



Universiteti i Prishtinës “Hasan Prishtina”

Tema: Aplikimi i algoritmit të grupimit Bisecting K-Means në për të
kllasteruar heronjtë në video-lojën Dota 2

Mentorë: Prof. Dr. Ing. Lule Ahmedi

Kandidati: Nora Ibrahim

Prishtinë, Dhjetorë 2019

Abstrakti

IntelliDota është një projekt i realizuar me anë të gjuhëve programuese Scala dhe Flutter, gërshetimi i së cilave sjell një aplikacion të mençur, të qëndrueshëm, të shpejtë dhe ndihmues, me anë të së cilit mund të analizojmë burime të të dhënave, vizualizojmë dhe të aplikojmë metrika dhe algoritme të ndryshme mbi to. Ky projekt është i ndarë në dy pjesë, ky dokumentin ka të bëjë me pjesën IntelliDota Clustering.

IntelliDota Clustering paraqet pjesën e dytë të projektit IntelliDota në të cilin përdoret një data set i marrë nga Kaggle. Data seti përbëhet nga 500000 reshte dhe 12 kolona, dhe në këtë data aplikohet algoritmi Bisecting K-Means, që është një version i K-Means. Data seti ka formatin CSV, që do të thotë vlera të ndara me presje (ang. *comma seperated values*).

Përpara aplikimit të algoritmit, së pari duhet para-procesuar data seti përkatës, Meqë nuk kemi pasur nevojë të rregullojmë gabime, si kolona që mungojnë etj. i vetmi para-procesim që duhet kryer është grupimi i vlerave në kolonën hero_id. Kjo kolonë paraqet numrin identifikues të heroit, dhe nëse në këtë grupim gjejmë mesataren për secilin hero, atëherë marrim disa vlera reprezentative që janë simbolikë e heroit përkatës. Me këtë para-procesim, ne derivojmë një data set final që përbëhet nga 110 rreshta dhe 12 kolona. 110 rreshta pasi ekzistojnë 110 heronj në lojë.

Pas kësaj faze, aplikojmë algoritmin Bisecting K-Means dhe ruajmë modelin e trajnuar në mënyrë që nëse dëshirojmë të kryejmë ndonjë kllasterim tjetër në kohë reale, mos të ri-trajnojmë modelin por të përdoret modeli i gatshëm.

Meqë ky raport përmban pjesën e dytë të projektit IntelliDota, dihet që ekziston një Docker imazh që nuk është i publikuar në ndonjë Cloud. Ky imazh do të publikohet në Google Cloud Platform nëpërmjet Google SDK, një bashkësi veglash që na ndihmojnë që të kryejmë operacione nëpërmjet një CLI. Pasi të publikohet imazhi, kemi një ueb publik që mundemi ti qasemi në çfarë do kohe.

Me mundësinë për qasje të shërbimeve në cdo kohë, është realizuar një aplikacion me anë të së cilit vizualizohen reprezentimet përkatëse për të dy data setet, qofshin ato metrika, qofshin faza të algoritmit që kanë kaluar, matrica korrelacioni apo shumë detaje të tjera. Pjesa kryesore e këtij aplikacioni është mundësia për të kryer predikim apo kllasterim në kohë reale, ku rezultati kthehet për një kohë shumë të shkurtër varësisht data setin që kemi përzgjedhur. Kjo është realizuar nëpërmjet Flutter, një vegël e ndërtuar nga *Google* për ndërtimin të aplikacioneve autokton (ang. *native*) për telefona të mençur, ekran apo shfletues. Lejon një zhvillim të ndërfaqes (ang. *interface*) në mënyrë të shpejtë, shprehëse dhe elastike. Gjuhë programuese ka *Dart*.

Tabela e figurave

Fig 1: Algoritmi i një loje tic-tac-toe.....	8
Fig 2: K-Means vizualisht	13
Fig 3: Bisecting K-Means vizualisht	15
Fig 4: Formacioni i lojtarëve të video-lojës Dota 2	17
Fig 5: Organizimi i ekipit.....	18
Fig 6: Mostra e data setit	19
Fig 7: Kolonat e data setit me tipet përkatëse	19
Fig 8: Rreshtat dhe kolonat e data setit të pre-procesuar	20
Fig 9: Ngarkimi i data setit	20
Fig 10: Zgjedhja dhe filtrimi në data set	20
Fig 11: Grupimi dhe heqja e kolonës hero_id	21
Fig 12: Leximi gjatë aplikimit të algoritmit.....	21
Fig 13: Funkzioni për riemërimin e data setit	22
Fig 14: Fazat nëpër të cilat ka kaluar algoritmi	22
Fig 15: Përdorimi i fazave (gypit) sipas funksionit fit.....	24
Fig 16: Ruajtja e modelit lokalisht.....	24
Fig 17: Listimi i imazheve Docker	25
Fig 18: Mundësia për zgjedhje dhe krijimit të projektit.....	25
Fig 19: Shoqërimi i etiketës dhe shtyrja për në Cloud.....	26
Fig 20: Mundësia për krijimin e një shërbimi në Cloud	26
Fig 21: Imazhi i publikuar nga CLI.....	27
Fig 22: Zgjedhja e regjionit më të përafërt	27
Fig 23: Lejimi i qasjeve të pa autorizuara	27
Fig 24: Memoria e alokuar në Cloud dhe maksimumi i kërkesave	28
Fig 25: Porti i hostit në Cloud.....	28
Fig 26: Ekzekutimi i imazhit me gabime gjatë procedurës	28
Fig 27: Ekzekutimi i imazhit pa gabime gjatë procedurës.....	28
Fig 28: Më shumë detaje rreth hostimit	29
Fig 29: Lidhja përfundimtare e qasshme në cdo kohë.....	29
Fig 30: Pamja hyrëse.....	30
Fig 31: Vartësia për rrëshqitje	30
Fig 32: Pamja e butoneve për klasifikim / kllasterim	31
Fig 33: Kolonat e diskretizuesit dhe rrëshqitësit.....	31
Fig 34: Diskretizuesi	31
Fig 35: Pamja e grafikut të Diskretizuesit	31
Fig 36: Pamja e parë e dy fazave.....	32
Fig 37: Faza Bucketizer	32
Fig 38: Pamja e vlerave në mostër.....	33
Fig 39: Pamja e kolonave në mostër.....	33
Fig 40: Matrica e korrelacionit të data setit Kaggle	34
Fig 41: Matrica e korrelacionit të data setit Steam	34
Fig 42: Vlerat fundore të korrelacionit.....	34
Fig 43: Struktura e data seteve.....	35
Fig 44: Pamja e klasifikimit në kohë reale	36

Fig 45: Kllasterët në mënyrë grafike.....	37
Fig 46: Grupimi i kllasterëve	40

Tabela e tabelave

Tab 1: Pika fundore index	44
Tab 2: Pika fundore getColumns.....	44
Tab 3: Pika fundore getSample	44
Tab 4: Pika fundore getStages.....	45
Tab 5: Pika fundore: getCorrelationMatrix.....	45
Tab 6: Pika fundore getGroupAndCount	45
Tab 7: Pika fundore getStages.....	45
Tab 8: Pika fundore getSchema	46
Tab 9: Pika fundore getDoubleGroup	46
Tab 10: Pika fundore getClusterStats.....	46
Tab 11: Pika fundore getClusterCount	46

Tabela e përmbajtjes

1	Hyrja	8
1.1	Motivimi.....	8
1.2	Përshkrimi i Problemit.....	9
2	Inteligjenca artificiale – Kllasterimi si mësim pa mbikëqyrje.....	11
2.1	Algoritmet e mësimi pa mbikëqyrje	12
2.1.1	K-Means	12
2.1.2	Mean-Shift.....	13
2.1.3	Bisecting K-Means.....	14
3	Hyrje në projekt	16
3.1	Loja kompjuterike Dota 2.....	17
3.2	Para-procesimi i të dhënave	18
3.3	Para-procesimi në Scala	20
3.4	Aplikimi i Algoritmit.....	21
3.5	Publikimi i imazhit Docker në Google Cloud Platform	24
4	Zhvillimi i Aplikacionit	30
4.1	Opsionet e menisë	31
4.1.1	Mundësia për zgjedhje.....	31
4.1.2	Diskretizuesi.....	31
4.1.3	Fazat për Trajnim.....	32
4.1.4	Mostrat e të Dhënave.....	33
4.1.5	Matrica e Korrelacionit.....	34
4.1.6	Struktura e Data Seteve	35
4.1.7	Klasifikimi dhe kllasterimi në Kohë Reale	36
4.1.8	Kllasterët	37
4.1.9	Grupimi i Kllasterëve	40
5	Lista e thirrjeve fundore të mundshme	44
5.1	index	44
5.2	getColumns.....	44
5.3	getSample	44
5.4	getStages.....	45
5.5	getCorrelationMatrix.....	45
5.6	getGroupAndCount	45

5.7	getStages.....	45
5.8	getSchema	46
5.9	getDoubleGroup.....	46
5.10	getClusterStats	46
5.11	getClusterCount.....	46
5.12	postCluster	47

1 Hyrja

Ashtu siç kureshtja e njeriut nuk shuhet kurrë, ashtu edhe përparimi i teknologjisë nuk ka të ndalur. Megjithëse nuk është e gabuar nëse themi se teknologjia gati ka arritur majat e veta, akoma ka shtigje të pashkelura që mund të themi se paraqesin një botë teknologjike në vete. Ndër to, dhe ndër më të pëlqyerat e kërkuarat nga njerëzit është inteligjenca artificiale (ang. *artificial intelligence*).

1.1 Motivimi

Inteligjenca artificiale, e referuar ndryshe si inteligjenca e makinave (ang. *machine intelligence*) ka të bëjë me stimulimin e inteligjencës së përvetësuar dhe zhvilluar nga makinat [1].

Kjo shkencë ndryshe definohet si fusha e studimit të agjentëve inteligjentë (ang. *intelligent agents*): një pajisje që kupton ambientin ku ndodhet dhe ndërmerr veprime të tilla që mundësia e arritjes së qëllimit të jetë maksimale.

Për një agjent themi se është racional nëse bën gjënë e duhur, pra secili veprim në rrethana të caktuara është i saktë. Por si mund të definojmë se çfarë është e saktë dhe çfarë jo? Nëse agjenti kalon nëpër një sekuencë veprimesh dhe gjendjesh që na kënaqin themi se agjenti ka pasur një performancë të mirë [2].

Pra, makinat tentojnë që të përvetësojnë aftësitë njerëzore të të kuptuarit dhe zgjidhjes së problemeve (ang. *learning and problem solving*).

Qëllimi i një agjenti inteligjentë mund të jetë i thjeshtë, si për shembull luajtja e një loje GO, apo kompleks siç është kryerja e operacioneve matematikore.

Parimi bazë i inteligjencës artificiale është përdorimi i algoritmeve. Algoritmet janë një grumbull instruksionesh që një makinë kompjuterike mund të ekzekutojë. Një algoritëm kompleks ndërtohet si bashkësi e algoritmeve të thjeshta.

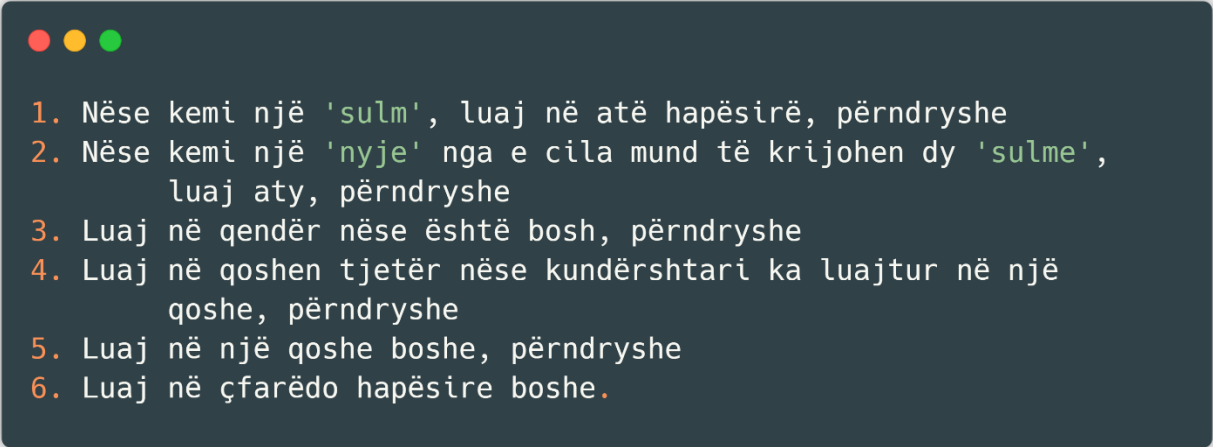
- 
1. Nëse kemi një 'sulm', luaj në atë hapësirë, përndryshe
 2. Nëse kemi një 'nyje' nga e cila mund të krijohen dy 'sulme', luaj aty, përndryshe
 3. Luaj në qendër nëse është bosh, përndryshe
 4. Luaj në qoshen tjetër nëse kundërshtari ka luajtur në një qoshe, përndryshe
 5. Luaj në një qoshe boshe, përndryshe
 6. Luaj në çfarëdo hapësire boshe.

Fig 1: Algoritmi i një loje tic-tac-toe

Disa algoritme, janë të aftë të mësojnë nga një grumbull të dhënash, si në rastin tonë, ku algoritmi mëson një strategji apo një mënyrë të mirë (ang. *rule of thumb*) të cilin e zbaton në të dhëna të reja, e disa algoritme të tjerë mund të vetë shkruajnë algoritme tjera.

Disa nga algoritmet që mësojnë, siç janë *fqinji më i afërt* (ang. *nearest-neighbor*), *pemët me vendime* (ang. *Decision Trees*), apo *rrjeti Bajesian* (ang. *Bayesian network*) munden teorikisht të përafrojnë çfarëdo të dhënash në një funksion të caktuar (nëse kanë memorie dhe kohë të pafundme). Që në kohët e hershme është stimuluar inteligjenca artificiale, e ndër qasjet më të njohura janë:

- Simbolizmi, njohur ndryshe si logjika formale: nëse personi ka të ftohtë, atëherë ai ka grip.
- Interference Bajesiane (ang. *Bayesian interference*): nëse personi ka të ftohtë, atëherë ekziston një probabilitet që ai të ketë edhe grip.
- *Vektorit Mbështetës i Makinës* (ang. *Support Vector Machine*) apo *Nearest-neighbor*: pas shqyrtimit të të dhënave të personave që kanë të ftohtë, duke përfshirë moshën, simptomat dhe faktorët tjerë, dhe këto faktorë përkasin me pacientin aktual, themi se pacienti ka grip.

Për një makinë thuhet se është inteligjente nëse kalon testin e Turingut, test ky i cili u dizajnuar në kohët e hershme që të sjell një përkufizim të kënaqshëm për inteligjencën. Ky test kalohet nëse është e pamundur të tregohet se a vijnë përgjigjet nga një njeri apo nga një makinë pas disa pyetjeve nga njeriu [3].

1.2 Përshkrimi i Problemit

Dihet se sot po thuajse në çdo fushë të jetës gjejmë aplikim të inteligjencës artificiale e sidomos në fushën e kompjuterikes, duke filluar nga sistemet rekomanduese, sistemet për predikim e shumë të tjera.

Duke pasur parasysh këto aplikime, ne kemi tentuar që të ndërtojmë një asistent për video-lojën e njohur Dota 2 pasi jo vetëm që do t'u ndihmonte njerëzve përgjatë lojës, por edhe do të pasuronte komunitetin me një data set unik (pasi po e krijojmë vet një të tillë nëpërmjet thirrjeve përkatëse) dhe kod burimorë që paraqet ecurinë e projektit dhe një shembull për të realizuar diçka të ngjashme si nga ana e inteligjencës artificiale, edhe nga ana e realizimit të thirrjeve dhe filtrimit të strukturës Json.

Duhet cekur se ekzistojnë shumë data sete të lidhura me këtë temë në internet, por për arsye të teknikave dhe metodologjive që dëshirojmë të implementojmë, kemi zgjedhur që ta krijojmë vet një të tillë.

Problemi jonë mund të përshkruhet nëpër disa faza, ku më të rëndësishmet janë:

- Të dhënat dhe algoritmi – fazë kjo që përmban hapat kryesorë në lidhje me mësimin e makinës. Duke filluar nga marrja e të dhënave, pastrimi i tyre, përkthimi i attributeve që të jenë të lexueshme për makinën dhe deri tek mundësia e përdorimit të algoritmit në data setin e përpunuar.
- Organizimi dhe Paraqitja në Flutter – kjo fazë përfshin komunikimin me thirrjet e krijuara në pjesën e Scala dhe përcaktimin e një dizajni të përshtatshëm dhe përgjegjshëm për pajisje mobile e njëherësh të lexueshëm për përdoruesin. Në fund ka pasuar lidhja logjike e pamjes me të dhënat dhe testimi në pajisje të ndryshme.

Pra, për aplikacionin tonë themi se ka kryer punën me sukses nëse është në gjendje të klasifikoj dhe kllasteroj të dhënat që i japim si hyrje në kohë reale, në mënyrë të shpejtë dhe të organizuar.

Pjesa inteligjente e aplikacionit mund të kryhet në shumë mënyra, duke filluar nga gjuha programuese Python me libraritë përkatëse që mundësojnë krijimin e një aplikacioni të ngjashëm me atë se çfarë duam të bëjmë ne e deri te shkrimi i një algoritmi nga ana jonë. Por meqë nuk është parë e arsyeshme të 'ri zbulojmë rrotën' ne kemi vendosur të përdorim gjuhën programuese Scala bashkë me Spark që ofron një arsenal të fuqishëm për manovrim me të dhëna dhe inteligjencë artificiale. Një përshkrim më në detaje se përse kemi zgjedhur këtë gjuhë me libraritë përkatëse do të shpjegohet më vonë.

E sa i përket pjesës së krijimit të të dhënave, ekzistojnë mënyra të tjera (siç janë përdorimi i një gjuhe programuese për të krijuar thirrjet) por jo metodologji, që do të thotë se data seti që kemi krijuar ne mund të merret vetëm nëpërmjet ueb-faqes zyrtare Steam nëpërmjet thirrjeve dhe filtrimeve përkatëse.

2 Inteligjenca artificiale – Kllasterimi si mësim pa mbikëqyrje

Në përgjithësi ekzistojnë forma të ndryshme të të mësuarit të makinës, por tri më konkretisht përfaqësojnë pothuajse çdo formë të të mësuarit. Kemi:

- Mësimin pa mbikëqyrje (ang. *unsupervised learning*) – agjenti mëson mënyra për të zgjidhur apo kalkuluar problemin e caktuar varësisht hyrjes edhe pse nuk ka reagime dalëse nga burime të caktuara (pra, algoritmi nuk shpërblehet e as nuk dënohet për veprimin e zgjedhur). Në thelb mësimi pa mbikëqyrje është sinonim për kllasterimin. Procesi i të mësuarit është i pa mbikëqyrur pasi shembujt hyrës nuk janë të etiketuar si klasë. Ne përdorim kllasterimin për të zbuluar klasa përbrenda të dhënave. Për shembull, një metodë e të mësuarit të pa mbikëqyrur mund të marr si hyrje një set të imazheve të numrave të shkruar më dorë. Supozojmë që i gjen 10 kllasterë me të dhëna. Këto kllasterë mund t'i korrespondojnë 10 numrave të veçantë nga 0 tek 9 respektivisht. Sidoqoftë, pasi që të dhënat e trajnimit nuk janë të etiketuar, modeli i mësuar nuk mund të na tregojë kuptimin semantik të kllasterëve të gjetur.
- Mësimi i sforcuar (ang. *reinforcement learning*) – agjenti mëson nga një seri e veprimeve dalëse, qofshin ato veprime dënuese apo shpërblyese. Për shembull, një bakshish në fund të vozitjes me taksi i tregon algoritmit se ka vepruar mirë, përndryshe ka vepruar keq. I takon algoritmit të vendos se cilat pjesë të tij kanë bërë që të kryejë veprimin mirë.
- Mësimi me mbikëqyrje (ang. *supervised learning*) – agjenti mëson nga çiftet hyrëse – dalëse, dhe mëson një funksion me anë të së cilit krijon një skemë ku orientohen vlerat e ardhshme hyrëse në ato dalëse. Në parim ky lloj i mësimit është sinonim i klasifikimit. Mbikëqyrja gjatë mësimit vjen nga shembujt e etiketuar në setin e trajnuar. Për shembull në problemin e njohjes së kodit postar, një set i kodeve postare të shkruara më dorë dhe përkthimet korresponduese të lexueshme nga makina përdoren si shembuj trajnues, të cilët mbikëqyrin mësimin e modelit klasifikues.

Në këtë aplikacion do të përdoret saktësisht të kllasterimi si mënyrë e të mësuarit pa mbikëqyrje, teknikë kjo që do të përshkruhet më poshtë saktësisht me algoritmet përkatëse.

2.1 Algoritmet e mësimit pa mbikëqyrje

Kllasterimi është një teknikë në inteligjencën artificiale që përfshinë grupimin e të dhënave, pra ne mund të përdorim kllasterimin për të klasifikuar të dhënat në grupe të caktuara. Në teori, të dhënat në grup të njëjtë duhen të kenë karakteristika dhe cilësi të njëjta, për derisa të dhënat në grupe të ndryshme duhet të kenë attribute dalluese [4].

Kllasterimi përndryshe është një teknikë e të mësuarit pa mbikëqyrje dhe përdoret shumë për statistika dhe analiza të të dhënave në fusha të ndryshme.

Në aplikacionin tonë, si teknikë kllasterimi ne kemi përdorur K-Means, por do të shfaqim edhe teknikën tjetër shumë të përdorur Mean-Shift nga e cila nuk ka nevojë fare të caktohet numri i pikave qendrore të quajtura ndryshe centroide.

2.1.1 K-Means

Është me shumë mundësi algoritmi më i përhapur dhe përdorur për kllasterim. Kjo për arsye se përdoret dhe kuptohet lehtë. Po përshkruajmë në hapa algoritmin për të sqaruar mënyrën se si funksionin.

- Fillimisht, zgjedhim një numër grupesh dhe në mënyrë të rastësishme vendosim pikat qendrore të tyre. Për të gjetur numrin që duam të përdorim, është praktikë e mirë që tu hedhim një sy të dhënave dhe të shohim 'me sy të lirë' nëse kanë ngjashmëri. Pikat qendrore janë po ashtu vektorë si çdo pikë tjetër aktuale.
- Secila pikë e të dhënave klasifikohet duke llogaritur distancën në mes pikës aktuale dhe secilit grup pikash qendrore, e pika e të dhënave klasifikohet në atë grup, distanca e së cilës është më e vogël ndër të gjitha grupet e pikave qendrore.
- Duke u bazuar në këto pika klasifikuese, ri-llogarisim pikat qendrore duke marrë vlerën mesatare të të gjitha grupeve qendrore.
- Ri përsërisim këto hapa për një numër të caktuar apo deri sa qendrat e grupeve nuk ndryshojnë shumë. Një mundësi tjetër është inicializimi në mënyrë të rastësishme i qendrave grupore disa herë dhe zgjedhim rezultatin që ka performuar më së miri.

K-Means ka avantazhin kryesorë në shpejtësi, pasi krejt çka bën është llogaritja e distancave ndërmjet pikave apo qendrave grupore; kalkulime shumë të vogla. Për këtë arsye ka kompleksitet linear $O(n)$ [5].

Një shembull i algoritmit K-Means duket kështu (Fig 2):

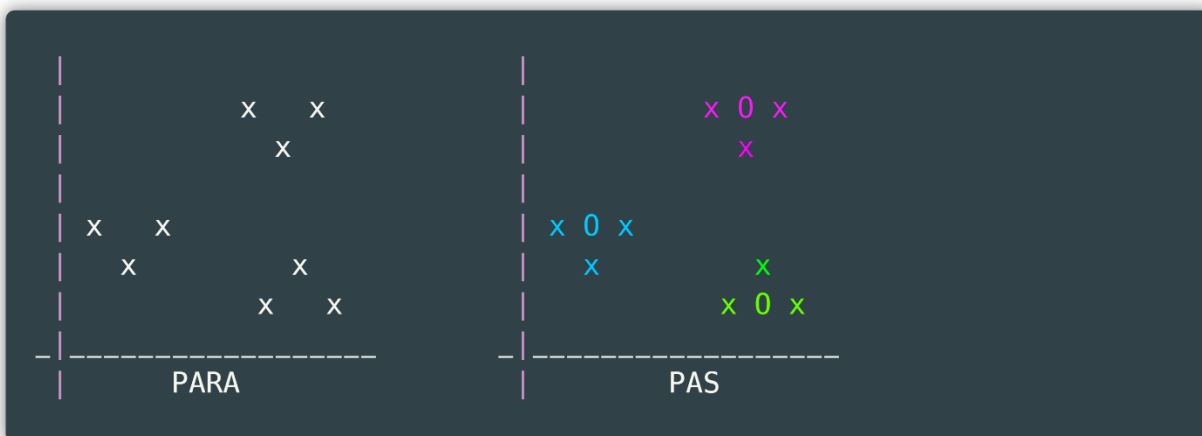


Fig 2: K-Means vizualisht

ku grafiku i parë paraqet të dhënat para kllasterimit, pra të dhëna të rëndomta pa ndonjë grupim, ndërsa grafiku i dytë paraqet të dhënat e grupuara në bazë të ngjyrës, ku shihen 3 pika qendrore të shënuara me 0 (nën supozimin se kemi zgjedhur 3 pika qendrore, pra $k = 3$). E nëse vjen një e dhënë e re, ajo grupohet në atë kllaster, distanca ndaj të cilit është më e afërt.

Në anën tjetër, ka disa mangësi, duke filluar prej asaj se duhet caktuar në mënyrë manuale numrin e grupeve të pikave qendrore (variabla k). Kjo nuk është diçka e mirë sepse ne duam të nxjerrim statistika të shëndosha nga data seti dhe do të ishte mirë po të merrej algoritmi me këtë pjesë.

Përveç kësaj, siç u cek edhe më lartë, K-Means i cakton qendrat grupore në mënyrë të rastësishme, për këtë arsye rezulton ndryshe nga secili ambient që ekzekutohet algoritmi. Pra, rezultatet nuk janë të ri përdorshëm dhe kemi mungesë qëndrueshmërie. Metodatat tjera janë më të qëndrueshme.

2.1.2 Mean-Shift

Mean Shift është një procedurë kllasterimi pa parametra, e famshme në fushën e pamjeve kompjuterike dhe përpunimit të imazheve, dalja e së cilës për dallim nga K-Means algoritmi, nuk varet nga supozimet eksplicite mbi formën e shpërndarjes së pikave, numrin e kllasterëve, ose çfarëdo forme të ndonjë inicializimi të rëndomtë [6].

Mean Shift ndërton mbi bazën e konceptit të vlerësimit të densitetit të bërthamës (ang. *kernel density estimation (KDE)*). KDE është një metodë për të vlerësuar shpërndarjen themelore, e quajtur edhe funksioni i densitetit të probabilitetit, për një seri të të dhënave. Funksionon në atë mënyrë që vendosë një bërthamë (ang. *Kernel*) në secilën pikë të data setit. Bërthama është një shprehje matematikore për një funksion peshues. Ka lloje të ndryshme të këtyre bërthamave, por më i famshmi është bërthama Gausiane. Duke i mbledhur të gjitha bërthamat individuale do të gjenerohet një sipërfaqe e probabilitetit (p.sh *funksioni i densitetit*).

Mean Shift e shfrytëzon këtë ide të KDE, duke imagjinuar se çfarë do të ndodhte nëse pikat do të ngjiteshin lartë në majën më të afërt në sipërfaqen KDE. Kjo vazhdon deri sa të gjithë pikat kanë arritur të jenë në maje dhe se këto pika eventualisht do të bashkohen tek një seri e pikave, afër maksimës lokale të shpërndarjes. Ato pika që konvergjojnë tek makisma e njëjtë lokale konsiderohen anëtarë të kllasterit të njëjtë, numri i të cilëve varet nga gjerësia e brezit të sipërfaqes KDE, kështu po të merrej një vlerë shumë e vogël do të rezultonte me numër të kllasterëve aq sa kemi pika në data set, dhe në të kundërtën do të kishim vetëm një kllaster të formuar.

2.1.3 Bisecting K-Means

Ideja prapa këtij algoritmi është pothuajse e njëjtë me atë të algoritmit K-Means, por diferencon në disa detaje të caktuara.

Ndryshe nga algoritmi K-Means, ky algoritëm nuk zgjedh pika të rastësishme për të matur më pas distancën, por ndan mostrën tonë përgjysmë duke vazhduar me pjesën me marzhë gabimi më të madhe deri sa të formohen K grupime [7].

Po përshkruajmë më poshtë hapat e saktë që ndërmerr ky algoritëm:

- Mbledh të dhënat në një grup të vetëm, që do të thotë se të gjitha të dhënat e data setit gjenden në një kllaster të vetëm.
- Aplikohet K-Means me vlerë 2, pra kllasteri jonë ndahet në dy pjesë relativisht të barabarta, pjesë këto të caktuara nga vetë algoritmi.
- Kalkulohet marzha e gabimit sipas metrikës RMSE dhe zgjidhet data seti me marzhë të gabimit më të madhe për tu ndarë përsëri në grupime më të vogla.
- Përsëritet hapi paraprak deri sa numri i grupimeve të jetë i barabartë me K.

Një shembull se si duket kjo grafikisht paraqitet më poshtë (Fig 3).

x		x	x		x		x		x
	x				x		x		x
x		x		x					x
	x					x		x	x

The first grid shows the initial array: [5, 10, 3, 4, 2, 8, 6, 9, 7, 1]. The second grid shows the array after the first pass: [5, 3, 4, 2, 8, 6, 9, 7, 1, 10].

$$\text{RMSE}(P2) > \text{RMSE}(P1)$$

The diagram illustrates the process of finding the minimum element in a binary tree. The left tree shows the root node 10 with children 6 and 12. Node 6 has children 3 and 9. Node 3 has children 1 and 5. Node 9 has children 7 and 11. The right tree shows the root node 10 with children 6 and 12. Node 6 has children 3 and 9. Node 3 has children 1 and 5. Node 9 has children 7 and 11. The minimum element 1 is highlighted in red in both trees.

Si rezultat kemi 3 grupime.

15

3 Hyrje në projekt

Me të përshkruar inteligjencën artificiale dhe algoritmet kryesore të kllasterimit si përfaqësues të të mësuarit pa mbikëqyrje të makinës, tani mund të kalohet në përshkrimin e zgjidhjes së ofruar të problemit, si dhe të rezultateve të fituara.

Po fillojmë njëherë me përshkrimin e veglave softuerike që janë përdorur, në mënyrë që të kuptohet më vonë përdorimi i tyre varësisht rastit.

- *IntelliJ* – vegla për ndërtim (ang. *Integrated Development Kit*, IDE) në të cilin është shkruar aplikacioni, është ndërtuar (ang. *build*) dhe provuar (ang. *test*). Kjo vegël është e shkruar në gjuhën programuese Java dhe është zgjedhur ndër shumë tjera për arsye të organizimit të lartë, mundësive të shumta që ofron si dhe paketave të gatshme si për Scala ashtu edhe për Play.
- *Postman* – i cili aktualisht është një nga mjetet më të njohura që përdoret në testimin e API, ku API qëndron për Ndërfaqe e Programimit të Aplikacioneve (ang. *Application Programming Interface*) dhe lejon aplikacionet softuerike të komunikojnë në mes vete përmes thirrjeve API.
- *Android Studio* – është mjedisi zyrtar i zhvillimit të integruar për sistemin operativ Android të Google, i krijuar posaçërisht për zhvillimin e Android. Ky mjedis është përdorur për zhvillimin e aplikacionit në Flutter, pasi ofron shumë lehtësi në kodim dhe gjithashtu është mjedis i rekomanduar nga Flutter.
- *Powershell* – vegla me anë të së cilës është menaxhuar imazhi Docker. Ndryshe nga CLI (Command Line Interface) të tjera, kjo vegël është më funksionale, më e pasur me funksione si dhe ka një ndërfaqe më të përdorshme.
- *Docker* – vegla me anë të së cilës është krijuar imazhi në mënyrë që ti publikojmë shërbimet e tona në internet të qasshme për këdo.
- *Git* – platforma në të cilën është koordinuar ecuria e punës. Pasi kjo platformë dominon në tregun e punës, është parë e arsyeshme që të praktikohet edhe më shumë kjo mënyrë e punës. Funksionon nëpërmjet ndërfaqes komanduese (ang. *Command Line Interface*, CLI).
- *Github* – platforma në të cilën *Git* vendos dosjet burimore. Kjo platformë në përgjithësi mbledh të gjitha projektet publike të shkruara nga çdo programues në botë dhe ia ofron kërkuesit në mënyrë që ato të kenë qasje edhe në kode burimore edhe në aplikacione që ndërtohen si ndihmesa, siç është edhe vegla e përdorur më poshtë *Ngrok*.


IntelliDota është një aplikacion i ndërtuar në Scala dhe Flutter që tenton të *klasteroj* (ang. *cluster*) dhe *klasifikoj* (ang. *cluster*) rezultatin e një video-loje që në rastin tonë është video-loja *Dota 2*. Do të emërtojmë me IntelliDota Classification pjesën e aplikacionit që kryen klasifikimin ndërsa IntelliDota Clustering pjesën e aplikacionit që kryen kllasterimin. Përpara se të vazhdojmë më tutje me aplikacionin, do të bëjmë një përshkrim të shkurtër se çfarë është kjo video-lojë dhe çfarë duam të predikojmë ne.

3.1 Loja kompjuterike Dota 2

Dota 2 është një nga lojërat e tipit MOBA që ka kuptimin arenë e betejës me shumë lojtar në kohë reale (ang. *Multiplayer Online Battle Arena*), e njohur edhe si strategji veprimi në kohë reale, e cila me masat rafinerisë të aplikuara gjatë viteve ka arritur një nivel të lartë sofistikimi dhe me këtë ka tërhequr jo vetëm lojtar të shumtë si të rastësishëm ashtu edhe profesional, por edhe është bërë pjesë e studimeve të shumta dhe eksperimenteve të ndryshme sidomos në fushën e Inteligjencës Artificiale.

Dota 2 qëndron për Mbrojtjen e Kullave (ang. *Defense of the Ancients*), ku lojtarët zgjedhin nga një grup prej më shumë se njëqind heronjsh, duke formuar dy ekipe që konsistojnë nga pesë lojtar secila, duke okupuar hapësirë të veçantë në hartë. Secili nga dhjetë lojtarët kontrollon në mënyrë të pavarur një karakter, të ashtuquajtur hero me aftësi unike dhe stile të ndryshme të lojës. Gjatë ndeshjes, lojtari dhe ekipi i tij grumbullojnë pikë eksperience dhe ari për të blerë artikuj për heronjtë e tyre, në mënyrë që të përshkojnë tek dymbëdhjetë kullat e ekipit kundërshtar. Një ekip fiton duke qenë e para për të shkatërruar bazën kryesore të armikut të quajtur "*Ancient*". *Dota 2* ka një theks të veçantë tek taktika dhe koordinimi i ekipit, dhe një fokus të madh në strategjinë e ndërtimit të forcës sa më shpejt që është e mundur dhe zgjedhjen e rendit të azhurnimit të magjive të heroit [8].

Ekipet e lojës posedojnë një ndarje të një niveli më të lartë, pra mbajtësit (ang. *carries*) dhe mbështetësit (ang. *supports*). Në esencë, çdo *support* mbështet një *carry* (Fig 4).



Lojtari	Pozicioni	Roli	Fusha
P1	1	Carry	Safe-lane
P2	2	Carry	Mid-lane
P3	3	Carry	Off-lane
P4	4	Support	Roamer
P5	5	Support	Hard-support

Fig 4: Formacioni i lojtarëve të video-lojës Dota 2

Ndërsa në këndvështrim të lartë, ekipi organizohet në formacion të tillë (Fig 5):



Fig 5: Organizimi i ekipit

Secili lojtarë, pavarësisht rolit ka qindra statistika, por ndër më të thepisurat dhe më të rëndësishmet janë *leaver_status*, *gold_per_min*, *leaver_status*, *xp_per_min*, *deaths*, *tower_damage* etj. Lista e plotë mund të gjendet në kapitullin e mëposhtëm në të cilin realizohet kllasterimi, pra `postCluster`.

3.2 Pamja e parë dhe para-procesimi i të dhënave

Si burim i të dhënave kemi përdorur Kaggle, ku Kaggle është platforma më e madhe për nga aspekti i shkencës së të dhënave (ang. *Data Science*). Kaggle ju mundëson shkencëtarëve të të dhënave dhe zhvilluesve të ndryshëm, të përfshihen në konteste të fushës së mësimi të makinës, të shkruajnë dhe të ndajnë kodin me të tjerët, të hostojnë baza të të dhënave dhe shumëçka tjetër [9]. Data seti përbëhet nga dhjetëra kolona dhe rreth 500.000 rreshta. Megjithëse procesi i kllasterimit nuk kërkon shumë fuqi llogaritëse, jemi munduar të marrim kolonat më përshkruese dhe më të shëndosha pavarësisht numrit të madh të rreshtave dhe kolonave.

Fillojmë me analizimin e data setit nga i cili fillimisht heqim kolonën *match_id* dhe *account_id* për arsye se nuk kanë peshë në kllasterim, pra janë të dhëna 'jo të mençura'. Shohim se kolona *hero_id* i përket një heroi përkatës, dhe secili hero në lojë shquhet me atributet përkatëse, pra kemi heronj që shkaktojnë dëme, që shërojnë, që shtyjnë kulla e të tjera, të dhëna këto të nxjerra nga pamja e parë e data setit.

Vazhdojmë me kolonën *player_slot* të cilën nuk e marrim parasysh po ashtu, pasi shërben vetëm për të treguar inventarin e heroit. Nëntë kolonat e ardhshme janë shumë përshkruese dhe që të gjitha do të jenë pjesë e data setit tonë të pastër, ato kolona janë *gold*, *gold_per_min*, *xp_per_min*, *kills*, *deaths*, *assists*, *denies*, *last_hits*, *hero_damage*, *hero_healing*, *tower_damage* dhe *level*.

Ndërsa kolonat tjerë përshkruajnë veglat e lojtarit që nuk janë përshkruese ngase nuk ndryshojnë varësisht rolit. Gjithsesi, tani më kemi një data set të llojit (Fig 6):

```
[{
  "gold": 1821.1461434370772,
  "gold_per_min": 447.6417456021651,
  "xp_per_min": 494.8359269282815,
  "kills": 9.657307171853857,
  "deaths": 9.190121786197563,
  "assists": 9.746278755074425,
  "denies": 5.939445196211096,
  "last_hits": 165.67016238159675,
  "hero_damage": 13487.634641407307,
  "hero_healing": 740.771650879567,
  "tower_damage": 1449.6248308525035,
  "level": 19.41982408660352
}, ...]
```

Fig 6: Mostra e data setit

Shohim tani skemën e data setit, pasi duhet ditur tipet e të dhënave në mënyrë që të ndryshojmë diçka apo të vazhdojmë më tutje ().

```
[ { "column": "gold",           "type": "DoubleType" },
  { "column": "gold_per_min",  "type": "DoubleType" },
  { "column": "xp_per_min",    "type": "DoubleType" },
  { "column": "kills",        "type": "DoubleType" },
  { "column": "deaths",       "type": "DoubleType" },
  { "column": "assists",      "type": "DoubleType" },
  { "column": "denies",       "type": "DoubleType" },
  { "column": "last_hits",    "type": "DoubleType" },
  { "column": "hero_damage",  "type": "DoubleType" },
  { "column": "hero_healing", "type": "DoubleType" },
  { "column": "tower_damage", "type": "DoubleType" },
  { "column": "level",        "type": "DoubleType" }]
```

Fig 7: Kolonat e data setit me tipet përkatëse

Shohim se të gjitha tipet e të dhënave janë të tipit *Double* që i bie që mund të aplikohen metrika të ndryshme numërore dhe kjo është ajo që na nevojitet neve. Po ashtu, shohim statistikën e data setit tonë (Fig 8):

```
{
  "rows": "110",    "columns": "12"
}
```

Fig 8: Rreshtat dhe kolonat e data setit të pre-procesuar

Pra, data seti jonë i përpunuar përmban 110 rreshta dhe 12 kolona. Kemi vetëm 110 rreshta për arsye se ekzistojnë 110 heronj, pasi kemi grupuar data setin në bazë të heronjve, ndërsa 12 kolona janë atributet e përzgjedhura nga ne.

3.3 Realizimi i para-procesimit në Scala

Së pari ngarkojmë të dhënat e shkarkuara nga Kaggle nëpërmjet Spark duke përdorur komandat (Fig 9):

```
var players = spark.read
  .option("header", true)
  .option("inferSchema", true)
  .csv(System.getenv("raw_kaggle_data"))
```

Fig 9: Ngarkimi i data setit

Pra, *players* aktualisht është një data set me 500.000 rreshta dhe rreth 30 kolona, ndërsa *raw_kaggle_data* është emri i variablës së mjedisit që e kemi emëruar ne për shkaqe të lehtësimit. Së pari zgjedhim kolonat e caktuara dhe filtrojmë ato, niveli (ang. *level*) i së cilave është më i madh se zero (Fig 10).

```
players = players
  .select(
    "hero_id", "gold", "gold_per_min", "xp_per_min", "kills", "deaths",
    "assists", "denies", "last_hits", "hero_damage", "hero_healing",
    "tower_damage", "level")
  .where(col("level") > 0)
```

Fig 10: Zgjedhja dhe filtrimi në data set

Pastaj aplikojmë ato në data setin *players* të ngarkuar më lartë. Tani meqë *players* është një element i pa ndryshueshëm (ang. *immutable*), ne mbishkruajmë data setin *players* me data setin e filtruar. Është një ide e mirë të filtrohet data seti në nivel më të madhe se zero pasi një nivel i tillë në lojë nuk ekziston dhe është mbushur hapësira si e korrumpuar nga zhvilluesit e video lojës. Si hap i ardhshëm është grupimi dhe mesatarja e secilës kolonë, e cila është realizuar si në vijim (Fig 11):

```
var groupedBy = players.groupBy("hero_id")
    .mean().drop("hero_id")
```

Fig 11: Grupimi dhe heqja e kolonës hero_id

e më pas vetëm kemi ruajtur lokalisht data setin në një lokacion të përshtatshëm për përdorim të më vonshëm. Funksionet e paraqitura më lartë, pra *mean* dhe *drop* i ofrohen koleksionit të data setit pas përdorimit të *group by* [10], kjo pasi mesatarja është vlera me e denjë për të përfaqësuar kolonën përkatëse, në rastin tone *mean* gjen mesataren aritmetike për secilën kolonë numerike, këtij funksioni i shtohet edhe largimi i *hero_id* pasi pas grupimit, na krijohet edhe një *avg(hero_id)*, që është asgjë më shumë se sa dublikatë e paraprakes.

Pra tash kemi data setin gati për të aplikuar metrika të tjera ose për të aplikuar algoritmin kryesorë, që është edhe qëllimi i këtij aplikacioni.

3.4 Aplikimi i Algoritmit

Ri ngarkojmë data setin tonë të pastër nëpërmjet librarisë Spark në mënyrë që të aplikojmë kllasterimin. Rikujtojmë se ky data set i ri përmban 110 rreshta dhe rreth 13 kolona. Së pari nëpërmjet këtyre rreshtave ngarkojmë data setin e pastruar (Fig 12).

```
var groupedBy = spark.read
    .option("header", true)
    .option("inferSchema", true)
    .csv(System.getenv("kaggle_data"))
```

Fig 12: Leximi gjatë aplikimit të algoritmit

dhe ua heqim emrat e pa përshtatshëm kolonave përkatëse. Me emra të pa përshtatshëm duam të themi se nëse aplikojmë metrika si grupimin (ang. *group by*) me shumë, apo me mesatare siç e kemi në rastin tonë, atëherë kolonave ju shtohet parashtesa e funksionit, në rastin tonë, secila kolonë merr emrin *avg(...)*.

E me anë të funksionit *RemoveBadNaming* e krijuar nga ne, kolona do të merr emrin që ka pasur para funksionit. Këtë metrikë ua aplikojmë të gjitha kolonave. Ky funksion duket si në vijim (Fig 13):

```
groupedBy = RenameBadNaming(groupedBy)
```

Fig 13: Funksioni për riemërimin e data setit

ku *groupedBy* paraqet data setin e filtruar dhe tanimë edhe të riemëruar. Pas këtij hapi, ne konstruktojmë fazat nëpër të cilat do të kalojë algoritmi jonë. Këto faza duken si në vijim (Fig 14):

```
val bucketizer = new Bucketizer()  
    .setInputCol("kills")  
    .setOutputCol("kills_out")  
    .setSplits(  
        Array(Double.NegativeInfinity,  
            3.0, 6.0, 9.0, 12.0,  
            Double.PositiveInfinity))  
val imputer = new Imputer()  
    .setInputCols(Array("hero_damage"))  
    .setOutputCols(Array("hero_damage_out"))  
    .setStrategy("median")  
val assembler = new VectorAssembler()  
    .setInputCols(args)  
    .setOutputCol("pre-features")  
val scaler = new StandardScaler()  
    .setInputCol("pre-features")  
    .setOutputCol("features")  
    .setWithStd(true)  
    .setWithMean(false)  
val kmeans = new BisectingKMeans()  
    .setK(5)  
    .setMaxIter(25)  
    .setSeed(1)  
val pipeline = new Pipeline()  
    .setStages(Array(bucketizer, imputer, assembler, scaler, kmeans))
```

Fig 14: Fazat nëpër të cilat ka kaluar algoritmi

Nga e cila mund të shohim se përdorim gjashtë klasa kryesore, ku:

- Klasa e parë, pra *Bucketizer* mundëson grupimin e të dhënave nga kolona *kills* në kolonën dalëse *kills_out*. Si metodë shihet që përdoret edhe *setSplits* nga e cila mundësohet që kolona të grupohet varësisht ndarjeve që kemi specifikuar ne. Ky grupim

ndihmon në kllasterimin e mëtutjeshëm të të dhënave, pasi kolona *kills* vetëm se ka vlera nga zero deri në mesatarisht 50. Saktësisht, ndarja i takon grupimeve të më poshtëm:

1. Prej intervalit infinitiv negativ deri në 3 të dhënat grupohen në vlerën 0.
 2. Prej intervalit 3 deri në 6 të dhënat grupohen në vlerën 1.
 3. Prej intervalit 6 deri në 9 të dhënat grupohen në vlerën 2.
 4. Prej intervalit 9 deri në 12 të dhënat grupohen në vlerën 3.
 5. Prej intervalit 12 e deri në infinitiv pozitiv të dhënat grupohen në vlerën 4.
- Klasa e dytë e quajtur *Imputer* përdoret për të mbushur vlerat e zbrazëta në data set, ku në rastin tonë përdoret strategjia *median* në atributin *hero_damage*. Në rastin tonë konkret nuk gjejmë ndonjë aplikim të kësaj por është përdorur si demonstrim.
 - Klasa e tretë *VectorAssembler* që mundëson grumbullimin e attributeve në një atribut të vetëm të quajtur *features* në rastin tonë. Ky funksion nevojitet në secilin algoritëm të mençur për arsye se mënyra e funksionimit të tyre është e tillë që algoritmi kërkon një kolonë hyrëse dhe jep një kolonë dalëse. Funksioni ka pra dy metoda, *setInputCols* që kërkon një varg me elementet që dëshirojmë të grumbullojmë në një atribut dhe metodën *setOutputCol* nga e cila mundësohet kolona dalëse.
 - Klasa e katërt është *StandardScaler* dhe është përdorur për të shkallëzuar të dhënat. Ky lloj shkallëzuesi, ndër të shumtë të llojit të tij, shërben për të konvertuar secilën vlerë në intervalin -1 deri në 1. Si kolonë hyrëse kemi marrë *pre-features* të përgatitura nga hapi paraprak dhe si kolonë dalëse kemi *features*.
 - Klasa e pestë quhet *Bisecting Kmeans* dhe është funksioni kryesorë me anë të së cilit aplikohet algoritmi i kllasterimit. Ndër shumë metoda tjerë me vlera të parazgjedhura, ne po aplikojmë dy parametra të ri në metodën *setK*. Kjo metodë cakton numrin e pikave qendrore të kllasterave që në rastin tonë është 5. Krahas kësaj, kemi edhe metodën *setMaxIter* që cakton numrin e unazave që në rastin tonë është 25, konkretisht kjo metodë provon 25 kombinime të mundshme të algoritmit dhe zgjedh rezultatin më të mirë.
 - Klasa e gjashtë quhet *Pipeline* dhe mundëson që të gjitha klasat ndihmëse të grupohen në faza dhe të ekzekutohen një pas një në mënyre automatike para se të përdoret algoritmi. Kjo teknikë lehtëson përdorimin e algoritmeve kryesore si dhe bën kodin më konciz, më të lexueshëm dhe më elastik. Nëse nuk do të kishim këtë teknikë, ne do të duhej të aplikonim secilën klasë të cekur më sipër në data set e kjo është jo praktike dhe redundante.

Me të aplikuar këto algoritme në data setin tonë me anë të funksionit (Fig 15):

```
val model = pipeline.fit(groupedBy)
```

Fig 15: Përdorimi i fazave (gypit) sipas funksionit fit

Na mbetet vetëm të ruajmë modelin lokalisht (Fig 16):

```
model.write.overwrite.save(System.getenv("clustered_model"))
```

Fig 16: Ruajtja e modelit lokalisht

Ne ruajmë modelin lokalisht për arsye se është më logjike dhe më praktike që të ruhet modeli i kllasteruar, në mënyrë që nëse duam të nxjerrim statistika apo të aplikojmë funksione në të, të mos trajnohet modeli përsëri pasi kërkon kohë dhe fuqi kompjuterike përpunuese. Kjo po ashtu na mundëson neve që të kryejmë kllasterime në kohë reale, që do të thotë mundemi të kllasterojmë çfarëdo rreshti të ri në kohë shumë të shkurtër.

3.5 Publikimi i imazhit Docker në Google Cloud Platform

Përpara se të vazhdojmë më tutje, duhet të përshkruajmë se çka është Docker dhe si funksionon ai. Docker është një vegël që ndihmon në krijimin, zhvillimin dhe publikimin e aplikacionit në ueb. Këtë e bën duke paketuar softuerin tonë në paketa të quajtur kontejnerë [11]. Kontejnerët janë të izoluar nga njëri tjetri, që do të thotë se posedojnë pavarësi të plotë. Motivimi për Docker, ndër shumë arsye tjera, ka ardhur nga ajo se për tu ekzekutuar programi nga dy sisteme të ndryshme është shumë më e lehtë të krijohet një Docker imazh që mund të ekzekutohet nga pala tjetër sesa pasimi i kodit burimorë. Arsyeja tjetër dhe kryesore është ajo se për të publikuar një sistem në ueb, që në rastin tonë ka qenë publikimi i API të krijuara nga pjesa backend, Docker është vegla më e përshtatshme, sidomos kur aplikacioni është i shpërndarë në disa pjesë, ku në këtë rast ne kemi pjesën e data seteve, modelet e trajnuara për klasifikim dhe kllasterim e të tjera.

Krahas Docker, kemi edhe Google Cloud Platform, që paraqet një aset të fuqishëm të ofruar nga Google për të kryer shumë operacione, duke filluar nga të mësuarit e makinës, një depo për dosje të ndryshme, vegla për DevOps si K8 e të tjera.

Ajo se çfarë ne kemi bërë me Google Cloud Platform është hostimi i API të ofruara nga ne. Përse na është nevojitur diçka e tillë? Për arsye që aplikacioni të mos varet nga ekzekutimi lokal i aplikacionit, pra thirrjet e pjesës frontend të kryhen pavarësisht se a po ekzekutohet aplikacioni lokalisht, pasi shërbimet e aplikacionit që është i ndezur nga një makinë lokale humben nëse ajo fiket. Me të cekur të gjitha këto detaje, kalojmë te pjesa e realizimit të kësaj procedure. Nga

pjesa IntelliDota Classification, vetëm se kemi krijuar një Docker imazh, verifikojmë se a ekziston një i tillë (Fig 17):

```
PS C:\Users\Nora> docker images
REPOSITORY          TAG
intellidota         latest

IMAGE ID            CREATED             SIZE
5534d846e77e        2 hours ago        525MB
```

Fig 17: Listimi i imazheve Docker

Pra shohim se kemi imazhin e quajtur intellidota me numër identifikues 5534d846e77e me madhësi rreth 525 MB. Kjo është e tëra që na nevojitet për ta kaluar në Google Cloud.

Kalojmë te pjesa e Google Cloud Platform [12], ku hapi i parë që duhet bërë është të krijojmë një projekt, në rastin tonë, ne vetëm se kemi krijuar diçka të tillë të emëruar *intellidotavfinal*. Krijimi i një projekti lehtë mund të realizohet nëpërmjet ndërfaqes kryesore si në vijim, pra tek butoni *New Project*. Shihet se ne kemi zgjedhur projektin *intellidotavfinal*, prandaj vazhdojmë tutje (Fig 18).

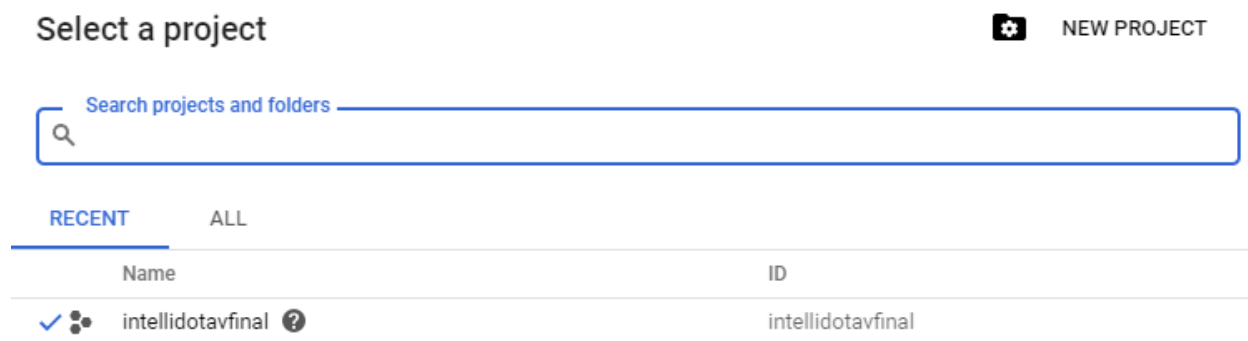


Fig 18: Mundësia për zgjedhje dhe krijimit të projektit

Me të krijuar projektin, hapim navigatorin dhe zgjedhim opsionin Cloud Run. Në esencë, Cloud Run është një platformë që lejon manipulimin me kontejnerë, në rastin tonë me imazhet Docker.

Më pas, me anë të një CLI (ang. *Command Line Interface*) publikojmë imazhin tonë lokal në Google Cloud Platform (Fig 19).

```

PS C:\Users\Nora\Documents\Github\intellidota>
docker tag intellidota gcr.io/intellidotavfinal/serverfinal_5
PS C:\Users\Nora\Documents\Github\Intellidota>
docker push gcr.io/intellidotavfinal/serverfinal_5
The push refers to repository [gcr.io/intellidotavfinal/serverfinal_5]
2523dd03774f: Layer already exists
3c0ad4e15a89: Layer already exists
06937e0115b9: Layer already exists
90f9a0ec06c5: Layer already exists
2579d88b8593: Layer already exists
0fca6ef80f69: Layer already exists
8794aa41f512: Layer already exists
79de1722d667: Layer already exists
1af2c1ddc23c: Layer already exists
83278a7e8e27: Layer already exists
4695e5f82a26: Layer already exists
13e4c0d00639: Layer already exists
767f936afb51: Layer already exists
latest: digest: sha256:cefa40ce2f5a8499c6ef4ab1ab64
98a1a95db5a4f43308d167a0b176b700dd83 size: 3048

```

Fig 19: Shoqërimi i etiketës dhe shtyrja për në Cloud

Komanda e parë i shoqëron projektin tonë lidhjen e faqes gcr.io, ku GCR qëndron për Google Cloud Repository, dhe më pas e shtyjme projektin tonë nëpërmjet lidhjes së shoqëruar. Lidhja në vazhdim përbëhet nga emri i projektit të GCP-së që në rastin tonë është *intellidotavfinal* ndërsa *serverfinal_5* është emri i degës, apo një super version në të cilën po bëhet publikimi.

Ky proces, pra ngarkimi, merr përafërsisht 10 minuta pasi madhësia e imazhit Docker është rreth 500 MB. Më poshtë vërejmë se shfaqen disa mesazhe si *Layer already exists*, kjo ndodhë për arsye se ne vetëm se kemi publikuar po të njëjtin version më herët (versionin në të cilin aktualisht është i ndezur serveri).

Pasi të kryejmë këtë veprim, kalojmë përsëri në GCP dhe vazhdojmë procesin në Cloud Run. Së pari krijojmë një shërbim të ri nëpërmjet butonit *Create Service* (Fig 20).

Google Cloud Platform intellidotavfinal						
<div> <div>Cloud Run</div> <div>Services</div> <div>CREATE SERVICE</div> <div>DELETE</div> <div>MANAGE CUSTOM DOMAINS</div> </div>						
Each Cloud Run service has a unique endpoint and autoscales deployed containers. Learn more						
Filter services						
<input type="checkbox"/>	Name ↑	Req/sec ?	Region	Authentication ?	Connectivity ?	Last deployed
<input checked="" type="checkbox"/>	serverfinal3	0	europe-west1	Allow unauthenticated	External	Dec 16, 2019, 10:15:03 AM

Fig 20: Mundësia për krijimin e një shërbimi në Cloud

Me klikimin e këtij butoni, na shfaqet dritarja në të cilën duhet të zgjedhim kontejnerin nga lista e të gjithë kontejnerëve të publikuar nga ne (Fig 21, ne zgjedhim kontejnerin e publikuar më herët, pra super versionin serverfinal_5 me etiketën *latest* – vetvetiu i shoqëruar nga Google).

Select Google Container Registry Image

Project: intellidotavfinal

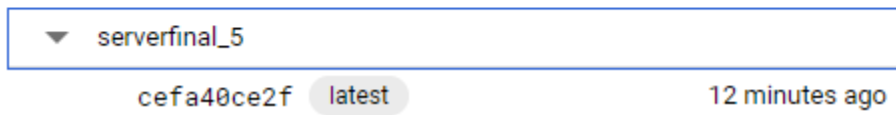


Fig 21: Imazhi i publikuar nga CLI

Caktojmë regjionin në mënyrë që thirrjet të realizohen në mënyrë më të shpejtë, me këtë duam të themi se nëse caktojmë një regjion me distancë të largët nga se ku jemi kyçur ne në internet, koha e transmetimit dhe pranimit të të dhënave nëpërmjet rrjetit është e lartë (Fig 22).

☒ Cloud Run (fully managed)

Region *

europa-west1

Region for this Service can't be changed later. [How to pick a region?](#)

Fig 22: Zgjedhja e regjionit më të përafërt

Lejojmë që thirrjet në shërbimet tona të jenë të realizueshme edhe nëse nuk janë të autentifikuara. Në këtë mënyrë, çdokush që dëshiron mund t'u qaset Json dokumenteve në mënyrë të drejtpërdrejtë duke shfrytëzuar lidhjen e aktuale që kemi mundësuar ne (Fig 23).

Authentication *



Allow unauthenticated invocations

Check this if you are creating a public API or website.

Fig 23: Lejimi i qasjeve të pa autorizuara

Ia ndajmë kontejnerit tonë 2GB memorie për arsye se aq kërkon edhe Spark në backend, që do të thotë se nëse i ndahen më pak njësi të memories, nuk mund të ndodhë procesimi, ndërkaq do të hidhen gabime, pasi nevojiten më shumë fuqi procesuse. Krahës kësaj, lejojmë edhe që maksimumi i kërkesave të përnjëhershme të jetë 80 (Fig 24). Kjo praktikë aplikohet në mënyrë që të shmangen sulmet e ndryshme, ndër më të famshmit që është mohimi i shërbimeve (ang. *DOS – Denial Of Service Attacks*).

Memory allocated
2 GiB

Memory to allocate to each container instance.

Maximum requests per container
80

The maximum number of concurrent requests that can reach each container instance.

Fig 24: Memoria e alokuar në Cloud dhe maksimumi i kërkesave

Më pas, kufizojmë një kohë që të kthehen përgjigjet që në rastin tonë është 300 sekonda dhe specifikohet porta e kontejnerit, që Google ia ka caktuar 8080 (Fig 25). Kjo mund të ndryshohet, por duhet cekur se kjo portë dhe porta në të cilën është ndërtuar Docker imazhi duhet të jenë të njëjta, për te realizuar këtë gjë gjithmonë në mënyrë dinamike, kemi specifikuar portin në Dockerfile siç është përshkruar në IntelliDota Classification si variabël mjedisi.

Request timeout
300 seconds

Time within which a response must be returned (maximum 3600 seconds).

Container port
8080

Requests will be sent to the container on this port. We recommend listening on \$PORT instead of this specific number.

Fig 25: Porti i hostit në Cloud

Pra nëse ky proces kryhet saktë, shërbimi jonë do të jetë i qasshëm në portin 8080. Me këto të dhëna, klikojmë butonin Create dhe presim që të krijohet shërbimi jonë.

Nëse kemi gabuar përgjatë procedurës, na shfaqet ky imazh bashkë me gabimin përgjatë procedurës (Fig 26).

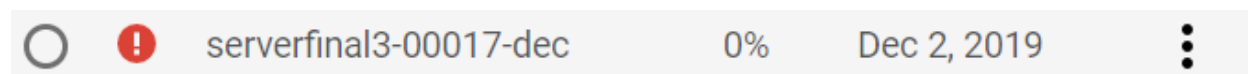


Fig 26: Ekzekutimi i imazhit me gabime gjatë procedurës

Përndryshe, themi se shërbimet tona janë të gatshme në ueb-faqen e specifikuar nga GPC, saktësisht na shfaqet imazhi në vijim (Fig 27).

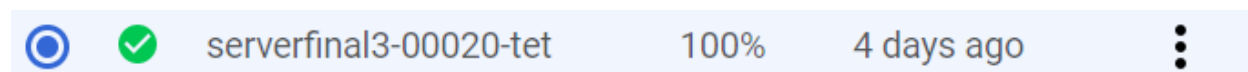


Fig 27: Ekzekutimi i imazhit pa gabime gjatë procedurës

Nëse na intereson të shohim më shumë të dhëna, kemi anën e djathtë të ekranit, pra (Fig 28):

Container image URL	gcr.io/intellicdotavfinal/serverfinal_5@sha256:44affb2...
Container port	8080
Autoscaling	Up to 1,000 container instances
CPU allocated	1
Memory allocated	2048Mi
Concurrency	80
Request timeout	300 seconds
Service account	983616342010-compute@developer.gserviceaccount.com

Environment variables

None

Fig 28: Më shumë detaje rreth hostimit

Përfundimisht kemi lidhjen <https://serverfinal3-qm4ka2ucaq-eë.a.run.app> të cilën nëse provojmë ti qasemi, shohim (Fig 29):

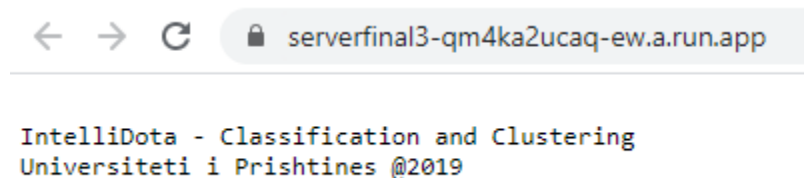


Fig 29: Lidhja përfundimtare e qasshme në cdo kohë
Pra, shërbimet tona janë të qasshme nga kushdo në çfarëdo kohe.

4 Zhvillimi i Aplikacionit

Aplikacioni është zhvilluar në gjuhën programuese Dart [13] e cila është gjuhë e orientuar në objekte, duke përdorur Flutter SDK. Flutter është një strukturë e zhvillimit të aplikacioneve mobile me burim të hapur e krijuar nga Google. Përdoret për të zhvilluar aplikacione për Android dhe iOS, si dhe është mënyra primare se si krijohen aplikacionet për Google Fuchsia. Flutter me mënyrën e krijimit të saj e lehtëson jashtëzakonisht punën duke ju dhënë zhvilluesve mundësi të shumta, gjë që krijon një eksperiencë të mirë dhe gjithashtu e përshejton kohën e zhvillimit. Mbështet disa mjedise për të ndërtuar aplikacione, të cilat janë IntelliJ, Visual Studio Code dhe Android Studio. Një ndër karakteristikat e Flutter është edhe ri-ngarkimi i aplikacionit i cili çdo

ndryshim të bërë në kod na mundëson ta vëzhgojmë në kohë reale.

Bazuar në pikat fundore të krijuara në Scala, janë krijuar thirrjet e veçanta në Flutter duke përdorur paketën `http` të zhvilluar nga Dart, ku në bazë të tyre është mundësuar krijimi i vizualizimeve të të dhënave.

Për të demonstruar më së miri mënyrën e funksionimit dhe strukturën e Flutter, fillojmë me pamjen hyrëse të aplikacionit në të cilën jepen detaje të përgjithshme për atë se çfarë është funksioni kryesor i tij (Fig 30). Në momentin që aplikacionit hapet, përdoruesit si pamje e parë i shfaqet ballina e aplikacionit, e cila është ndërtuar me anë të komponentëve (ang. *Widgets*) të cilat janë karakteristika kryesore e mbi të cilat është ndërtuar gjithçka në Flutter. Në këtë pjesë ne mund të fitojmë një kuptim të shkurtër mbi atë se çfarë përfshihet në projekt, kjo arrihet duke zvarritur panelin e poshtëm (Fig 31), i cili është krijuar duke përdorur paketën e mëposhtme e cila ofron një *Widget* të gatshëm që ofron këtë pamje.

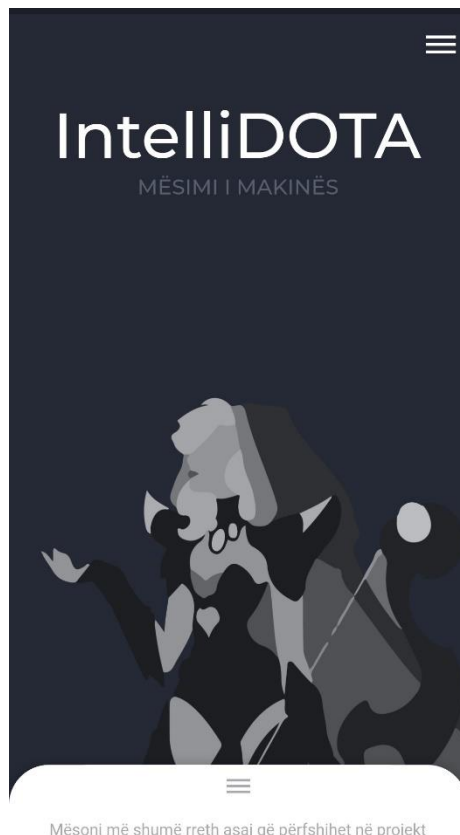


Fig 30: Pamja hyrëse

Për të manovruar tek pjesët tjera të aplikacionit ne mund të shfrytëzojmë menynë e pozicionuar në pjesën e epërme të ballinës e cila do të hap pamjen me listën e ekraneve, ku kemi gjithsej tetë ekrane të mundshme tek të cilat mund të navigojmë.

```
dependencies:  
  sliding_up_panel: ^0.3.6
```

Fig 31: Vartësia për rrëshqitje

4.1 Opsionet e menusë

4.1.1 Mundësia për zgjedhje

Meqenëse ne kemi dy data sete të trajnuara, metrikët mund të aplikohen në data setin e krijuar vetë, pra Steam apo Kaggle. Se në cilin data set duam të operojmë, kemi krijuar dy butona (ang. *Tabs*, Fig 32) me anë të së cilëve vendoset lloji i data setit përkatës. Duhet cekur se ky opsion nuk ekziston tek menyja Kllasterët dhe tek menyja Grupimi i Kllasterëve që funksionojnë vetëm për kllasterim. Ne si shembull kemi zgjedhur 'Kllasterim' që do të thotë se metrikët operohen në data setin Kaggle dhe si kolona automatikisht rreshtohen të data setit përkatës, në rastin tonë të data setit Kaggle.



Fig 32: Pamja e butoneve për klasifikim / kllasterim

4.1.2 Diskretizuesi

Diskretizuesi është një nga veglat më të dobishme në asetin tonë, që paraqet transformimin e vlerave të vazhdueshme në vlera të numërueshme. Meqenëse data setet përbëhen nga vlera të vazhdueshme me madhësi të lartë, ka qenë e arsyeshme ndërtimi i një vegle të tillë që diskretizon të dhënat në njërin nga grupet e zgjedhura nga rrëshqitësi (ang. *slider*). Ky ekran mund të hapet me zgjedhjen e *Diskretizuesi* nga pjesa e menysë siç shihet në vijim (Fig 33):



Fig 33: Diskretizuesi

Tek ky ekran ne kemi mundësinë që të shohim shpërndarjen e të dhënave në mënyrë grafike.

Së pari nga lista e shfaqur në pjesën e poshtme të ekranit ne zgjedhim njërin nga kolonat për të cilën dëshirojmë të bëjmë këtë vizualizim dhe më pastaj me anë të rrëshqitësit (ang. *Slider*) ne mund të vendosim për numrin e ndarjeve në të cilat do të grupohen vlerat e kolonës përkatëse (Fig 34). Kemi vendosur që të lejojmë mundësinë e zgjedhjes të një nga 4 vlerat e ndryshme të rrëshqitësit, meqë nga vlerësimet e bëra këto vlera reprezentojnë më së miri voluminozitetin e të dhënave.

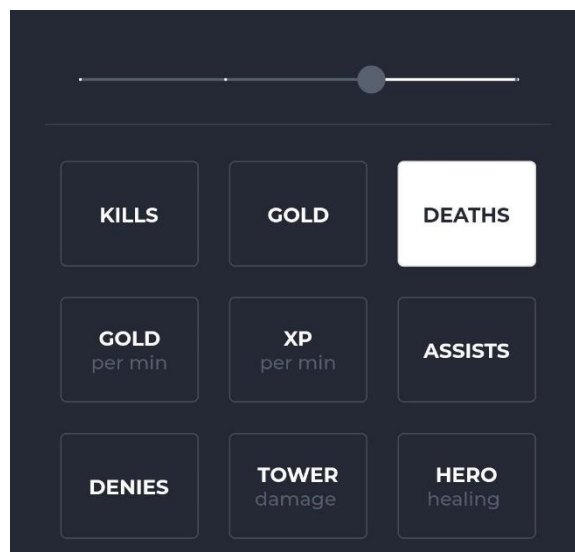


Fig 34: Kolonat e diskretizuesit dhe rrëshqitësit

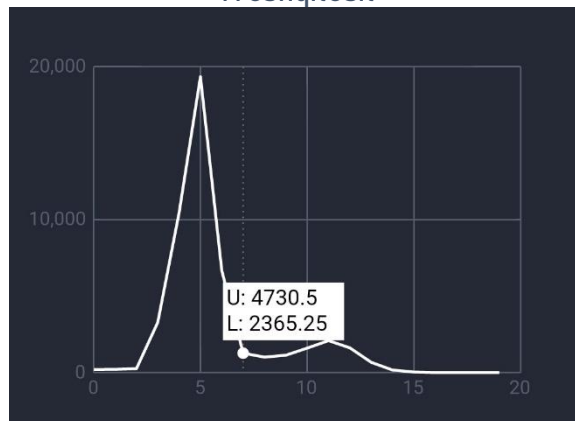


Fig 35: Pamja e grafikut të Diskretizuesit

Me zgjedhjen e njëres prej kolonave, kërkesa do të procesohet dhe si rezultat do të shfaqet një reprezentim grafik i cili demonstron në boshtin X numrin e grupeve të zgjedhura nga ne, qofte ai 5, 10, 15 apo 20 grupe ndërsa në boshtin Y sasinë e grupimit (Fig 35). Për të parë se cili është rangi i vlerave të grupuara, klikojmë në ndonjë pikë në drejtëz dhe shfaqet vlera më e ulët dhe më e lartë, ku shkronja U qëndron për lartë (ang. *Upper*) dhe L për poshtë (ang. *Lower*). Si shembull kemi imazhin në fig ku vlera më e vogël është 2365.5 dhe vlera më e lartë 4730.5, ku në rastin tonë janë rreth 2000 vlera siç vërehet në boshtin Y, ndërsa grupimi i takon kovës me numër 7 në boshtin X. Kjo vlerë është fituar nga të dhënat e grumbulluara për klasifikim, duke përdorur atributin *gold_per_min*. Duhet cekur se ky ekran është i ndërtuar me anë të API-të *getGroupAndCount*.

4.1.3 Fazat për Trajnim

Gjatë para-procesimit të të dhënave deri tek përzgjedhja e algoritmit në mënyrë që saktë të shikohen fazat nëpër të cilat kanë kaluar algoritmet tona deri sa janë trajnuar, kemi vizualizuar ato me renditje nga lartë poshtë, që do të thotë se faza më e lartë është ekzekutuar e para ndërsa faza më poshtë paraqet fazën e fundit.

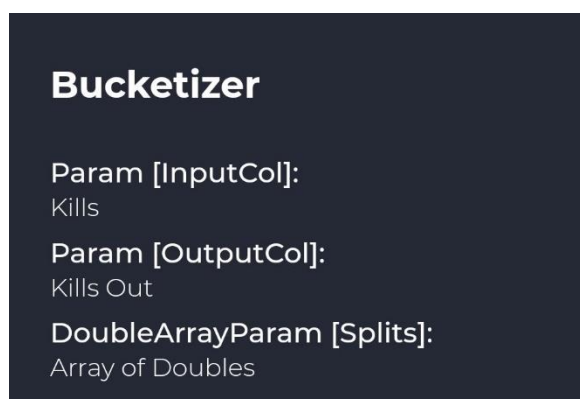


Fig 37: Faza Bucketizer

Nëse për instancë duam të shqyrtojmë se çfarë metodash janë përdorur në krijimin e një modeli të trajnuar, kemi, si shembull një Bucketizer ku kërkon disa parametra si kolona hyrëse, kolona dalëse dhe numri i ndarjeve (Fig 37). Duhet cekur se nëse një metodë është përdorur dy apo më shumë herë, metodës së përdorur më vonë i shtohet një numër i rastësishëm, për shembull, nëse përdorim një metodë tjetër të Bucketizer atëherë metoda tjetër emërohet Bucketizer4 (apo një numër të rastësishëm).

Një bashkësi e këtyre metodave formon një gyp (ang. *pipeline*) të tërë që prodhon një model të trajnuar. Nëse shqyrtojmë një gyp, për shembull Klasifikimin, mund të themi se së pari ndodhë normalizimi i të dhënave, ku si parametër hyrës kemi një numër kolonash ndërsa si dalje, jep një kolonë të vetme të quajtur Features ku vlerat janë të shkallëzuara ndërmjet 0 dhe 1. E pas kësaj përdoret algoritmi Random Forest Classifier me numër pemësh 10 (Fig 36). Duke vazhduar kështu, konstruktohet gypi i tërë.

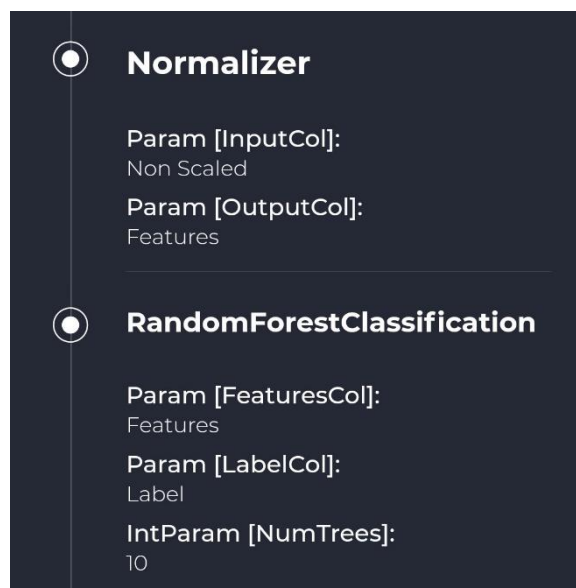


Fig 36: Pamja e parë e dy fazave

Gypat janë të dobishëm jo vetëm që procesi të jetë optimal por edhe për lexueshmëri. Gypi formohet në mënyrë dinamike dhe varësisht metodave të cilat i përdorim gjatë periudhës fillestare të mësimin të makinës, krijohen fazat përkatëse. Ky ekran është realizuar me anë të API-të *getStages*.

4.1.4 Mostrat e të Dhënave

Për të shfaqur më mirë për përdoruesin të dhënat e grumbulluara, kemi krijuar një listë me emrat e attributeve të data seteve përkatëse, listë kjo e cila gjenerohet dinamikisht, ku secili element përfshin në vete një përqindje (20%) të vlerave të cilat japin një reprezentim për llojin e të dhënave të cilat shtrihen përgjatë fushave (Fig 39, Fig 38). Një shembull i kolonave të gjeneruara dhe të vizualizuara duket kështu.

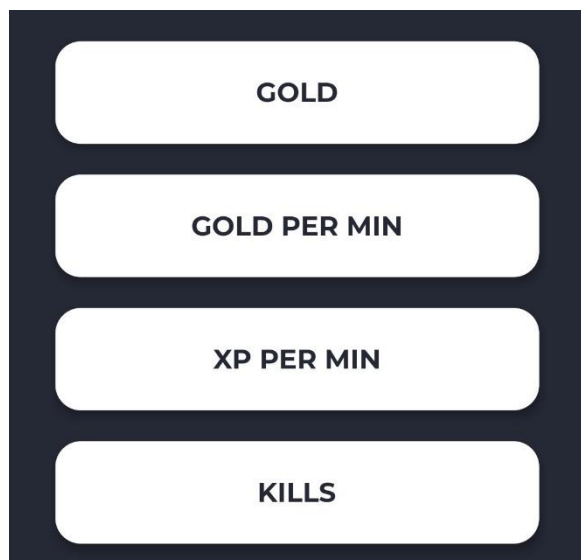


Fig 39: Pamja e kolonave në mostër

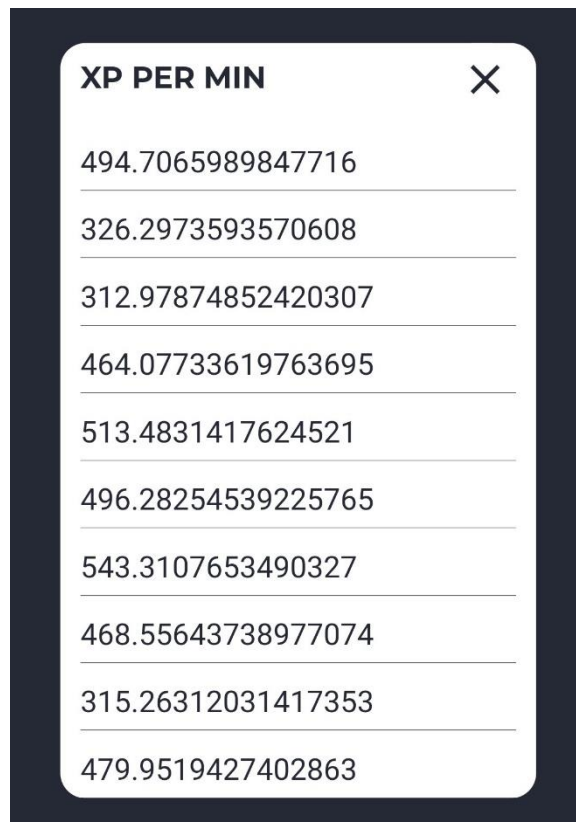


Fig 38: Pamja e vlerave në mostër

Ndërsa me klikimin e atributit *XP PER MIN* do të hapet një fushe me vlerat e këtij atributi, përshtypja e parë e së cilave na len të kuptohet se janë vlera numerike decimale dhe se i përkasin një rangui nga 300 deri 600. Ky ekran është realizuar me anë të API-të *getSample* që pranon dy parametra, data setin që në këtë rast është Kaggle dhe një përqindje që në këtë rast është 20. Meqenëse është e vështirë të vizualizohen të dhënat e një mostre në tabelë në një sistem operativ iOS apo Android atëherë për shkak të dukjes, është parë më e arsyeshme që të shfaqen kolonat vertikalisht.

4.1.5 Matrica e Korrelacionit

Matrica e korrelacioni është një lloj tabele që tregon koeficientet e korrelacionit ndërmjet variablave [14]. Secila celulë në tabelë tregon lidhjen midis dy variablave. Matrica është në formë katrore, duke figuruar të njëjtat emra atributesh në të dyja boshtet. Shërben për të përmbledhur sasi të mëdha të të dhënave, ku qëllimi kryesor është që të fitohet një model, e ku tek ne në rastin e data setit të Kllasterimit vërehet se atributet janë shumë të lidhura me njëra tjetrën, ndërsa tek Klasifikimi mbizotëron më shumë pavarësia ndërmjet attributeve. Kodimi i korrelacioneve është i rangut të vlerave nga -1 deri në 1, ku 0 (Fig 40) tregon se vlerat aspak nuk korrelojnë me njëra-tjetrën, 1 do të thotë se variablat korrelojnë shumë ndërsa -1 do të thotë se ekziston korrelacion invers i lartë.

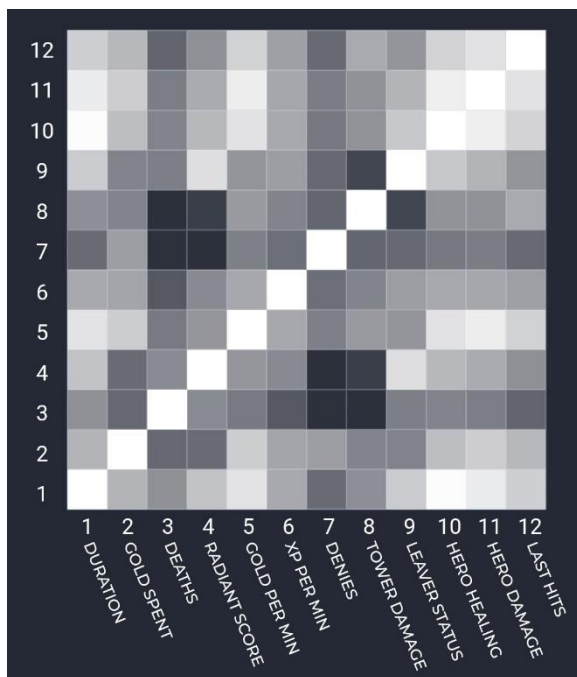


Fig 42: Matrica e korrelacionit të data setit
Kaggle

Kodimi mund të bëhet edhe në rangje të tjera por kjo nuk ka efekt të madh pasi çfarë do vlerë mund të pasqyronte vartësinë në mënyrë të duhur.

Për të shkallëzuar variablat nuk është përdorur ndonjë librari për këtë qëllim por metoda *Correlation* në Scala ka një shkallëzues unik të integruar në vete. Meqë në Flutter nga paketa e cila është përdorur për vizualizim nuk ka pasur ndonjë *Widget* të gatshme për krijimin e matricës së korrelacionit, është përdorur një klasë e avancuar e Flutter, *CustomPaint*, e cila ofron një kanvacë (ang. *Canvas*) për vizatimin e formave të ndryshme të cilat nuk do të mund të krijoreshin në mënyrë të saktë me anë të *Widgets* të cilat ekzistojnë. Kështu nga kjo klasë të cilën ne e trashëgojmë dhe përdorim metodën vizato (ang. *paint*), tek e cila edhe shënojmë funksionet për të vizatuar në mënyrë dinamike celulat e vartësisë. Ky ekran është realizuar me anë të API-të *getCorrelationMatrix* që pranon si parametër data setin që duam të aplikojmë matricën (Fig 42, Fig 41).

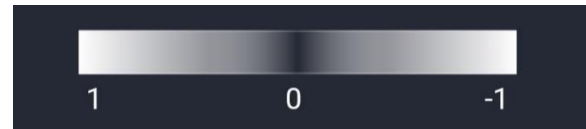


Fig 40: Vlerat fundore të korrelacionit

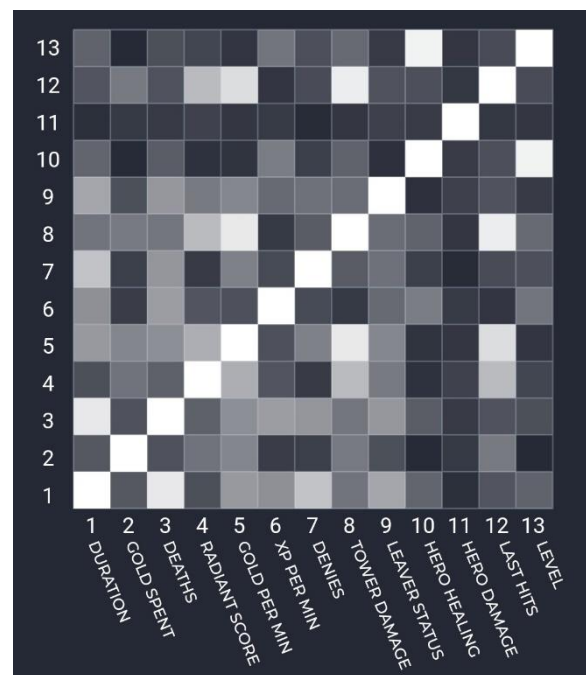


Fig 41: Matrica e korrelacionit të data setit
Steam

4.1.6 Struktura e Data Seteve

Në këtë pjesë të aplikacionit kemi ofruar një pamje e cila na informon për numrin e attributeve dhe fushave të përdorura për secilin data set, si dhe kemi mundësinë të shohim tipin e të dhënave për secilin atribut, ndonëse thuajse të gjithë kanë tip të njëjtë të të dhënave dhe dallojnë vetëm nga data seti në data set. Në rastin aktual (Fig 43), si data set po përdorim Kaggle nga shohim se tipi i atributit të klikuar është Double ndërsa data seti përbëhet nga 12 kolona dhe 110 rreshta. Rikujtojmë se ky data set është i para-procesuar dhe se data seti origjinal përmban përafërsisht 500.000 rreshta dhe rreth 70 kolona.

Ky ekran është realizuar me gërshetimin e dy API të përdorur, që janë *getSchema* që mundëson listimin e kolonave bashkë me tipin e tyre, ndërsa *getStats* ofron numrin e kolonave, rreshtave dhe burimin e data setit për të cilin është gjeneruar statistika.

Duhet cekur se ky ekran kërkon kohën më të lartë për procesim dhe hapje, pasi kalkulimi i rreshtave dhe kolonave bëhet në kohë reale, pra me klikimin e butonit, njihen këto të dhëna dhe shfaqen në ekran, që do të thotë se nuk ekzistojnë vlera të ruajtura paraprakisht në aplikacion.

GOLD	GOLD PER MIN	XP PER MIN
KILLS	DEATHS	ASSISTS
DENIES	LAST HITS	DOUBLETTYPE
HERO HEALING	TOWER DAMAGE	LEVEL

ROWS	COLUMNS	SOURCE
110	12	KAGGLE

Fig 43: Struktura e data seteve

4.1.7 Klasifikimi dhe kllasterimi në Kohë Reale

Ndër pjesët kryesore të aplikacionit është klasifikimi dhe kllasterimi në kohë reale. Me kohë reale nënkuptojmë mundësinë që të përdoren modelet e trajnuara kurdo që klikojmë butonin *Predict* me kolonat e caktuara. Pamja e parë duket kështu si për klasifikim edhe për kllasterim. Shohim () se tek secila hapësirë në të cilën mund të shkruajmë kemi emrin e kolonës që i përket hapësirës dhe një vlerë numerike, që paraqet vlerën e para definuar të kolonës. Ky ekran përbëhet nga 3 thirrje fundore, të cilat janë:

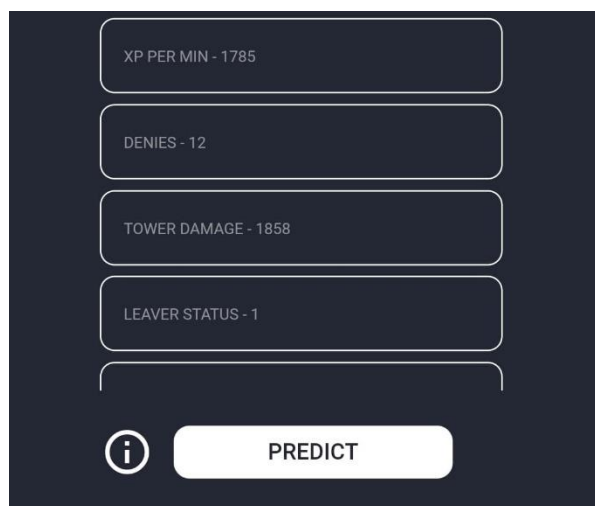


Fig 44: Pamja e klasifikimit në kohë reale

- `postPredict` - për të realizuar klasifikimin ku si hyrje jepen kolonat e data setit ndërsa si dalje ofrohet një tekst ku tregohet se a e kemi fituar lojën apo humbur.
- `postCluster` - për të realizuar kllasterimin ku si hyrje jepen kolonat e data setit ndërsa si dalje ofrohet numri i kllasterit në të cilin janë grupuar të dhënat.
- `getColumnns` - që liston emrat kolonave bashkë me vlerat e para definuara.

Me kryerjen e procesit të klasifikimit apo kllasterimit, na shfaqet një tekst në të cilin tregohet se sa është vlera e predikuar apo se cili kllaster i përkasin vlerat e reja.

Bazuar në të dhënat e ofruara, me shumë gjasë ju do të humbisni

MBYLL

Tek pamja e parë shihet se me këtotribute të para definuara ne do të humbim lojën, siç mund të kuptohet edhe nga teksti që shfaqë 'Bazuar në të dhënat e ofruara, me shumë gjasë ju do të humbisni'.

Bazuar në të dhënat e ofruara ju do i takoni kllasterit 1

MBYLL

Ndërsa me atributet e para definuara të kllasterimit, pra pamjes së dytë, shohim se do t'i takojmë grupimit numër 1, pra 'Bazuar në të dhënat e ofruara ju do i takoni kllasterit 1'.

Se çfarë të dhënash ka grupimi 1, do të mund të kuptohet nëse lexohet pjesa e Kllasterët.

4.1.8 Kllasterët

Ekrani i kllasterimit paraqet në mënyrë grafike të dhënat e kllasteruara në një renditje të caktuar. Shpjegojmë këtë menu duke u bazuar tek pamja më poshtë (Fig 45). Kjo menu vlenë

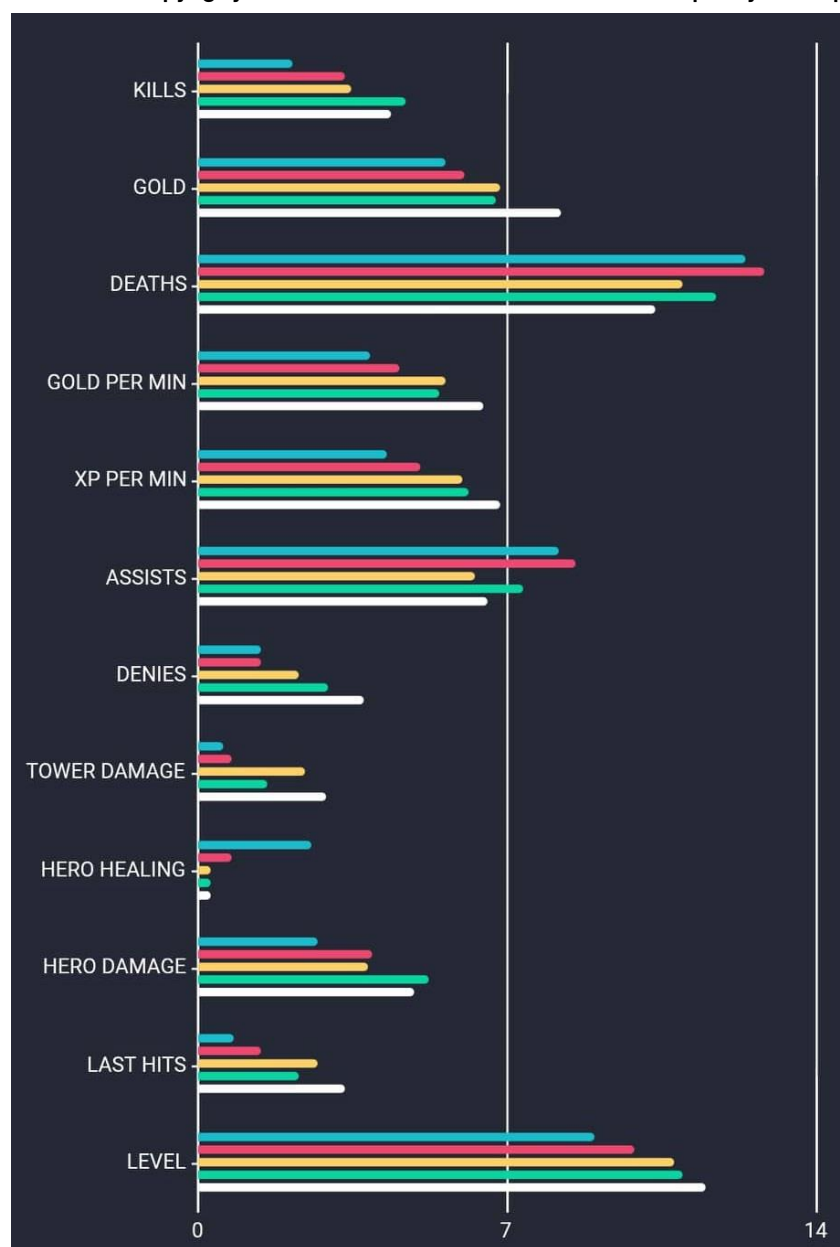


Fig 45: Kllasterët në mënyrë grafike

vetëm për procesin e kllasterimit, pra nuk ekziston opsioni për rrëshqitje. Shohim se në të majtë, kemi një varg emrash që reprezentojnë atributet e data setit Kaggle. Secili atribut shoqërohet me 5 ngjyra të ndryshme, ku renditja prej lart – poshtë paraqet numrin e kllasterit, që do të thotë se ngjyra e kaltër i përket kllasterit 1, ngjyra e kuqe i përket kllasterit 2, ngjyra e verdhë kllasterit 3, ngjyra e gjelbër kllasterit 4 dhe në fund ngjyra e bardhë i përket kllasterit 5.

Me të identifikuar kllasterët, fillojmë me analizën e tyre deri sa të nxjerrim të dhëna të shëndosha dhe praktike për secilin grupim. Rikujtojmë se Dota 2 është një video lojë ku lojtarët janë të ndarë në role të caktuara, diçka që edhe ne do të stimulojmë të nxjerrim varësisht attributeve.

Po fillojmë me kllasterin 1, duke vazhduar tutje, duke krijuar një tabelë për secilin anëtarë

4.1.8.1 Kllasteri me ngjyrë të kaltër - *Mbështetësit mjekues*

Nga grafiku, shohim se ky grupim shquhet me shumë pak vrasje (*kills*), pak para (*gold*), shumë vdekje (*deaths*), pak para për minutë (*gold per min*) dhe shumë asistime (*assists*). Me këto statistika, konstatojmë se ky grupim nuk është prej vrasësve, pasi vrasësit do të kishin para me anë të së cilave do të furnizoheshin me vegla dhe do të vrisnin hero tjerë, kjo do të thotë që me këto aftësi, numri i vdekje do të ishte më i vogël (pasi vetë mbrojtja do të ishte më e madhe). Me një numër të madhe asistimesh, themi se ky grup i takon më shumë mbështetësve, por le të vazhdojmë me analizim të më tutjeshëm që saktë të emërojmë këtë grupim. Themi se ky grup ka pak mohime, ju shkakton pak dëme kullave dhe mjekojnë shkëlqyeshëm. Me këto të dhëna themi se ky grup me saktësi i takon mbështetësve pa vazhduar më tutje, pasi krahasimi i këtij grupimi me të tjerë në aspektin e mjekimit ka një avantazh të dukshëm. Pak goditje të fundit dhe një level i ulët vetëm sa i përforcojnë këto që thamë. Konkludojmë se ky grupim i takon mbështetësve, atyre që sakrifikojnë veten për një rol tjetër.

Do e quajmë këtë grup *Mbështetësit mjekues* pasi karakterizohen dukshëm me një atribut të ndryshëm. Duhet cekur se mbështetësit mund të mos jenë mjekues tërë kohën, në lojë mund të mbështetesh një hero edhe pa e mjekuar.

4.1.8.2 Kllasteri me ngjyrë të kuqe - *Mbështetësit e dorës së dytë*

Me pak vrasje, pak para dhe një numër konsiderueshëm të lartë të vdekjeve, themi se as ky grup nuk është një që kryen vrasje. Megjithatë, duke u bazuar në attribute, ky grup akoma mund t'i takoj një grupi që shtynë kulla, që mbështetin apo që janë të parët në linjë. Vazhdojmë me një numër të vogël të eksperiencës për minutë dhe një numër të madh asistimesh. Pak mohime dhe pak shtyrje kullash. Përfundimtare mundësitë që grupimi ti takoj shtyrësve të kullave. Mbetet të shikohet se a është grup që janë të parët në linjë apo një mbështetës por jo mjekues. Vazhdojmë me atributet dhe shohim se ky grupim mjekon në një sasi të caktuar dhe shkakton dëme heronjve të tjerë në një masë. Ka pak goditje përfundimtare dhe pak nivele. Me këto të dhëna, themi se edhe ky grup i takon mbështetësve, por në këtë rast jo mjekuesve, por *Mbështetësve të dorës së dytë*.

4.1.8.3 Kllasteri me ngjyrë të verdhë - *Të parët në linjë*

Ky grupim shquhet me vrasje mesatare, dhe para mbi mesataren, pak numër vdekjesh dhe eksperiencë relativisht të madhe. Me shumë pak asistime, shumë mohime dhe shtyrje kullash të madhe, por pak mjekim. Me këto informacione, bie teoria se ky grupim mund t'i takojë ndonjë grupi tjetër të mbështetësve, pasi ato shquheshin për mjekim apo asistime dhe relativisht pak para dhe eksperiencë. Ekzistojnë akoma të hapura mundësitë që ky grup ti takojë vrasësve, shtyrësve të kullave apo të parëve në linjë. Vazhdojmë me atributet tjera; Pak dëmtime heronjve të tjerë dhe shumë goditje përfundimtare, dhe nivel të lartë. Me të thëna këto, emri i këtij grupi mund të jetë *Të parët në linjë*. Bazuar nga eksperiencia në lojë, këto heronj, përveç këtyre attributeve, shquhen për lojë të vetmuar (pak asistime) dhe shtyjnë kulla por jo më së shumti, megjithëse roli i tyre e kërkon këtë.

4.1.8.4 Kllasteri me ngjyrë të gjelbër – *Vrasësit / Playmakerët*

Numër i madh vrasjesh, numër i parave dhe vdekjeve janë karakteristikat që e dallojnë këtë grupim për të vazhduar me eksperiencë të lartë dhe numër të madh asistimesh. Një eksperiencë e lartë dhe një numër i madh asistimesh nuk janë diçka që i karakterizon mbështetësit, andaj ky opsion nxirret nga loja. Shumë mohime dhe konsiderueshëm dëme kullave por jo mjekim. Vazhdojmë me dëmtim masiv ndaj heronjve dhe numër të lartë të goditjeve përfundimtare dhe nivel shumë të lartë. Ne mund të identifikojmë këtë grupim me emrin *Vrasës* apo *Playmaker* për disa arsye. Vrasës pasi i ka të thepisura atributet që identifikojnë këtë emër, bashkë me paratë dhe eksperiencën. Ndërsa *Playmaker* pasi ka numër të madh vdekjesh dhe nuk shquhet për shtyrje masive kullash. Numri i asistimeve e ndihmon këtë emërtim që i kemi vënë e edhe niveli i lartë.

4.1.8.5 Kllasterimi me ngjyrë të bardhë – *Ndjekësit e objektivave*

Me një numër të lartë të vrasjeve, një numër maksimalisht të lartë të parave dhe numër të vogël të vdekjeve, shohim se ky grupim i të dhënave është unik, pasi për secilin atribut të lartcekur ka një marzhë ideale. Me shumë eksperiencë dhe numër mesatarë të asistimeve, themi se ky grup definitivisht nuk i takon mbështetësve. Vazhdojmë me një numër të madh mohimesh dhe numër maksimal dëmsh të kullave, shumë pak mjekim dhe shumë dëmtime heronjve. Shumë goditje përfundimtare dhe maksimalisht nivele. Ekzistojnë shumë emërime që mund të ketë ky atribut, por nuk mund të grupohet tek të parët në linjë pasi ka numër të madh të vrasjeve dhe dëme heronjve. Mund të futet në kategorinë e vrasësve por as ky emërim nuk e identifikon këtë kategori në mënyrë ideale pse ka shtyrje maksimale ndaj kullave. Ky kllaster ka të dhëna ideale për secilin atribut, të dhëna që fitohen vetëm nëse loja luhet në mënyrë perfekte dhe ndiqen objektivat në mënyrë të saktë, prandaj do e quajmë *Ndjekësit e objektivave*.

Nëse bëjmë një përmbledhje të këtyre grupimeve dhe emrave, themi se:

<i>Kllasteri</i>	<i>Ngjyra</i>	<i>Emri</i>
I	E kaltër	<i>Mbështetësit mjekues</i>
II	E kuqe	<i>Mbështetësit e dorës së dytë</i>
III	E verdhë	<i>Të parët në linjë</i>
IV	E gjelbër	<i>Vrasësit / Playmakerët</i>
V	E bardhë	<i>Ndjekësit e objektivave</i>

4.1.9 Grupimi i Kllasterëve

Në fund, në mënyrë që të shohim disa statistika bazike për kllasterimin e realizuar më lartë, kemi ekranin Grupimi i Kllasterëve (Fig 46). Ky ekran mundësohet nga API *getClusterCount* që në princip paraqet një numërues të thjeshtë. Nga figura shohim se në:

- Kllasterin numër 0, të quajtur *Mbështetësit mjekues*, janë grupuar rreth 25 vlera (me ngjyrë të gjelbër të hapur).
- Kllasterin numër 1, të quajtur *Mbështetësit e dorës së dytë*, janë grupuar rreth 32 vlera (me ngjyrë të kuqe).
- Kllasterin numër 2, të quajtur *Të parët në linjë*, janë grupuar rreth 16 vlera (me ngjyrë të verdhë).
- Kllasterin numër 3, të quajtur *Vrasësit / Playmakerët*, janë grupuar rreth 24 vlera (me ngjyrë të gjelbër të errët).
- Kllasterin numër 4, të quajtur *Ndjekësit e objektivave*, janë grupuar rreth 13 vlera (me ngjyrë të bardhë).

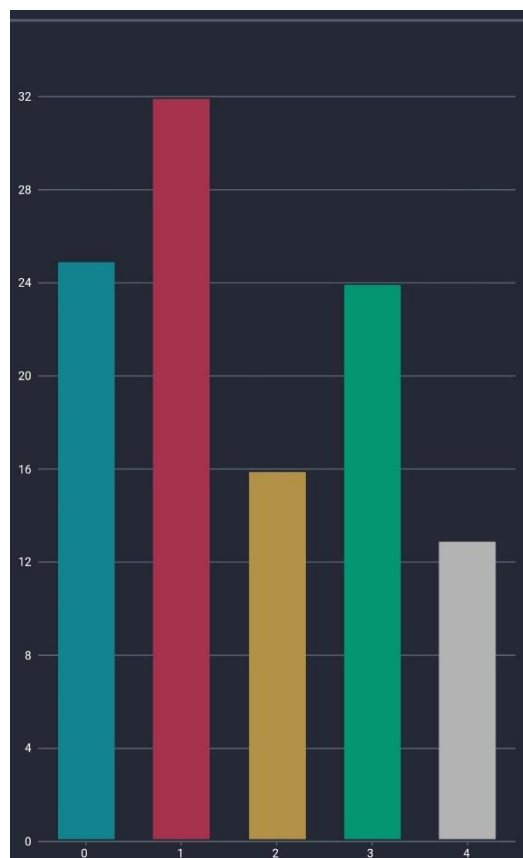


Fig 46: Grupimi i kllasterëve

5 Konkluzioni

Projekti është i ndarë në dy pjesë, pjesa *frontend* dhe pjesa *backend* nga edhe kanë ardhur vështirësitë më të shumta, pasi vizualizimi dhe përshkrimi i funksioneve kryesore është dashur të bëhet në koordinim të plotë. Gjithsesi, eksperiencia e krijimit të një platforme të tillë në të cilën koordinohen pjesa *frontend* dhe pjesa *backend* ka qenë e veçantë dhe do të hyjë në përdorim shumë në të ardhmen, pasi në ditët e sotme rrallë herë punohet me *frontend* të integruar në skeletin (ang. *framework*) përkatës.

Megjithëse në industri ekzistojnë projekte të ngjashme, asnjë nuk është i tillë. Ndër projektet më të spikatura që edhe nga jemi inspiruar janë Opendota dhe Dotabuff, që integrojnë grafike me statistika në kohë reale dhe japin një ndërfaqe të këndshme në të cilën lehtë mund të arrihet improvizimi, megjithatë asnjëri nuk integron në to të mësuarit e makinës, qoftë ai kllasterim apo klasifikim, apo ndonjë lloj të mësuarit tjetër.

Pra, në IntelliDota Clustering është përdorur algoritmi Bisecting K-Means, që është një version i K-Means. Ky algoritëm është përdorur në projektin tonë me vlerën e K-së 5 dhe me 25 iterime, në mënyrë që rezultati të jetë më i saktë. Algoritmin e trajnuar e kemi ruajtur me anë të imazhit Docker. Këtë algoritëm e kemi aplikuar në një data set të para-procesuar, pasi data seti jo i para-procesuar përmban 500000 rreshta dhe 12 kolona, e një kllasterim në një data set të tillë nuk do të kishte kuptim.

Para-procesimi përmban kryesisht grupimin e data setit në kolonën *hero_id*, kolonë kjo që identifikon heroin në mënyrë unike. Pra data seti normal përbëhet nga 500000 rreshta, e secili rresht përshkruan sjelljen e një heroi për një lojë përkatëse. Me një grupim të tillë, reduktohet madhësia në 110 rreshta dhe si agregim, kemi përdorur mesataren. Pra data seti përfundimtarë në të cilin aplikohet algoritmi përmban 110 rreshta dhe 12 kolona. Nga pjesa e parë e projektit IntelliDota, dihet që ekziston një imazh që nuk është i publikuar në të cilin shërbehen ndërfaqen programuese të aplikacionit, për dy data setet.

Publikimi i këtij imazhi ndërthur Docker dhe Google Cloud SDK, pra imazhin e krijuar e ngarkojmë në Google Cloud Platform me anë të Google SDK, që paraqet një set veglash që mundësojnë ngarkimin nëpërmjet ndonjë CLI. Publikimi në Cloud kryhet duke i shoqëruar imazhit një nofkë e duke e shtyrë më pas për në server. Pason konfigurimi i tij në ndërfaqen e Google sic është përshkruar më lartë.

Realizimi i aplikacionit, i kryer me anë të Flutter që përdor gjuhën programuese Dart, përfshinë mundësinë për vizualizim, për të aplikuar metrika si dhe për analiza. Ndër tërë këto opsione, më konkretisht, kemi mundësinë për diskretizim të të dhënave, ku të dhënat me vlera të vazhdueshme grupohen varësisht numrit që zgjedhim ne, qoftë ai 5, 10, 15 apo 20 dhe shfaqet grafiku me vlerat përkatëse sipas grupimeve, kemi mundësinë për të parë fazat e modelit të trajnuar, mundësinë për të parë matricën e korrelacioneve, mundësinë për predikim dhe kllasterim në kohë reale, mundësinë për të parë grupimet e kllasterëve pasi të shkallëzohen vlerat e të tjera.

Ndërkaq, projekti jonë aktualisht nuk mund të zgjerohet më shumë por mund të përdoret si një aplikacion bazë për statistika interesante. Megjithatë, gjithsesi se do të vazhdohet të punohet me të mësuarit e makinës dhe të nxirren analiza më të thella si për video loja, edhe për raste konkrete. Në video lojëra, mund të hidhet edhe një hap i ardhshëm si për shembull, nxjerrja e analizave se cilat ishin arsyet kryesore që një ekip fitoi. Niveli i të mësuarit të makinës që duhet zhvilluar për këtë aplikacion nuk është më i avancuar se sa niveli që po përdorim aktualisht, por aftësitë analitike duhet rritur, pasi gjithçka përsëri nxirret nga grafet.

Elemente tjera me rëndësi edhe për projektin edhe për të ardhmen kanë qenë përdorimi i strukturës Json, pra manipulimi i attributeve nëpërmjet Scalas, përdorimi i Gson si librari ndërmjetësuese e Json dhe Scalas dhe marrja e të dhënave nëpërmjet vartësive të Scalas, si *lihaoyi* me anë të së cilës kemi realizuar kërkesat. Ja vlen të ceket edhe njëherë aplikacioni *ngrok* me anë të së cilit kemi mundur të publikojmë pikat fundore tonat ashtu që të mund të realizohet pjesa e dytë, pra ajo *frontend*.

Të gjitha pjesët e këtij aplikacioni janë publike, duke filluar nga data seti i krijuar, tek Scala, lidhja ndërmjet tyre me REST modelin si dhe algoritmi i klasifikuar dhe kllasteruar në mënyrë që zhvilluesit të ndihmohen sado pak.

6 Shtojcat

Më poshtë janë listuar kodet burimore dhe data seti:

- Backend: <https://github.com/LabinotVila/IntelliDota>
- Modelet e trajnuara: <https://github.com/noraibrahimi/IntelliDotaIC>
- Frontend: <https://github.com/noraibrahimi/IntelliDota-mobile>
- Steam data seti: <https://www.kaggle.com/labinotvila/dota-2-steam-api-fetched-dataset>

7 Lista e thirrjeve fundore të mundshme

7.1 index

host / index

Parametrat:

Përshkrimi: Kthen përmbajtjen e faqes filluese, që është lista e pikave fundore të ofruara nga ne.

Tab 1: Pika fundore index

7.2 getColumns

host / getColumns

Parametrat: **kind** – lloji i datasetit [steam / Kaggle]

Përshkrimi: kthen një listë të kolonave të datasetit përkatës

Shembull: /getColumns?**kind**=steam

Tab 2: Pika fundore getColumns

7.3 getSample

host / getSample

Parametrat: **kind** – lloji i datasetit [steam / Kaggle]

percentage – përqindja e monstrës [0 deri 100]

Përshkrimi: kthen një monstër të datasetit përkatës, varësisht përqindjes

Shembull: /getSample?**kind**=steam&**percentage**=10

Tab 3: Pika fundore getSample

7.4 getStages

host / getStages

Parametrat:	kind	- lloji i datasetit [steam / Kaggle]
Përshkrimi:	kthen një varg të fazave nëpër të cilët kanë kaluar të dhënat	
Shembull:	/getStages? kind =kaggle	

Tab 4: Pika fundore getStages

7.5 getCorrelationMatrix

host / getCorrelationMatrix

Parametrat:	kind	- lloji i datasetit [steam / Kaggle]
Përshkrimi:	kthen një varg numrash që tregojnë ndërlidhjen mes kolonave të datasetit përkatës	
Shembull:	/getCorrelationMatrix? kind =kaggle	

Tab 5: Pika fundore: getCorrelationMatrix

7.6 getGroupAndCount

host / getGroupAndCount

Parametrat:	kind	- lloji i datasetit [steam / Kaggle]
	attribute	- grupimi ndodhë sipas këtij atributi
	partitions	- numri i ndarjeve të të dhënave numerike
Përshkrimi:	kthen një varg grupesh dhe intervalin përkatës që të dhënat i përkasin	
Shembull:	/getGroupAndCount? attribute =xp_per_min& partitions =3	

Tab 6: Pika fundore getGroupAndCount

7.7 getStats

host / getStats

Parametrat:	kind	- lloji i datasetit [steam / Kaggle]
Përshkrimi:	kthen sasinë e rreshtave dhe kolonave të datasetit përkatës	
Shembull:	/getStats? kind =steam	

Tab 7: Pika fundore getStats

7.8 getSchema

host / getSchema

Parametrat:	kind	- lloji i datasetit [steam / Kaggle]
Përshkrimi:	kthen kolonat dhe tipin përkatës të datasetit të caktuar	
Shembull:	/getSchema? kind =steam	

Tab 8: Pika fundore getSchema

7.9 getDoubleGroup

host / getDoubleGroup

Parametrat:	kind	- lloji i datasetit [steam / Kaggle]
	col1	- kolona e parë
	col2	- kolona e dytë
Përshkrimi:	kthen grupimin dhe njehsimin sipas kolonave përkatëse	
Shembull:	/getDoubleGroup? kind =steam& col1 =leaver_status& col2 =radiant_win	

Tab 9: Pika fundore getDoubleGroup

7.10 getClusterStats

host / getClusterStats

Parametrat:	
Përshkrimi:	Kthen pikat qendrore të të dhënave për secilin kllaster, pra informata për secilin atribut të data setit
Shembull:	/getClusterStats

Tab 10: Pika fundore getClusterStats

7.11 getClusterCount

host / getClusterCount

Parametrat:	
Përshkrimi:	Kthen numrin e sasisë së të dhënave për secilin kllaster
Shembull:	/getClusterCount

Tab 11: Pika fundore getClusterCount

7.12 postCluster

host / getClusterCount

Parametrat:	gold	- paratë
	gold_per_min	- paratë për minutë
	xp_per_min	- eksperiencia për minutë
	kills	- vrasjet gjatë lojës
	deaths	- vdekjet gjatë lojës
	assists	- asistimet
	denies	- mohimet e vrasjeve
	last_hits	- goditja përfundimtare për vrasje
	hero_damage	- dëmtimet ndaj kundërshtarëve
	hero_healing	- ndihmesa ndaj ekipit
	tower_damage	- dëmtimet ndaj kullave
	level	- nivelet
Përshkrimi:	Kthen atributet përkatëse bashkë me një atribut të ri që tregon llojin e kllasterimit në të cilin përket data seti	
Shembull:	<code>/postCluster?</code> <code>gold=2000.0&gold_per_min=5113.0&xp_per_min=1231.0&kills=21.2</code> <code>&deaths=3.3&assists=10.0&denies=34.3&last_hits=411.1</code> <code>&hero_damage=56000.2&hero_healing=0.0&tower_damage=22000.0</code> <code>&level=25.0</code>	

8 Referencat

- [1] P. Norvig dhe S. J. Russell, *Artificial Intelligence: A Modern Approach*, Harlow, United Kingdom: Prentice Hall, 2009.
- [2] P. Norvig dhe S. J. Russell, «Artificial Intelligence: A Modern Approach,» në *Artificial Intelligence: A Modern Approach*, Harlow, United Kingdom, Prentice Hall, 2019, p. 37.
- [3] P. Norvig dhe S. J. Russell, «Artificial Intelligence: A Modern Approach,» në *Artificial Intelligence: A Modern Approach*, Harlow, United Kingdom, Prentice Hall, 2009, p. 2.
- [4] J. Han, M. Kamber dhe J. P. , *Data Mining: Concepts and Techniques*, Elsevier, 2020.
- [5] I. Dabbura, «K-Means Clustering Algorithm Applications,» 17 September 2018. [Në linjë]. Available: <https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a>. [Qasja 11 11 2019].
- [6] D. C. «Mean Shift,» [Në linjë]. Available: <https://courses.csail.mit.edu/6.869/handouts/PAMIMeanshift.pdf>. [Qasja 11 11 2019].
- [7] S. Devs, «Spark Clustering,» [Në linjë]. Available: <https://spark.apache.org/docs/latest/ml-clustering.html>. [Qasja 13 11 2019].
- [8] A. Agarwala dhe M. Pearce, «Learning Dota 2 Team Compositions,» [Në linjë]. Available: <http://cs229.stanford.edu/proj2014/Atish%20Agarwala,%20Michael%20Pearce,%20Learning%20Dota%202%20Team%20Compositions.pdf>. [Qasja 15 11 2019].
- [9] Wikipedia, «Kaggle,» [Në linjë]. Available: <https://en.wikipedia.org/wiki/Kaggle>. [Qasja 12 11 2019].
- [10] P. Bugnion, P. R. Nicolas dhe A. Kozlov, *Scala: Applied Machine Learning*, Packt Publishing, 2017.
- [11] K. Rimple, *A Gentle Intro to Docker for Developers*, Chariot Solutions, 2019.
- [12] Google, «Google Cloud Platform,» Google, 2017. [Në linjë]. Available: <https://cloud.google.com/>. [Qasja 29 11 2019].
- [13] D. Devs, «Dart Documentation,» 10-12 October 2011. [Në linjë]. Available: <https://dart.dev/guides>. [Qasja 23 10 2019].

- [14] T. Bock, «What is a Correlation Matrix,» [Në linjë]. Available: <https://www.displayr.com/what-is-a-correlation-matrix/>. [Qasja 24 11 2019].