea automată și cele 3 probleme enumerate mai înainte, de ce sunt ele necesare, care sunt câteva dintre abordările care ar putea fi considerate pentru fiecare în parte, cu exemple de algoritmi prezentați în mod succint și cu informații cât mai clare;
- în cea de a doua jumătate voi trata practic, pas cu pas, problema de învățare automată pe un set de date cu înregistrări legate de cancerul de sân, generarea unui set cu date lipsa, motivația din spatele alegerii unei anumite metrici pentru tratarea problemei, compararea abordărilor alese și opinii privind cea pe care am considerat-o optimă. De asemenea voi prezenta și probleme pe care le-am întâlnit atunci când am început aprofundarea unora dintre subiecte;
- în final voi enumera posibile îmbunătățiri care ar putea fi aduse lucrării;

1 Noțiuni de bază

Învățarea automată este două tipuri: învățare supervizată si învățare nesupervizată. Multe probleme practice se încadrează în prima categorie. Astfel, o mare parte din probleme de învățare automată se rezumă la identificarea unei funcții, a unei relații între niște variabile de intrare și ieșirile (etichetele) setului de date.

$$f(x) + \epsilon = y$$

Epsilon reprezintă eroarea ireductibilă, acel aspect aleatoriu care nu poate fi prevazut, dupa care nu se poate modela.

Primul lucru care ar trebui stabilit în momentul în care se începe studiul pe o problema specifică de învățare automată este dacă aceasta reprezintă o problema de regresie sau una de clasificare. În funcție de tipul acesteia, diferite abordări bazate pe algoritmi de învățare pot fi recomandate.

Clasificare

La problemele de clasificare, etichetele înregistrarilor (valorile țintă) sunt categorice, discrete și iau valori dintr-o mulțime finită de elemente. Scopul final este de a identifica apartenența fiecărei înregistrări la o mulțime specifică.

Există două tipuri de clasificare: clasificare binară în care există doar două valori țintă și clasificarea pe mai multe clase. Problema predicției cancerului (malign/ benign) abordată în această lucrare, sau cea a clasificării tipului de e-mail (*spam* sau nu) reprezintă exemple de clasificări binare. Identificarea unei cifre specifice dintr-o imagine, sau a unui tip de obiect sunt exemple de clasificări din cea de-a doua categorie.

Regresie

La problemele de regresie, etichetele sunt continue, valorile aparținând unui interval și nu unei mulțimi finite. Scopul este de a prezice sau a estima un rezultat pe baza valorilor atributelor. Un exemplu pentru cazurile de regresie l-ar putea reprezenta predicția unor prețuri.

Subiectivitate (*bias*)

Este o sursă de erori dintr-un model, care măsoară cât de îndepărtate sunt elementele estimate de valorile țintă.

Varianța (*variance*)

Varianța măsoară cât de împrăștiate sunt elementele de media lor. Reprezintă senzitivitatea la zgomotul care apare în date. Un model cu varianța ridicată se modelează prea mult după setul de antrenament și nu poate generaliza cât să poată prezice cu precizie ridicată și pe setul de test.

2 Datele lipsă și tratarea acestora

Datele lipsa reprezintă una dintre marile probleme care pot interveni în procesul de lucru cu seturi de date, de asemenea și una dintre principalele surse de erori în cod. Tratarea acestora este o provocare și trebuie adresată cu atenție pentru obținerea performanței, în plus, o mare parte din algoritmii de învățare automată nu pot lucra cu astfel de date. Se poate încerca înlăturarea înregistrărilor care prezintă valori lipsa pentru unele atribute, dar adeseori nu este o soluție viabilă.

Primul pas în tratarea datelor lipsă o reprezintă analiza setului de date, trebuie înțeles motivul pentru care lipsesc anumite valori, ce procentaj din totalul acestuia îl reprezintă și care este abordarea potrivită ce poate fi folosită. Datele pot lipsi complet aleatoriu, însemnând că nu sunt influențate de nicio altă valoare, de niciun alt atribut sau pot lipsi cu un motiv specific, depinzând de niște valori ipotetice sau fiind dependențe de alte atribute. Pentru primul caz, dacă numărul de date lipsă nu este considerabil, se poate considera înlăturarea, dar pentru al doilea, deoarece datele sunt dependențe, se poate produce un bias în cadrul modelului.

2.1 Tratarea datelor lipsă prin ștergere

- Înlăturarea unui întreg atribut (coloana) poate fi posibilă atunci când mai mult de jumătate din înregistrările prezente au valori lipsă în dreptul atributului respectiv. De cele mai multe ori trebuie precedată de o analiză atentă a setului de date și a importanței atributului respectiv; de obicei se preferă ca atributele să fie păstrate
- Înlăturarea unei înregistrări presupune ștergerea unei întregi linii din setul de date atunci când există date lipsă pe oricare dintre atributele acesteia. Simplitatea acestei abordări reprezintă un mare avantaj, în schimb poate provoca probleme majore în cazul multor astfel de situații cu date lipsă. Poate fi utilă atunci când numărul acestora este relativ mic. Dacă înregistrarea ce urmează a fi înlăturată diferă în mare parte de toate celelalte cazuri complete din setul de date se pot crea parametri și rezultate subiective (cu bias). Dacă numărul acesta este foarte mare, înlăturarea lor poate reduce din puterea de predicție a modelului antrenat, limitându-se considerabil numărul înregistrărilor

2.2 Tratarea datelor lipsă prin completarea cu noi valori

- Calcularea mediei, a medianei și a modulului (valoarea cea mai frecventă) este o metodă ușoară, accesibilă de a trata datele lipsa. Media și mediana pot fi folosite pentru date continue (se iau valorile prezente din cadrul atributului cu un câmp lipsă și este completat cu media respectiv valoarea de mijloc a acestora) iar modulul pentru date categorice (valoarea cea mai des întâlnită pentru atributul respectiv). Calculul acestora este rapid, dar prezintă numeroase dezavantaje, unele dintre acestea fiind faptul că reduce varianța în setul de date și compromite relațiile dintre atribute, nu ține cont de acestea; motiv pentru care este recomandată o astfel de abordare doar în cazurile în care un număr mic de date lipsesc [11]
- KNN – se aleg k cei mai apropiați vecini care să fie luați în considerare și o metrică de distanță. Calculul se poate face atât pentru atribute discrete (cea mai frecventă valoare dintre cei k vecini) cu o distanță precum distanța Hamming, cât și pentru atribute continue (media dintre cei k vecini), cu distanțe precum cea Euclidiană sau Manhattan. Distanța Euclidiană este recomandată atunci când variabilele de input sunt similare ca și tip, iar cea Manhattan când nu sunt. Problema acestui algoritm este că în ciuda ușurinței cu care poate fi implementat și înțeles, poate fi ineficient din punct de vedere al timpului atunci când se lucrează cu un set de date de dimensiuni mari (ca și număr de atribute, număr de înregistrări) și din punct de vedere al diferențierii vecinilor atunci când sunt prea multe caracteristici
- Predicția valorilor folosind un model antrenat – se poate face setând atributul cu valori lipsa ca și etichetă, valoare țintă. Se antrenează modelul cu înregistrările fără valori lipsă, iar testarea se face pe cele care au, înregistrările fiind completate cu cele prezise de model

3 Selecția atributelor

Atunci când se lucrează cu seturi de date, adeseori se poate observa că dimensiunile acestora sunt relativ mari, nu doar ca număr de înregistrări ci și ca număr de atribute. Astfel de seturi cu un număr ridicat de date este dorit, ajutând în procesul de antrenare al modelului, însă un număr prea mare de caracteristici poate prezenta una dintre următoarele probleme:

- crește timpul de antrenare al modelului
- crește complexitatea modelului antrenat atunci când toate sunt utilizate, ceea ce poate duce la o modelare prea precisă pe setul de antrenament și la incapacitatea de a generaliza pe setul de test (*overfitting*)
- scade din precizia modelului, atributele în plus comportându-se ca și datele cu zgomot (*noisy data*)

Prin selecția atributelor se înțelege alegerea variabilelor cele mai importante în modelul creat, multe dintre ele fiind reduntate, neajutând semnificativ în procesul de antrenare. Uneori, un astfel de subset de atribute poate produce rezultate mai bune decât cel cu dimensiunile originale, pe lângă avantajele clare de îmbunătățire a vitezei de antrenare, performanță, complexitate spațiu și simplicitate.

La momentul actual, există trei metode de selecție a caracteristicilor:

- filtrare (*filter*)
- încorporare (*embedded*)
- învelire (*wrapping*)

3.1 Filtrare

Filtrarea reprezintă un pas de preprocesare, selectarea unui subset de atribute făcându-se pe baza caracterisicilor lor generale, pe unicitatea datelor, printr-o funcție predefinită de evaluare. Nu se aplică niciun algoritm de învățare și nu se testează cât de optimă a fost alegerea subsetului de atribute pe baza rezultatelor modelului antrenat. Se alege un criteriu de clasificare, iar un număr de atribute care au îndeplinit cel mai bine criteriul respectiv sunt selectate.

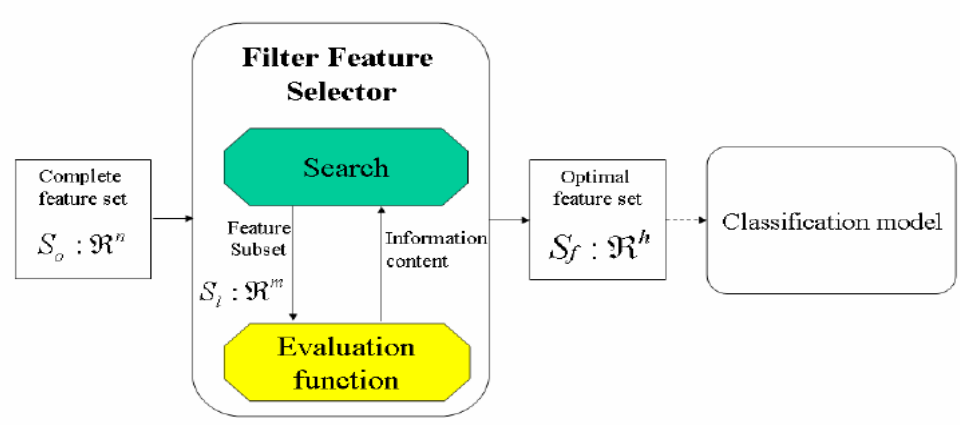

Fig. 1 – Filter Feature Selection[1]

3.1.1 Chi-pătrat (chi2, *chi-square*)

Folosit pe seturile de date cu etichete care au valori categorice, se aplică o statistică chi2 care determină dependența dintre două variabile. Astfel, pentru fiecare atribut se verifică relația acestuia cu variabila țintă, dacă acestea sunt independente atunci atributul nu ajută în decizia stabilirii apartenenței înregistrării la una dintre clase și nu este selectat. Mai precis, se verifică dacă apariția unui atribut este independentă de cea a unei clase specifice (eticheta de predicție). Cu cât valoarea returnată este mai mică, cu atât cele două sunt mai independente iar atributul mai nesemnificativ în procesul de clasificare. [15]

$$\chi^2 = \frac{N(AN - MP)^2}{PM(N - P)(N - M))}$$

Valorile din formulă reprezintă numarul total pentru:

- N- înregistrări
- M- înregistrări în care apare atributul
- P- înregistrări pentru clasa pozitivă
- A- înregistrări pentru clasa pozitivă în care apare atributul

[1] Imagine preluata de la https://www.researchgate.net

3.1.2 Clasificare cu câștig de informație (*mutual info classif*)

Informația mutuală a două trăsături este o măsură a dependenței reciproce dintre acestea, a câștigului de informație pe care îl obținem pentru una dintre caracteristici prin analiza celeilalte.

$$I(X;Y) = \sum_{x,y} P_{XY}(x,y) \log \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)}$$

Știm din formula independenței a două evenimente că dacă X și Y sunt independente, atunci $P_{xy}(x,y)=P_x(x)*P_y(y)$, de unde reiese că raportul dintre cele două este 1, iar valoarea logaritmului 0. Astfel câștigul de informație este 0.

Scopul acestui clasificator este de a descoperi dependența dintre fiecare atribut și etichetă.

3.2 Metode de tip *wrapping*

Metodele de tip *wrapping* oferă rezultate superioare față de cele de tip filtrare deoarece se axează și sunt optimizate pentru relațiile dintre setul de date, atributele acestuia și clasificator, folosindu-se un clasificator pentru a calcula performanța caracteristicilor, mai precis, modelul este antrenat pe subsetul de atribute selectat la fiecare pas.

Deși în principiu găsesc cele mai utile atribute, metodele de tip *wrapping* sunt mai costisitoare din cauza antrenărilor repetate și de asemenea sunt mai susceptibile la *overfitting*.

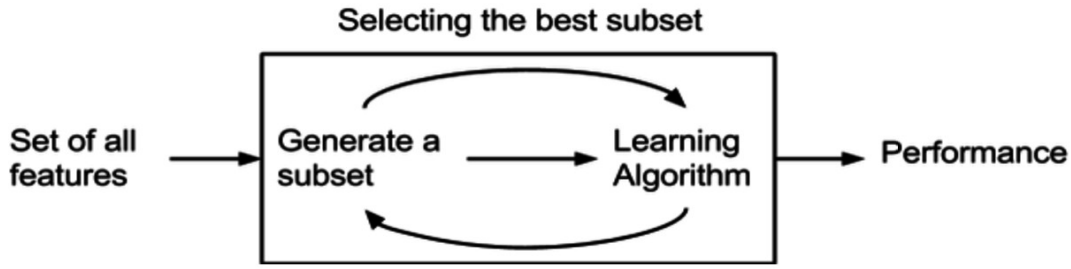

Fig. 2 – Filter Feature Selection[2]

[2] Imagine preluata de la https://www.analyticsvidhya.com

3.2.1 Selecția înainte (*forward selection*)

Este un algoritm de tip greedy, un proces iterativ prin care se selectează câte un atribut la fiecare pas, în funcție de îmbunătățirea pe care o aduce modelului. Se pornește de la o submulțime vidă de atribute, iar la fiecare pas se testează rezultatele modelului adăugând fiecare atribut pe rând, performanța acestuia testându-se prin cross-validare. Se continuă astfel până nu se mai îmbunătățește performanța modelului sau s-a ajuns la un număr stabilit de atribute.

3.2.2 Eliminarea recursivă (*recursive elimination*)

Folosind un clasificator extern (un estimator), se antrenează modelul pe întregul set de atribute și se calculează un coeficient de importanță pentru fiecare dintre acestea, pe baza clasificatorului ales. Atributul cu coeficientul cel mai mic este înlăturat din setul de date iar modelul este antrenat din nou pe noul set. Procesul este repetat până se ajunge la numărul dorit de caracteristici.

3.3 Metode de tip e*mbedded*

Metodele de tip *embedded* încearcă o îmbinare a celor două metode prezentate anterior, filtrare și *wrapping*, fiind similare cu metodele de tip *wrapping*, dar îmbunătățind cele două aspecte negative ale acestora, costurile computaționale și susceptibilitatea la *overfitting*.

3.3.1 Regularizare L1 (*L1-regularization, LASSO*)

Regularizarea încearcă să reducă din complexitatea modelului prin penalizarea unei funcții de pierdere în ideea de a face ca modelul antrenat să generalizeze mai bine pe setul de test, de a evita overfitting-ul.

$$L(x, y) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} |\theta_i|$$

Funcția de pierdere (*loss function*) este suma pătratelor diferenței dintre valorile țintă și valorile prezise. Prin regularizare, la această funcție i se mai adaugă un termen de regularizare care are scopul de a menține ponderile mici.[6]

În cazul regularizării L1, termenul de regularizare îl reprezintă valoarea absolută a ponderilor înmulțit cu un coeficient de penalizare. Cu cât coeficientul de penalizare este mai mare cu atât ponderile sunt penalizate mai mult, modelul devine insuficient de complex și foarte puține atribute contribuie la rezultat. Aceasta se datorează faptului că prin adăugarea valorii absolute, ponderile atributelor pot deveni 0, însemnând că atributul respectiv nu mai contribuie la decizia modelului. Când atributul are pondere 0, se consideră că poate fi înlaturat, metoda fiind foarte utilă atunci când numărul atributelor este mare.[16]

4 Metode de tip ansamblu

Una dintre principalele probleme în crearea unui model constă în alegerea unui clasificator potrivit care să producă rezultate optime pe setul de date ales. Nu există o soluție universală care să se poată potrivi în totalitate în fiecare situație, pentru că fiecare set de date are particularitățile lui.

De asemenea, acuratețea unui model antrenat are la baza două concepte importante, subiectivitatea și varianța (*bias*, *variance*), aspectul cel mai de dorit în crearea unui model optim fiind reducerea acestor două valori cât mai mult. Problema este că în cadrul unui singur model, când se urmărește reducerea uneia dintre ele, cealaltă crește. De exemplu, metoda arborilor de decizie: cu cât arborele are tot mai multe nivele de decizie, este mai complex, deci subiectivitatea este mică, cu atât modelarea se face tot mai specifică setului de antrenament (varianța a crescut). Astfel pot apărea erori de decizie când se vor testa predicțiile modelului pe setul de test deoarece modelul nu este capabil să generealizeze (acest fenomen mai este numit și *overfitting*). [13]

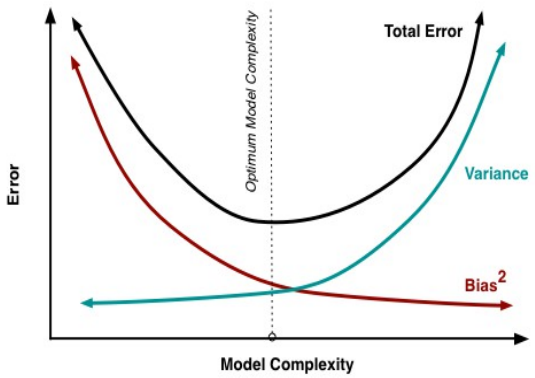

Fig.3 – Variance/ bias tradeoff[3]

Există însă posibilitatea de a reduce ambele valori simultan combinând mai multe modele într-unul singur. De aceea a apărut conceptul de ansamblu. Ansamblurile sunt sisteme cu clasificatori de bază multipli, clasificatorii pot fi diferiți sau pot pot avea la bază același algoritm de clasificare, dar antrenat pe diferite subseturi ale aceleiași mulțimi. Fiecare model este antrenat în mod independent (cu excepția ansamblurilor de tip *boosting*), iar apoi combinate împreună pentru a îmbunătăți puterea de predicție a unui model final. Scopul central este de a avea un set de antrenament pe care fiecare clasificator învață aspecte diferite ale acestuia. Rezultatele acestora sunt apoi îmbinate în diferite moduri.

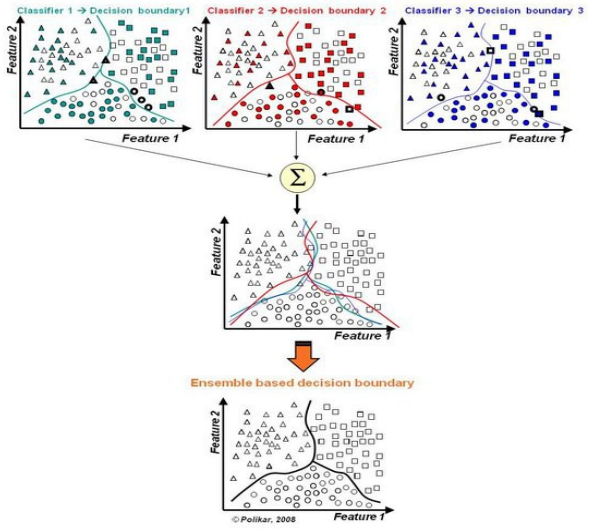

Fig.4 – Basic ensemble example[4]

[3] Imagine preluata de la https://www.analyticsvidhya.com/blog/2015/08/introduction-ensemble-learning/
[4] Imagine preluata de la http://www.scholarpedia.org

4.1 Vot majoritar

Este o metodă simplă de tip ansamblu folosită în problemele de clasificare, predicția fiecărui membru este considerată un vot, urmând ca votul majoritar să reprezinte rezultatul modelului. Echivalentul acestei metode pentru problemele de regresie este media rezultatelor.

Un aspect important care trebuie luat în calcul atunci când este folosit votul majoritar ca metodă de ansamblu este corelarea modelelor alese. Deseori, în cazul modelelor puternic corelate, rezultatul predicțiilor ansamblului creat nu îl depășește pe cel al modelului care produce cele mai bune rezultate. Un exemplu pentru acest caz (unde 1 reprezintă o clasificare corectă):

1101101011 ~ 70%
1100101011 ~ 60%
0111101001 ~ 60%

1101101011 ~ 70%

De asemenea pot apărea și cazurile în care acuratețea maximă scade, dar în același timp acest lucru nu trebuie să reprezinte un aspect negativ dacă ceea ce se urmărește este reducerea erorilor de tip fals negativ sau fals pozitiv.

4.2 *Boostrap aggregation* (sau *Bagging*)

O întrebare importantă care survine în timpul creării unui ansamblu este “Cum s-ar putea folosi în mod cât mai eficient un set de date astfel încât antrenând mai multe modele cu același algoritm de clasificare pe aceleași proprietăți (structura, atribute, etc.), fiecare să învețe particularități diferite ale acestuia astfel încât atunci când toate sunt combinate într-un singur model, ansamblu, să contribuie la îmbunătățirea puterii de predicție și să nu producă toate același rezultat?”. Un răspuns la această întrebare l-ar putea reprezenta conceptul de *bootstrapping*.

Bootstrapping-ul presupune că dintr-o mulțime A se creează o mulțime A(prim) selectând câte o înregistrare pe rând în mod aleatoriu din mulțimea originală și introducând-o în cea finală. După ce înregistrarea a fost aleasă, este introdusă înapoi în mulțimea inițială și poate fi selectată în mod aleatoriu din nou.

Bagging-ul are la bază ideea de *bootstrapping*, în sensul că din mulțimea inițială se creează mai multe mulțimi de aceleași dimensiuni folosind această metodă. Pentru fiecare din mulțimile respective de înregistrări se antrenează câte un model folosind același algoritm de clasificare, iar agregarea acestora se face prin metodele de bază de tip ansamblu, vot majoritar pentru problemele de clasificare și media predicțiilor pentru cele de regresie. [14]

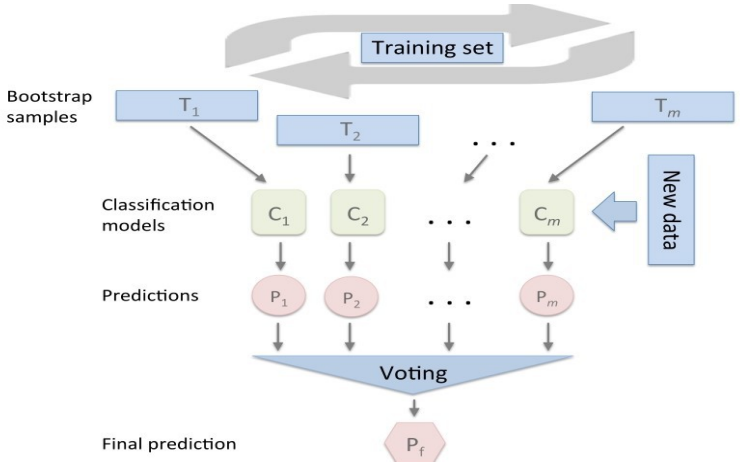

Fig.5 – Bagging[5]

Unul dintre algoritmii de clasificare care beneficiază cel mai mult dintr-o metodă de ansamblu de tip *bagging* este cel de tip arbore de decizie, deoarece orice modificare ce intervine asupra setului inițial de antrenament poate provoca modificări în arborele generat, chiar dacă nu a reprezentat o modificare majoră.

4.2.1 *Random Forest*

Este un algoritm de tip *bagging* unde fiecare model de clasificare este antrenat folosind arbori de decizie. Procesul aleatoriu este introdus în timp ce arborii de decizie sunt creați. În momentul alegerii unui nod de decizie, atributul cel mai important este selectat dintr-o mulțime aleatorie (cu o metodă precum câștigul maxim de informație) și nu din întreg setul de atribute. Astfel se elimină una din probleme întâlnite la arborii de decizie cu numeroase nivele, felul în care clasificatorul se poate modela după setul de antrenament, fenomenul de overfitting. Varianța este redusă, fiecare arbore antrenat nu prezice într-un mod extrem de precis, dar agregarea rezultatelor acestora se va apropia în medie de predicțiile așteptate. [12]

[5] Imagine preluata de la https://subscription.packtpub.com/book/big_data_and_business_intelligence

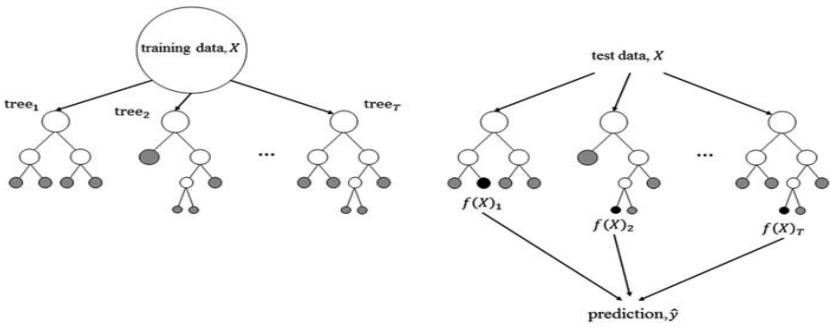

Fig.6 – Random Forest[6]

Un aspect care poate ajuta în optimizarea model antrenat folosind algoritmul Random Forest o reprezintă alegerea hiperparametrilor. Mai precis, alegerea numărului de arbori de decizie care va fi inclus în pădure și adâncimea maximă a fiecăruia dintre ei, mai precis numărul de atribute selectate aleatoriu ca noduri de decizie. Acești doi parametri reprezintă principalii parametri care pot influența cel mai vizibil rezultatele modelului obținut.

4.3 *Boosting*

Este o metodă iterativă, secvențială în care cu fiecare etapă se încearcă rezolvarea erorilor survenite în modelul anterior, scopul fiind de a mari acuratețea unui clasificator. Această metodă combină o serie de clasificatori slabi de învățare a căror predicții produse depășesc cu puțin pe cele în care deciziile s-ar lua in mod aleator.

Fiecare înregistrare, dată din setul de antrenament primește o pondere. Inițial, toate aceste ponderi sunt egale. Pe subsetul creat se antrenează un model folosind clasificatorul ales, apoi se calculează erorile de predicție a modelului antrenat. Fiecărei înregistrări incorect clasificate i se atribuie o pondere mai mare decât cea precedentă.

Fiecare astfel de ciclu este repetat, modelul final fiind reprezentat de votul majoritar cu ponderi a predicțiilor anterioare (sau suma cu ponderi pentru problemele de regresie).

Boosting-ul diferă de *bagging* prin faptul că funcționează într-o manieră secvențială, pe când *bagging*-ul funcționează în paralel, deoarece niciuna din mulțimile *bootstrapped* nu depinde una de alta ci modelele sunt antrenate pe acestea în mod independent unele de altele.

[6] Imagine preluata de la https://www.researchgate.net

4.3.1 AdaBoost

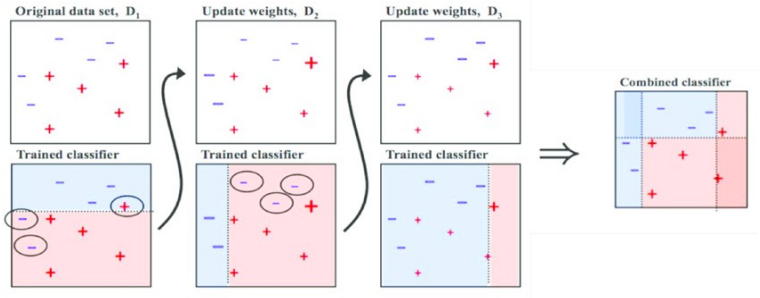

Fig.7 – AdaBoost[7]

Reprezintă un algoritm de tip *boosting*, funcționând pe principiile menționate, ecuația finală pentru clasificare fiind suma produselor dintre fiecare clasificator și ponderea acestora.

$$F_M(x) = sign(\sum_{m=1}^{M} \alpha_m f_m(x))$$

sign returnează 0 dacă suma produselor este mai mică decât 0 și 1 în caz contrar.

Se inițializează valorile inițiale ale ponderilor w[i]=1/N unde i=1..N și într-o structură iterativă, se antrenează câte un model $f_m(x)$ pe setul de antrenament folosind ponderile curente. Se calculează apoi eroarea pentru iterația curentă.

$$\varepsilon_m = \sum_{i=1}^{N} w_i I(y_i \neq f_m(x_i))$$

Se calculează coeficientul 'alpha', care arată cât de important este modelul curent pentru cel final. Cu cât predicțiile clasificatorului sunt mai bune, cu atât 'alpha' este mai mare.

$$\alpha_m = \frac{1}{2} log \left[\frac{1 - \varepsilon_m}{\varepsilon_m} \right]$$

Valorile țintă sunt din mulțimea {-1,1} și nu {0,1}, astfel când valoarea prezisă de model și cea așteptată sunt de semne opuse (fals pozitiv sau fals negativ), ponderea înregistrării va crește, iar când sunt de același semn, ponderea va scădea. Actualizarea aceasta are loc pentru a face clasificatorii să se concentreze asupra înregistrărilor care sunt dificil de clasificat.

$$w_i = w_i exp\left[-\alpha_m y_i f_m(x_i)\right]$$
$$w_i = \frac{w_i}{\sum_{t=1}^{N} w_{t=1}}$$

[7] Imagine preluata de la https://medium.com/diogo-menezes-borges

Se poate observa că ponderile sunt normalizate după calculul acestora (pentru ca suma lor să fie mereu 1). Motivul fiind acela că, o înregistrare care continuă să fie clasificată greșit va avea o pondere în continuă creștere și pondere extrem de mică pentru cazul contrar, ceea ce poate duce la instabilități numerice după un număr mare de iterații.

Se stochează "alpha" și modelul curent "$f_m(x)$" și se trece la următoarea iteratie (până se atinge numărul setat de modele de baza). [17]

5 Implementare

5.1 Corpus

Scopul lucrării este acela de a prezenta pașii care pot fi realizați atunci când se lucrează cu o problemă de învățare automată (din domeniul medical). Problema considerată în acest studiu este predicția cancerului la sân. Pentru unele seturi de date nu există informații atașate acestora legate de tipurile de date sau de semnificația fiecărui atribut în parte, de aceea am ales să abordez problema ca și cum nu există nicio informație referitoare la caracteristici și nici semnificația acestora nu este cunoscută. Setul de date folosit este "Breast Cancer Wisconsis (Diagnostic) Data Set", și poate fi descărcat de la adresa "https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29". Detaliile setului de date sunt redate mai jos, cu scop informativ.

```
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
id                         569 non-null int64
diagnosis                  569 non-null object
radius_mean                569 non-null float64
texture_mean               569 non-null float64
perimeter_mean             569 non-null float64
area_mean                  569 non-null float64
smoothness_mean            569 non-null float64
compactness_mean           569 non-null float64
concavity_mean             569 non-null float64
concave points_mean        569 non-null float64
symmetry_mean              569 non-null float64
fractal_dimension_mean     569 non-null float64
radius_se                  569 non-null float64
texture_se                 569 non-null float64
perimeter_se               569 non-null float64
area_se                    569 non-null float64
smoothness_se              569 non-null float64
compactness_se             569 non-null float64
concavity_se               569 non-null float64
concave points_se          569 non-null float64
symmetry_se                569 non-null float64
fractal_dimension_se       569 non-null float64
radius_worst               569 non-null float64
texture_worst              569 non-null float64
perimeter_worst            569 non-null float64
area_worst                 569 non-null float64
smoothness_worst           569 non-null float64
compactness_worst          569 non-null float64
concavity_worst            569 non-null float64
concave points_worst       569 non-null float64
symmetry_worst             569 non-null float64
fractal_dimension_worst    569 non-null float64
Unnamed: 32                0 non-null float64
dtypes: float64(31), int64(1), object(1)
```

Fig.8 – Detalii generale atribute

Din fig.8, se observă că setul de date este o matrice cu 569 linii și 33 de coloane. Fiecare înregistrare reprezintă o linie și fiecare atribut o coloană. Se poate observa că pentru fiecare

caracteristică există câte 569 de instanțe nenule (non-null), ceea ce înseamnă că nu există date lipsă în setul curent. Excepția o reprezintă ultimul atribut “Unnamed: 32”, a cărui valori sunt toate de tip null, de aceea va fi înlăturat înainte de antrenarea modelului.

Un alt aspect care atrage atenția îl reprezintă tipul de date al fiecărui atribut, toate sunt de tip float, deci sunt date de tip continuu, cu excepția primelor două: id și diagnosis. [1]

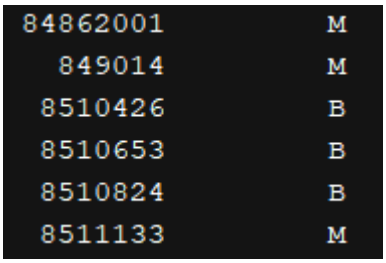

Fig.9 – Id și diagnosis

La afișarea numărului de instanțe unice pentru cele două caracteristici am obținut următoarele valori: 569 și 2. Pentru prima valoare, fiind de tip int și având un număr de înregistrări diferite egale cu numărul total de linii, se poate presupune că reprezintă un identificator unic pentru fiecare pacient/ înregistrare. Cea de a doua va reprezenta coloana de diagnostic, valorile țintă pentru fiecare linie (înregistrare), din mulțimea {'M'-malign,'B'-benign}.

Cunoscând aceste informații, se pot înlătura cele 2 coloane, 'id' și 'Unnamed: 32' deoarece nu sunt necesare, iar 'diagnosis' va fi tratat ca și output pentru antrenare și testare.

5.2 Generare seturi de date

Faptul că nu există date lipsă în acest set de date reprezintă un aspect pozitiv. Deoarece am dorit o abordare completă, în care să prezint și tratarea datelor lipsa, voi genera din acesta un alt set cu valori lipsa luând în calcul următoarele cazuri care pot fi întâlnite:

- valori lipsă in mod total aleatoriu
- valori lipsă în cadrul unui singur atribut
- valori lipsă cu dependențe (lipsesc în funcție de valorile unui alt atribut)

Pentru aceasta, am utilizat următoarea abordare:

- generare aleatoare; funcția implementată are următoarele argumente: probabilitatea ca o valoare să lipsească –coeficient_random (de exemplu, pentru un grad=5, există 1/5 șanse

ca o valoare să fie înlocuită), denumirea atributului de clasificare – classif_column (cel de output, ieșire căruia nu îi înlăturăm valorile), un argument prin care se specifică atributele în dreptul cărora se vor face modificările – atribut. Această metodă înlocuiește valorile inițiale prezente în setul de date cu NaN (Not a Number în python).

```
Total missing values for radius_mean are 13
Total missing values for texture_mean are 15
Total missing values for perimeter_mean are 13
Total missing values for area_mean are 8
Total missing values for smoothness_mean are 9
Total missing values for compactness_mean are 7
Total missing values for concavity_mean are 13
Total missing values for concave points_mean are 10
Total missing values for symmetry_mean are 12
Total missing values for fractal_dimension_mean are 4
Total missing values for radius_se are 12
Total missing values for texture_se are 12
Total missing values for perimeter_se are 10
Total missing values for area_se are 12
Total missing values for smoothness_se are 15
Total missing values for compactness_se are 8
Total missing values for concavity_se are 14
Total missing values for concave points_se are 14
Total missing values for symmetry_se are 16
Total missing values for fractal_dimension_se are 15
Total missing values for radius_worst are 13
Total missing values for texture_worst are 11
Total missing values for perimeter_worst are 15
Total missing values for area_worst are 15
Total missing values for smoothness_worst are 16
Total missing values for compactness_worst are 8
Total missing values for concavity_worst are 12
Total missing values for concave points_worst are 10
Total missing values for symmetry_worst are 15
Total missing values for fractal_dimension_worst are 14
(569, 31)
(292, 31)
```

Fig.10 – Numărul valorilor lipsă

5.2.1 Rezultatele obținute pentru generare aleatoare pe întregul set de date

Prima încercare a reprezentat-o generarea unui set aleatoriu, cu un coeficient aleatoriu 50 pe toate atributele, ceea ce a rezultat la ~2% din valorile de pe fiecare coloană să lipsească. Am încercat înlăturarea înregistrărilor cu date lipsa, dar, se poate observa că acesta nu este un caz favorabil pentru o astfel de abordare, numărul înregistrărilor fiind redus la aproape jumătate față de cel inițial, valorile lipsa neaparținând acelorași linii.

```
Acurracy on the initial dataset:0.947%
Acurracy on the dataset with eliminated rows:0.714%
Acurracy on the dataset with mean compute:0.935%
Acurracy on the dataset with mode compute:0.929%
Acurracy on the dataset with median compute:0.929%
Acurracy on the dataset with KNN compute:0.941%
```

Fig.11 – Acuratețile pe rezultat 1

Am ales un algoritm simplu pentru antrenarea modelelor și compararea preciziilor, KNN. Am antrenat un model pe setul de date inițial, iar acuratețea pe care am obținut-o pe modelul acesta a reprezentat valoarea față de care am încercat să mă apropii cu celelalte 5 metode. După cum se poate observa din fig.11, eliminarea înregistrărilor nu reprezintă o soluție optimă atunci când datele lipsă sunt prezente în cadrul a mai multe caracteristici. Dintre cele 3 metode simple de completare a datelor lipsă (medie, mod și mediană), media s-a apropiat cel mai mult de valoarea țintă. O explicație o poate reprezenta și faptul că multe dintre valori sunt compacte, de tip continuu din intervalul [0,1] sau alte intervale restrânse, valorile cu zgomot neavând un impact la fel de mare ca și pe intervale mai puțin restrictive.

Cele mai bune rezultate au fost obținute folosind KNN cu 5 cei mai apropiați vecini. Setul de date curent reprezintă și un mediu favorabil, neavând dimensiuni mari ca și număr de înregistrări și de atribute (la un număr mare de atribute, diferența dintre cel mai îndepărtat și cel mai apropiat vecin poate fi foarte mică).

5.2.2 Rezultatele obținute utilizând generarea pe un singur atribut

Pentru alegerea unui atribut am verificat relevanța acestora antrenând algoritmi de tip RandomForest. Am selectat un atribut care apărea mai des cu rezultate nefavorabile (*concavity_worst*) și unul care se află mereu în jumătatea superioară (*perimeter_mean*). Fiind un algoritm oarecum aleator în alegerea atributelor, l-am folosit mai mult cu scop orientativ, pentru a-mi forma o idee generală asupra importanței acestora și nu una conclusivă. Am ales două caracteristici, pentru a compara diferența între rezultate atunci când un atribut important (coloană) este înlăturat față de unul cu o contribuție mai puțin semnificativă.

Alegerea unei coloane specifice nu a produs diferențe foarte mari, dar așa cum era de așteptat, modelele antrenate cu atributul mai semnificativ lipsă au produs rezultate puțin mai slabe.

```
Acurracy on the initial dataset:0.947%
Acurracy on the dataset with eliminated rows:0.957%
Acurracy on the dataset with eliminated columns:0.888%
Acurracy on the dataset with mean compute:0.947%
Acurracy on the dataset with mode compute:0.918%
Acurracy on the dataset with median compute:0.923%
Acurracy on the dataset with KNN compute:0.947%
Acurracy on the dataset with RF compute:0.947%
```

Fig.12 – Acuratețile pe rezultat 2

Pentru cazul curent, am introdus două alte metode de calcul a datelor lipsă: eliminarea atributului și predicția valorilor lipsă folosind RandomForestRegressor (deoarece toate caracteristicile din setul de date sunt de tip continuu). Cu a două metodă îmi setez ca și valoare țintă coloana pe care vreau să o completez și antrenez modelul pe înregistrările în care este prezentă, prezicerile făcându-se pe cele lipsă.

Pe mai mulți algoritmi de învățare, atât KNN cât și RF au reușit să producă aceleași rezultate (sau extrem de apropiate) ca și pe setul de date inițial, urmate de completarea prin valoarea mediei. Eliminarea coloanei a obținut în cele mai slabe rezultate. Un aspect interesant îl reprezintă faptul că atât pe modele antrenate cu KNN cât și cu Bayes Naiv, se produc mici îmbunătățiri în predicție atunci când unele înregistrări sunt înlăturate.

La un număr mai ridicat de coloane cu valori lipsa (3), RF a produs rezultatele cele mai optime, urmat de KNN. Însă, o astfel de abordare nu e indicată, în special când numărul acestora crește, deoarece valorile înlocuite se modelează prea precis pe celelalte variabile. De aceea KNN pe un set de dimensiuni medii este mai recomandat.

5.3 Alegerea acurateții ca metrică potrivită

Un subiect important neabordat până în acest punct în cadrul lucrării o reprezintă alegerea metricii potrivite pentru comparație, mai exact acuratețea, precizia sau recall-ul.

În cadrul problemei curente de clasificare, mai exact clasificarea unui caz de cancer la sân ca fiind malign sau benign, este mai importantă descoperirea cazurilor de tip malign decât o acuratețe generală ridicată. Justificarea fiind faptul că este mai prioritară descoperirea cazurilor grave de cancer și mai urgentă, de aceea eticheta țintă pe care o vom urmări va fi 'M'-malign.

Astfel, definim cazurile maligne corect clasificate ca și PA (pozitive adevărate), înregistrări incorect clasificate ca fiind maligne cu PF (pozitive false) și înregistrările care au fost clasificate ca fiind benigne deși erau maligne cu NF (negative false).

Acuratețea reprezintă raportul dintre înregistrările corect clasificate și totalul acestora (NA+PA)/total. *Recall*-ul este PA/(PA+NF), mai exact cât la sută dintre cazurile maligne sunt identificate, iar precizia este PA/(PA+PF), mai exact cât la sută dintre cazurile pe care le-am identificat ca fiind maligne au fost corect identificate și nu erau cazuri de tip benign.

Se poate ajunge la concluzia că metrica cea mai importantă ar trebui să fie *recall*-ul astfel, dacă dorim identificarea tuturor cazurilor maligne, dar o astfel de abordare ar fi incorectă. Am

putea pur și simplu să eliminăm mare parte dintre cazurile care sunt etichetate ca fiind benigne la antrenament și cu siguranță modelul va prezice cu un recall de aproape 100% cazurile maligne, precizia fiind aproape de 0, dar un astfel de model este eronat, pacienții diagnosticați în mod incorect cu o formă severă de cancer ar trebui să suporte costuri suplimentare pentru investigații ulterioare în mod nejust. De aceea este importantă balansarea preciziei și a recall-ului și menținerea amândurora la un nivel ridicat în cadrul îmbunătățirilor aduse.

Pentru alegerea metricii astfel, după abordarea inițială cu tratarea datelor lipsa, am realizat că ar trebui să acord o atenție sporită asupra căreia mă axez, așa că în urmă a 100 de iterații, am făcut o medie pentru cele 3 metrici în cadrul celor 8 metode prezentate în capitolul 5.2 și am observat următoarele lucruri:

- pentru aproape fiecare metodă, precizia era cu ~0.02% mai mare decât acuratețea
- pentru fiecare metodă, *recall*-ul era cu ~0.08% mai mic decât acuratețea
- singura metodă care nu s-a încadrat celor două observații a reprezentat-o eliminarea coloanelor, care stabilisem deja că nu este cea mai optimă abordare

	Accuracy	Precision	Recall
Initial	0.93%	0.94%	0.86%
Elim rows	0.92%	0.94%	0.83%
Elim colm	0.91%	0.90%	0.84%
Mean	0.90%	0.93%	0.80%
Mode	0.90%	0.93%	0.81%
Median	0.90%	0.92%	0.81%
KNN	0.92%	0.94%	0.85%
RF	0.93%	0.94%	0.86%

Astfel am realizat că precizia și *recall*-ul sunt foarte strâns corelate de acuratețe și că îmbunătățirile aduse acesteia sunt direct proporționale cu îmbunătățirile aduse celorlalte două. Din acest motiv, am ales ca metrică folosită în cadrul lucrării, acuratețea, aceasta incapsulandu-le și pe restul și deoarece menținea simplitatea înțelegerii îmbunătățirilor aduse.

Un motiv pentru corelarea dintre cele 3 îl reprezintă și repartiția datelor, 60% dintre acestea sunt clasificate ca fiind benigne, iar 40% ca fiind maligne. Astfel, setul de date nu este foarte nebalansat, ceea ce nu ne forțează în a încerca să favorizăm una dintre cele două clase de

clasificare, axându-ne pe *recall*.

Dacă comportamentul acesta nu ar fi fost prezent și nu exista o legătură atât de strânsă între cele 3, aș fi urmat una dintre următoarele două situații:

- dacă costurile unor investigații ulterioare pentru cazurile clasificate ca fiind maligne ar fi fost extrem de mici, m-aș fi axat strict pe îmbunătățirea *recall*-ului, dar fără a neglija în totalitate precizia (prin eliminarea cazurilor benigne)
- dacă costurile nu ar fi fost mici, aș fi urmărit maximizarea unui scor F1 care reprezintă media armonică dintre precizie și recall: 2*P*R/(P+R); astfel se menține o balanță între cele două

5.4 Selectia atributelor

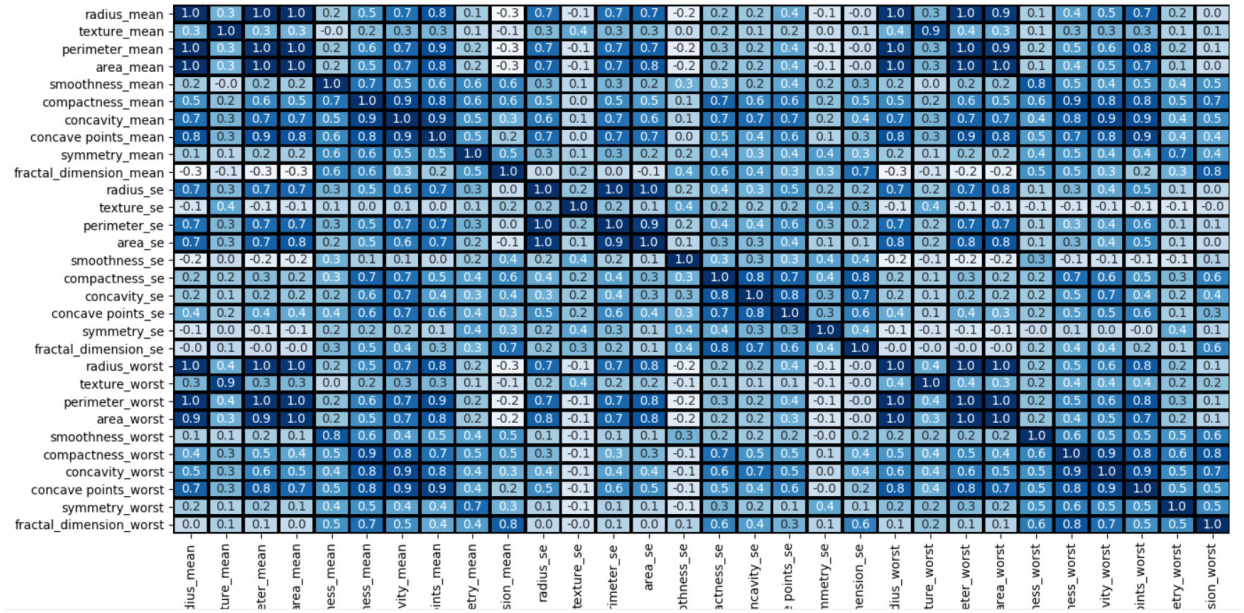

Fig.13 – Matricea de corelare

După tratarea datelor lipsă, este importantă o analiză detaliată a atributelor. De aceea, am început cu observarea corelării dintre acestea.[8] În fig.13, culorile cât mai închise reprezintă un coeficient de corelare directă tot mai mare între două caracteristici (-1.0 și 1.0 fiind corelările maxime, ca și valoare). Corelarea pozitivă presupune că atunci când una dintre valori crește și cealaltă urmărește aceeași tendință, pe când coeficientul negativ înseamnă că atunci când una crește, cealaltă scade, fenomen cunoscut ca și corelare inversă. Diagonala principală va avea corelare 1.0 întotdeauna, comparația făcându-se între două coloane de același tip.

Din imagine, se poate observa că valorile de același tip (_mean _se sau _worst) pentru radius, perimeter și area sunt puternic corelate între ele, de asemenea și valorile lor _mean cu cele _worst, mai exact, gradul lor de corelare este 1.0.

5.4.1 Selecția manuală a atributelor

Pentru înlăturarea manuală a atributelor, stabilesc o limită inferioară pentru corelarea maximă, pentru cele care depășesc limita respectivă, compar două câte două distribuțiile acestora pe cele două clase de diagnoză, în cazul prezent, “malign” și “benign”, mai exact, compar separarea graniței de decizie. De asemenea încerc să iau în calcul și corelarea dintre atributele rămase, cele alese manual, pentru a nu selecta din cazuri diferite două atribute puternic corelate.

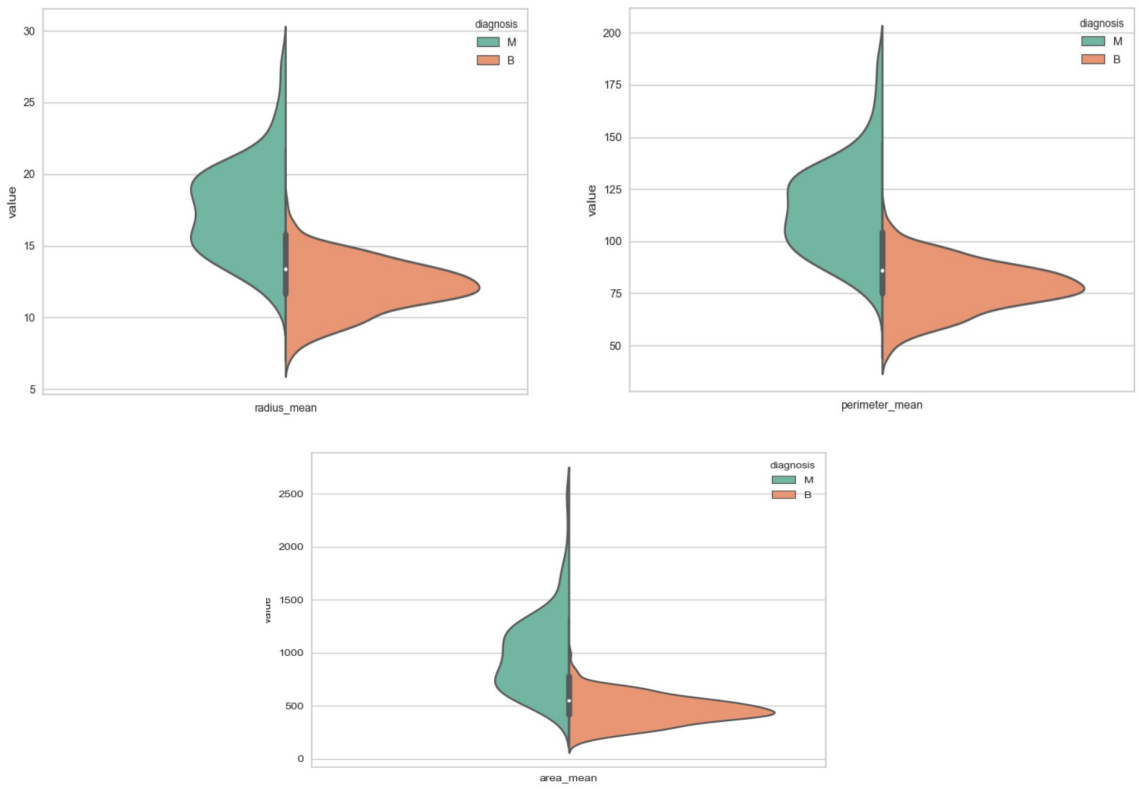

Fig.14 – Distribuții radius, perimeter, area_mean

Cele trei sunt puternic corelate, așa că pentru alegerea uneia dintre ele mă voi uită la distribuțiile acestora. Sunt foarte similare ca și distribuție, dar perimeter_mean pare să aibe mai multe valori care nu se suprapun, granițele fiind mai bine definite. Celelalte două le voi elimina.

Limita inferioară aleasă a fost de 0.9. Am continuat procesul de eliminare și comparație până când nu mai există nici o corelare mai mare sau egală cu limita inferioară.

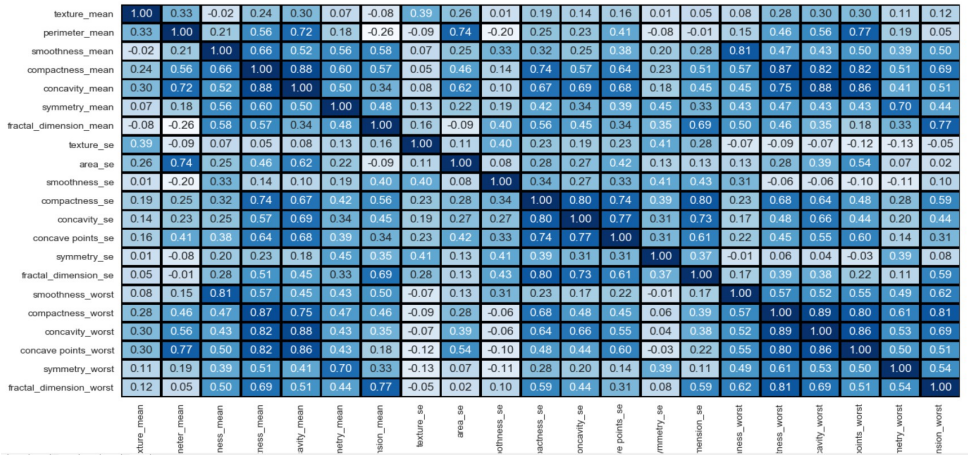

Fig.15 – Matricea de corelare finală

În fig.15 se poate observa că nu mai există nicio corelare mai mare decât 0.9 și că numărul de trăsături a fost redus de la 30 la 21. Pentru a vedea dacă s-au adus îmbunătățiri prin modificările făcute, am realizat o comparație pe două modele cu clasificator AdaBoost, antrenate și testate pe aceleași distribuții de seturi, dar pe atribute diferite (unul pe întregul set și celălalt pe cel nou).

Acuratețea a crescut de la 0.959% la 0.976% în urma selecțiilor manuale.

5.4.2 Determinarea unui număr optim de atribute

Un aspect important în alegerea caracteristicilor cele mai bune pentru antrenarea modelului îl reprezintă și alegerea numărului optim de atribute.

Inițial am testat ca și clasificator pentru estimare algoritmul RandomForest. Însă cum selecția atributelor se face în mod aleatoriu atunci când se creează arborii de decizie, numărul de atribute optime returnate în mod recursiv cu cross validare nu era constant, era diferit la fiecare pas. Așa că am ales ca și clasificator un algoritm de tip boosting, AdaBoost, numărul optim returnat fiind 15 de fiecare dată. Aceasta va fi valoarea pe care o voi folosi în cadrul tuturor celorlalte metode.

5.4.3 Comparația metodelor de selecție a atributelor

Am realizat o comparație a tuturor celor trei metode prezentate de selecție a atributelor: *filter*, *wrapper* și *embedded*. Am ales câte un algoritm pentru fiecare dintre aceste metode, selecția chi2 pentru filtrare, regularizare l1 (sau LASSO) pentru *embedded* și eliminarea recursiva a atributelor pentru *wrapping*. Comparația este realizată strict pe acestea, însă au fost adăugate rezultate și pentru alți algoritmi.

5.4.3.1 Rezultatele obținute pentru un număr “optim” predefinit de atribute

```
Initial accuracy:0.95906432748538018
No of features to select:  15
Attributes selected by chi2: Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'concavity_mean', 'radius_se', 'perimeter_se', 'area_se',
       'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst'],
      dtype='object')
Accuracy after chi2 selection:0.96491228070175448
Attributes selected by RFE: Index(['texture_mean', 'area_mean', 'compactness_mean', 'concave points_mean',
       'symmetry_mean', 'area_se', 'compactness_se', 'radius_worst',
       'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst',
       'concavity_worst', 'concave points_worst', 'symmetry_worst'],
      dtype='object')
Accuracy after RFE selection:0.96491228070175448
Attributes selected by LASSO: Index(['radius_worst', 'perimeter_worst', 'concave points_worst'], dtype='object')
Accuracy after LASSO selection:0.92397660818713458
```

Fig.16 – Acuratețile inițiale după selecția atributelor

Primul lucru care se poate observa e că metoda prin regularizare L1 a produs rezultatele cele mai slabe, iar numărul de atribute selectate este de doar 3. Celelalte două au avut număr predefinit de atribute care trebuie selectate (15, aflat anterior).

Metoda de la care așteptam rezultatele cele mai bune era cea de tip *embedded* și cum numărul de coloane selectat a fost atât de mic comparativ cu cel obținut inițial, am tras concluzia că există o problema cu felul în care a fost ales coeficientul alfa, de aceea am căutat să analizez cum pot ajusta valoarea respectivă în mod optim. Pentru aceasta am folosit cross validare cu diferite valori predefinite pentru alfa și am observat eroarea care se produce.

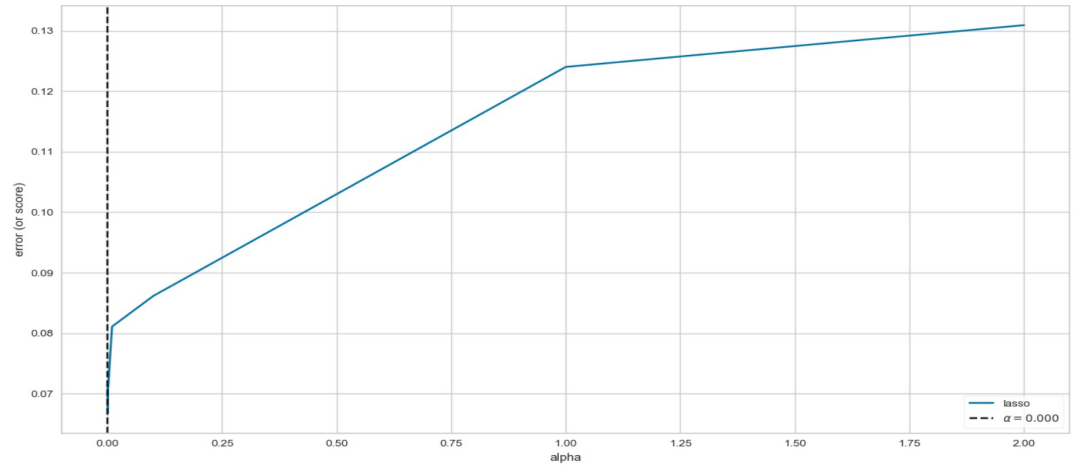

Fig.17 – Rata de eroare pentru coeficientul alfa

Se poate observa din graficul de la fig.17 că alfa trebuie ales cu atenție (pragul este foarte fin). Alfa trebuie să fie diferit de 0, de aceea am ales alfa=10^{-4}. După schimbarea făcută s-au obținut următoarele rezultate.

```
Attributes selected by LASSO: Index(['texture_mean', 'smoothness_mean', 'compactness_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'area_se', 'smoothness_se', 'compactness_se',
       'concavity_se', 'concave points_se', 'fractal_dimension_se',
       'radius_worst', 'texture_worst', 'area_worst', 'smoothness_worst',
       'concavity_worst', 'concave points_worst', 'symmetry_worst',
       'fractal_dimension_worst'],
      dtype='object')
Accuracy after LASSO selection:0.9766081871345029%
```

Fig.18 – Caracteristici selectate cu L1 regularizare

Numărul de atribute selectate a crescut de la 3 la 21 iar acuratețea de la 0.923% la 0.976%. Astfel, după ajustarea respectivă, toate cele trei metode au îmbunătățit rezultatele obținute pe întreg setul de caracteristici. Nu doar acuratețea reprezintă un aspect important în selecția atributelor, ci și timpul care este economisit prin procesul respectiv. Deci dacă precizia nu creștea, dar timpii de antrenare și testare scădeau, tot s-ar fi considerat că prin selecția acestora s-a adus o îmbunătățire modelului.

Am testat de asemenea și alte 4 metode existente în librăriile din Python, sklearn și mlxtend: f_classif, mutual_info_classif, ExtraTreesClassifier și SequentialFeatureSelector[8]. Primele două au produs rezultate apropiate de cel inițial, deci îmbunătățirea ar fi fost din punct de vedere al timpului, clasificatorul cu arbori nu a produs rezultate pozitive și erau aleatorii întodeauna. *Forward feature selection* a atributelor a produs același rezultat ca și întreg setul de caracteristici, deși numărul acestora a fost înjumătățit. Un avantaj al acestei metode îl reprezintă faptul că putem identifica cum și ce atribute sunt alese la fiecare pas, în schimb durata de selecție a fost mai mare decât la celelalte. Din aceste motive am ales să prezint rezultatele doar pentru cele 3 metode alese inițial.

5.4.5 Rezultatele obținute cu încercări multiple pentru fiecare metodă

Metoda care folosește regularizare L1 a găsit 21 de atribute optime față de 15 cum au fost alese inițial. Datorită acestei observații, am încercat să încerc printr-o metodă exhaustivă să aflu pentru fiecare abordare în parte numărul optim de caracteristici care ar fi putut fi selectat, deoarece valoarea aleasă era specifică eliminării secvențiale recursive cu cross validare.

Pentru aceasta, am implementat o funcție care pentru fiecare număr posibil de coloane, calculează acuratețea pe modelele antrenate cu fiecare metodă în parte. Valorile sunt atașate unei

[8] http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

liste, iar la final sunt reprezentate grafic.

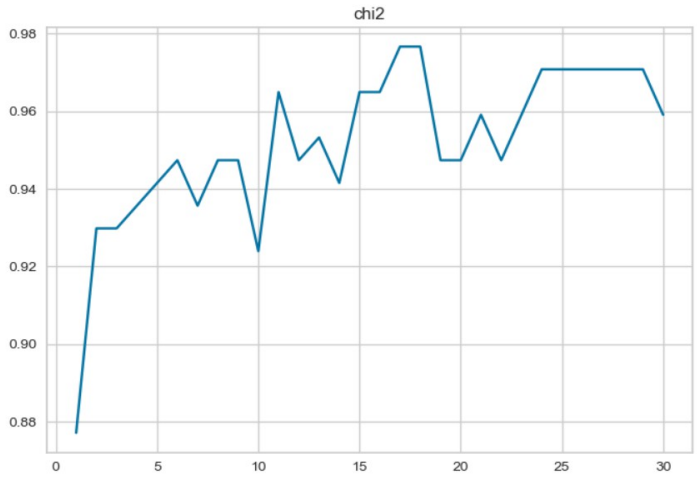

Fig.19 – Rata de acurateţe chi^2

În fig.19, sunt reprezentate rezultatele prin selecţia chi2. Se poate observa că la adăugarea a încă 2 atribute pentru selecţie, acurateţea ar creşte de la 0.964% la 0.976%.

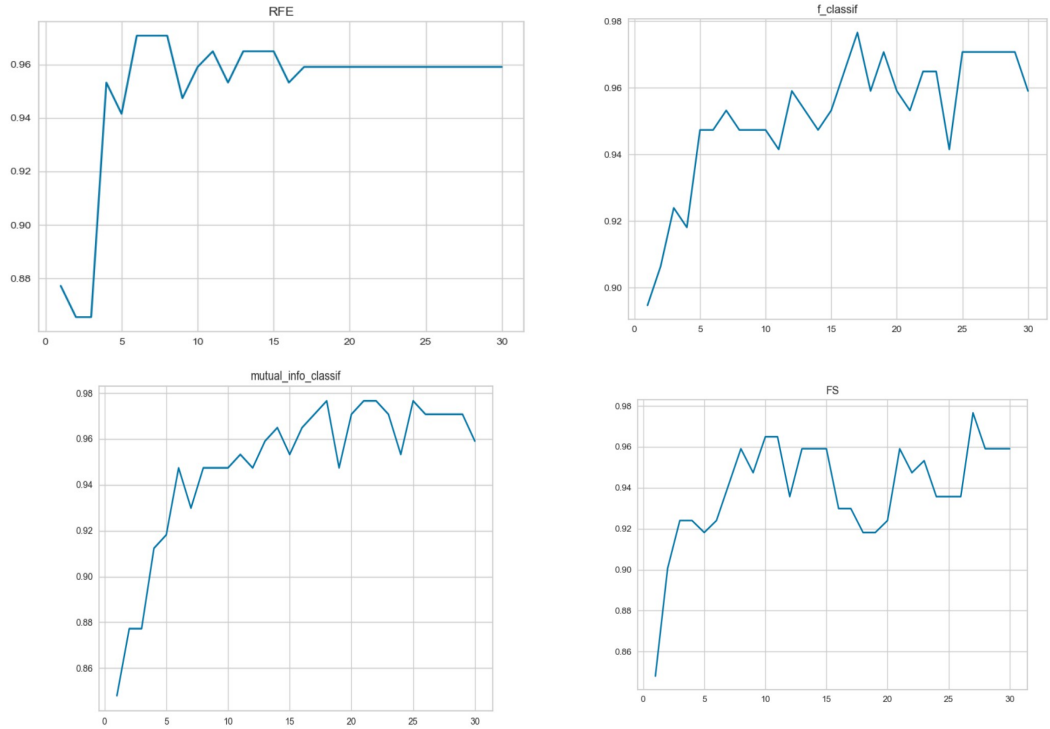

Fig.20 – Comparaţie acurateţi

Din figurile de mai sus se poate observa că rezultatele metodei de tip wrapping sunt mai consistente decât cele obținute de filtrare. Un număr mai mic de atribute este necesar pentru a atinge maximul global și orice valoare mai mare decât aceasta produce rezultate asemănătoare. De 3 ori mai puține caracteristici au fost necesare pentru a produce aproximativ același rezultat ca și metodele de filtrare, în schimb, dezavantajul acestora l-a reprezentat timpul computațional.

Pentru metoda de tip embedded nu am mai rulat procesul exhaustiv, acesta fiind echivalentul ajustării parametrului alfa prezentat la pasul 1.

O astfel de abordare nu este recomandată, fiind tot mai costisitoare cu cât dimensiunile setului de date crește. Am folosit-o pentru a face o comparație între cele două metode și pentru a vedea cât de mult se puteau optimiza rezultatele obținute la încecarea inițială.

5.5 Metode de tip ansamblu

Am considerat următoarele obiective în analiza metodelor de tip ansamblu:

- ajustarea hiperparametrilor pentru clasificatorul de tip RandomForest și obținerea unei valori medii precise
- analiza unor ansambluri simple (cu votul maxim) folosind algoritmi de bază
- găsirea unui ansamblu de modele necorelate care să îmbunătățească rezultatul maxim obținut până acum (cu RandomForest, AdaBoost și modelele simple)

Considerând că RandomForest și AdaBoost sunt metode de tip ansamblu, am inclus rezultatele acestora fără nicio ajustare în secțiunea curentă. Ajustările care se pot face pentru AdaBoost ar reprezenta cele aduse atributelor, prezentate anterior. În schimb, pentru RandomForest, se pot modifica parametrii acestuia. Valorile inițiale obținute pentru RandomForest și AdaBoost au fost de 0.951% (cu valorile standard) respectiv 0.959%.

5.5.1 Ajustare RandomForest

Algoritmul RandomForest din cadrul librăriei python “sklearn.ensemble”[9] are 2 parametri importanți care ajustați pot îmbunătăți rezultatele modelului antrenat.: n_estimators și max_features. n_estimators reprezintă numărul de arbori de decizie prezenți în ansamblu, iar max_features numărul de atribute care pot fi luate în calcul atunci când se introduce un nod de

[9] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

decizie. Voi căuta valori optime și pentru criteriul de calcul și adâncimea maximă a unui arbore.

Deoarece există un element aleatoriu în predicțiile cu un clasificator de tip RandomForest, am ales să antrenez un număr mai mare de modele pe aceleași setări, iar valoarea care va fi luată în calcul să o reprezinte media acestora.

Pentru selecția parametrilor am folosit o metodă exhaustivă, mai exact GridSearchCV[10]. Pe niște parametri specificați, predefiniti, se antrenează modelul pe fiecare combinație posibilă de parametri și se optimizează cu cross validare.[7]

```
parameters = {
    'n_estimators': [100, 200, 500, 1000],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [None, 3, 4, 5, 6, 7, 8, 9, 10],
    'criterion': ['gini', 'entropy']
}
```

Fig.21 – Parametri de optimizat

Pentru structura din fig.21 s-au obținut următoarele valori pentru parametri.

```
{'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 500}
```

Fig.22 – Valorile optime selectate pentru argumente

Primele 3 valori obținute sunt valorile standard pentru parametrii respectivi, deci nicio modificare nu se face pentru aceștia. Singura care se modifică este cea pentru n_estimators, mai exact numărul de arbori de decizie. Adăugarea a tot mai multor arbori de decizie nu va reduce niciodată din acuratețe, în schimb va încetini modelul. Deci singurul lucru care trebuie luat în calcul este costul computațional.

Astfel am antrenat câte 100 de modele pentru ambele valori, 500 și 1000 pentru n_estimators pentru a compara dacă dublarea numărului de arbori de decizie produce îmbunătățiri semnificative.

După o medie a predicțiilor, diferența de precizie între cele două setări a fost de 0.001%, mai exact 0.959% față de 0.96%, o îmbunătățire a acurateții mult prea mică pentru a justifica costul computațional care a fost dublat.

Cu tot cu selecția atributelor, RandomForest a putut fi adus la aproximativ aceeași precizie ca și cea de la rezultatele prezentate pe AdaBoost.

[10] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

5.5.2 Ansambluri cu vot maxim pe clasificatori simpli

Pentru partea de ansamblu de bază, simplu, am ales 3 clasificatori: Bayes Naiv, Arbori de decizie și KNN. Ceea ce am urmărit a fost ca indiferent de modul în care sunt selectate seturile de antrenament și de test, rezultatele ansamblului să fie mai mari sau egale cu rezultatele obținute de unul dintre cei trei clasificatori. În aproape toate situațiile lucrul acesta se întâmplă, acuratețea crescând semnificativ sau fiind egală cu cea mai bună predicție.

Fig.23 – Acuratețea pe ansamblu

Pentru o mai bună generalizare a acurateții, am memorat 500 de astfel de instanțe cu rezultatele lor și am făcut o medie pe acestea.

Fig.24 – Acuratețile medii

5.5.3 Ansambluri cu vot maxim

Ultima încercare a reprezentat-o adăugarea a încă două modele la structura ansamblului deja existent. Ce am încercat să observ a fost comportamentul noului ansamblu atunci când adaug două ansambluri, RandomForest și AdaBoost și să arăt că așezând mai multe nivele de ansambluri poate aduce îmbunătățiri, dar care nu justifică neapărat și costurile/ complexitatea.

Fig.25 – Rezultatele medii pe ansamblul complex

Rezultatul a fost cu puțin îmbunătățit față de mediile celor mai buni doi clasificatori. Într-un astfel de caz, luând în calcul timpul de antrenament și structura ansamblului, utilizarea ultimului clasificator în mod separat ar fi mai potrivită. Rezultatul poate fi în schimb îmbunătățit printr-o selecție potrivită a membrilor ansamblului, in urma unei analize detaliate a relațiilor dintre aceștia.

6 Concluzii

Învățarea automată este un domeniu vast, iar abordările și tehnicile existente sunt numeroase. Am reușit să abordez doar câteva din acestea și în ciuda unor rezultate pozitive, nu

neg că tehnicile și algoritmii folosiți pot fi înlocuiți de alții mai optimi pe tema curentă, pe setul de date prezentat.

Lucrarea a avut ca scop crearea unui punct de intrare, o prezentare a aspectelor de baza din învățarea automată, iar în urma aprofundării acestora am dorit să le aplic practic pe un subiect de interes. Tehnicile, pașii executați în cadrul analizei din această lucrare pot și de cele mai multe ori, trebuie, să fie urmați și în cadrul altor probleme care pot fi tratate folosind învățarea automată.

Rezultatele obținute sunt promițătoare, dar insuficiente pentru un caz din domeniul medical. Există deja o oarecare reticență față de sistemele automate în domeniul acesta din cauza naturii de “black box” a acestora, și pentru a fi luat în calcul un astfel de sistem, mașina ar trebui să reproducă ceea ce un om poate face deja, cu o acuratețe mai mare sau să poată ușura într-un fel munca medicului.

Lucrarea prezentă are un scop introductiv, este o prezentare a conceptelor care formează fundația învățării automate, o prezentare care poate facilita studii ulterioare.

7 Bibliografie

[1] Manish Kumar, Basic machine learning with cancer
https://www.kaggle.com/gargmanish/basic-machine-learning-with-cancer

[2] Scikit-learn https://scikit-learn.org/stable/

[3] Rahul C. Deo. Machine Learning in Medicine, Circulation 132 (20), 1920-1930, 2015
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5831252/

[4] Mathworks, Machine learning
https://www.mathworks.com/discovery/machine-learning.html

[5] Seaborn https://seaborn.pydata.org/index.html

[6] Valeria Fonti, Feature selection using LASSO, 2017
https://beta.vu.nl/nl/Images/werkstuk-fonti_tcm235-836234.pdf

[7] Will Koehrsen, Hyperparameter tuning for Random Forest
https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

[8] Heatmap customization using Seaborn
https://python-graph-gallery.com/91-customize-seaborn-heatmap/

[9] EliteDataScience, tutorial seaborn https://elitedatascience.com/python-seaborn-tutorial

[10] Alvira Swalin, Handling missing data
https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4

[11] Zhongheng Zhang. Missing data imputation: focusing on single imputation, Ann Transl Med. 4(1): 9, 2016
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4716933/

[12] Niklas Donges, Random Forest algorithm
https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

[13] Tavish Srivastava, Introduction to ensemble learning
https://www.analyticsvidhya.com/blog/2015/08/introduction-ensemble-learning/

[14] Dietterich, T.G. Machine Learning (2000) 40: 139.
https://doi.org/10.1023/A:1007607513941

[15] Learn for Master, chi-square for feature selection
http://www.learn4master.com/machine-learning/chi-square-test-for-feature-selection

[16] Renu Khandelwal, L1 regularization
https://medium.com/datadriveninvestor/l1-l2-regularization-7f1b4fe948f2

[17] AdaBoost tutorial
https://www.freetutorials.eu/ensemble-machine-learning-in-python-random-forest-adaboost-1/