

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**Studiul comparativ al unor modele de invatare
automata in cadrul jocului video PUBG**

propusă de

Tudor Manoleasa

Sesiunea: iulie, 2020

Coordonator științific

Dr. Eugen Croitoru

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

**Studiul comparativ al unor modele de
invatare automata in cadrul jocului
video PUBG**

Tudor Manoleasa

Sesiunea: iulie, 2020

Coordonator științific

Dr. Eugen Croitoru

Cuprins

Introducere	2
1 Planul de lucru	3
1.1 Metodologia CRSIP-DM	3
1.2 Planul analizei exploratorii	5
1.3 Masuratori statistice	6
1.3.1 Matricea de confuzie	6
1.3.2 TPR&TNR, PPV&NPV	6
1.3.3 Acuratetea	7
1.3.4 Acuratetea echilibrata	8
1.3.5 Graficul ROC	8
1.3.6 Scorul Silhouette	9
2 Intelegerea datelor si pregatirea lor	10
2.1 Operatii de preprocesare	11
2.2 Eliminarea valorilor aberante	12
2.3 Alegerea subsetului optim de attribute	12
2.3.1 Corelatia	14
2.3.2 Selectia de caracteristici cu Random Forest	15
2.3.3 Standardizare	16
2.3.4 Diminuarea dimensiunilor	17
3 Analiza nesupervizata	19
3.0.1 K-means	19
3.0.2 DBSCAN	22
4 Analiza supervizata	26
4.1 AdaBoost	27

4.2	Al doilea set de test	29
4.2.1	Naive Bayes	30
4.2.2	Regresie Logistica	33
4.2.3	Legatura dintre Naive Bayes si EM	35
4.2.4	Random Forest	37
4.2.5	Arbore de decizie & Masini vector suport	43
	Tehnologii Folosite	47
	Concluzii	48
	Bibliografie	49

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

În cadrul platformei Kaggle [1] se regăsește o competiție deschisă numită *"PUBG Finish Placement Prediction"* [2], ce are drept scop predicția locului pe care un jucător va termina un meci, în funcție de diferiți parametri din cadrul partidei. Întrucât există mai multe moduri de joc (*solo*, *duo*, *echipa de 3*, *echipa de 4*), seturile de date de care dispun conțin statistici pentru toate tipurile enunțate între paranteze. Mă voi concentra doar asupra meciurilor *solo* (eng. *single-player*), încercând să prezic cine va ajunge în primii 10 și cine nu. Atât setul de antrenament original cât și cel de test original conțin milioane de interogări. Am ales să le esantionez pe ambele, construind 2 seturi auxiliare, fiecare a câte 80.000 de intrări și 29 (28 + coloana de ieșire) 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)* [3], standardul din industrie în materie de proiecte ce vizează lucrul cu date. Modelul surprinde ciclul de viață al unui produs analitic ca al meu, nefiind foarte restrictiv cu privire la etapele pe care trebuie să le urmez.



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 attributele 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* [?], 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 reducere 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 grupuri, 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 surprinde 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 metrice 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* [4] 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.
- **TN (ENG. *True Negatives*):** Numarul de intrari clasificate negativ (nu sunt in top 10) care sunt intr-adevar negative.

		Realitate		Total
		Pozitiv	Negativ	
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 pre-zise 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 **specificitate** (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. *Specificitatea* 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:

$$BACC = \frac{\text{senzitivitate} + \text{specificitate}}{2}$$

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* [5] 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, $[0, 1]$.

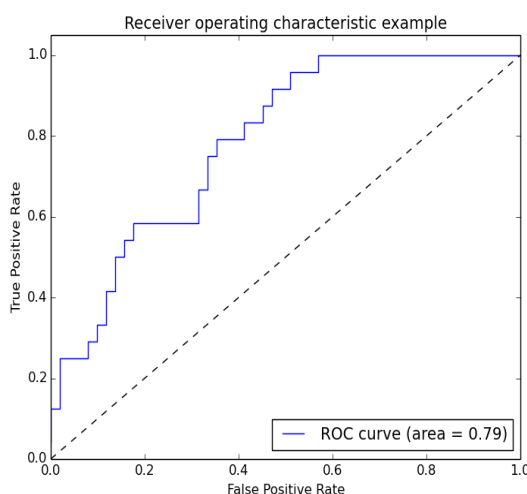


Figura 1.1: Exemplu de grafic ROC

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. *thresholds*) 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 $threshold = 85\%$ atunci intrarea va fi clasificata

negativ. Pentru toate thresholdurile generate, obținem matrici de confuzie diferite a caror FPR și TPR constituie un nou punct pe grafic. După ce am epuizat numărul de praguri, unim punctele de pe grafic pentru a obține curba zimțată albastră. Același lucru îl putem face și cu un alt algoritm, obținând o a doua curba zimțată și urmând ca la final să calculăm ariile suprafețelor cuprinse de cele două curbe. Empiric, aria mai mare este specifică clasificatorului mai bun.

Ca observație, linia punctată marchează locurile unde $TPR = FPR$. De asemenea, dacă se dorește aflarea threshold-ului ideal pentru un algoritm, utilizatorul ar trebui să urmărească punctul curbei care se apropie cel mai mult de colțul stanga-sus al imaginii. În acel colț, sensibilitatea este maximă, $TPR = 1$, iar FPR este minim, $FPR = 0$.

1.3.6 Scorul Silhouette

Spre deosebire de învățarea supervizată unde metricile sunt calculate în funcție de acuratețea predicțiilor făcute pe coloana ieseire, la analiza nesupervizată acest lucru devine imposibil, eticheta de *output* fiind inexistentă. În plus, scopul algoritmilor nesupervizați este de a găsi sabloane explicative cu privire la datele de care dispunem. Prin urmare, metrica ideală ar trebui să măsoare gradul de separare a partițiilor găsite, similaritatea sau diferențele componentelor din cluster etc.

Scorul Silhouette [6] este cel pe care l-am folosit pe tot parcursul proiectului întrucât cuprinde cele 2 proprietăți ideale de mai sus. O parte din avantaje și dezavantaje sunt:

- Scor ușor de interpretat, cuprins între $[-1,1]$, unde -1 semnifică clusterizare incorectă, 1 clusterizare ideală iar 0 marchează partiții care se suprapun
- Un scor ridicat indică cluster dense și bine separate
- Scorul este de obicei mai mare pentru cluster convexe și mai mic pentru alte tipuri de cluster, precum cele bazate pe densitate

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

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

b = distanța medie dintre o observație și toate celelalte observații din cel mai apropiat cluster

a = distanța medie dintre o observație și toate celelalte observații din același cluster

Capitolul 2

Intelegerea 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 sub-tipuri. Imaginea urmatoare surprinde cel mai bine partitionarea despre care voi vorbi in continuare:

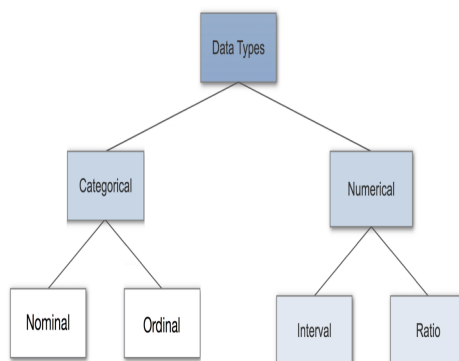


Figura 2.1: Tipuri de date

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** - aceeași semnificație ca la cele nominale, diferența fiind că ordinea contează. (ex. satisfacția unui client pe o scară 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
- 2 coloane categoriale nominale si 2 categoriale ordinale 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 attribute specifice echipelor, fie attribute 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

În descrierea seturilor de pe *Kaggle* se menționează că pentru coloanele *'damageDealt'*, *'longestKill'*, *'walkDistance'* și *'weaponsAcquired'* au fost observate și valori aberante, datorate *hacker-ilor* (rom. trisori). Pentru a elimina aceste situații folosesc metoda *intervalului dintre cuantile* [7].

Prima dată, pentru fiecare coloană enumerată mai sus, se calculează diferența dintre cunatila 3 (granita până unde se află 75% din date) și cuantila 1 (granita până unde se află 25% din date) $IQR = Q3 - Q1$. Mai apoi construiesc capetele de interval : $[Q1 - 3 \cdot IQR, Q3 + 3 \cdot IQR]$. Intrările specifice valorilor atributelor, abia menționate, care depășesc intervalul sunt sterse.

Deși în mod normal se lucrează cu regula de $1.5 \cdot IQR$, am ales-o pe cea cu $3 \cdot IQR$ din dorința de a fi mai puțin punitiv cu datele. Dacă aș fi păstrat $1.5 \cdot IQR$ riscam să elimin multe observații din top 10, lucru de nedorit întrucât avem un set de antrenament foarte *inbalanced* (rom. inegal). Desigur, acum s-ar putea spune că există șanse mult mai mari să fi păstrat *hackeri*. Un lucru adevărat, dar acest impediment nu încurcă prea mult deoarece numărul de *hackeri* este unul relativ mic, deci eliminarea doar acelor cazuri extreme este una convenabilă.

2.3 Alegerea subsetului optim de attribute

O problemă universală cu care se confruntă orice inginer pe parcursul unei analize exploratorii este selectarea acelor caracteristici (eng. *features*) care sunt cu adevărat relevante și eliminarea celor redundante sau inutile. Între **redundanta** și **inutilitate** există o diferență majoră deoarece putem avea attribute utile, dar redundante în raport cu altele, în timp ce attributele inutile nu reflectă nimic (ex. coloanele eliminate la 2.4).

Avantajele pastrării 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 existente*
- *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 [8] 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:

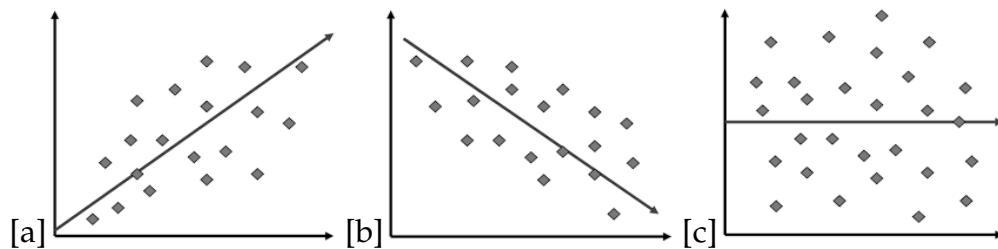


Figura 2.2: (a) Corelatie pozitiva (b) Corelatie negativa (c) Fara corelatie

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 attributele 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:

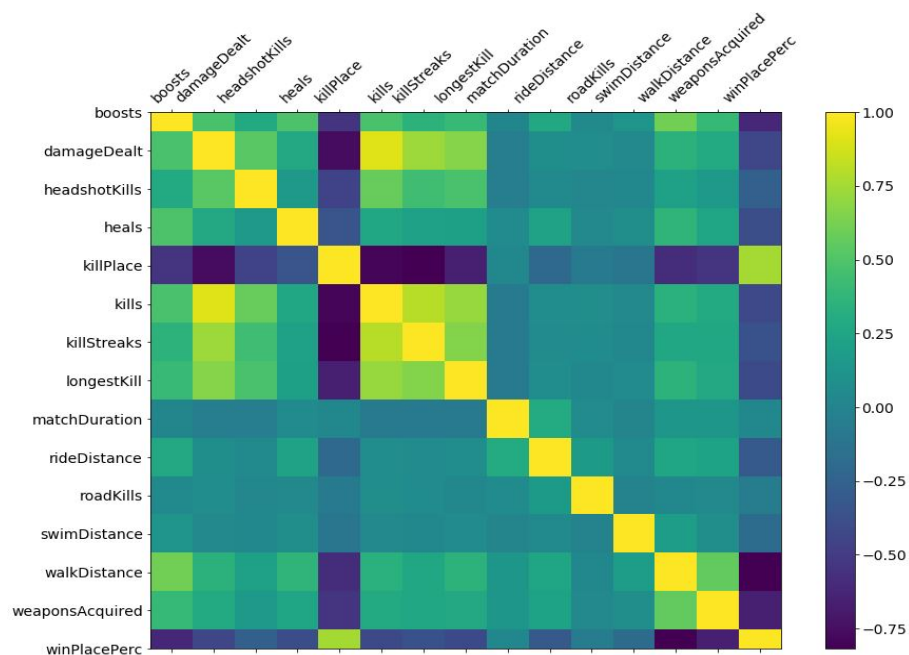


Figura 2.3: Tipuri de date

2.3.2 Selectia de caracteristici cu Random Forest

Asa cum reiese 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 attribute pentru care se calculeaza scoruri astfel [9]:

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

$feature_imp[f] = \frac{\sum_{i \in \text{all nodes that split on } f} node_imp[i]}{\sum_{k \in \text{all nodes}} node_imp[k]}, \forall f \in \text{all features}$

$normalised(feature_imp[f]) = \frac{feature_imp[f]}{\sum_{j \in \text{all feature}} feature_imp[j]}$

$RF_feature_imp[f] = \frac{\sum_{j \in \text{all trees}} normalised(feature_imp[f])}{T}$

✓ T = numarul total de arbori din ansamblu

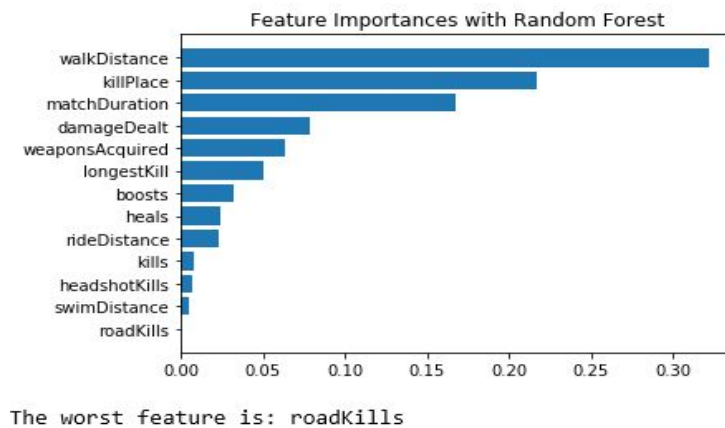


Figura 2.4: Scorurile feature-urilor din **train.csv**, calculate de Random Forest

2.3.3 Standardizare

Un alt aspect important îl reprezintă scalarea caracteristicilor (eng. *Feature Scaling*). În majoritatea cazurilor, unitățile de măsură ale atributelor sunt diferite, motiv pentru care magnitudinile observațiilor sunt foarte variate. De exemplu, greutatea unei persoane ia valori, în general, de la 4 la 100 de kilograme în timp ce aceeași greutate în grame poate fi și de 1000 de ori mai mare ca magnitudine. Algoritmii de învățare automată care calculează distanțe euclidiene sau realizează *Gradient Descent* au nevoie de scalare. În cazul distanțelor, caracteristicile cu o magnitudine mare au o pondere mai semnificativă în momentul realizării calculului, problema aceasta fiind rezolvată de scalare. De asemenea, în cazul *Gradient Descent*-ului, procesul de "coborare" este mai rapid.

În literatura de specialitate, se face distincția între 2 moduri generale de a face *Feature Scaling*: **Standardizare** și **Normalizare**. Normalizarea scalează caracteristicile în așa fel încât ele să aibă valori cuprinse în intervalul $[0, 1]$. Standardizarea scalează caracteristicile pe baza distribuției normale, fiecare *feature* având în final o medie de $\mu = 0$ și deviația standard $\sigma = 1$. Empiric, a fost stabilit că cea de-a doua metodă funcționează în practică mai bine.

Transformarea pe care o voi realiza în cadrul acestei lucrări este una de standardizare, numită *Yeo-Johnson* [10]. Facând parte din familia de transformări parametrice monotone, *Yeo-Johnson* încearcă să-și mapeze datele într-o distribuție normală, cu scopul de a reduce varianța și a minimiza skewness. Această mapare se va dovedi utilă ulterior, atunci când voi aplica algoritmul **Naive Bayes** care face prezumția că variabilele de intrare urmează o distribuție Gaussiană. De asemenea, standardizarea este de folos și următorului sub-capitol, de diminuare a dimensiunilor, tehnica folosită acolo având nevoie de scalarea atributelor. Iată transformarea *Yeo-Johnson*, formulată 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 parametrizată de λ care se obține prin estimarea probabilității maxime (eng. *maximum likelihood estimation*). Iată cum arată histogramele datelor înainte și după 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** [11]. 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 conform unei metricei (ex. *Ward-linkage*)

→ Se unesc clusterelor 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 attributele 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 aceasta 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:

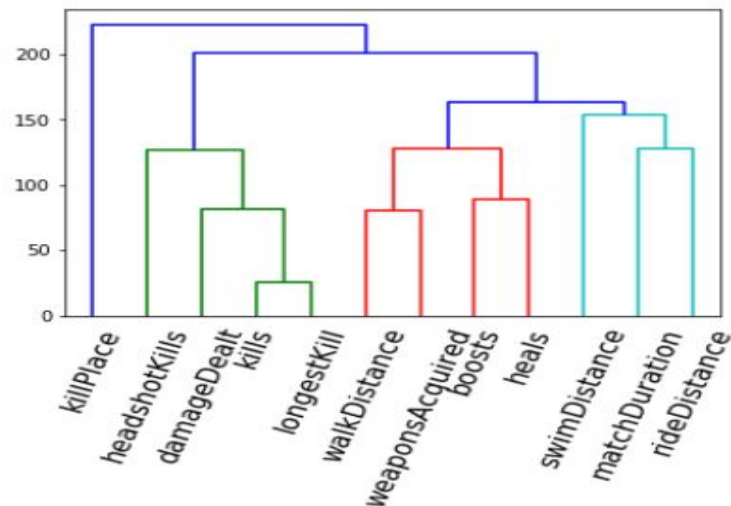


Figura 2.5: Ierarhia de caracteristici

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 clusterelor cu putine elemente pentru a reduce sansele ca la o prima taiere a arborului ierarhizat sa obtinem cluster *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 surprinde combinatia dintre attributele 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 attributele 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:

Algorithm: K-means [12]

INITIALIZARE

Alege numarul K de clustere dorite

Initializeaza centroizii μ^1, \dots, μ^K a celor K clustere cu observatii alese random

CORP ITERATIV

repetă

→ pentru $\forall x^i \in \text{training_set}$ calculează $d(x^i, \mu^j)$, $j \in 1..K$

→ asignează fiecare observatie x^i clusterului C^j pentru care $d(x^i, \mu^j)$ este minima

→ recalculează centroizii $\mu^j = \frac{\sum_{e \in C^j} e}{|C^j|}$

cat timp o conditie este satisfacuta

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 algoritmului 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++**, algoritmul 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 clusterurile. Mai jos este pseudocodul corespunzator:

Algorim: K-means++ [12]

INITIALIZARE

Stabileste numarul de cluster K dorite

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

CORP ITERATIV

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

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

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

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

REZULTAT

Returneaza $\mu = [\mu^1, \dots, \mu^K]^T$

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

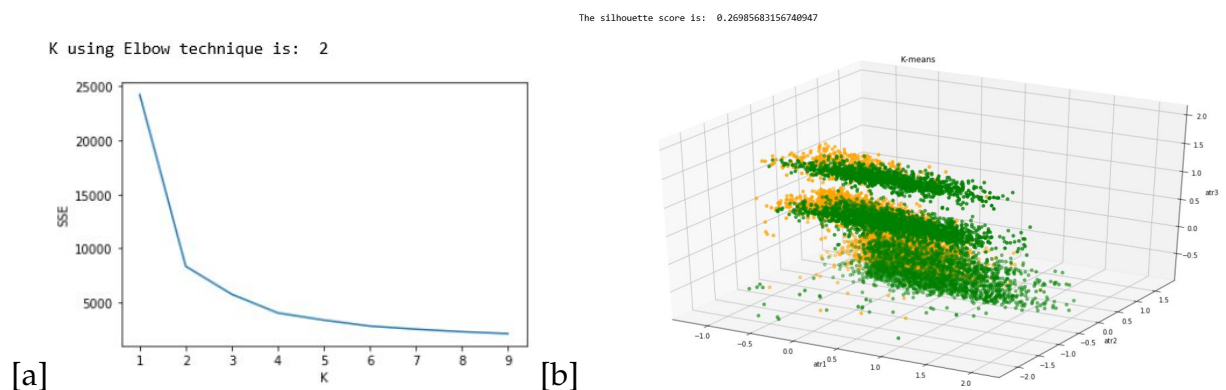


Figura 3.1: (a) Valoarea optima K cu tehnica Elbow (b) Rezultate K-means pe date

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

par sa surprinda foarte bine forma datelor. Posibilele principale motive ce stau la baza esecului 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 [13] (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, \dots, 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 $minPts$

× Senzitivitate maxima la ϵ si $minPts$

× 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 $i \leftarrow 1, \dots, n$

$label[i] = undefined$

CORP ITERATIV

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

daca $label[p] \neq undefined$

continua

$N_\epsilon[p] = RangeQuery(p)$, *RangeQuery* e functia ce returneaza vecinii din raza ϵ

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

$label[p] = noise$

continua

$S = N_\epsilon[p]$; $C = C + 1$; $label[p] = C$

pentru $q \in S$

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

continua

$label[q] = C$

$N_\epsilon[q] = RangeQuery(q)$

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 acest 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 de sus al graficului, cum se observa si in imaginea de mai jos.

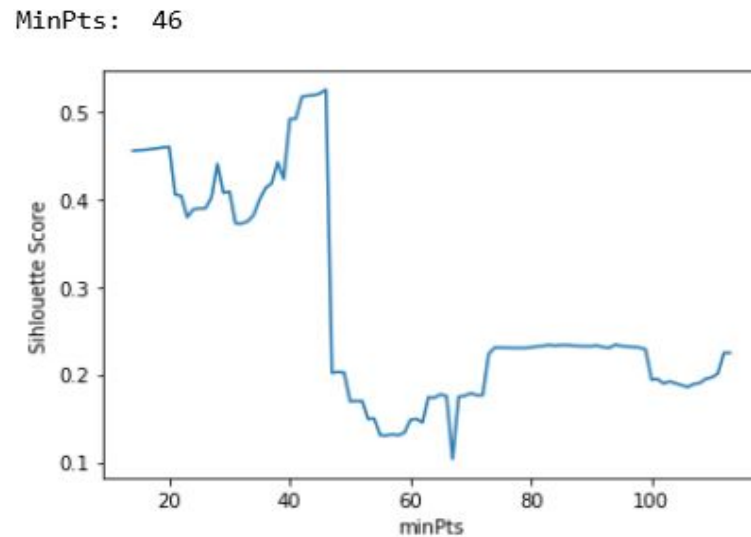


Figura 3.2: Cautarea valorii optime a lui minPts in functie de ϵ gasit deja

Number of points that are noise: 216 out of 11366
 [0 1 2]
 The cluster with the worst average placement: orange
 The silhouette score is: 0.44936189353465983

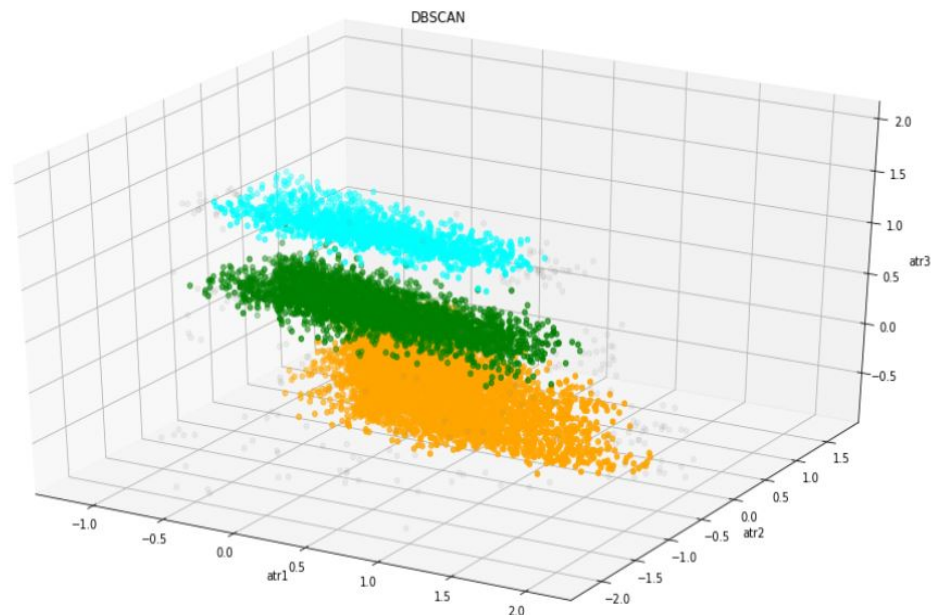


Figura 3.3: Rezultatul clusterizarii cu DBSCAN (punctele gri reprezinta *Outliers*)

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, clusterelor **azur** si **verde** sunt combinate in cadrul de date (eng. *DataFrame*) *training_set_1* iar cel **portocaliu** (clusterul 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 cluster, 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:

Out of 6720 samples that are in the worst cluster, only 147 are in top 10.

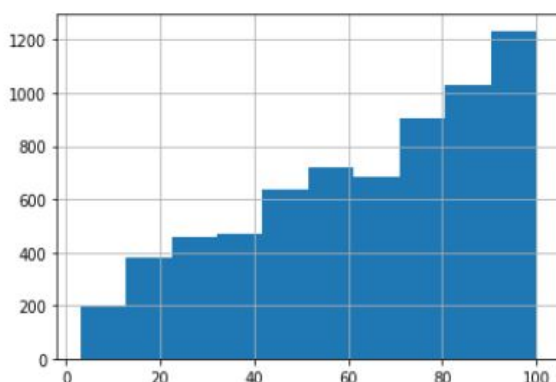


Figura 4.1: Locurilor ocupate la finalul meciului de jucatorii din clusterul portocaliu

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. *thresholds*) pe care algoritmul le considera importante
- ✓ Trateaza usor atributele irelevante pe care le inlatura la fiecare nod, dar le reconsidereaza 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 specificitate 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 aproape 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 informatiile 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 predictorii 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 attribute continue si converge relativ repede la un optim datorita functiei 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 [14]:

Algorim: AdaBoost

INITIALIZARE

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

CORP ITERATIV

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

→ 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

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

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': 2.001176999026374, '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.

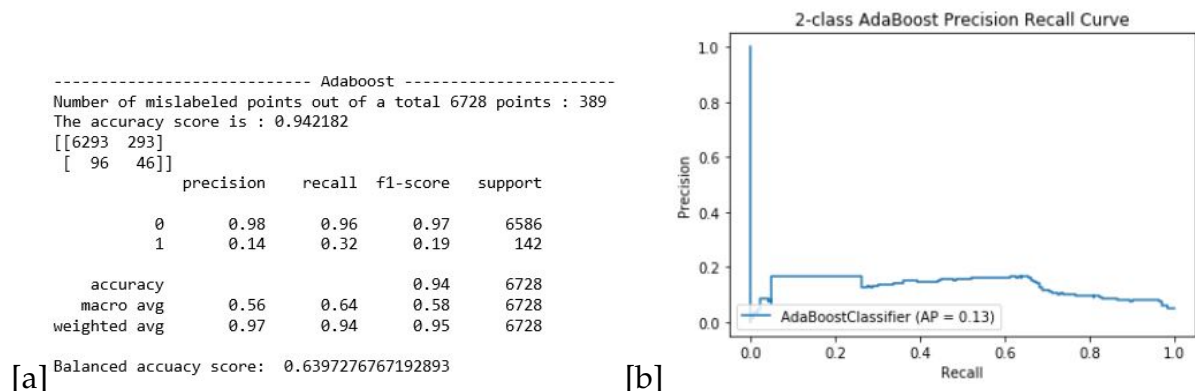


Figura 4.3: (a) Statistici AdaBoost (b) Graful Precision-Recall K-means pe date

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 cluster), 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 special pus pe aceasta 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* [15] (presc. *NB*) bazat pe *attribute continue* ar putea fi un predictor puternic atat din punct de vedere a timpului de executie, cat si a acuratetii.

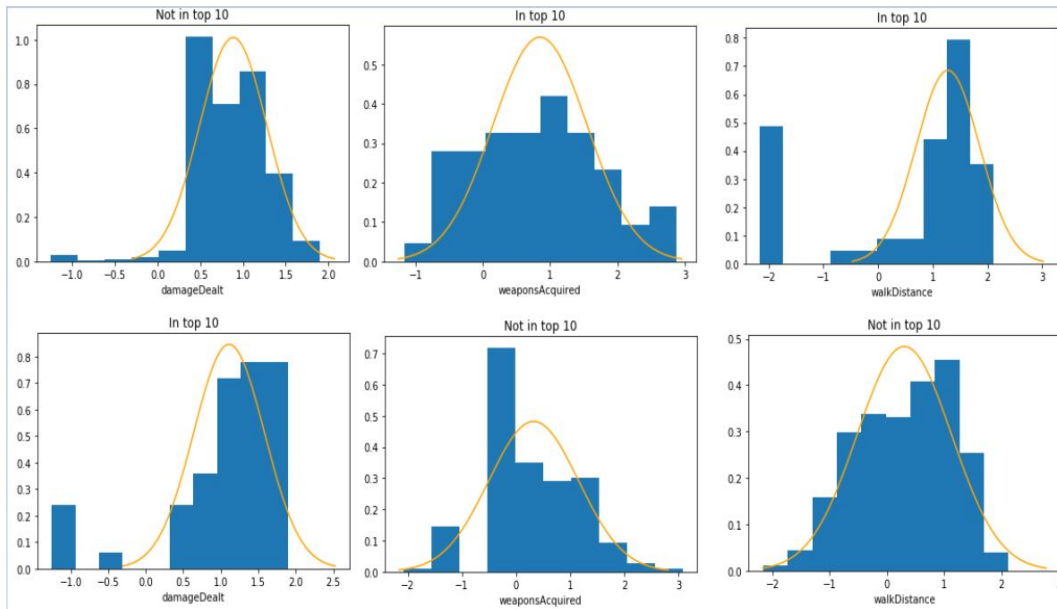


Figura 4.4: Histogramele atributelor in functie de etichete

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 probabilitate 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 urmatoar:

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

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

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?*".

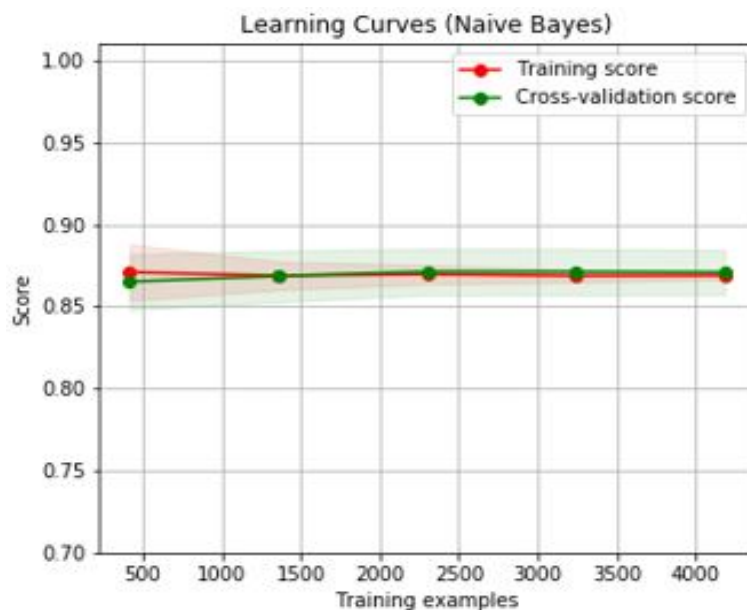


Figura 4.5: Graficul Learning Curve in cazul Naive Bayes

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 deviatii de la medie a celorlalte esantioane. Ambele linii sunt foarte apropiate, sugerand faptul ca Naive Bayes nu face overfitting, 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)

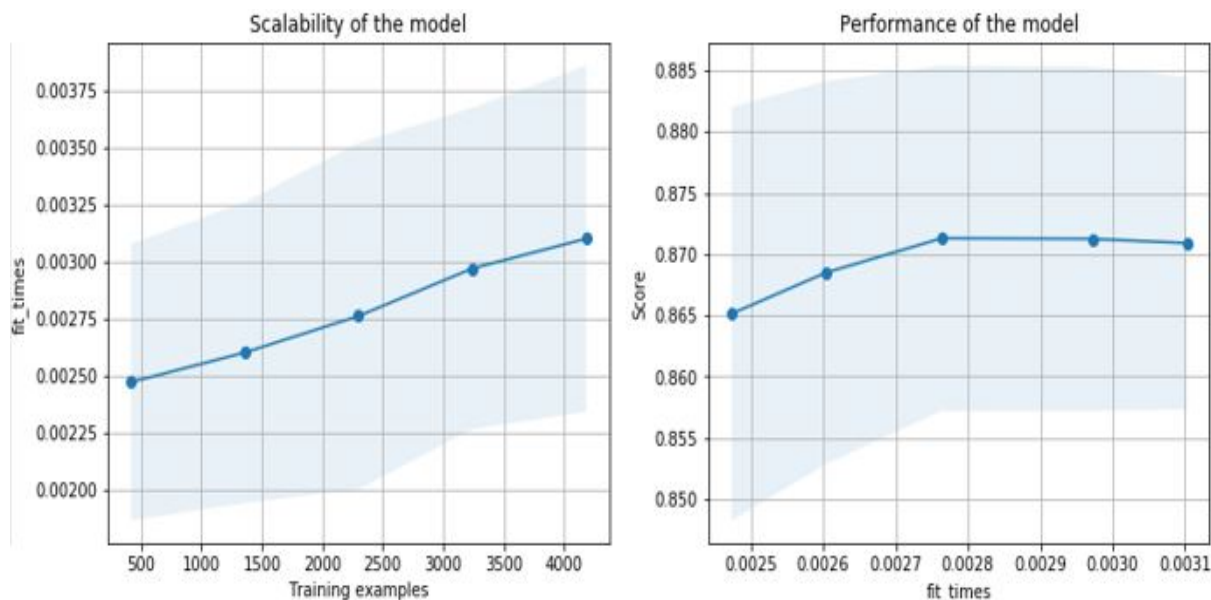


Figura 4.6: Scalabilitatea si performanta Naive Bayes

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 accuracy 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 acestuia 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
- **Precizia** 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 **specificitatii** de 96% si a **NPV** de 88% e nevoie de o crestere a **NPV**.

4.2.2 Regresie Logistica

Am ales *Regresia Logistica* [16] 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 prezumptia 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:

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

$$\begin{aligned}
&= \frac{1}{1 + \frac{(\prod_{i=1}^n p(X_i|Y=0))P(Y=0)}{(\prod_{i=1}^n p(X_i|Y=1))P(Y=1)}} = \frac{1}{1 + e^{\ln(\prod_{i=1}^n \frac{p(X_i|Y=0)}{p(X_i|Y=1)}) \frac{P(Y=0)}{P(Y=1)}}} = \frac{1}{1 + e^{\ln(\prod_{i=1}^n \frac{p(X_i|Y=0)}{p(X_i|Y=1)}) + \ln(\frac{P(Y=0)}{P(Y=1)})}} = \\
&p(X_i|Y=0) \sim \mathcal{N}(\mu_{i0}, \sigma_{i0}^2), p(X_i|Y=1) \sim \mathcal{N}(\mu_{i1}, \sigma_{i1}^2), \sigma_{i0}^2 = \sigma_{i1}^2 = \sigma_i^2, \pi = P(Y=1) \\
&= \frac{1}{1 + e^{\ln \frac{1-\pi}{\pi} + \sum_{i=1}^n \frac{2X_i(\mu_{i0}-\mu_{i1}) + \mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2}}} = \frac{1}{1 + e^{\ln \frac{1-\pi}{\pi} + \sum_{i=1}^n \frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2}}} = \frac{1}{1 + e^{\ln \frac{1-\pi}{\pi} + \sum_{i=1}^n \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} + \sum_{i=1}^n \frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2} X_i}} \\
&= \frac{1}{1 + e^{w_0 + \sum_{i=1}^n w_i X_i}} = \sigma(w^T X)
\end{aligned}$$

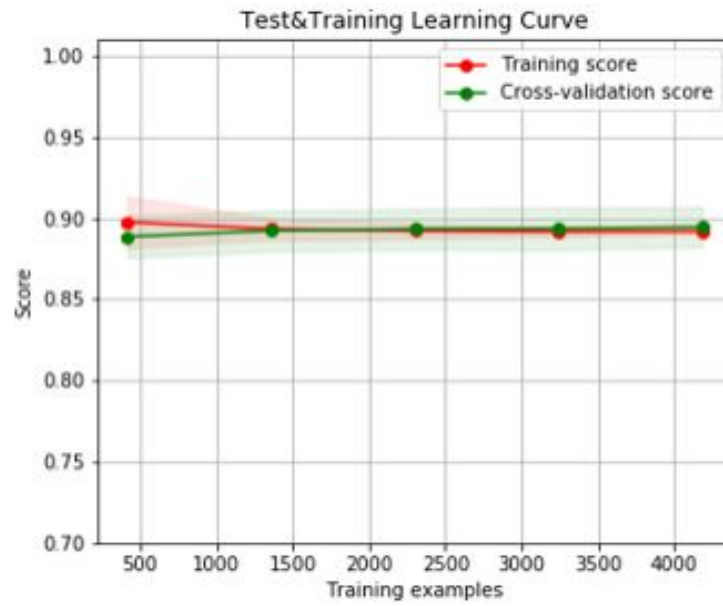


Figura 4.8: Linia de invatare a regresie logistice

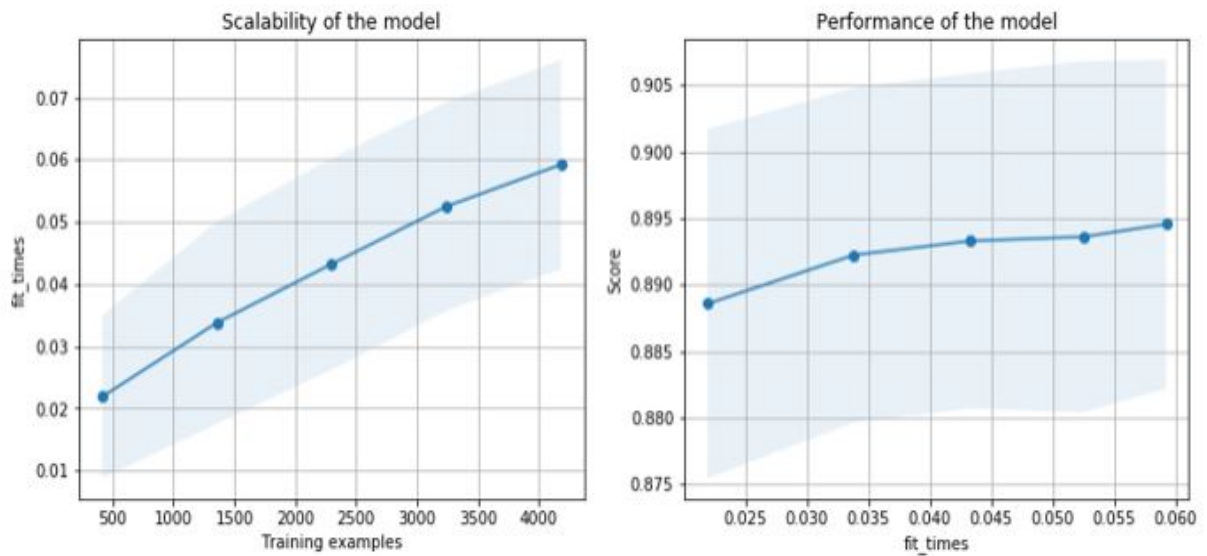


Figura 4.9: Scalabilitatea si performanta regresiei logistice

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 accuracy score: 0.6429639570552147

```

Figura 4.10: Performanta regresiei logistica

Recall-ul a scazut cu 7 procente, dar *precizia* este acum de 80%. Aceasta observatie se traduce in felul urmatoar: desi putine intrari pozitive sunt precise corect din totalul celor pozitive, algoritmul nu prezice multe observatii drept fals pozitive, fiind foarte atent la raspunsul pe care il da in cazul etichetelor nenule. Din punctul de vedere al investitorului, regresia logistica e cel mai bun model (in comparatie cu *NB*), oferindu-i increderea necesara. Totodata, *specificitatea* 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 [17]

INITIALIZARE

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

CORP ITERATIV (Pasul Estimativ, Pasul Maximizator)

pentru $t \leftarrow 1, \dots, max_iter$

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 $j \leftarrow 1, \dots, 2$

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

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 accuracy score: 0.7921721044803867
```

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 *specificitatea* 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* [18] este o alta 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.

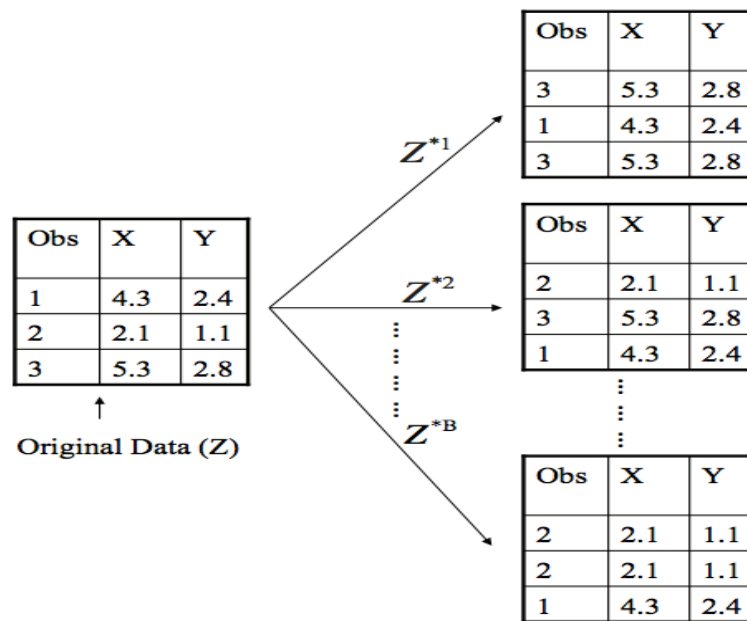


Figura 4.11: Exemplu de Bootstrapping

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.

Algorithm: Pseudocod Bagging [19]

INPUT

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

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

CORP ITERATIV

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

a) genereaza un esantion Bootstrap de dimensiune n

b) construiește 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 $\{\eta_b(x) | b = 1, \dots, B\}$

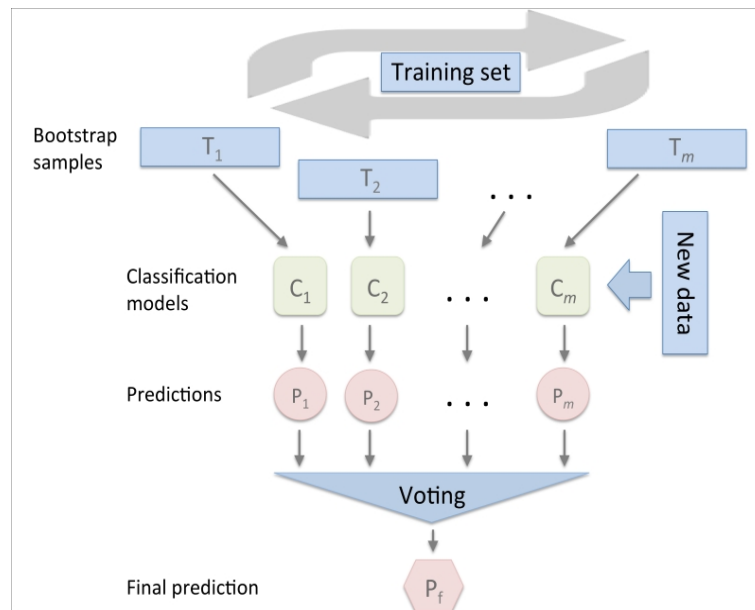


Figura 4.12: Reprezentare grafica a Bagging-ului

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:

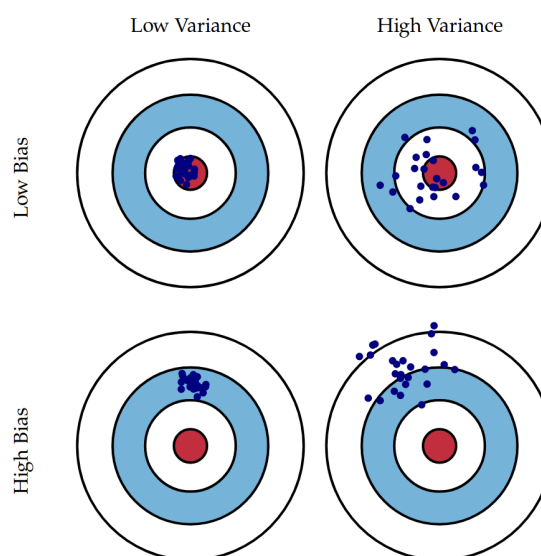


Figura 4.13: Experimentul "Darts" cu privire la bias si varianta

Random Forest foloseste tehnica de bagging la care mai adauga un strat (eng. *layer*) de randomizare. Setul de attribute 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:

Algorithm: Pseudocod Random Forest [19]

INPUT

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

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

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

CORP ITERATIV

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

a) genereaza un esantion Bootstrap de dimensiune n

b) construiește un clasicator η_b folosind esantionul de mai sus drept set de antrenament si alege la fiecare nod atributul cel mai bun din subsetul de caracteristici alese aleator pentru aceasta iteratie

PREDICTIE

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

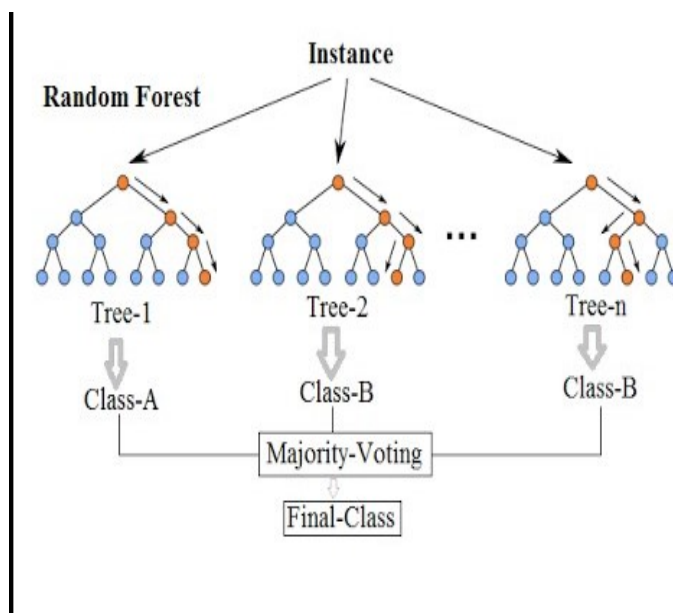


Figura 4.14: Reprezentare grafica a modului in care Random Forest functioneaza

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 echilibrăm 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)
```

Asa cum se observa, am evitat setarea manuala a unui interval de valori pentru parametri, in favoarea unui generator automat dintr-o distributie uniforma folosind (*scipy.stats.randint*) pentru fiecare hiper-parametru. Totodata, prin setarea lui *n_jobs = -1* am folosit toate core-urile procesorului pentru a reduce timpul de executie a cautarii prin distributia task-urilor pe mai multe nivele ale CPU. Rezultatele obtinute pe baza parametrilor gasiti de *GridSearch* sunt:

```
----- Random Forest -----
Number of mislabeled points out of a total 5946 points : 845
The accuracy score is : 0.857888
[[4330  560]
 [ 285  771]]
      precision    recall  f1-score   support

         0         0.94      0.89      0.91      4890
         1         0.58      0.73      0.65      1056

   accuracy          0.86          0.86      5946
  macro avg          0.76      0.81      0.78      5946
weighted avg          0.87      0.86      0.86      5946

Balanced accuracy score: 0.8077971044803867
```

Scorurile F sunt cele mai mari, pe ambele etichete, in raport cu intreaga gama de algoritmi folositi in lucrarea de fata. De asemenea, *acuratetea echilibrata* este si ea maximala in raport cu tehnicile precedente. 73% dintre etichetele cu adevarat pozitive au fost prezise corect cu o precizie, relativ medie, de 58%. Practic, investitorul se afla intr-o situatie destul de incerta atunci cand pariaza, avand sansa de 42% sa finanteze un jucator prezis fals pozitiv. In contrast, increderea in predictia jucatorilor din afara top 10 este una foarte mare, lucru intalnit totusi si la modelele anterioare.

In continuare voi vorbi de performanta computationala a modelului, bazandu-ma pe graficele prezentate in urmatoarea pagina (metoda prin care au fost realizate este cea descrisa la pagina 31 de la sectiunea [4.2.1]). In figura de la [4.15] ambele linii de pe acest prim graf sunt apropiate, dovada ca algoritmul nu face overfitting si nici nu are un comportament imprevizibil (caz in care umbrele ar fi fost mult mai dense). Totodata, modelul stagneaza de la un punct incolo deoarece datele de antrenament se repeta. A se observa pe cel de-al doilea grafic [4.16] cum timpul de executie creste substantial odata cu dimensiunea setului de antrenament, motivul fiind numarul mai mare de praguri de decizie ce sunt create si luate in considerare si durata mai lunga de constructie a unor arbori cu mai multe nivele (daca nu este setata o adancime maxima). Performanta din chenarul secundar de la [4.16] stagneaza si ea, indiferent de timpul de executie mai ridicat asociat unui set de antrenament mai mare. De asemenea, dupa o anumita valoare pentru numarul de estimatori din cadrul ansamblului, performanta nu va mai creste.

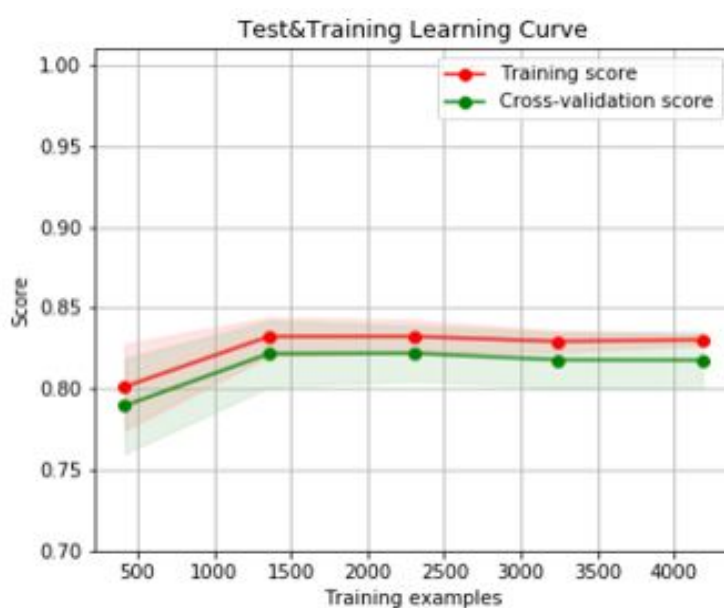


Figura 4.15: Linia de invatare a Random Forest

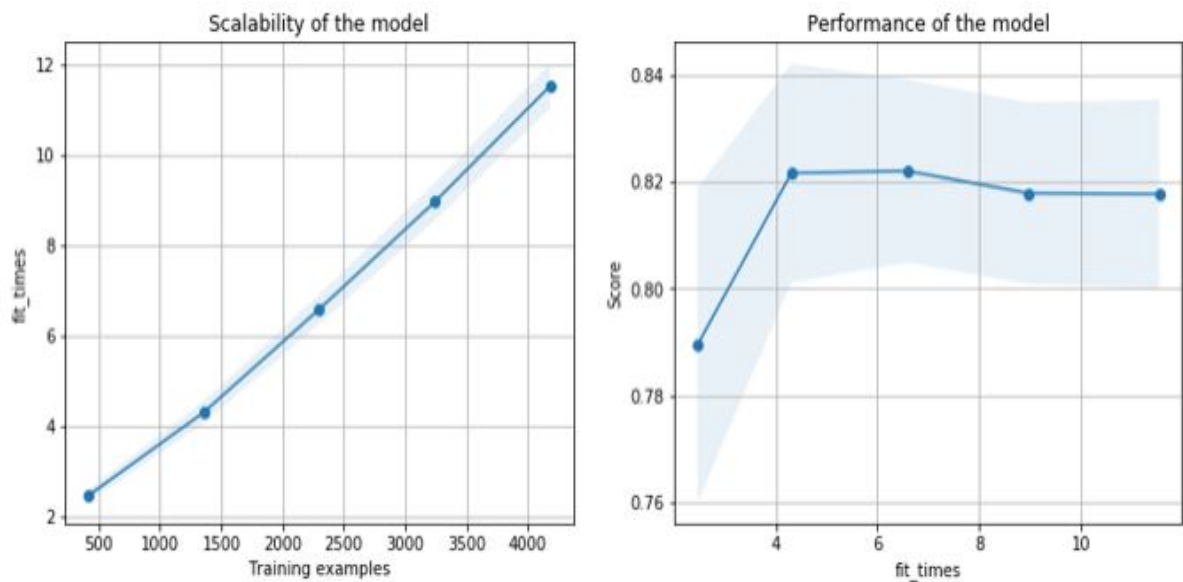


Figura 4.16: Performanta si scalabilitatea a Random Forest

4.2.5 Arbore de decizie & Masini vector suport

Unul dintre cei mai utilizati algoritmi de invatare automata este arborele de decizie (eng. *Decision Tree*) [19]. Practic atat in industrie, cat si la nivel teoretic, un astfel de model sta la baza intelegerii unor concepte derivate precum *Boosting*, *Random Forest* etc. Arborele de decizie este un clasificator neliniar ce construiesc o structura arborescenta a carei noduri surprind diferite praguri (eng. *thresholds*) ale atributelor de intrare din setul de antrenament. Pe baza acestor praguri se face divizarea unui nod. Fiind un algoritm greedy, *decision tree-ul* va incerca sa aleaga la fiecare iteratie caracteristica ce ofera, spre exemplu, castigul de informatie maxim. Un dezavantaj al caracterului greedy este acela ca arborele optim global nu este garantat obtinut.

Mecanismul de predictie este foarte rapid si consta in parcurgerea unui subarboare de la radacina pana la nodul frunza corespunzator drumului. Odata ajuns la un nod terminal, observatiei de test ii va fi asociata eticheta majoritara. Rezultatele sunt usor de interpretat, granitile de decizie mereu vizibile iar drumul parcurs de datele de test poate oferi informatii suplimentare cu privire la attributele mai relevante sau locul unde ar fi necesara retezarea (eng. *prunning*). Arborele de decizie se adapteaza atat la attributele discrete, cat si la cele continue, unde discretizeaza spatiul de cautare prin stabilirea unor praguri de decizie acolo unde etichetele corespunzatoare unor valori difera. Totodata, un *decision tree* poate lucra simultan cu attribute de ambele feluri.

Modelul parcurge exhaustiv toate posibilitatile de divizare a nodurilor arborelui pana cand acest lucru nu mai este posibil. Cu alte cuvinte, se incearca adaptarea cat mai exacta la datele de antrenament. Astfel, fenomenul de *overfitting* este o problema foarte cunoscuta cu care se confrunta algoritmul. O modalitate de rezolvare este tehnica *prunning*, adica retezarea unor nivele sau noduri care nu aduc imbunatatiri semnificative sub-arborilor construiti de ele. Am realizat acest lucru prin tehnica de *GridSearch*, cautand parametrii optimi si obtinand urmatoarele rezultate:

```
----- Decision Trees -----
Number of mislabeled points out of a total 5946 points : 1148
The accuracy score is : 0.806929
[[3970  920]
 [ 228  828]]
      precision    recall  f1-score   support

      0         0.95      0.81      0.87       4890
      1         0.47      0.78      0.59       1056

 accuracy          0.81       5946
 macro avg         0.71      0.80      0.73       5946
 weighted avg      0.86      0.81      0.82       5946

Balanced accuracy score: 0.7979759248931029
```

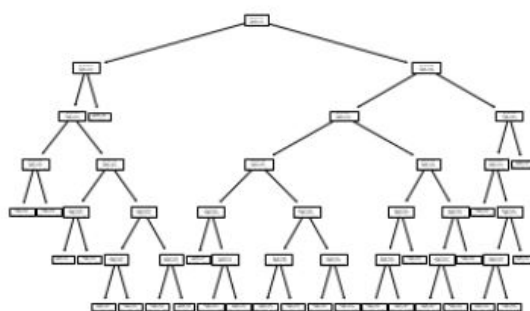


Figura 4.17: Rezultate metrice pentru arborele de decizie simplu

Pentru un clasificator singular, masuratorile din tabel sunt foarte bune, apropiindu-se de cele ale *Random Forest*. Totodata, adancimea arborelui a fost redusa de la 26 (numarul maxim de nivele construite de arborele complet fara prunning) la 7, eficienta timp si spatiu fiind mult superioara. Totusi, in acest fel, frunzele nu vor mai fi pure iar predictia se va face prin vot majoritar, existand sansa ca o mare parte din intrarile a caror etichete sunt pozitive sa fie clasate gresit si vice-versa. Din acest motiv, ca optimizare suplimentara, *voi analiza nodurile terminale unde diferenta in procente dintre etichetele pozitive si cele nule este mai mare de 10% cu ajutorul masinilor de vectori suport* (eng. *Support Vector Machines*). In momentul in care o intrare de test ajunge la o frunza cu proprietatea descrisa mai sus, vom folosi la predictie masina vector suport specifica.

Masini vector suport

Tehnica masinilor cu suport vectorial (eng. *Support vector machines*) [20] este cea mai cunoscuta abordare neprobabilista din domeniul invatarii automate. Mecanismul de functionare este urmatorul: *se incearca gasirea unui hiperplan care sa separe liniar datele*. Diferenta fata de alti algoritmi similari, precum perceptron-ul, este data de incercarea SVM-ului de a maximiza distantele/marginile dintre cele mai apropiate observatii, adica vectorii suport, si plan cu scopul de trasa granita de separare optima.

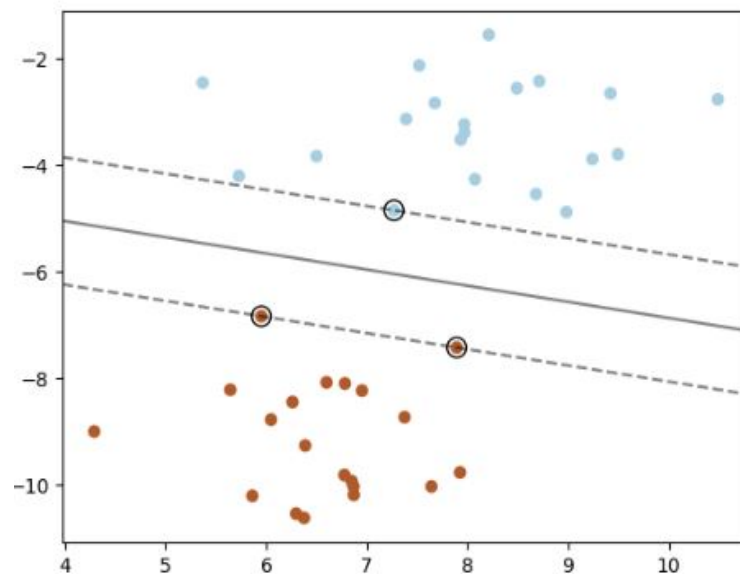


Figura 4.18: Exemplu teoretic de masini cu suport vectorial

De asemenea, modelul permite si relaxarea conditiei de separabilitatea exacta prin introducerea unui regularizator ξ a carui pondere C poate fi manual setata in functie de ce dorim sa obtinem: o clasificare perfecta sau una care permite si greseli in detrimentul unei margini mai mari. Forma primala de baza a SVM, specifica unui separator liniar, care inglobeaza cele descrise pana acum este:

$$\min_{w,b,\xi} \frac{1}{2} w^T \cdot w + C \sum_{i=1}^n \xi_i \quad \text{st} \quad y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \quad \xi \geq 0$$

Cum datele de care dispun *nu sunt separabile liniar*, este nevoie de o optimizare a formei matematice de mai sus. Aceasta evolutie o reprezinta *forma duala* care face referire la functii de tip *kernel* ce au rolul de a mapa datele intr-o dimensiune mai mare, pentru ca mai apoi sa se decida asupra planului de separare. *Scikit-learn* lucreaza doar cu forma duala a carei formulare este urmatoarea:

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha, \quad Q_{ij} = y_i y_j K(x_i, x_j)$$

În mod empiric, s-a stabilit ca funcția kernel cea mai potrivită este RBF. De asemenea, fixarea parametrului C se face prin încercare și eroare (eng. *trial and error*), pornind de la o valoare mare și scăzând-o treptat pe parcursul mai multor iterații. La fel ca și în cazurile precedente, *GridSearch* a fost opțiunea ideală pentru găsirea parametrilor optimi. Rezultatele obținute în urma hibridului dintre arborele de decizie și SVM sunt:

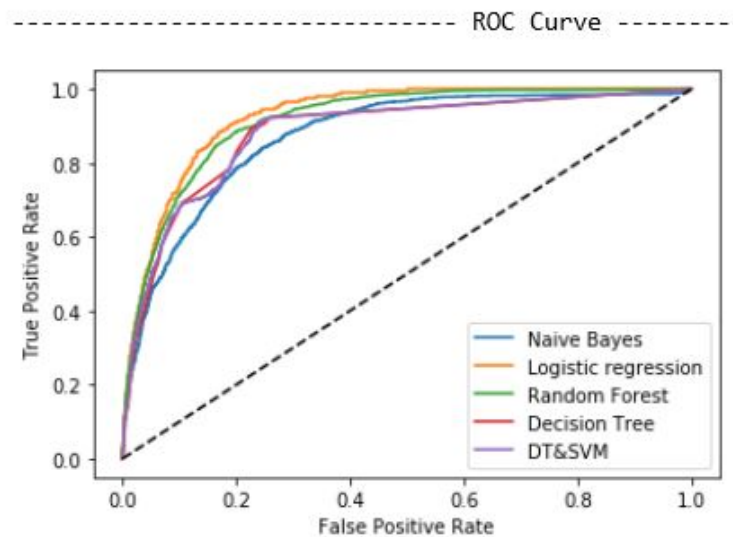
```
----- Decision Trees & Support Vector Machines -
Number of mislabeled points out of a total 5946 points : 855
The accuracy score is : 0.856206
[[4503 387]
 [ 468 588]]
      precision    recall  f1-score   support

      0       0.91      0.92      0.91      4890
      1       0.60      0.56      0.58      1056

 accuracy          0.86      5946
 macro avg       0.75      0.74      0.75      5946
 weighted avg    0.85      0.86      0.85      5946

Balanced accuracy score: 0.7388385387618517
```

Figura 4.19: Statisticile cumulate ale arborelui cu SVM



```
AUC for Gaussian Naive Bayes: 0.871563
AUC for Logistic Regression: 0.926419
AUC for Random Forest: 0.914426
AUC for Decision Tree: 0.882900
AUC for DT&SVM: 0.882464
```

Figura 4.20: Grafic ROC final

Tehnologii Folosite

La nivel de tehnologii, am folosit numai limbajul de programare *Python*. Având o sintaxa ușor de înțeles și folosit, un mecanism de indentare ce face codul lizibil, Python mai îmbină și programarea orientată obiect cu cea funcțională. Astfel, am reușit să reduc porțiuni dense de cod la doar câteva linii. Alocarea memorie se face automat, utilizatorul nefiind nevoit să se concentreze asupra tipurilor de date sau a gestiunii interne. De asemenea, Python este prima opțiune (eng. *state of the art language*) în industrie, atunci când se dorește o analiză de tip învățare automată sau minare de date. Motivul principal este suportul limbajului pentru modulele cunoscute precum *scikit-learn* [21], principala bibliotecă care conține cele mai noi implementări ale tuturor modelelor de machine learning, *numpy* [23], un pachet fără de care operațiile de algebră liniară și lucrul cu vectori în general ar fi mult îngreunate, *pandas* [22], o unealtă folosită la manipularea datelor și *matplotlib* [25], instrumentul cu care au fost realizate toate graficele prezente în lucrarea de față. Amintesc aici și *scipy* [24], modulul ce furnizează/modelează distribuții probabiliste.

Teza este scrisă în mediul *Jupyter Notebook* [26], care reprezintă un ansamblu de foi (eng. *programming sheets*) cu cod particular pe fiecare din ele. Practic, acest instrument permite etapizarea programului în mai multe stadii, cu posibilitatea de revenire asupra lor, fără a rula de fiecare dată întregul soft. Totodată, are înglobate toate funcționalitățile limbajului Python și nu numai, fiind o alternativă superioară oricărui alt kernel. De asemenea, un fișier Jupyter Notebook este un document JSON (Javascript Object Notation) mai special ce nu depinde de un editor, limbaj sau mediu de programare pentru a putea fi deschis, ci oferă opțiunea de vizualizare a lucrării în format HTML. Pe parcursul evoluției soft-ului scris de mine, am ținut evidența versiunilor noi pe *GitHub*, un serviciu de găzduire web ce permite crearea și mentenanța unui seif virtual (eng. *repository*) privat. Tehnologiile folosite sunt standardul practic și teoretic atât în industrie, cât și în mediul academic dacă ne referim la modelarea datelor.

Concluzii

Nevoia unui ansamblu de modele menite sa gaseasca *pattern-uri* explicative ale statisticilor jocului *Player Unknown Battlegrounds* este una evidenta. O industrie a jocurilor video aflata in plina ascensiune, dorinta companiilor de a investi in jucatori cu scopul promovarii propriilor produse cat si simplele pariuri puse pe posibillii finalisti ai unui meci reprezinta componenta economica de baza ce a dat startul acestei lucrari.

Pe tot parcursul tezei, am urmarit metodologia CRSIP-DM, lucru evidentiat si prin structura capitolelor care surprind cu exactitate fiecare etapa. In prima parte m-am ocupat de procesul de curatare a datelor, care mai apoi au fost supuse selectiei caracateristicilor relevante, standardizate iar in unele cazuri attributele au fost reduse la o combinatie a unora dintre ele. Pre-procesarea a fost necesara pentru numerosi algoritmi a caror functionalitate ar fi afectata altfel. *Analiza nesupervizata*, reprezentata de K-means si DBSCAN, a pus in evidenta 3 categorii de clustere ce au dus la partitionarea setului de antrenament in 2 subseturi: unul relativ echilibrat iar altul mai putin. Fiecare dintre acestea a fost exploatat cu tehnici specifice. Daca la primul, tehnicile clasice de *analiza supervizata* precum regresia logistica, bayes naiv sau arborii de decizie au functionat ideal, la al doilea a fost nevoie de boosting pentru a creste ponderea/relevanta asociata etichetelor nenule aflate in minoritate.

Metrici precum precizia, senzitivitatea sau specificitatea, inglobate in grafice explicative de genul ROC, dar si ilustrararile vizuale ale scalabilitatii sau a ratei de invatare sunt elementele cheie ce compun studiul comparativ. Cu alte cuvinte, in urma acestui proiect, au fost studiate si analizate rezultatele modelelor descrise mai sus cu scopul de a stabili care din ele este de preferat in functie de ce doreste investitorul. Spre exemplu, daca se vrea o precizie cat mai mare a predictiilor jucatorilor din top 10, programul ne spune ca regresia logistica este varianta ideala. Prin urmare, lucrarea de fata trebuie privita drept un reper care ghideaza utilizatorii ei spre algoritmi care raspund cel mai bine cererilor oamenilor.

Bibliografie

- [1] <https://www.kaggle.com>
- [2] <https://www.kaggle.com/c/pubg-finish-placement-prediction>
- [3] <https://www.sv-europe.com/crisp-dm-methodology/>
- [4] <http://www.medicalbiostatistics.com/Sensitivity-specificity.pdf>
- [5] <https://www.sciencedirect.com/science/article/abs/pii/S0031320396001422>
- [6] <https://scikit-learn.org/stable/modules/clustering.html#clustering-evaluation>
- [7] https://robertlbeach.weebly.com/uploads/1/2/3/8/12382487/data_notes.pdf
- [8] <http://uregina.ca/~gingrich/corr.pdf>
- [9] <https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-part-1-what-is-feature-importance>
- [10] <https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing>
- [11] <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.FeatureAgglomeration.html>
- [12] <https://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf>

- [13] <https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf>
- [14] <https://web.stanford.edu/~hastie/Papers/samme.pdf>
- [15] <https://www.cs.ubbcluj.ro/~gabis/ml/ml-books/McGrawHill%20-%20Machine%20Learning%20-Tom%20Mitchell.pdf>
- [16] http://www.cs.cmu.edu/~tom/10701_sp11/slides/LR_1-27-2011.pdf
- [17] <https://profs.info.uaic.ro/~ciortuz/SLIDES/cluster.pdf>
- [18] <https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf>
- [19] <https://profs.info.uaic.ro/~ciortuz/SLIDES/ml3.pdf>
- [20] https://www.researchgate.net/publication/221621494_Support_Vector_Machines_Theory_and_Applications
- [21] <https://scikit-learn.org/stable/>
- [22] <https://pandas.pydata.org/>
- [23] <https://numpy.org/>
- [24] <https://www.scipy.org/>
- [25] <https://matplotlib.org/>
- [26] <https://jupyter.org/>