UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

PUBG

propusă de

Tudor Manoleasa

Sesiunea: iulie, 2020

Coordonator științific

Croitoru Eugen

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

PUBG

Tudor Manoleasa

Sesiunea: iulie, 2020

Coordonator științific

Croitoru Eugen

Avizat,

Îndrumător lucrare de licență,

Croitoru Eugen.

Data: ..... Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Manoleasa Tudor** domiciliat în **Iasi**, născut la data de **03 martie 2020**, identificat prin CNP **1980303226746**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2020, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **PUBG** elaborată sub îndrumarea domnului **Croitoru Eugen**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

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

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

Data: ..... Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **PUBG**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

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

Absolvent **Tudor Manoleasa**

Data: ..... Semnătura: .....

Cuprins

Introducere

Player Unknown BattleGrounds, cunoscut drept PUBG, este un joc de lupta de tip arena (eng. *Battle-Royale*) in care un numar de 100 de jucatori concureaza unii impotriva celorlalti iar castigator este acela care "ramane ultimul in viata". Desigur, definitia castigatorului se poate extinde si la echipe, jocul permitand meciuri tot de 100 de jucatori, dar impartiti in echipe de 2, 3 sau 4 coechipieri. In cazul acesta, ultima echipa ramasa in viata este castigatoare. Un meci se desfasoara pe o harta unde participantii isi aleg locatia de start. Acestia trebuie sa gaseasca arme cu care sa ii invinga pe ceilalti. Mai multi jucatori pot avea aceeasi locatie de start, fapt care accelereaza dinamica jocului, unii pierzand chiar din primele minute. Pe parcursul meciului, participantii pot cauta arme mai bune, kit-uri medicale pentru a-si creste viata inapoi la 100 daca au fost atacati, vehicule pentru a se deplasa mai repede pe harta, obiecte care le pot creste acuratetea armelor si multe altele. La prima vedere, elementele care ar putea prezice locul pe care un jucator termina meciul sunt cele enuntate mai sus. Totusi, PUBG este un joc care mizeaza foarte mult pe noroc, cei mai buni jucatori putand fi eliminati aproape involuntar de catre altii. Din acest punct de vedere, predictia reprezinta o adevarata provocare pentru cel care incearca sa gaseasca tipare explicative (eng. *patterns*) ale statisticilor jucatorilor. Se impune astfel o analiza exploratorie a datelor pentru a avea o intelegere (eng. *insight*) mai buna asupra lor. Aceasta analiza ofera mediului competitiv diferite repere si ii ajuta atat pe profesionisti, cat si pe novici sa inteleaga mai bine utilitatea pe care anumite actiuni o ofera. PUBG are in prezent rangul de *joc-sportiv* (eng. *e-Sport*), multe persoane fiind sponsorizate financiar sa se antreneze pentru ca mai apoi sa participe la competitii unde premiile in bani depasesc 1.000.000$. Popularitatea, expunerea la un public variat, caracterul competitiv-sportiv si interesul firmelor de a sponsoriza jucatori sustin dorinta de a cauta modele explicative ale jocului, implicatiile financiare fiind evidente.

Capitolul 1

Planul de lucru

In cadrul platformei Kaggle [1] se regaseste o competitie deschisa numita ***"PUBG Finish Placement Prediction"*** [2], ce are drept scop predictia locului pe care un jucator va termina un meci, in functie de diferiti parametri din cadrul partidei. Intrucat exista mai multe moduri de joc (*solo*, *duo*, *echipa de 3*, *echiap de 4*), seturile de date de care dispun contin statistici pentru toate tipurile enuntate intre paranteze. Ma voi concentra doar asupra meciurilor *solo* (eng. *single-player*), incercand sa prezic cine va ajunge in primii 10 si cine nu. Atat setul de antrenament original cat si cel de test original contin milioane de interogari. Am ales sa le esantionez pe ambele, construind 2 seturi auxiliare, fiecare a cate 80.000 de intrari si 29 (28 + coloana de iesire) respectiv 28 de coloane.

1.1 Metodologia CRSIP-DM

Metodologia pe care o voi folosi la acest proiect este ***Cross-Industry Standard Process For Data Mining(CRSIP-DM)***, standardul din industrie in materie de proiecte ce vizeaza lucrul cu date. Modelul surpinde ciclul de viata al unui produs analitic ca al meu, nefiind foarte restrictiv cu privire la etapele pe care trebuie sa le urmez.

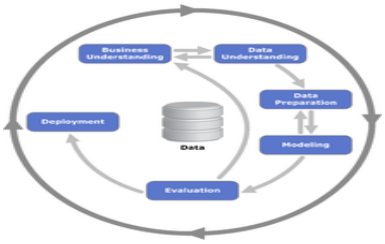


Figura 1.1: Etapele CRSIP-DM

Dupa cum se poate observa, exista sageti duble intre anumite procese, indicand posibilitatea de revenire la etapa anterioara si de reluare a acesteia. Metodologia poate fi considerata costisitoare, dar ofera avantajul esential: un proiect cu o acuratete mult mai mare. CRISP-DM face, asadar, un compromis (eng. *trade-off*) cost-acuratete.

Prezint acum o scurta descriere a etapelor si cum au fost ele integrate de mine:

- BUSINESS UNDERSTANDING: Care este scopul?(predictia top 10), Cui se adreseaza proiectul?(mediului competitiv), Care sunt asteptarile?(un model cu o acuratete cat mai mare), Pot aplica tehnici de invatare automata?(Da, atat invatare supervizata cat si ne-supervizata)

- DATA UNDERSTANDING: Cu ce fel de date lucrez? Imi ofera ele o imagine de ansamblu asupra problemei pe care vreau sa o rezolv? In cazul de fata, toate atributele sunt continue iar o mare parte din ele par sa fie relevante pentru un posibil model. De asemenea, numele atributelor sunt foarte clare (damageDealt, kills, heals etc.) iar in lipsa unei descrieri, rolul lor ar putea fi dedus usor.

- DATA PREPARATION: Aceasta etapa implica procesele de reducere a dimensiunilor (eng. *Dimensionality Reduction*), selectie a atributelor esentiale (eng. *Feature Selection*) si scalare (eng. *Feature Scaling*). Aici am aplicat tehnici cunoscute precum analiza in perechi a corelatiilor (eng. *pair-wise correlation*), calculul scorurilor atributelor prin *Random Forest* [3], scalare si normalizare printr-o transformare non-liniara (Yeo-Johnson) si altele descrise in sectiunile urmatoare.

- MODELING: Modelarea are la baza aplicarea unor algoritmi de invatare automata pe datele pregatite la etapa anterioara. In cazul acestui proiect, am folosit atat algoritmi nesupervizati (*k-Means, DBSCAN, EM-GMM*) cat si supervizati (*Naive Bayes, Regresie Logistica sau Random Forest*).

- EVALUATION: Evaluarea este primordiala in orice proiect de invatare automata. Metricile folosite aici ne spun daca modelele antrenate sunt cele bune sau am gresit, caz in care ne intoarcem la prima etapa, cea de Business Understanding. Amintesc aici doar cateva dintre principalele masuratori utilizate: *Precizie, Senzitivitate, ROC, acuratete echilibrata*.

- DEPLOYMENT: Refactorizarea codului, ultimele modificari aduse(eng. *last touch*) si livrarea produsului final.

1.2 Planul analizei exploratorii

Intr-o prima faza, am realizat "curatarea" datelor (eng. *data cleansing*) pentru ca mai apoi sa aplic tehnici de invatare nesupervizata asupra lor. Motivatia este aceea ca in majoritatea proiectelor de analiza se pot identifica sabloane (eng. *patterns*) ce releva informatii necunoscute la prima vedere. Deoarece sunt multe atribute de intrare, a fost nevoie mai intai de o reducerea a dimensionalitatii la 3D, pentru a putea vizualiza rezultatele obtinute de invatarea nesupervizata. Algoritmul nesupervizat care explica cel mai bine datele mele este *DBSCAN*, o metoda de clusterizare foarte cunoscuta, robusta si care se pliaza pe clustere cu densitati mari.

Dupa ce *DBSCAN* a impartit setul de antrenament in 3 grupari, am hotarat sa analizez in detaliu clusterul care in medie avea cel mai slab loc pe care jucatorii l-au ocupat la finalul unui meci. Evident, in acel cluster exista putini competitori care sa fi terminat in top 10. In felul acesta, reusesc sa fac o bipartitionare a datelor, separand clusterul cu o medie slaba de celelalte doua.

Prima partitie, cea slaba, e formata acum dintr-un set de observatii foarte inegal (eng. *unbalanced set*). Aici se regasesc multi jucatori din afara top 10 si putini din top 10. Atunci cand am incercat sa construiesc un arbore de decizie pt clasificare, acesta imi clasa gresit aproape toate etichetele pozitive (competitorii din top 10), dar corect pe cele negative (in afara top 10). Cu alte cuvinte, se producea fenomenul de *underfitting* pe eticheta pozitiva. Problema a fost rezolvata prin *boosting*, mai exact algoritmul *Adaboost*, care a produs o acuratete echilibrata (eng. *balanced accuracy*) buna.

A doua partitie contine un set mai echilibrat, motiv pentru care am aplicat tehnici de invatare supervizata specifice: *Regresie Logistica, Random Forest sau Naive Bayes Gaussian*. Cea mai interesanta metoda a fost un hibrid dintre *Naive Bayes Gaussian* si *EM-GMM*. De altfel, aceasta suprinde cea mai buna acuratete echilibrata la testare. Alegerea unui singur algoritm de clasificare este imposibila intrucat fiecare dintre cei enumerati mai sus se comporta specific unui anumit task. Spre exemplu, daca as dori sa maximizez numarul de jucatori prezisi corect ca fiind in top 10, alegerea potrivita este hibridul dintre *Naive Bayes Gaussian* si *EM-GMM*, dar daca vreau sa maximizez numarul de jucatori prezisi corect ca **nefiind** in top 10, aleg ceilalti algoritmi. In concluzie, utilizatorul acestui proiect poate alege dintr-o gama variata de algoritmi pe acela care se potriveste cel mai bine cu ce vrea sa obtina, avand drept ghid numeroase metrici de evaluare.

1.3 Masuratori statistice

Asa cum am mentionat mai sus, scopul lucrarii de fata nu este de a gasi un algoritm general si robust pentru orice sarcina de lucru, ci mai degraba un ansamblu de metode care sa ofere o viziune clara asupra rezultatelor pe care vrem sa le obtinem. Claritatea este data de *metricile de evaluare* pe care le voi descrie in continuare:

1.3.1 Matricea de confuzie

Matricea de confuzie (eng. *Confusion Matrix*) este un tabel 2x2, realizat pe setul de test, in care sunt condensate 4 statistici relevante pentru calculul altor metrici. Poate fi vazut drept un tabel auxiliar ce sta la baza oricarei evaluari. Elementele din matrice reprezinta:

- TP (ENG. *True Positives*): Numarul de intrari clasificate pozitiv (in top 10) care sunt intr-adevar pozitive.
- FP (ENG. *False Positives*): Numarul de intrari clasificate pozitiv (in top 10) care in realitate sunt negative.
- FN (ENG. *False Negatives*): Numarul de intrari clasificate negativ (nu sunt in top 10) care in realitate sunt pozitive.
- FN (ENG. *True Negatives*): Numarul de intrari clasificate negativ (nu sunt in top 10) care sunt intr-adevar negative.

		Realitate		
		Pozitiv	Negativ	Total
Predictie	Pozitiv	TP	FP	$TP + FP$
	Negativ	FN	TN	$FN + TN$
	Total	$TP + FN$	$FP + TN$	N

1.3.2 TPR&TNR, PPV&NPV

Cele 4 metrici din titlul acestei sub-sectiuni sunt calculate cu ajutorul statisticilor enumerate anterior. Luate pe rand, fiecare inseamna:

1) ***TPR*** = dintre toate intrarile care sunt in realitate pozitive, ce procent sunt prezise corect ca fiind pozitive? ***TPR*** este acronimul de la *True Positive Rate* si mai este

cunoscut in literatura de specialitate drept ***senzitivitate*** (eng. *sensitivity or recall*). Formula de calcul este:

$$TPR = \frac{TP}{TP+FN}$$

2) ***TNR*** = dintre toate intrarile care sunt in realitate negative, ce procent sunt prezise corect ca fiind negative? ***TNR*** este acronimul de la *True Negative Rate* si mai este cunoscut in literatura de specialitate drept ***specificacitate*** (eng. *specificity*). Formula de calcul este:

$$TNR = \frac{TN}{TN+FP}$$

3) ***PPV*** = dintre toate intrarile prezise ca fiind pozitive, ce procent sunt cu adevarat pozitive? ***PPV*** este acronimul de la *Positive predicted value* si mai este cunoscut in literatura de specialitate drept ***precizie*** (eng. *precision*). Formula de calcul este:

$$PPV = \frac{TP}{TP+FP}$$

4) ***NPV*** = dintre toate intrarile prezise ca fiind negative, ce procent sunt cu adevarat negative? ***NPV*** este acronimul de la *Negative predicted value*. Formula de calcul este:

$$NPV = \frac{TN}{TN+FN}$$

Se poate observa ca perechile TPR&TNR, PPV&NPV sunt oarecum redundante la nivel de logica. *Specificacitatea* o pot privi ca o *senzitivitate* pe etichete negative iar *NPV*-ul nu este altceva decat *precizia*, dar pe etichete negative. Modulul *scikit-learn*, pe care l-am folosit in cadrul proiectului, tine cont de aceste redundante la apelul anumitor functii.

1.3.3 Acuratetea

Acuratetea este probabil cea mai simpla metrica de evaluare, folosita drept prima-alegere (eng. *first-choice*) atunci cand se doreste o evaluare rapida a unui algoritm:

$$ACC = \frac{TP+TN}{TP+FP+FN+TN}$$

1.3.4 Acuratetea echilibrata

Acuratetea echilibrata (eng. *balanced accuracy*) este o metrica, derivata de la acuratetea de baza, definita drept media senzitivitatilor obtinute pe cele doua etichete de output. Formula este:

$$BACC = \frac{senzitivitate + specificacitate}{2}$$

Avantajul mare este acela ca functioneaza bine pe seturi dezechilibrate (eng. *imbalanced sets*), asa cum e cazul acestei lucrari.

1.3.5 Graficul ROC

Graficul *Receiver Operating Characteristic* este una dintre cele mai cunoscute tehnici de comparare a unor algoritmi supervizati de invatare automata in vederea stabilirii celui "mai bun". Este bidimensional, pe prima axa aflandu-se valori ale FPR (eng. *False Positive Rate*), tradus ca 1-specificacitatea, iar pe a doua axa avand valori ale senzitivitatii. Intervalul de valori pe ambele axe este, in mod evident, [0, 1].

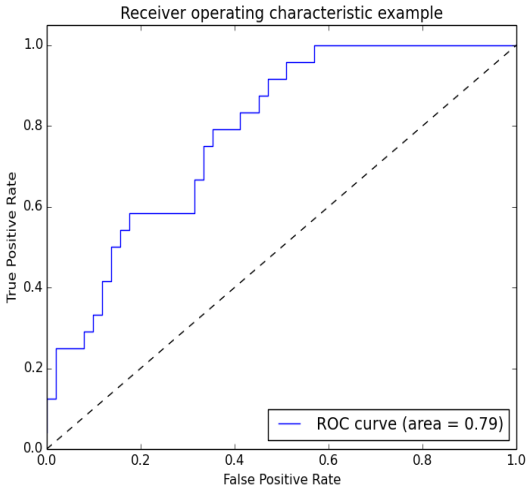


Figura 1.2: Exemplu de grafic ROC

Curba zimtata albastra surprinde diferite perechi de valori *(FPR, TPR)* care sunt obtinute din mai multe matrici de confuzie ale aceluiasi algoritm. Spre exemplu, clasificatorul *Naive Bayes* categorizeaza o intrare (eng. *sample*) ca fiind pozitiva cu o probabilitate anume. Am putea impune diferite praguri (eng. *tresholds*) pe care intrarile (eng. *the samples*) trebuie sa le depaseasca pentru a fi clasate drept pozitive. Spre exemplu, daca $P(+) = 80\%$ pentru o intrare iar $treshold = 85\%$ atunci intrarea va fi clasificata

negativ. Pentru toate tresholdurile generate, obtinem matrici de confuzie diferite a caror *FPR si TPR* constituie un nou punct pe grafic. Dupa ce am epuizat numarul de praguri, unim punctele de pe grafic pentru a obtine curba zimtata albastra. Acelasi lucru il putem face si cu un alt algoritm, obtinand o a doua curba zimtata si urmand ca la final sa calculam ariile suprafetelor cuprinse de cele doua curbe. Empiric, aria mai mare este specifica clasificatorului mai bun.

Ca observatie, linia punctata marcheaza locurile unde *TPR = FPR*. De asemenea, daca se doreste aflarea treshold-ului ideal pentru un algoritm, utilizatorul ar trebui sa urmareasca punctul curbei care se apropie cel mai mult de coltul stanga-sus al imaginii. In acel colt, senzitivitatea este maxima, *TPR = 1*, iar FPR este minim, *FPR = 0*.

1.3.6 Scorul Silhouette

Spre deosebire de invatarea supervizata unde metricile sunt calculate in functie de acuratetea predictiilor facute pe coloana iesire, la analiza nesupervizata acest lucru devine imposibil, eticheta de *output* fiind inexistenta. In plus, scopul algoritmilor nesupervizati este de a gasi sabloane explicative cu privire la datele de care dispunem. Prin urmare, metrica ideala ar trebui sa masoare gradul de separare a partitiilor gasite, similaritatea sau diferentele componentelor din cluster etc.

Scorul Sillhouette este cel pe care l-am folosit pe tot parcursul proiectului intrucat cuprinde cele 2 proprietati ideale de mai sus. O parte din avantaje si dezavantaje sunt:

- Scor usor de interpretat, curpins intre [-1,1], unde -1 semnifica clusterizare incorecta, 1 clusterizare ideala iar 0 marcheaza partitii care se suprapun
- Un scor ridicat indica clustere dense si bine separate
- Scorul e de obicei mai mare pentru clustere convexe si mai mic pentru alte tipuri de clustere, precum cele bazate pe densitate

$$s = \frac{b-a}{max(a,b)},$$

$$TotalScore = \frac{\sum_{i=1}^{n} s_i}{n}, \text{ n = nr total de observatii}$$

b = distanta medie dintre o observatie si toate celelalte observatii din cel mai apropiat cluster

a = distanta medie dintre o observatie si toate celelalte observatii din acelasi cluster

Capitolul 2

Intelgerea datelor si pregatirea lor

Am la dispozitie doua seturi de date in format **CSV**, *train.csv* si *test.csv*. Primul semnifica setul de antrenament iar al doilea setul dat spre testare. Primul pas in vederea unei analize exploratorii il reprezinta intelegerea datelor si a tipurilor acestora. Din punct de vedere ierarhic, putem imparti datele in 2 tipuri, fiecare a cate alte 2 subtipuri. Imaginea urmatoare suprinde cel mai bine partitionarea despre care voi vorbi in continuare:

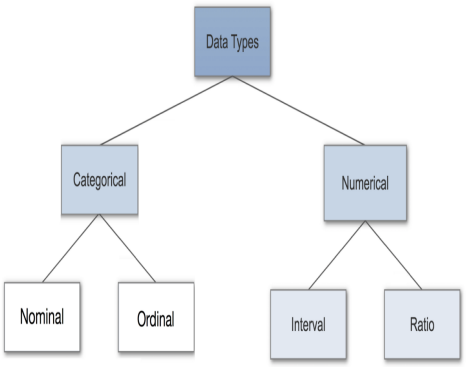


Figura 2.1: Tipuri de date

Categorical Data (rom. *date categoriale*) - reprezinta caracteristici, precum genul unei persoane sau limba vorbita de aceasta. Datele categoriale pot lua si valori numerice (ex. 1-limba engleza, 2-limba romana), dar fara a avea vreun sens matematic.

1) ***Nominale*** - valori discrete care pot fi privite drept etichete (eng. *labels*) (ex. sexul unei persoane)

1) ***Ordinale*** - aceeasi semnificatie ca la cele nominale, diferenta fiind ca ordinea conteaza. (ex. satisfactia unui client pe o scara de la 1 la 10)

Numerical Data (rom. *date numerice*) - pot fi discrete, deci numarabile, sau continue, adica masurabile. Datele numerice sunt cele mai intalnite tipuri in practica.

Voi descrie acum informatiile pe care le-am obtinut in urma analizei celor 2 seturi de date *train.csv* si *test.csv*.:

- 80.000 de observatii per set
- 21 de coloane numerice continue per set
- 1 coloana categoriala nominala per set
- 3 coloane cu valori unice per set
- 1 coloana numerica continua ce semnifica atributul de iesire (evident, aceasta nu se regaseste in *train.csv* intrucat trebuie prezisa)

2.1 Operatii de preprocesare

Intrucat multe dintre coloanele seturilor de date sunt irelevante, o mare parte dintre observatii nu corespund cerintelor acestui proiect iar in celulele care alcatuiesc tabelele **CSV** se regasesc si valori lipsa sau nedefinite, este nevoie de o "curatare" (eng. *data cleansing*). In acest sens, am hotarat sa ***elimin intrarile care au pe cel putin o celula in care valoarea lipseste***. De asemenea, imi pastrez doar observatiile care sunt de tipul ***solo sau solo-fpp***, meciurile de echipa nefiind de interes. Totodata, scot coloanele care din punct de vedere logic nu imi ofera informatii relevante. Ele sunt fie atribute specifice echipelor, fie atribute cu caracter unic neinformativ. O alta problema e prezentata de coloana de iesire a carei valori sunt subunitare, motiv pentru care e nevoie de o conversie.

```
df = pd.read_csv(r"C:\Users\Tudor\Desktop\Licenta2020\train.csv")
new_df = df.dropna()
new_df = new_df[new_df.columns][(new_df['matchType'] == 'solo-fpp')
                                | (new_df['matchType'] == 'solo')]
new_df = new_df.drop(["Id", "groupId", "matchId", "killPoints",
"matchType", "numGroups", "rankPoints", "teamKills", "DBNOs",
"maxPlace", "revives", "vehicleDestroys", "winPoints", "assists"],
```

```
axis = 1)
new_df['winPlacePerc'] = 100 -
                          (new_df['winPlacePerc'] * 100).astype('int32')
new_df.drop(new_df[(new_df['winPlacePerc'] < 1) |
            (new_df['winPlacePerc'] > 100)].index, inplace = True)
```

2.2 Eliminarea valorilor aberante

In descrierea seturilor de pe *Kaggle* se mentioneaza ca pentru coloanele *'damageDealt', 'longestKill', 'walkDistance' si 'weaponsAcquired'* au fost observate si valori aberante, datorate *hacker-ilor* (rom. trisori). Pentru a elimina aceste situatii folosesc metoda ***intervalului dintre cuantile***.

Prima data, pentru fiecare coloana enumerata mai sus, se calculeaza diferenta dintre cunatila 3 (granita pana unde se afla 75% din date) si cuantila 1 (granita pana unde se afla 25% din date) IQR=Q3-Q1. Mai apoi construiesc captele de interval : [Q1-3*IQR, Q3+3*IQR]. Intrarile specifice valorilor atributelor, abia mentionate, care depasesc intervalul sunt sterse.

Desi in mod normal se lucreaza cu regula de 1.5*IQR, am ales-o pe cea cu 3*IQR din dorinta de a fi mai putin punitiv cu datele. Daca as fi pastrat 1.5*IQR riscam sa elimin multe observatii din top 10, lucru de nedorit intrucat avem un set de antrenament foarte *inbalanced* (rom. inegal). Desigur, acum s-ar putea spune ca exista sanse mult mai mari sa fi pastrat *hackeri*. Un lucru adevarat, dar acest impediment nu incurca prea mult deoarece numarul de *hackeri* este unul relativ mic, deci eliminarea doar acelor cazuri extreme este una convenabila.

2.3 Alegerea subsetului optim de atribute

O problema universala cu care se confrunta orice inginer pe parcursul unei analize exploratorii este selectarea acelor caracteristici (eng. *features*) care sunt cu adevarat relevante si eliminarea celor redundante sau inutile. Intre **redundanta** si **inutilitate** exista o diferenta majora deoarece putem avea atribute utile, dar redundante in raport cu altele, in timp ce atributele inutile nu reflecta nimic (ex. coloanele eliminate la 2.4).

Avantajele pastrarii numai a acelor caracteristici relevante sunt:

- *simplificarea modelului pentru a fi mai bine interpretat*
- *durata antrenarii mai scurta*
- *evitarea blestemului dimensiunilor (eng. Curse of Dimensionality, desi aici nu este cazul*)
- *generalizarea modelelor, fiind mai specifice contextului problemei si nu datelor existene*
- ***Occam's razor** afirma ca ne dorim modele simple si inteligibile, proprietati pe care le pierdem cu un numar mare de caracteristici (eng. features).*

Metodele de *feature selection* pot fi impartite in 3 categorii: ***Metode Filter**, **Metode Wrapper**, **Metode Embedded***. In cadrul unui proiect de analiza ele pot fi intercalate in functie de ce dorim sa obtinem. Nu exista nicio recomandare empirica cu privire la practici bune (eng. *best practices*) cand vine vorba de astfel de metode, dar in urma unei analize s-au facut urmatoarele observatii:

Metode Filter	**Metode Wrapper**	**Metode Embedded**
Metode care nu incorporeaza algoritmi specifici de invatare automata	Evaluate pe un algoritm de invatare automata, cauta in mod greedy un subset optim de caracteristici	Scoruri calculate pentru caracteristici incorporate in cadrul algoritmilor
Rapide ca si complexitate timp	Complexitate timp mare pentru un set cu multe caracteristici	Complexitatea timp este undeva intre cea de la metodele Filter si cele Wrapper
Sanse mici de a face overfitting	Sanse mari de a face overfitting	In general, reduce overfitting-ul
Exemple: Corelatia, testul Chi-Square, ANOVA	Exemple: Eliminare Recursiva de Caracteristici	Exemple: Random Forest , Regresie Lasso

In cadrul acestei proiect am folosit o metoda Filter: *Corelatia* si o metoda Embedded: *scorurile caracteristicilor (feature-urilor) dintr-o Random Forest.*

2.3.1 Corelatia

Corelatia este o relatie statistica dintre 2 variabile aleatoare care ne poate explica gradul de legatura dintre cele 2 (ex. relatia dintre pretul unui produs si numarul de oameni dispusi sa il cumpere). Intervalul din care corelatia poate lua valori este [-1, 1]. Valorile pozitive semnifica o crestere simultana pe axe, cele negative o crestere a unei variabile sustinuta de o scadere a celeilalte iar valorile apropiate de 0 indica o dispersie foarte mare. Cele 3 tipuri sunt prezentate intuitiv in imaginile de mai jos:

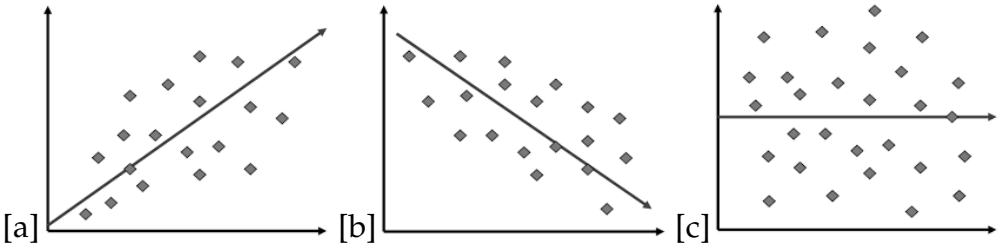


Figura 2.2: (a) Corelatie pozitiva (b) Corelatie negativa (c) Fara corelatie

Un aspect important este acela ca **corelatia nu implica cauzalitate**! In informatica, daca doua variabile de intrare sunt strans corelate, ne dorim sa o eliminam pe una din ele intrucat aceea este **redundanta**. Asadar, planul ar fi sa analizam corelatiile dintre atributele de intrare iar din perechea cu o corelatie in modul mai mare de 0.8 sa eliminam atributul cu o mai slaba legatura fata de coloana de iesire. Pragul de 0.8 a fost ales in mod empiric. Iata o vizualizare grafica, la prima vedere, a tuturor corelatiilor:

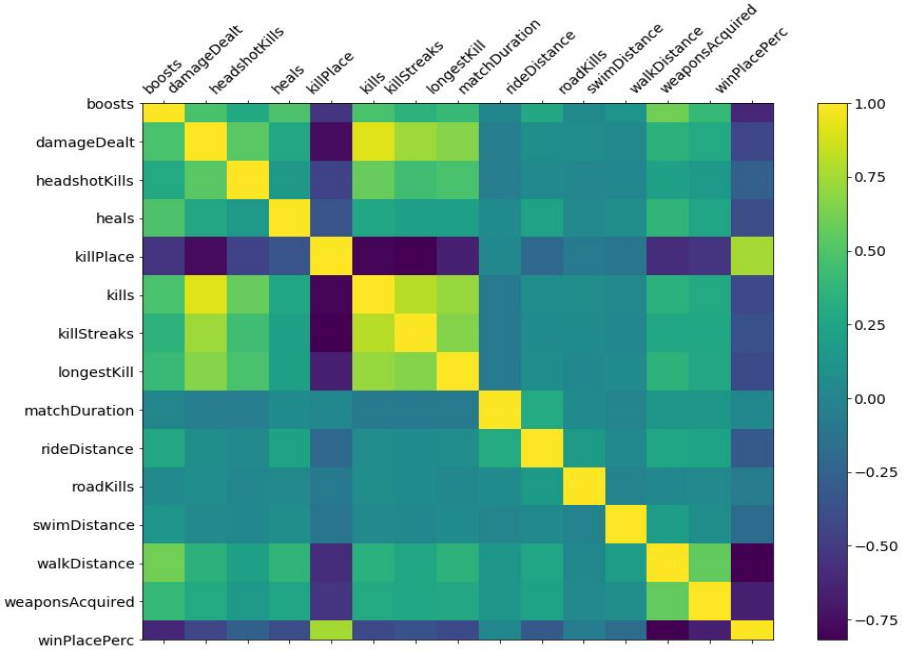


Figura 2.3: Tipuri de date

2.3.2 Selectia de caracteristici cu Random Forest

Asa cum reise din tabelul de la pagina 13, metodele *Filter* par sa fie cele mai echilibrate din punct de vedere al timpului de executie si al overfitting-ului. Totusi, acestea nu ofera in general informatii cu privire la **contributia** caracteristicilor, spre deosebire de metodele *Embedded* care generalizeaza bine iar scorurile atribuite feature-ilor sunt usor de interpretat. De exemplu, un feature a carui corelatie cu coloana de iesire este mare nu inseamna neaparat ca va juca un rol esential in cadrul unui algoritm de invatare automata. Asadar, importanta poate fi reflectata de un algoritm precum ***Random Forest***. Acesta este un ansamblu de arbori de decizie, in nodurile carora sunt plasate atribute pentru care se calculeaza scoruri astfel:

---

pentru fiecare arbore din ansamblu

$node\_imp[i] = w[i] * H[i] - w[left(i)] * H[left(i)] - w[right(i)] * H[right(i)]$ *unde*:

- ✓*node_imp[i] = importanta nodului i*
- ✓*w[i] = cate observatii ajung la nodul i impartit la numarul total de observatii*
- ✓*H[i] = entropia nodului i*

$$feature\_imp[f] = \frac{\sum_{i \in allnodesthatsplitonf} node\_imp[i]}{\sum_{k \in allnodes} node\_imp[k]}, \forall f \in allfeatures$$

$$normalised(feature\_imp[f]) = \frac{feature\_imp[f]}{\sum_{j \in allfeature} feature\_imp[j]}$$

$$RF\_feature\_imp[f] = \frac{\sum_{j \in alltrees} normalised(feature\_imp[f])}{T}$$

- ✓*T = numarul total de arbori din ansamblu*

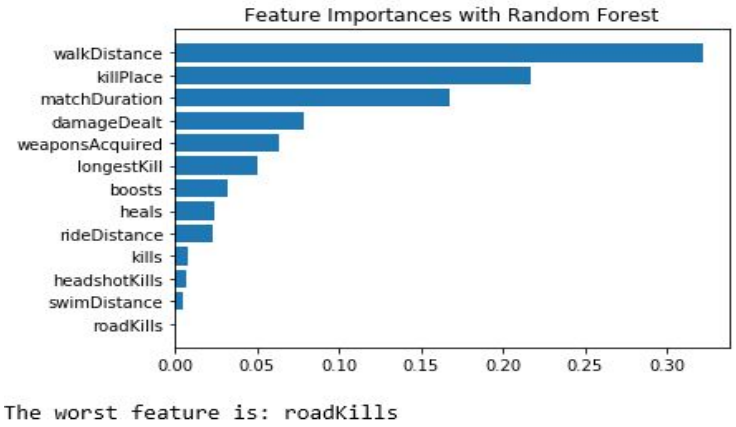


Figura 2.4: Scorurile feature-urilor din **train.csv**, calculate de Random Forest

2.3.3 Standardizare

Un alt aspect important il reprezinta scalarea caracteristicilor (eng. *Feature Scaling*). In majoritatea cazurilor, unitatile de masura ale atributelor sunt diferite, motiv pentru care magnitudinile observatiilor sunt foarte variate. De exemplu, greutatea unei persoane ia valori, in general, de la 4 la 100 de kilograme in timp ce aceeasi greutate in grame poate fi si de 1000 de ori mai mare ca magnitudine. Algoritmii de invatare automata care calculeaza distante euclidiene sau realizeaza *Gradient Descent* au nevoie de scalare. In cazul distantelor, caracteristicile cu o magnitudine mare au o pondere mai semnificativa in momentul realizarii calculului, problema aceasta fiind rezolvata de scalare. De asemenea, in cazul *Gradient Descent-ului*, pocesul de "coborare" este mai rapid.

In literatura de specialitate, se face distinctia intre 2 moduri generale de a face *Feature Scaling*: **Standardizare** si **Normalizare**. Normalizarea scaleaza caracteristicile in asa fel incat ele sa aiba valori cuprinse in intervalul [0, 1]. Standardizarea scaleaza caracteristicile pe baza distributiei normale, fiecare *feature* avand in final o medie de $\mu = 0$ si deviatia standard $\sigma = 1$. Empiric, a fost stabilit ca cea de-a doua metoda functioneaza in practica mai bine.

Transformarea pe care o voi realiza in cadrul acestei lucrari este una de standardizare, numita *Yeo-Johnson*. Facand parte din familia de transformari parametrice monotone, *Yeo-Johnson* incearca sa si mapeze datele intr-o distributie normala, cu scopul de a reduce varianta si a minimza skewness. Aceasta mapare se va dovedi utila ulterior, atunci cand voi aplica algoritmul **Naive Bayes** care face prezumtia ca variabilele de intrare urmeaza o distributie Gaussiana. De asemenea, standardizarea este de folos si urmatorului sub-capitol, de diminuare a dimensiunilor, tehnica folosita acolo avand nevoie de scalarea atributelor. Iata transformarea *Yeo-Johnson*, formulata matematic:

$$x_i^\lambda = \begin{cases} [(x_i+1)^\lambda - 1]/\lambda, & \lambda \neq 0, x_i \geq 0 \\ ln(x_i+1), & \lambda = 0, x_i \geq 0 \\ -[(-x_i+1)^{2-\lambda} - 1]/(2-\lambda), & \lambda \neq 2, x_i < 0 \\ -ln(-x_i+1), & \lambda = 2, x_i < 0 \end{cases}$$

Transformarea e parametrizata de λ care se obtine prin estimarea probabilitatii maxime (eng. *maximum likelihood estimation*). Iata cum arata histogramele datelor inainte si dupa standardizare.

2.3.4 Diminuarea dimensiunilor

Pentru a face trecerea la analiza *nesupervizata*, este nevoie de reducerea dimensionalitatii setului de antrenament. Avand 12 caracteristici ramase, dupa operatiile de mai sus, vizualizarea lor pe un grafic (eng. *plot*) este imposibila. Din acest motiv, ne dorim o diminuare a spatiului in 3 dimensiuni sau 2 dimensiuni. Exista numeroase variante de a realiza astfel de operatii iar cea pe care am ales-o este bazata pe ***clusterizare ierarhica aglomerativa***. Diferenta majora va fi ca pe dendograma, in locul observatiilor, se vor afla caracteristicile setului de antrenament. Algoritmul este unul relativ simplu:

INITIALIZARE

→ Fiecare caracteristica este initial cluster *singleton*

CORP ITERATIV

→ Se calculeaza distantele dintre clustere confrom unei metrici (ex. *Ward-linkage*)

→ Se unesc clusterele care au distanta minima intre ele, formand un nou cluster

→ Pasii anteriori sunt reluati pana cand se ajunge la un singur cluster

INTERPRETARE

Dendograma obtinuta este taiata cu o linie verticala imaginara, incepand de la primul nivel, decupand astfel numarul de clustere dorit.

Uniunea finala dintre atributele ce alcatuiesc un cluster intr-o noua componenta globala se realizeaza facand media aritmetica, ca in exemplul urmator:

Feature1	Feature2		Feature1&Feature2
1	2	→	1.5
1	2	→	1.5
1	3	→	2.0

Am ales acesta metoda, cunoscuta in modulul sci-kit drept *Feature Agglomeration*, intrucat este rapida din punct de vedere computational, se pliaza foarte bine pe caracteristici deja scalate la aceeasi unitate de masura si este usor de interpretat. In cadrul

proiectului, ierarhia de *feature-uri* poate fi vizualizata:

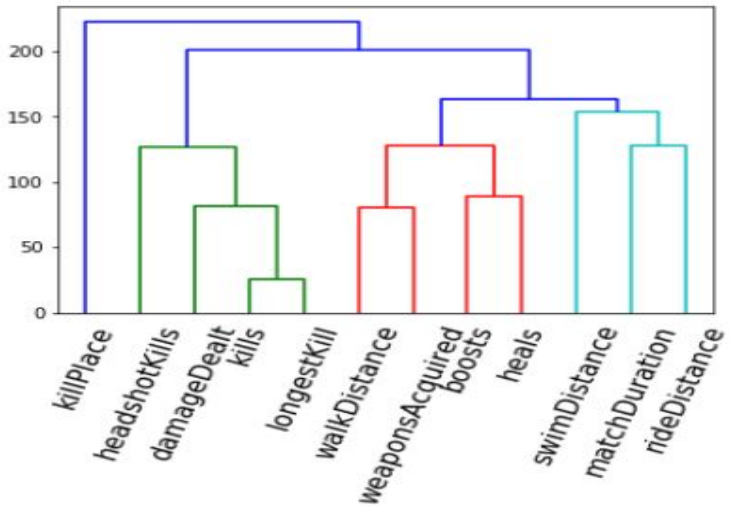


Figura 2.5: Ierarhia de caracteristici

Se observa ca daca as face o singura taietura, as ramane cu 2 componente, una *singleton* si alta cu restul atributelor. Prin urmare, din dorinta de avea un grad de informatie mai ridicat, aleg sa fac retezarea dubla. De mentionat este si metrica **Ward**, fara de care dendograma era una cu totul diferita. Similaritatea de tip Ward penalizeaza componentele dense, incercand mai intai sa grupeze clusterele cu putine elemente pentru a reduce sansele ca la o prima taiere a arborerlui ierarhizat sa obtinem clustere *singleton*. Altfel spus, Ward evita formarea clusterelor elipsoide.

Capitolul 3

Analiza nesupervizata

Avand in vedere ca datele din setul de antrenament au fost prelucrate corespunzator, invatarea nesupervizata poate incepe. In urma operatiilor descrise in paginile anterioare, cadrul de date (eng. *Data Frame*) arata in felul urmator (a se lua in vedere ca tabelul de mai jos surprinde primele 5 observatii dintr-un tabel de 11366):

atr1	atr2	atr3	winPlacePerc
0.700062	0.400335	-0.472270	51
0.178045	-0.808024	0.624001	48
0.294052	-0.561770	0.711124	57
0.871228	0.101547	-0.808441	33
-0.716307	1.225031	-0.448755	90

Prima coloana surpinde combinatia dintre atributele a caror culori pe dendograma de la [2.5] sunt azur si rosu, a doua coloana este singulara, specifica lui *'killPlace'* iar a treia coloana cuprinde atributele de culoare verde. Cele 3 dimensiuni sunt rezultatul aglomerarii de caracteristici (eng. *Feature Agglomeration*) de la sectiunea 2.3.4. Ca scurta observatie, **'atr1'** poate fi vazut ca un atribut de lupta (eng. *combat features*) intrucat inglobeaza statistici specifice iar **'atr3'** sumarizeaza valori de suport (eng. *non-combat features*). Ne dorim ca pe tot parcursul analizei nesupervizate sa aflam mai multe informatii calitative ce ascund *pattern-uri* nestiute in date.

3.0.1 K-means

Primul algoritm pe care l-am aplicat a fost **K-means**. Cunoscut drept un *general purpose clustering algorithm*, este cea mai folosita tehnica de analiza a unui set mare de date. Este prima alegere (eng. *firsthand option*) intrucat ofera o viziune de ansamblu

asupra unor posibile sabloane. Algoritmul este unul simplu, descris in pseudocodul urmator:

Algoritm: K-means

INITIALIZARE

*Alege numarul **K** de clustere dorite*

*Initializeaza centroizii $\mu^1, ..., \mu^k$ a celor **K** clustere cu observatii alese random*

CORP ITERATIV

repeta

$\rightarrow$ *pentru* $\forall x^i \in training\_set$ *calculeaza* $d(x^i, \mu^j)$, $j \in 1..K$

$\rightarrow$ *asigneaza fiecare observatie* x^i *clusterului* C^j *pentru care* $d(x^i, \mu^j)$ *este minima*

$\rightarrow$ *recalculeaza centroizii* $\mu^j = \frac{\sum_{e \in C^j} e}{|C^j|}$

cat timp o conditie este satisfacute

K-means urmareste minimizarea unui criteriu intitulat **suma patratelor erorilor** (eng. *sum of squared errors sau SSE*), definit ca:

$$\sum_{i=1}^{n} \min_{\mu^j \in C} (||x^i - \mu^j||^2)$$

Cu alte cuvinte, ne dorim clustere care sa minimizeze distantele dintre punctele membre si centroizii specifici. Acest scop este dependent de urmatoarele 2 observatii:

- Este nevoie de specificarea unui numar potrivit de K clustere care sa explice datele
- Alegerea initiala a centroizilor are o pondere foarte mare, influentand atat timpul de executie, cat si convergenta la o partitionare optima

Avand doar un singur parametru ce trebuie specificat, si anume K, algoritmul poate alege valoarea ideala a acestuia in functie de SSE, prin metoda *Elbow technique*. Ea presupune repetarea algorimtului cu valori diferite ale lui K si marcarea grafica a perechilor (K, SSE) pentru fiecare rulare. Tuplul cautat este varful de pe figura, care din punct de vedere vizual seamana cu un punct de cotitura.

In mod normal, centroizii initiali sunt alesi la intamplare din randul instantelor de clusterizat. Daca datele sunt foarte bine separate atunci e posibil ca la finalul initializarii sa existe macar un cluster din care nu a fost selectat niciun centroid. Astfel, timpul

de executie este mult mai mare iar, la final, riscam sa nu obtinem partitionarea dorita a datelor. Din aceste motive, initializarea centroizilor se va face prin **K-means++**, algoritm care incearca sa selecteze, drept centroizi, observatii care sunt cat mai distantate unele de altele. In felul acesta, exista o probabilitate mai mare de a selecta membri din toate clusterele. Mai jos este pseudocodul corespunzator:

Algorim: K-means++

INITIALIZARE

Stabileste numarul de clustere K dorite

Alege $\mu^1 = x^i$ *in mod aleatoriu, cu probabilitate uniforma*

CORP ITERATIV

pentru $j \leftarrow 2, ..., K$

$\rightarrow D^i = \min_{j` \in 1,..,j-1} \|x^i - \mu^{j`}\|_2^2$

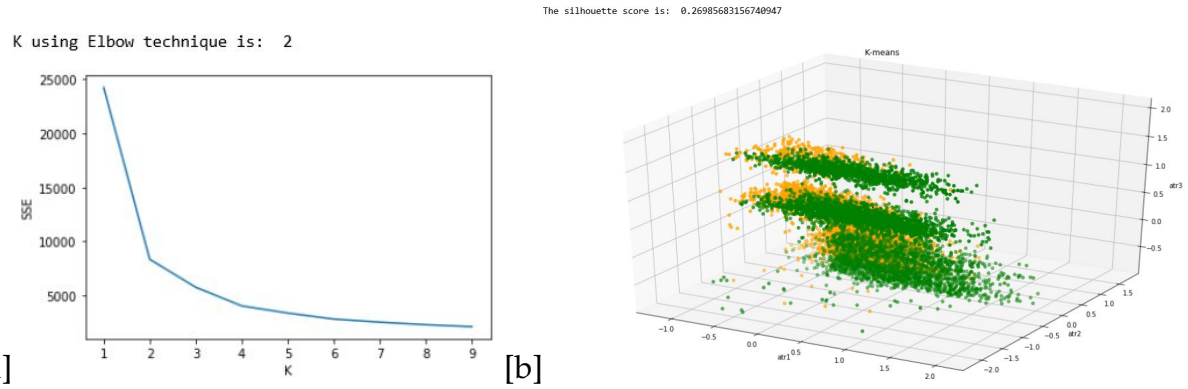
$\rightarrow P(x^i) = \frac{D^i}{\sum_{l=1}^{n} D^l}, \forall i = 1, .., n$

$\rightarrow$ *alege* x^i *in mod aleatoriu cu probabilitatea de mai sus si fixeaza* $\mu^j = x^i$

REZULTAT

Returneaza $\mu = [\mu^1, .., \mu^k]^T$

Algoritmul abia prezentat rezolva partial problema initializarii centroizilor. Caracterul este tot unul aleator, dar putem reduce si mai mult acest lucru prin rularea K-means de un numar specific de ori (100 in cazul nostru), de fiecare cu initializari diferite date de K-means++. Rezultatul optim este rularea care da cea mai mica inertie.



[a] [b]

Figura 3.1: (a) Valoarea optima K cu tehnica Elbow (b) Rezultate K-means pe date

Rezultatul este unul slab, un scor Silhouette mic si clustere suprapuse care nu

par sa surprinda foarte bine forma datelor. Posibelele principale motive ce stau la baza eseculu sunt senzitivitatea la valori extreme, distributia elongata a punctelor sau initializarile eronate. Totodata, densitatea mare semnaleaza nevoie unui alt algoritm.

3.0.2 DBSCAN

DBSCAN (eng. *Density based spatial clustering algorithm of applications with noise*) este principala tehnica de partitionare aplicata seturilor de date a caror densitate este semnificativa. Un algoritm simplu si intuitiv care lucreaza cu vecinatatile punctelor pe care trebuie sa le clusterizeze. Avand 2 parametri principali:

- ϵ = raza maxima pe care se vor cauta vecini
- *minPts* = numarul de puncte in ϵ-vecinatatea unui punct

DBSCAN impune concepte noi de clustere, construite in jurul acestor variabile. Astfel, un **"Core point"** este o observatie care pe o raza ϵ are minim *minPts* vecini, un **"Border point"** este o observatie care pe o raza ϵ NU are minim *minPts* puncte, dar se afla in jurul granitelor-ϵ ale unui "Core point" iar un **"Outlier"** este o observatie ce nu respecta niciuna din definitiile "Core point" sau "Border point". Notiunile anterioare definesc acum principiile pe care elementele unui cluster dens trebuie sa le respecte:

- *Accesibilitate directa* = un element Q este direct accesibil din P daca Q se afla in interiorul granitelor-ϵ ale "Core point-ului" P
- *Accesibilitate* = Q este accesibil din P daca exista un drum $p_1, .., p_l$ unde $p_1 = P$ si $p_l = Q$ si fiecare p_{i+1} este accesibil direct din p_i
- *Conexiune densa* = P si Q sunt conectate daca exista un element O astfel incat P si Q sunt accesibile din O

In continuare voi prezenta o lista cu cateva dintre avantajele si dezavantajele DBSCAN:
✓ Eficient pentru seturi mari de date
✓ Robust la valori extreme (eng. *outliers*)
✓ Se pliaza bine la clustere cu forme arbitrare
✓ Gaseste automat numarul optim de clustere, spre deosebire de K-means
× Densitatile foarte diferite in randul clusterelor pot reprezenta o problema atunci cand se aleg ϵ si *minPts*

$\times$ Senzitivitate maxima la ϵ si *minPts*

$\times$ Poate aparea "Blestemul Dimensiunilor Mari" daca spatiul este prea mare

Algoritmul implementat in *Scikit* este cel standard, de complexitate timp $\mathcal{O}(n^2)$, care arata in felul urmator:

Algorim: DBSCAN

INITIALIZARE

C = 0

pentru $i \leftarrow 1, ..., n$

    label[i]=undefined

CORP ITERATIV

pentru $p \leftarrow 1, .., n$

    daca label[p] != undefined

        continua

    N_ϵ*[p] = RangeQuery(p), RangeQuery e functia ce returneaza vecinii din raza* ϵ

    daca $|N_\epsilon[p]| < minPts$

        label[p] = noise

        continua

    S = N_ϵ*[p]; C=C+1; label[p]=C*

    pentru $q \in S$

        daca label[q] $\notin (undefined, noise)$

            continua

        label[q]=C

        N_ϵ*[q] = RangeQuery(q)*

        daca $|N_\epsilon[q]| \geq minPts$

            S=S $\cup N_\epsilon$*[q]*

REZULTAT

Returneaza vectorul labels

Intrebarea ramane cum selectam ϵ si *minpts*? Mai intai am selectat cei mai apropiati 1% vecini ai tuturor observatiilor. Pentru fiecare observatie in parte am calculat distanta medie de la ea la toti membrii vecinatatii. Se obtine un vector de distante medii care se aduna si se impart la numarul total de observatii. In acestl fel obtin ϵ. Pentru *minPts*

am oferit mai multe valori dintr-un interval, avand deja ϵ calculat, si am afisat pe un grafic perechile de *(scor Silhouette, minPts)*. Solutia optima este data de varful graficului, cum se observa si in imaginea de mai jos.

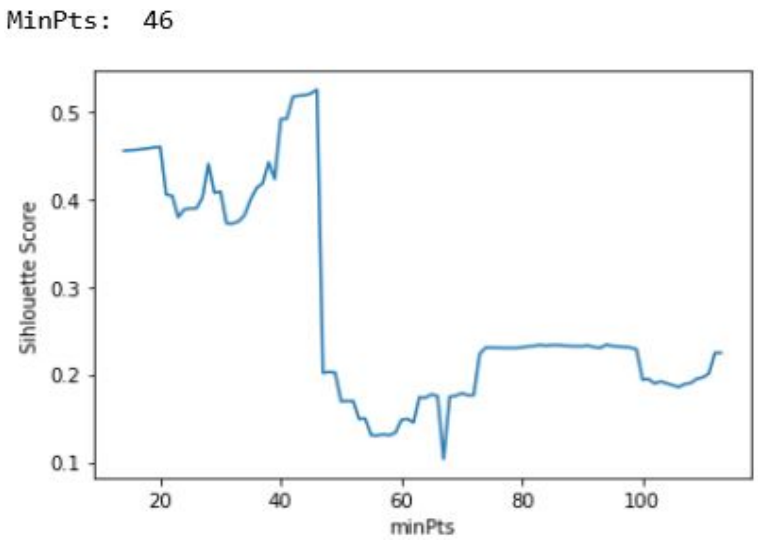


Figura 3.2: Cautarea valorii optime a lui minPts in functie de ϵ gasit deja

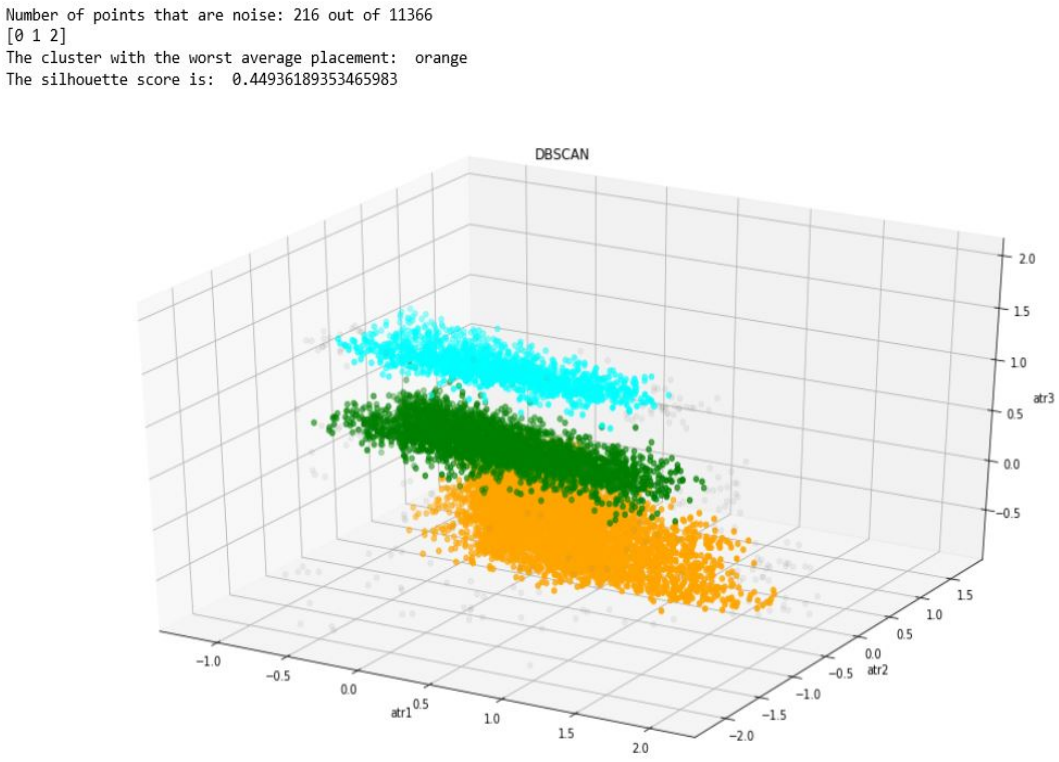


Figura 3.3: Rezultatul clusterizarii cu DBSCAN (punctele gri reprezinta *Outliers*)

Din cele 3 clustere rezultate ne intereseaza acela care cuprinde jucatorii a caror pozitie de final in cadrul unui meci este in medie foarte slaba. In mod evident, aici se

vor afla putini competitori din top 10, lucru care ne da posibilitatea de a face o noua separare a datelor. Cu alte cuvinte, clusterele azur si verde sunt combinate in cadrul de date (eng. *DataFrame*) *training_set_1* iar cel portocaliu (clsuterul slab) formeaza *training_set_2*. Primul set e unul echilibrat (eng. *balanced*) iar al doilea este profund inegal (eng. *unbalanced*), avand o majoritate clara de etichete nule (jucatori ce nu au terminat meciul in top 10). Se preteaza acum o analiza supervizata pe 2 module, descrisa in capitolul urmator. La momentul testarii, vom vedea mai intai daca o intrare corespunde clusterului portocaliu, caz in care aplicam anumite tehnici, sau celorlalte 2 clustere, caz in care aplicam alte tehnici.

Capitolul 4

Analiza supervizata

Prima parte a acestui capitol se va focusa asupra clusterului portocaliu din figura [3.3]. Asa cum am spus anterior, majoritatea jucatorilor din aceasta partitie au terminat meciul pe un loc slab, observatie furnizata vizual prin urmatoarea histograma:

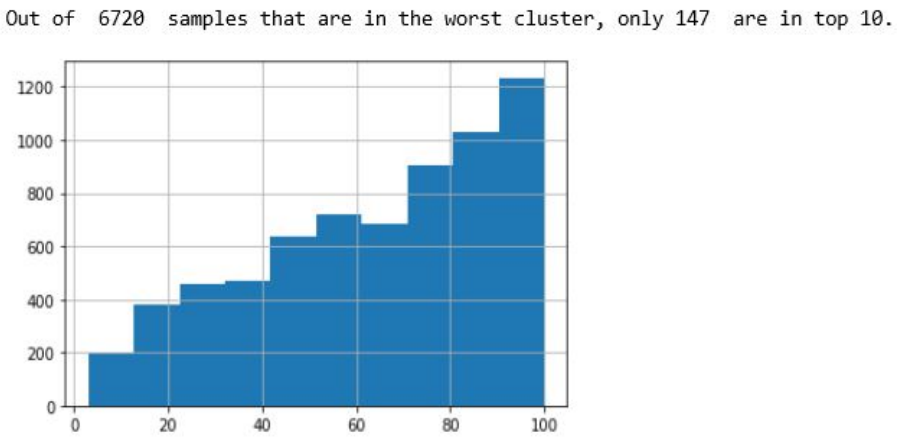


Figura 4.1: Locurilor ocupate la finalul meciului de jucatorii din clusterul portocaliu

Avand in vedere majoritatea clara de observatii (eng. *samples*) care nu sunt in top 10, ne putem gandi la un algoritm rapid, ca timp, ce va atribui pe scara larga eticheta 0. In acest sens, m-am gandit la un **Arbore de decizie** (eng. *Decision Tree*) din urmatoarele considerente:

✓ Este usor de interpretat si ofera informatii esentiale cu privire la granitele (eng. *tresholds*) pe care algoritmul le considera importante

✓ Trateaza usor atributele irelevante pe care le inlatura la fiecare nod, dar le reconsidera in iteratiile urmatoare

✓ Poate suporta valorile lipsa intrucat lucreaza cu intervale (atributele noastre sunt continue, deci pragurile de separare sunt de ordin comparativ)

✓ Foarte rapid la testare, complexitatea fiind de $O(adancime)$

Una dintre problemele principale aduse de un astfel de algoritm este ***overfitting-ul***, fenomen caracterizat printr-o specificacitate prea ridicata a modelui vis-a-vis de setul de antrenament. Un arbore construit cu o adancime maxima are sanse mari de a deveni prea particular. Acest efect este evitat, in general, prin operatia de ***retezare*** (eng. *prunning*) a nivelelor. Am aflat adancimea maxima a arborelui de decizie complet iar apoi, din acesta, am taiat pe rand cate un nivel pana cand am ajuns la un compas de decizie (eng. *decision stump*). Pentru fiecare arbore retezat am aplicat ***validare incrucisata leave one out (sau CVLOO)*** cu scopul de a alege drept arbore final pe acela a carui eroare la validare este minima. In urma acestei proceduri, am obtinut un arbore de decizie optim de adancime 4 cu urmatoarea matrice de confuzie:

		Realitate	
		Negativ	Pozitiv
Predictie	Negativ	6572	132
	Pozitiv	1	15

Desi intrarile cu etichete nule sunt prezise corect in proportie de aprope 100%, celelalte sunt clasate in mod eronat intr-un numar foarte mare. Fenomenul de ***underfitting*** este cel cu care ne confruntam acum. Modelul nu beneficiaza de destule informatii relevante cu privire la jucatorii din top 10 (sau informatile despre ei nu au un caracter predictibil), motiv pentru care esueaza in a-i categoriza corect. O strategie cunoscuta, menita sa combata un astfel de comportament este ***Boosting-ul***.

4.1 AdaBoost

Boosting-ul este o metoda de tip ansamblu (eng. *ensemble learning*) care imbina mai multi clasificatori slabi (eng. *weak classifiers*), adica predictori putin mai buni decat alegerea la intamplare, intr-un singur algoritm robust. Unul dintre cei mai folositi algoritmi de boosting este cel care da numele sub-sectiunii. Am ales ***AdaBoost*** deoarece se adapteaza bine la atribute continue si converge relativ repede la un optim datorita functie de pierdere (eng. *loss function*) de tip exponentiala. Mai mult, rezultatul predictiei este dat de o combinatie liniara a tuturor clasificatorilor slabi si ponderile asociate, interpretabilitatea erorilor devenind usoara intrucat e de ajuns sa ne uitam la impactul fiecarui *weak classifier* pentru a stabili unde este problema. Mai jos implementarea [4]:

Algorim: AdaBoost

INITIALIZARE

$w_i^1 = \frac{1}{n}, \quad \forall i = \overline{1,n}$

CORP ITERATIV

pentru $t \leftarrow 1, ..., T$

$\rightarrow$ *aplica un clasificator slab* h^t *pe date*

$\rightarrow \epsilon^t = \sum_{i=1}^{n} w_i^t | y^i \neq h^t(x^i)$ **eroare ponderata**

$\rightarrow \alpha^t = \eta \cdot ln\frac{1-\epsilon^t}{\epsilon^t}$ **pondere clasificator slab**

$\rightarrow w_i^{t+1} = \frac{1}{Z^t} \cdot w_i^t \cdot e^{-\alpha^t \cdot y^i \cdot h^t(x^i)}, \quad \forall i = \overline{1,n}$

PREDICTIE

$H^t(z) = sgn(\sum_{t=1}^{T} \alpha^t \cdot h^t(z))$

Numarul de clasificatori slabi si rata rata de invatare η (eng. *learning rate*) sunt parametrii de care depinde algoritmul. Un ansamblu dens, cu multi clasificatori slabi, este de preferat deoarece ofera un spatiu de cautare mare. Totusi, rata de invatare este primordiala intrucat ea stabileste ponderea fiecarui *weak classifier*. Prin urmare, exista un compromis (eng. *trade-off*) intre aceste 2 variabile iar felul in care alegem valorile optime este esential. In acest sens, am aplicat o tehnica cunoscuta, numita ***Randomized Grid-Search***. Ea presupune rularea unui algoritm de mai multe ori, cu valori ale *hyper-parametrilor* (parametrii de care depinde modelul) luate dintr-un interval specific. La fiecare iteratie avem o pereche de *hyper-parametri* noua. Optimalitatea modelului se masoara cu o metrica specificata de utilizator. In cazul meu, am folosit acuratetea echilibrata (eng. *Balanced Accuracy*). De asemenea, in cadrul implementarii din *scikit*, functia de *Grid-Search* realizeaza si ***validarea incrucisata de tip 5-fold*** pentru fiecare iteratie. Pentru a reduce si mai mult din particularitatea unei perechi de parametri, validarea incrucisata este facuta de 3 ori, la final calculandu-se media scorurilor obtinute pe seturile de test.

param_learning_rate	param_n_estimators	params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score
2.00118	445	{'learning_rate': 2.001178999026374, 'n_estima...	0.88381	0.835528	0.921725	0.880355	0.035274

Figura 4.2: Valorile optime pentru parametrii algoritmului AdaBoost

Daca rulam acum AdaBoost pe setul de antrenament specific clusterului portocaliu obtinem urmatoarea matrice de confuzie:

		Realitate	
		Negativ	Pozitiv
Predictie	Negativ	5888	7
	Pozitiv	685	140

A se observa ca, in cazul etichetelor nule, acuratetea a scazut in detrimentul clasificarii mai bune a intrarilor pozitive. Ramane de vazut cum se va comporta algoritmul pe datele de test, dar mai intai trebuie sa stabilim care sunt observatiile ce pot fi prezise cu tehnica descrisa pana acum. Acest lucru implica gasirea celor *minPts* vecini din *training_set* ai fiecarui *sample* din *test_set*. Daca mai mult de 95% dintre vecinii unei observatii sunt din clusterul portocaliu, o putem incadra ca fiind viabila pentru AdaBoost. Astfel, am descoperit ca 6728 de intrari pot fi clasificate cu o metoda de Boosting.

[a]

```
---------------------------- Adaboost -----------------------
Number of mislabeled points out of a total 6728 points : 389
The accuracy score is : 0.942182
[[6293  293]
 [  96   46]]
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      6586
           1       0.14      0.32      0.19       142

    accuracy                           0.94      6728
   macro avg       0.56      0.64      0.58      6728
weighted avg       0.97      0.94      0.95      6728

Balanced accuacy score:  0.6397276767192893
```

[b]

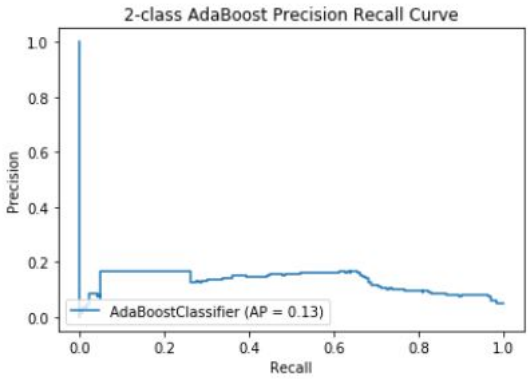


Figura 4.3: (a) Statistici AdaBoost (b) Graful Precision-Recall K-means pe date

Am facut o modificare la graficul ***ROC***, inlocuind *False Positive Rate-ul*, de pe axa Ox, cu *Precision*. *FPR*-ul nu este influentat de un set de date inegal (eng. *unbalanced set*) intrucat nu include *TN(True Negatives)*, deci graficul este mai explicativ.

4.2 Al doilea set de test

Asa cum am mentionat in capitolele anterioare, cel de-al doilea set de test este relativ mai echilibrat. Numara 5946 de observatii dintre care pozitive, 1056, iar nule, 4890. Aici nu vom mai aplica reducerea dimensiunilor (datele sunt evident din celelalte 2 clustere), ci vom lasa caracteristicile in forma data de operatia de standardizare.

Majoritatea jucatorilor din top 10 se afla la aceasta sub-sectiune, deci clasarea lor corecta are o pondere mai mare in momentul alegerii modelului ideal. Este cadrul de date (eng. *Dataframe*) pe care investitorii/sponsorii sau pariorii il urmaresc pentru a se hotara asupra unei investii. Accentul principal va fi in pus pe acesta sectiune!

4.2.1 Naive Bayes

In urma unei analize a datelor, am descoperit ca 3 dintre atribute (*'damageDealt'*, *'walkDistance'*, *'weaponsAcquired'*) au distributiile specifice histogramelor asociate foarte asemanatoare cu cele ***Normale/Gaussiene***. Din acest motiv, un model ***Naive Bayes*** (presc. *NB*) bazat pe *atribute continue* ar putea fi un predictor puternic atat din punct de vedere a timpului de executie, cat si a acuratetii.

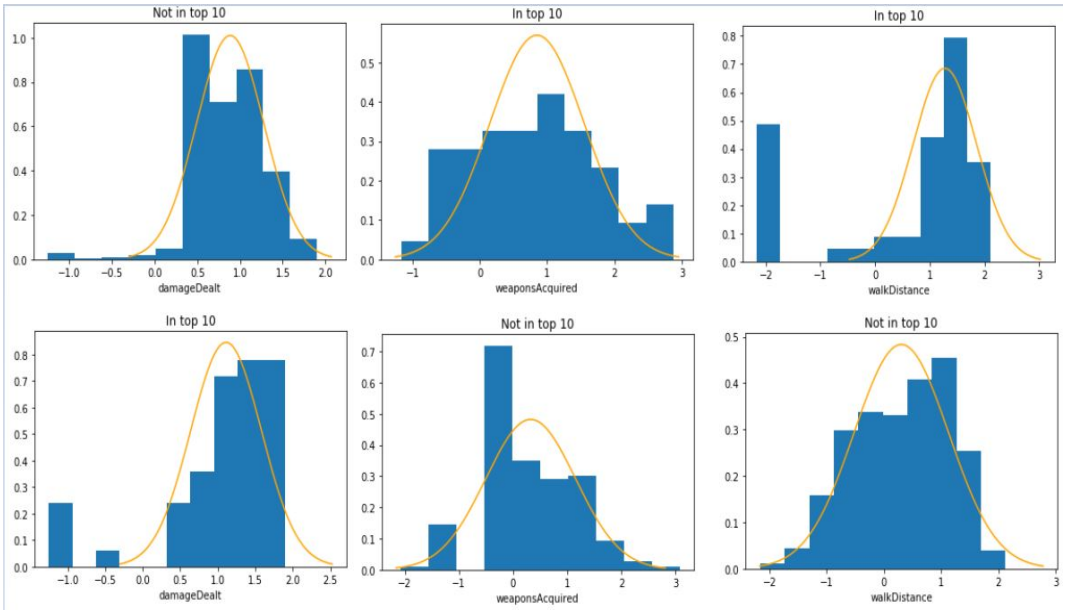


Figura 4.4: Histogramele atributelor in functie de etichete

Eficienta timp este data de *supozitia de independenta conditionala a unui atribut fata de altul, in raport cu eticheta*. Aceasta presupunere reduce numarul de estimari de probabilite pe care o distributie corelata le face. Practic, daca in mod normal ar trebui sa se realizeze $2^{n+1}+1$ operatii de estimare, acum e nevoie doar de $2n+1$ (n=numarul de atribute). NB este un model generativ liniar care ia decizii in felul urmator:

$$\underset{Y\in\{0,1\}}{\operatorname{argmax}} P(Y|X) = \underset{Y\in\{0,1\}}{\operatorname{argmax}} \frac{p(X|Y)P(Y)}{p(X)} = \underset{Y\in\{0,1\}}{\operatorname{argmax}} p(X|Y)P(Y) = \underset{Y\in\{0,1\}}{\operatorname{argmax}} (\prod_{i=1}^{n} p(X_i|Y))P(Y)$$

unde $X = [X_1, .., X_n]^T, \quad P(X_i|Y) \sim \mathcal{N}_Y^i(\mu, \sigma^2) \quad \forall i = \overline{1,n}$

La antrenare am aplicat validare incrucisata pentru a ma asigura ca algoritmul nu face overfitting. De aceasta data, tehnica a fost una diferita, alegand sa impart setul de antrenament in 90% date *train* si 10% date *validare*, de 100 de ori. Obtinem o viziune mult mai exacta daca esantionam atat de des, pentru dimensiuni diferite ale seturilor de *train* si *test*. Cu alte cuvinte, urmam exact pasii specifici liniei de invatare (eng. ***Learning Curve***), o metrica ce raspunde la intrebarea ***"Devine un model mai bun daca invata mai mult?"***.

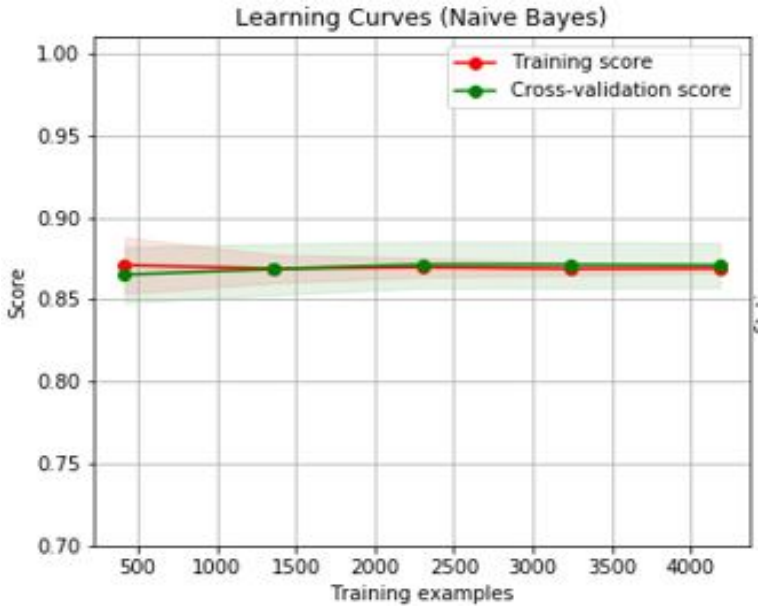


Figura 4.5: Graficul Learning Curve in cazul Naive Bayes

O pereche rosu - verde de puncte in aceeasi pozitie identifica acuratetea (in medie) la antrenare, respectiv validare a unui set format din atatea observatii cate arata coordonata Ox. Umbrele (eng. *shades*) ce cuprind liniile sunt deviatiile de la medie a celorlalte esantioane. Ambele linii sunt foarte apropiate, sugerand faptul ca Naive Bayes nu face overfiting, iar intervalele de deviatie mici atesta un comportament previzibil al algoritmului. De asemenea, modelul se stabilizeaza de la un numar de observatii incolo, indicand un grad de incredere ridicat in momentul unei predictii.

Alt aspect important de care se tine cont la antrenarea unui model este ***Scalabilitatea***. Partea de *training* tinde sa ocupe foarte mult timp in cadrul unei analize de *Data Mining*, lucru care in practica cauzeaza pierderi financiare majore la produsele software. In cazul de fata, suntem in egala masura interesati de viteza, cat si de acuratete. Pentru a raspunde ambelor cerinte, am efectuat inca 2 grafice explicative din care reies urmatoarele rezultate:

✓ NB este rapid in raport cu numarul de observatii, dar oscileaza mult in functie de

esantion (deviatie care totusi nu deranjeaza, fiind de ordinul milisecundelor)

✓ Performanta modelului stagneaza dupa un anumit timp de antrenare (fapt evident deoarece la un moment dat nu mai invata date utile si doar consuma secunde; chiar daca are observatii in plus, ele tind sa se repete)

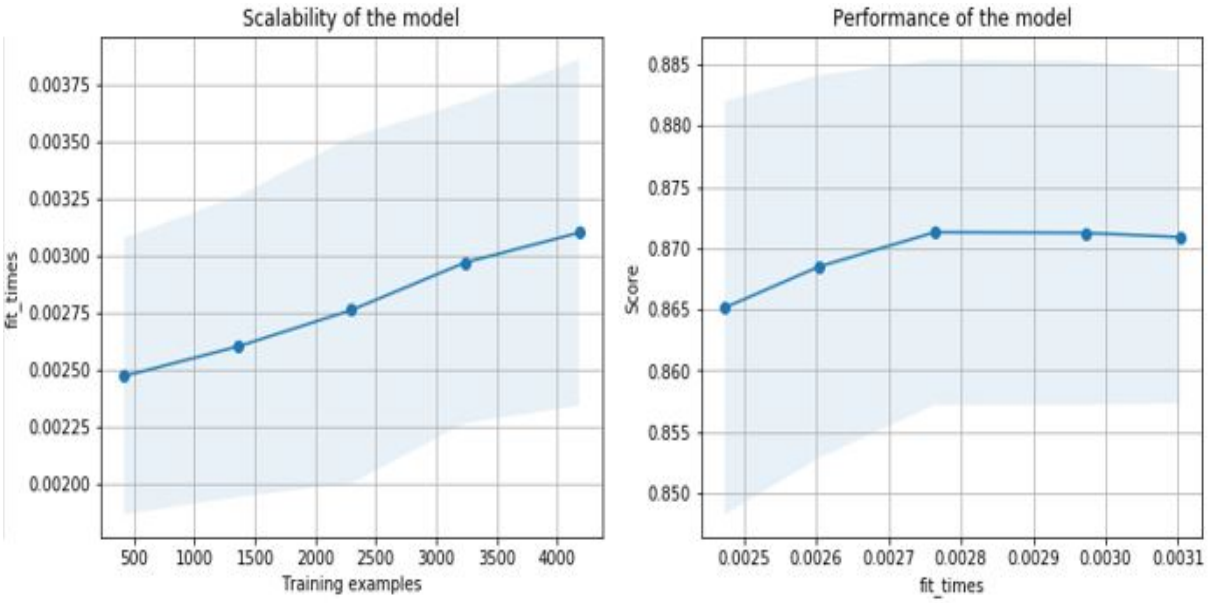


Figura 4.6: Scalabilitatea si performanta Naive Bayes

La nivel de statistici avem rezumatul (eng. *metric summary*) de mai jos:

```
---------------------------- Gaussian Naive Bayes ------------

Number of mislabeled points out of a total 5946 points : 838
The accuracy score is : 0.859065
[[4714  176]
 [ 662  394]]
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      4890
           1       0.69      0.37      0.48      1056

    accuracy                           0.86      5946
   macro avg       0.78      0.67      0.70      5946
weighted avg       0.84      0.86      0.84      5946

Balanced accuacy score:  0.6685571202825804
```

Figura 4.7: Rezumatul statisticilor Naive Bayes

Desi ***acuratetea generala*** este una buna, etichetele pozitive sunt clasificate corect intr-un numar prea mic, asa cum indica ***recall-ul*** de 37%. Cu alte acuvinte, doar 37 de procente din cele 1056 observatii de ***suport*** au tag-ul corespunzator. Totusi, ***precizia*** de 69% cu care intrarile (eng. *entries*) din top 10 sunt prezise e una relativ buna. Investitorii pot fi destul de siguri ca predictia algoritmului pentru cei din top 10 este una fondata pe adevar, si nu pe un proces aleator. Doar 31% ar pierde bani de pe seama

jucatorilor. ***Precizia*** este sinonima cu ***increderea*** iar o rata mare a acesteia atrage mai multe investitii deoarece oamenii vad potentialul de castig. Doua observatii principale pot fi formulate:

- ***Senzitivitatea*** modelului este importanta pentru dezvoltatori intrucat ei isi doresc cat mai multi jucatori corect prezisi in top 10
- ***Prezicia*** modelului este importanta pentru investitori intrucat le ofera increderea ca investitiile au sanse mari de a fi alocate jucatorilor ce vor termina in top 10

		Precizie	
		Mica	Mare
Senzitivitate	Mica	model slab	investitii mai putine, dar mari
	Mare	investitii mai multe, dar mici	model ideal

In cazul ***specificacitatii*** de 96% si a ***NPV*** de 88% e nevoie de o crestere a ***NPV***.

4.2.2 Regresie Logistica

Am ales *Regresia Logistica* drept urmatorul clasificator datorita legaturii puternice a acestuia cu *Naive Bayes*. Daca *NB* este un model **generativ** ce calculeaza distributiile $P(X|Y)$ si $P(Y)$, estimand parametrii lor, *Regresia Logistica* este echivalentul **discriminativ** care estimeaza in mod direct parametrii distributiei $P(Y|X)$. Totusi, echivalenta este reala doar **atunci cand prezumptia de independenta conditionala este adevarata**. O astfel de ipoteza ne-ar conduce spre rezultate foarte asemanatoare ale celor 2 algoritmi, mai ales daca setul de antrenament contine multe date a caror distributii vor tinde spre cele reale. Daca prezumtia nu este adevarata, *RL* va da predictii mai bune.

Ca si timp de executie, etapa de antrenare a regresiei logistice ar putea dura mai mult, in functie de rata de invatare (eng. *learning rate*) a gradientului descendent (eng. *Gradient Descent*) sau a numarului de parametri. Graficul de scalabilitate ne va da mai multe informatii.

Mai jos prezint "scurtatura" (eng. *trick*) pentru a ajunge de la *NB* la *RL* si rata de invatare, scalabilitatea si performanta algoritmului:

$$P(Y=1|X) = \frac{P(X|Y=1)P(Y=1)}{P(X|Y=1)P(Y=1)+P(X|Y=0)P(Y=0)} = \frac{1}{1+\frac{P(X|Y=0)P(Y=0)}{P(X|Y=1)P(Y=1)}} =$$

$$= \frac{1}{1 + \frac{(\prod_{i=1}^{n} p(X_i|Y=0))P(Y=0)}{(\prod_{i=1}^{n} p(X_i|Y=1))P(Y=1)}} = \frac{1}{1 + e^{ln(\prod_{i=1}^{n} \frac{p(X_i|Y=0)}{p(X_i|Y=1)}) \frac{P(Y=0)}{P(Y=1)}}} = \frac{1}{1 + e^{ln(\prod_{i=1}^{n} \frac{p(X_i|Y=0)}{p(X_i|Y=1)}) + ln(\frac{P(Y=0)}{P(Y=1)})}} =$$

$$p(X_i|Y=0) \sim \mathcal{N}(\mu_{i0}, \sigma_{i0}^2), p(X_i|Y=1) \sim \mathcal{N}(\mu_{i1}, \sigma_{i1}^2), \sigma_{i0}^2 = \sigma_{i1}^2 = \sigma_i^2, \pi = P(Y=1)$$

$$= \frac{1}{1+e^{ln\frac{1-\pi}{\pi} + \sum_{i=1}^{n} \frac{2X_i(\mu_{i0}-\mu_{i1}) + \mu_1^2 - \mu_{i0}^2}{2\sigma_i^2}}} = \frac{1}{1+e^{ln\frac{1-\pi}{\pi} + \sum_{i=1}^{n} \frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_1^2 - \mu_{i0}^2}{2\sigma_i^2}}} = \frac{1}{1+e^{ln\frac{1-\pi}{\pi} + \sum_{i=1}^{n} \frac{\mu_1^2 - \mu_{i0}^2}{2\sigma_i^2} + \sum_{i=1}^{n} \frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2} X_i}}$$

$$= \frac{1}{1+e^{w_0 + \sum_{i=1}^{n} w_i X_i}} = \sigma(w^T X)$$

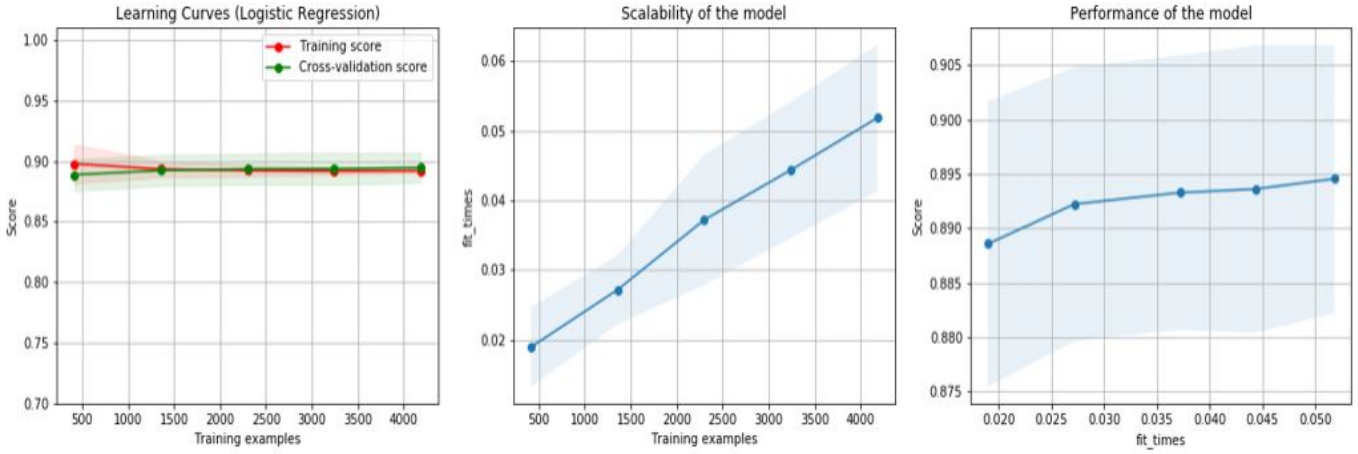


Figura 4.8: Performanta regresiei logistica

La antrenare am aplicat exact aceeasi tehnica de validare incrucisata ca la *Naive Bayes*, avand 90% date *train* si 10% date *validare*. Primul grafic este aproape identic cu cel de la figura [4.5], diferenta fiind data de acuratetea putin mai buna a regresiei. Comportamentul acesteia este, asadar, echivalent cu *NB* iar prezumtia de mai sus, adevarata. Scalabilitatea *LR*, desi mai putin oscilanta, scade, dovada ca metoda gradientului descendent este costisitoare. La nivel de statistici, obtinem o mica crestere:

```
---------------------------- Logistic Regression ------------

Number of mislabeled points out of a total 5946 points : 816
The accuracy score is : 0.862765
[[4811   79]
 [ 737  319]]
              precision    recall  f1-score   support

           0       0.87      0.98      0.92      4890
           1       0.80      0.30      0.44      1056

    accuracy                           0.86      5946
   macro avg       0.83      0.64      0.68      5946
weighted avg       0.86      0.86      0.84      5946

Balanced accuacy score:  0.6429639570552147
```

Figura 4.9: Performanta regresiei logistica

Recall-ul a scazut cu 7 procente, dar ***precizia*** este acum de 80%. Din punctul de vedere al investitorului, regresia logistica e cel mai bun model, oferindu-i increderea necesara. Totodata, ***specificacitatea*** creste putin, indicand o mai buna detectie a jucatorilor slabi.

4.2.3 Legatura dintre Naive Bayes si EM

Asa cum arata histogramele din figura [4.4], anumite atribute par sa aiba date generate de **2 distributii normale**. In acest sens, o optimizare a algoritmului *Naive Bayes* este evident necesara. Ideea este ca acolo unde se pot distinge vizual 2 distributii, sa se stabileasca mediile μ_1, μ_2 pentru ca mai apoi sa fie rulat algoritmul de ***Expectation Maximization*** (eng. *EM*) pentru mixturi de modele Gaussiene. Vectorul de medii returnat va fi folosit la calculul functiilor de densitate-probabilitate (pdf) iar observatiile vor fi asociate distributiilor pentru care se obtin valori mai mari ale pdf. Algoritmul ***EM*** este urmatorul:

Algorim: EM-GMM

INITIALIZARE

Se initializeaza vectorul de medii $\mu = [\mu_1, \mu_2]^T$

CORP ITERATIV (Pasul Estimativ, Pasul Maximizator)

pentru $t \leftarrow 1, ..., max\_iter$

pentru fiecare x^i *din training_set_2*

$$E[Z_j^i] = \frac{p(x^i|Z_j^i=1,\mu_j)}{\sum_{l=1}^{2} p(x^i|Z_l^i=1,\mu_l)} = \frac{e^{-\frac{(x^i-\mu_j)^2}{2\sigma^2}}}{\sum_{l=1}^{2} e^{-\frac{(x^i-\mu_l)^2}{2\sigma^2}}}$$

pentru $j \leftarrow 1, ..., 2$

$$\mu_j = \frac{\sum_{i=1}^{n} E[Z_j^i]x^i}{\sum_{i=1}^{n} E[Z_j^i]}$$

REZULTAT

Returneaza vectorul $\mu = [\mu_1, \mu_2]^T$

Fiecare componenta are propria sa varianta, mediile sunt initializate manual de catre mine, evitand astfel necesitatea ajustarii automate a unor *hyper-parametri* costisitori, iar *max_iter* reprezinta numarul maxim de iteratii ale algoritmului, si anume, 100 (prag specificat de mine). Daca convergenta este atinsa mai devreme, implementarea din *sci-*

kit se opreste inainte de 100, returnand valorile gasite la acel moment.
Atributele pe care am aplicat tehnica descrisa mai sus sunt: *damageDealt* (atat pentru top 10, cat si in afara lui), *walkDistance* (doar pentru cei din top 10) si *weaponsAcquired* (atunci cand jucatorii nu sunt in top 10). In total, avem 9 distributii pentru cele 3 caracteristici din set. Cu ajutorul acestora se vor calcula probabilitatile necesare algoritmului *Naive Bayes*. Statisticile hibridului sunt:

```
---------------------------- GMM Gaussian Naive Bayes --------

Number of mislabeled points out of a total 5946 points : 1517
The accuracy score is : 0.744871
[[3515 1375]
 [ 142  914]]
              precision    recall  f1-score   support

           0       0.96      0.72      0.82      4890
           1       0.40      0.87      0.55      1056

    accuracy                           0.74      5946
   macro avg       0.68      0.79      0.68      5946
weighted avg       0.86      0.74      0.77      5946

Balanced accuacy score:  0.7921721044803867
```

Figura 4.10: Performanta Naive Bayes cu modele de mixturi Gaussiene

Din seria tehnicilor de clasificare aplicate pana acum, aceasta se comporta cel mai bine, daca ar fi sa ne raportam la ***acuratetea echilibrata***. De asemenea, ***senzitivitatea*** pe eticheta pozitiva a crescut substantial, lucru ce indica predictia corecta a unui numar mai mare de jucatori care au terminat in top 10. Din pacate, ***precizia*** cu care sunt clasati competitorii de top a scazut la jumatate. Ne aflam pe linia a doua, prima coloana din tabelul de la pagina 33: investitii multe, dar mici. O observatie interesanta se poate face asupra ***NPV-ului*** de 96%. Practic, investitorilor le este aproape garantat ca cei prezisi in afara top 10, chiar sunt exclusi de pe podium. Desi ***specificacitatea*** a scazut, siguranta cu care sunt prezise etichetele nule este ideala. ***Scorul F***, adica media echilibrata dintre ***precizie*** si ***senzitivitate***, este cel mai ridicat pe eticheta pozitiva dintre cele 3 modele prezentate pana acum. Daca un investitor nu ar fi interesat doar de o singura masuratoare a calitatii unui algoritm, ***scorul F*** ii poate oferi o viziune de ansamblu, fiind o metrica combinata. Computational, acest model hibrid este mai lent intrucat algoritmul de ***EM-GMM*** este unul ce consuma timp aditional pentru estimarea mediilor caracteristicilor. O baza de date statica, cu relativ putine intrari, permite functionarea decenta a modelului, dar intr-un context *"Big-Data"*, cu baze de date dinamice, rularea algoritmului va deveni aproape imposibila. Asadar, timpul de executie are un rol esential iar o optimizare in acest sens este primordiala.

4.2.4 Random Forest

Algoritmul ***Random Forest*** este o altra strategie de tip *ansamblu*, la fel ca si algoritmul *AdaBoost*, prezentat mai sus. Se lucreaza iar cu mai multi estimatori, care aici sunt arbori de decizie cu mai mult de un nivel, spre deosebire de *AdaBoost*, unde lucram cu compasi de decizie. Aceasta observatie motiveaza complexitatea timp si spatiu crescuta a padurilor de arbori care folosesc memorie suplimentara pentru a retine structurile dense ale estimatorilor ce furnizeaza predictii intr-un timp mai mare. Totusi, exista un schimb (eng. *trade-off*) intre un predictor singular si unul asamblist: ***puterea de clasificare a ansamblului este mult mai mare decat cea a predictorului unic, dar resursele consumate de ansamblu cresc exponential***. Ramane astfel la latitudinea celui care face analiza datelor sa stabileasca ce primeaza.

Un concept esential in cadrul acestei sub-sectiuni este ***Bootstrapping-ul***, care reprezinta esantionarea aleatoare cu replasarea a unui set de observatii. Probabilitatea de alegere a intrarilor (eng. *entries*) urmeaza o distributie uniforma iar dimensiunea esantioanelor construite este mereu egala cu cea a *training_set-ului*.

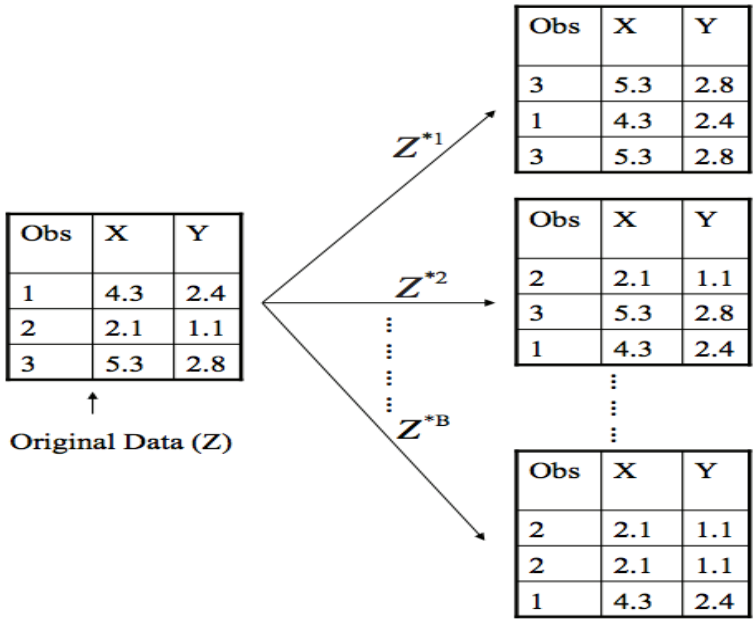


Figura 4.11: Exemplu de Bootstrapping

Bagging-ul (eng. *Bootstrap Aggregating*) foloseste fiecare esantion unic construit prin *Bootstrap* drept set de antrenament pentru fiecare clasificator unic din ansamblul. Predictia unei observatii este mai apoi data prin vot majoritar. Cu alte cuvinte, unei instante *"x"* ii va fi asociata cea mai comuna eticheta prezisa de arborii de decizie.

Algoritm: Pseudocod Bagging

INPUT

$\{(x_i, y_i)|i = 1, ..., n\}$-*datele de antrenament*

$B \in \mathbb{N}^*$- *numarul de estimatori*

CORP ITERATIV

pentru $b \leftarrow 1, ..., B$

a) genereaza un esantion Bootstrap de dimensiune n

b) construieste un clasificator η_b *folosind esantionul de mai sus drept set de antrenament*

PREDICTIE

Data o instanta x, asigneaza-i cea mai comuna eticheta din setul $\{\eta_b(x)|b = 1, .., B\}$

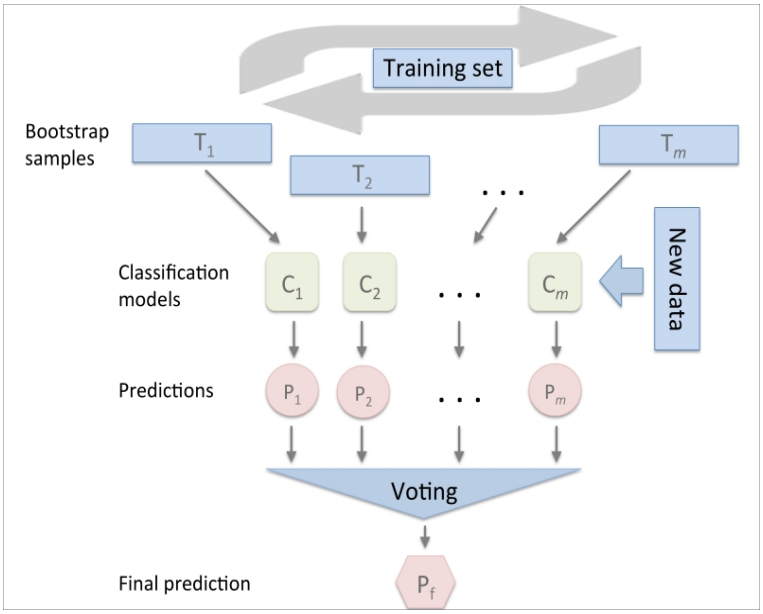


Figura 4.12: Reprezentare grafica a Bagging-ului

Totodata, Bagging-ul reduce ***varianta*** modelului, care se traduce prin scaderea *overfitting-ului*, fenomen intalnit extrem de des, mai ales atunci cand se lucreaza cu arbori de decizie. De asemenea, problema unui ***bias*** crescut este solutionata de numarul mare de estimatori ce alcatuiesc ansamblul (cu cat numarul lor este mai mare, cu atat cresc sansele ca predictiile sa fie cele corecte, datorita tendintei de aplatizare la o acuratete constanta a padurii de arbori). Ne aflam in scenariul ideal al experimentului *Darts*, si anume, coltul stanga sus:

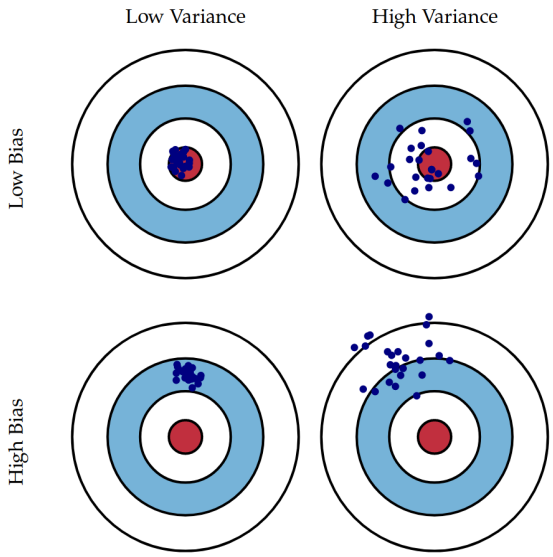


Figura 4.13: Experimentul ”Darts” cu privire la bias si varianta

Random Forest foloseste tehnica de bagging la care mai adauga un strat (eng. *layer*) de randomizare. Setul de atribute din care se poate alege in functie de cine se face divizarea unui nod este acum restrans la un subset de dimensiune mai mica. Caracteristicile din subset sunt alese aleator la fiecare predictor in parte. Pseudocodul algoritmului este acelasi cu cel de Bagging, dar are in plus *layer-ul* tocmai prezentat:

Algoritm: Pseudocod Random Forest

INPUT

$\{(x_i, y_i)|i = 1, ..., n\}$ - *datele de antrenament*

$B \in \mathbb{N}^*$- *numarul de estimatori*

$m \in \mathbb{N}^*$ - *cardinalul subsetului de caracteristici*

CORP ITERATIV

pentru $b \leftarrow 1, ..., B$

a) genereaza un esantion Bootstrap de dimensiune n

b) construieste un clasificator η_b *folosind esantionul de mai sus drept set de antrenament si alege la fiecare nod atributul cel mai bun din subsetul de caracteristicti alese aleator pentru aceasta iteratie*

PREDICTIE

Data o instanta x, asigneaza-i cea mai comuna eticheta din setul $\{\eta_b(x)|b = 1, .., B\}$

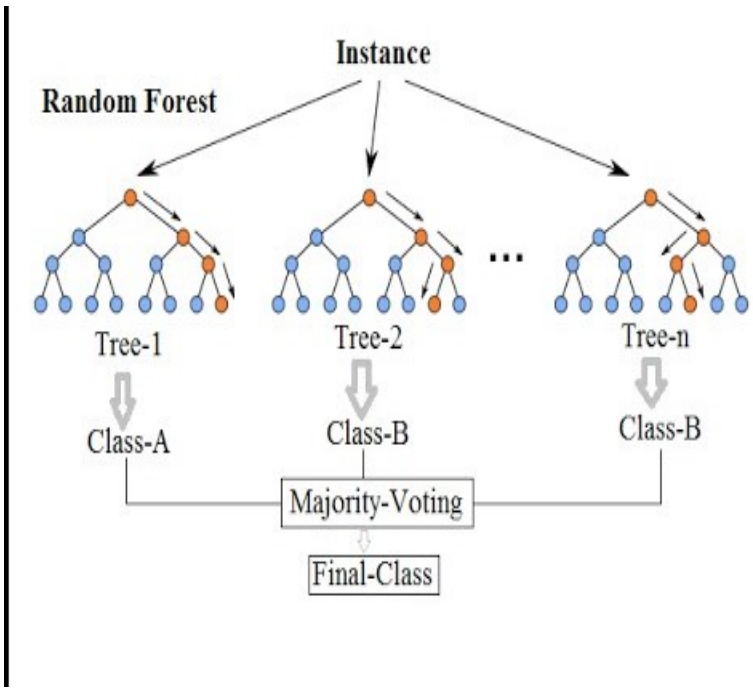


Figura 4.14: Reprezentare grafica a modului in care Random Forest functioneaza

Implementarea din *scikit-learn* dispune de o varietate mare de parametri care nu pot fi alesi decat printr-un ***GridSearch***, asa cum am facut si in cazul *AdaBoost*. Mecanismul de functionare este acelasi ca la [4.1]. Alaturi de numarul de estimatori si cardinalul subsetului de caracteristici, trebuie sa ne mai decidem asupra adancimii maxime a fiecarui predictor, a numarului minim de observatii necesare dintr-un nod intern pentru a realiza divizarea si daca echilibram setul prin ponderi sau nu.

```
grid_search = RandomizedSearchCV(
estimator = RandomForestClassifier(criterion = 'entropy'),
param_distributions = {
    'n_estimators' : scipy.stats.randint(1, 1001),
    'max_depth' : scipy.stats.randint(1, depth_full_tree),
    'min_samples_split' : scipy.stats.randint(2, 50),
    'min_samples_leaf' : scipy.stats.randint(1, 50),
    'max_features' : ["sqrt", "log2", None],
    'class_weight' : ['balanced', 'balanced_subsample']},
    n_iter = 1000,
    scoring = 'balanced_accuracy',
    n_jobs = -1, cv = None)
```

Concluzii

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nunc mattis enim ut tellus elementum sagittis vitae et. Placerat in egestas erat imperdiet sed euismod. Urna id volutpat lacus laoreet non curabitur gravida. Blandit turpis cursus in hac habitasse platea. Eget nunc lobortis mattis aliquam faucibus. Est pellentesque elit ullamcorper dignissim cras tincidunt lobortis feugiat. Viverra maecenas accumsan lacus vel facilisis volutpat est. Non odio euismod lacinia at quis risus sed vulputate odio. Consequat ac felis donec et odio pellentesque diam volutpat commodo. Etiam sit amet nisl purus in. Tortor condimentum lacinia quis vel eros donec. Phasellus egestas tellus rutrum tellus pellentesque eu tincidunt. Aliquam id diam maecenas ultricies mi eget mauris pharetra. Enim eu turpis egestas pretium.

Bibliografie

Bibliografie

[1] *https://www.kaggle.com*

[2] *https://www.kaggle.com/c/pubg-finish-placement-prediction*

[3] *https://www.stat.berkeley.edu/ breiman/randomforest2001.pdf*

[4] *https://web.stanford.edu/ hastie/Papers/samme.pdf*