Prepoznavanje vizuelnih šablona učenika zasnovanih na tehnologiji praćenja oka tokom polaganja

Boris Bibić E2 26/2019
Računarstvo i automatika
Fakultet tehničkih nauka
Univerzitet u Novom Sadu
borisbibic1996@gmail.com

Apstrakt — Eye tracker se koristi za praćenje pogleda očiju, a primenu je našao u obrazovanju, medicini i neuromarketingu. U ovom radu eye tracker tehnologija se koristiti za dobijanje sekvenci regiona od interesa prilikom testiranja učenika. Ove sekvence su grupisane u klase paterna, što dalje može pomoći nastavnicima da razumeju kako učenici posmatraju delove testa, te kako mentalno rezonuju pri polaganju e-testova. Sekvence regiona od interesa su prikupljene uz pomoć napisanog softvera za testiranje koje je sprovedeno nad 5 učenika sa ukupno 115 pitanja, a potom su transformisane u posmatrane atribute, klasterovane i podeljene u dobijene klase. Ovako uređen skup sekvenci regiona od interesa sa pripadajućim klasama korišćen je za treniranje prediktivnog modela koji prepoznaje šablone (definisane klase) u pogledu. Analiza atributa 12 prepoznatih klasa pokazala je svojstva različitih paterna pogleda učenika. Na kraju je kreirana softverska biblioteka koja se može koristiti za laku obradu i prepoznavanje šablona na osnovu prethodno sačuvanih sekvenci pogleda.

Ključne reči — paterni pokreta oka; obrazovanje; mašinsko učenje; klasterizacija; identifikacija obrasca

1. UVOD

Poslednjih godina u obrazovanju dolazi do prelaska sa papira na računarske tehnologije, a pogotovo sada usled pandemije novim korona virusom. Kako tehnologija postaje sve jeftinija i dostupnija svima, škole su jedna od institucija koje je koriste. Postoje mnoge oblasti u kojima obrazovanje ima koristi od razvoja tehnologije, ali ovaj rad se fokusira na tehnologiju koja može pomoći nastavnicima da procene kako učenici gledaju testove, u cilju razumevanja učenikovih kognitivnih procesa tokom polaganja. Praćenje pogleda i analiza paterna istih može pomoći i učenicima jer će omogućiti da se testovi pojedinačno prilagode učeniku, a ne učenik testu kao što je to do sada bio slučaj. *Eye tracker* [1] je tehnologija koja prati poglede očiju u prostoru i izračunava *x* i *y* koordinate pogleda (sa još puno drugih informacija koje nisu od značaja za ovaj projekat) u skoro realnom vremenu. Praćenjem ovih informacija tokom dužeg vremenskog perioda mogu se identifikovati paterni – ponavljajuće sekvence pogleda, koje imaju iste ili slične komponente pogleda, npr. iste sekvence regija od interesa ili vreme zadržavanja na određenim regijama od interesa. Nastavnici bi mogli iz šablona pogleda učenika da zaključe kako učenici razumeju određene aspekte testa (slika, pitanje, ponuđeni odgovori,...) i tako prilagode testove i/ili gradivo pojedinačnim grupama učenika. Takođe, učenici se mogu grupisati u klase na osnovu njihovih paterna pogleda dok rade test, dajući mogućnost nastavnicima da obrate više pažnje na određene grupe učenika (tokom testa, ali i tokom pripreme za test).

Cilj projekta je napraviti softversku biblioteku koja procesira rezultate praćenja pokreta oka učenika i klasifikuje učenika na neki od prethodno identifikovanih šablona pogleda. Rezultati pokreta oka su sekvenca regiona od interesa koje je učenik gledao, kao i vremenski interval koji je učenik proveo gledajući određeni region od interesa.

U poglavlju dva su predstavljena slična rešenja koja su se bavila tehnologijom praćenja pogleda i klasterizacijom podataka dobijenih ovom tehnologijom. Poglavlje tri se bavi procesom izrade željenog rešenja, kao i opisom prikupljanja skupova podataka, metodama i algoritama koji su korišćeni. Četvrto poglavlje predstavlja analizu dobijenih rezultata, a peto poglavlje sumarizaciju čitavog rada i diskusiju o mogućnostima unapređenja.

2. PRETHODNA REŠENJA

Među pronađenim rešenjima nema puno onih koji se bave problematikom prikupljanja, klasterovanja i identifikacije šablona pogleda. Ipak, postoje neka rešenja koja se mogu iskoristiti za dalja istraživanja, a koja su opisana u daljem tekstu poglavlja.

Autori rada [2] su koristili regije od interesa (engl. *Regions of interest - ROI*) koje su predstavljene kao slova. Na ovaj način problem sekvenci regiona transformisan je u problem sekvence slova. Ovaj problem se može rešiti na nekoliko načina, a autori su se opredelili za *Levenštajnovu distancu* za merenje različitosti stringova, *Nidlmen-Vanšov* algoritam koji koristi promenljivu matricu sličnosti za optimizaciju sličnosti sekvenci slova, kao i *hijerarhijsku klasterizaciju* uz grafičko klasterovanje. Autori ovog rada se nisu dalje bavili prepoznavanjem paterna što jeste bitan deo predloženog ~~projekta~~rada. Za razliku od [2] gde su sekvence *ROI* transformisane u sekvencu slova, u ovom radu su one pretvorene u vektore atributa koji sadrže informacije o dužini

trajanja i frekvenciji pogleda na *ROI*, kao i informacije o redosledu regija.

Eye tracker podaci imaju vremensku varijablu koja igra ključnu ulogu u rešavanju ovog problema. Kada je u pitanju klasifikacija vremenskih serija podataka, autori [3] su koristili *rekurentnu rezidualnu mrežu* za sekvencijalno učenje, kao i *Skip-Connected LSTM* i poredili ih sa "klasičnim" *LSTM*. Pokazali su da je *Skip-Connected LSTM* najbolji za klasifikaciju dugačke sekvence sa duboko povezanim podacima, što pogoduje dugačkim sekvencama *ROI*. Ipak, autori nisu koristili *ROI* dobijene uz pomoć *eye tracker-a*, pa ostaje na drugima da provere da li bi predložen algoritam uspešno klasifikovao *eye tracker* podatke. Predložen rad nije koristio neuronske mreže, jer su se one pokazale inferiornije u odnosu na klasične metode mašinskog učenja nad autorovim skupom podataka. U ovom radu, najbolje rezultate je dao *Logistic Regression* algoritam.

Rad [4] se bavio problemom automatske klasterizacije podataka kroz nenadgledane algoritme, gde su podaci dobijeni uz pomoć *eye tracker-a*. Da bi izvršili nenadgledanu klasterizaciju autori ovog rada su testirali tri algoritma: *KMeans*, *Spectral clustering* i *DBscan*. Podatke za klasterizaciju su izvlačili iz *eye tracker-a* da bi kreirali vlastite podatke, koje su kasnije klasterovali - ovim postupkom je možda izgubljena sekvencijalnost *ROI*, ali je problem transformisan u problem velike dimenzionalnosti podataka. Iako su klasterizacioni algoritmi iz [4] mogli biti korišćeni i za ovaj rad, autor se odlučio za *Dirichlet*-ov proces jer podaci iz *eye tracker-a* imaju vremenski karakter.

Slično, u [5] autori su prikupili skup podataka od 612 pokreta oka dok su test subjekti gledali kartografske mape. Iz kretanja oka izvučeno je 229 osobina dok se u predloženom radu koriste samo dve – regije koje su posmatrane i trajanje pogleda na te regije. Ovo ima za cilj da sačuva informacije o vremenu i redosledu gledanja *ROI*, koje autori [5] zanemaruju. Regije od interesa koriste se i u radu [6] gde je cilj da se paterni pokreta oka grafički prikažu, ali nije vršena klasterizacija istih, za razliku od ovog rada.

Razlike ovog rada i gorenavedenih jeste u celokupnosti procesa (od prikupljanja, preko analize i obrade podataka, do klasterizacije i identifikacije), kao i u konačnom proizvodu – softverska *Python* biblioteka koja se može koristiti bez znanja o programiranju ili poznavanja tehnologije koja se nalazi u pozadini.

3. METODOLOGIJA

Metodologija ovog rada uključuje tri oblasti: prikupljanje i obrada podataka, klasterovanje podataka i treniranje modela za predikciju. U daljem tekstu biće detaljno opisan svaki korak koji je uključen u metodologiju ovog rada.

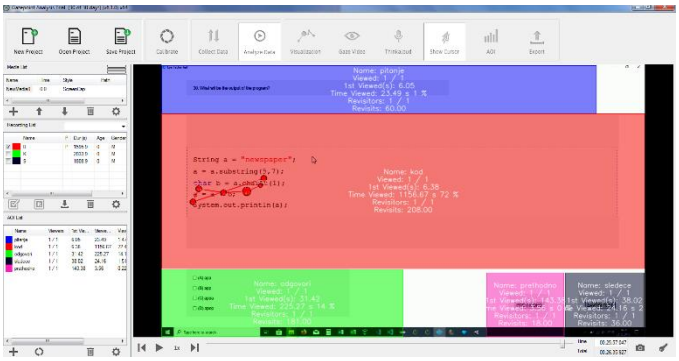
3.1 Prikupljanje podataka

Pre prikupljanje podataka, kreirana je aplikacija za testiranje učenika. Ova aplikacija omogućava da učenik prolazi kroz test, pitanje po pitanje, i bira jedan od četiri ponuđena odgovora. Ova aplikacija predstavlja autorovu verziju alata za e-testiranje, a mogla je da posluži i bilo koja od postojećih aplikacija za testiranje. Prednost autorove aplikacije jeste mogućnost određivanja pozicije određenih regija testa, tako da regije budu dovoljno udaljene jedna od druge. Ovo pomaže da se minimizuje greška *eye tracker* softvera gde postoje male fluktuacije u beleženju pogleda. Na slici 1 je prikazan primer jednog test pitanja u ovoj aplikaciji.

Slika 1. Prikaz ekrana aplikacije za e-testiranje učenika

Pripremljene su dve verzije testa. Prvu verziju sa po 5 pitanja je radilo 5 učenika, dok je druga verzija imala 30 pitanja i testirana su 3 učenika. Ovim je prikupljen skup podataka od 115 pitanja.

Prikupljanje pogleda oka i generisanje sekvenci regija od interesa urađeno je uz pomoć komercijalno dostupnog alata *Gazepoint* [7]. Za potrebe testiranja i izrade rada, ~~R~~regije od interesa su podeljene ~~na osnovu testa~~ na: *pitanje*, *kod*, *odgovori*, *prethodno*, *sledece*, *prazno* i *pitanje-kod*. Ove regije su uz pomoć alata nacrtane (slika 2), a sam alat je generisao rezultate pogleda. Softverska biblioteka dozvoljava unos proizvoljnog broja regiona proizvoljnog naziva. U radu će se koristiti 7 gore definisanih regiona za jednostavnije razumevanje alata.



Slika 2. Prikaz označenih regija u *Gazepoint* alatu

U ~~ove~~ rezultate alata [7] spadaju i podaci koji nisu od interesa, kao na primer prečnik zenice, pa je uz pomoć skripte generisan skup podataka koji se sastoji od 115 *csv* fajlova koji imaju strukturu prikazanu u tabeli 1:

Duration	ROI	Question
7,9	kod	T1_K_Q1
2,6	odgovori	T1_K_Q1
2,1	kod	T1_K_Q1
2,8	odgovori	T1_K_Q1
0,3	kod	T1_K_Q1
0,3	odgovori	T1_K_Q1

1,7	sledece	T1_K_Q1

Tabela 1. Primer jednog podatka – sekvenca pogleda

Jedan *csv* fajl sadrži dužinu gledanja na regiju u sekundama (*Duration*), naziv regije koja je gledana (*ROI*), kao i identifikator pitanja (*Question*). Ovo predstavlja jednu sekvencu pogleda učenika za jedno pitanje. Identifikator pitanja je sastavljen od tri dela: verzija testa, identifikator učenika i broj pitanja na testu.

3.2 Obrada podataka i priprema za klasterovanje

Klasterizacija podataka uz pomoć *Dirichlet*-ovog procesa [8] je zahtevala da se podaci formatiraju u jedan jedinstven *tsv* fajl. Redovi u tom fajlu predstavljaju pitanja, dok kolone predstavljaju vrednosti atributa paterna gledanja. Ovi atributi su dobijeni procesiranjem pojedinačnih paterna, tako da je svaki redosled gledanja *ROI* transformisan u vektor od 37 atributa i jedan *id* atribut. Atributi koji čine vektor su opisani ispod:

- *Ukupna dužina trajanja pitanja u sekundama*
- *Frekvencija/ukupan broj pogleda u pitanju*
- *Ukupan broj sekundi provedenih na regiji* (vektor od 7 vrednosti)
- *Frekvencija/broj pogleda po regijama* (vektor od 7 vrednosti)
- *Srednja vrednost broja sekundi provedenih na regiji* (vektor od 7 vrednosti)
- *Najčešće posmatrana regija nakon regije x* (vektor od 7 vrednosti)
- *Najčešće posmatrana regija pre regije x* (vektor od 7 vrednosti)

Ovi atributi su odabrani kao reprezentativni u sekvenci regija od interesa koja je prethodno definisana. Dužina vektora zavisi od broja regiona od interesa i računa se preko sledeće formule:

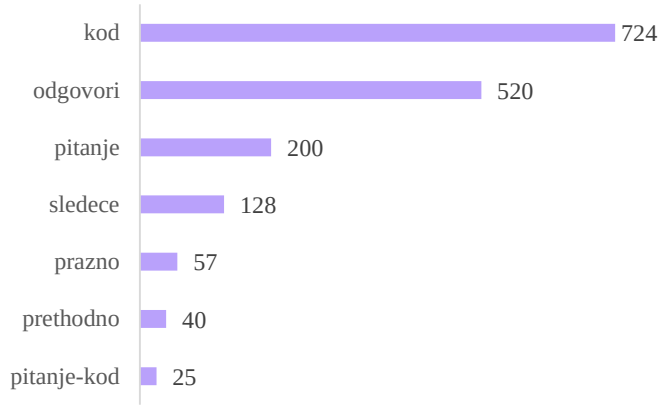
$$len(V) = 2 + 5 * len(ROI),$$

gde je $len(V)$ dužina vektora, a $len(ROI)$ je broj različitih regiona od interesa.

Dva bitne osobine paterna jesu redosled pojavljivanja regija i vreme gledanja u pojedinačne regije. Redosled regija je očuvan sa poslednja dva atributa, dok su ostali atributi orijentisani prema trajanju pogleda.

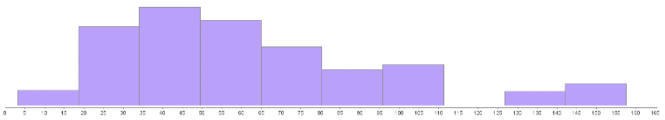
3.3 Statistika prikupljenih podataka

Nakon testiranja 5 ispitanika i 115 pitanja, izračunato je prosečno vreme po pitanju i ukupan broj gledanja u određene regije za sva pitanja zajedno. Statistika sa slike 3 pokazuje da su ispitanici najčešće vremena utrošili na čitanju koda (42,74%), a zatim na odgovorima (30,7%) i pitanju (11,81%).



Slika 3. Graf broja gledanja po regijama od interesa

Distribucija vremena provedenog na pitanju je prikazana na slici 4. Minimum provedenog vremena je 3,2 sekunde, maksimum je 157,6 sekundi, dok je prosečno vreme 63,213±35,35 sekundi.



Slika 4. Graf distribucije broja sekundi provedenih na pitanju

3.4 Klasterovanje podataka

Klasterovanje nelabeliranih podataka je služilo za dobijanje klastera, a zatim labeliranje podataka u dobijene klastere, tj. klase. Za ovo je iskorišćen već postojeći algoritam [9] koji radi klasterovanje gena na osnovu njihove ekspresije u toku vremena. *DP_GP_cluster* algoritam generiše klastere gena na osnovu njihove ekspresije tokom vremenskog perioda koristeći *Dirichlet*-ov model *Gauss-ovog* procesa. Ovaj algoritam je na ulazu primao obrađen *tsv* fajl, a na izlazu je davao, između ostalog, klase kojima pripadaju paterni pogleda iz skupa podataka. Detaljnija analiza klastera sledi u poglavlju 4.

3.5 Treniranje prediktivnih modela

Labeliranje redosleda gledanja regija od interesa u dobijene klase je urađeno ručno. Dobijeni skup podataka je činio jedan *csv* fajl od 115 redova i 37 kolona (primer sa 7 *ROI*), ne računajući zaglavlje kolone i reda. Ovaj skup podataka je testiran nad nekoliko različitih klasifikacionih algoritama: *Logistic Regression*, *Random Forest Classifier*, *MLP Classifier* i *SVM*. Izabran je najbolji klasifikacioni algoritam (*Logistic Regression*), koji je istreniran u *Python*-u i ubačen u softversku biblioteku za klasifikaciju učenika u neki od prethodno identifikovanih šablona pogleda.

Treba imati na umu da je prilikom definisanja novih regiona od interesa potrebno ponovo istrenirati klasifikacioni model sa podacima koji sadrže novodefinisane ROI. Ovo je posledica osobine prediktivnih modela da se klasifikacija ne može vršiti nad podacima sa promenljivom dužinom. Ukoliko se promeni broj ROI, menja se i dužina informacionog vektora koji predstavlja ulaz u predikcioni model.

3.6 Softverska biblioteka za identifikaciju šablona pogleda

Napisana biblioteka se može koristiti kao samostalna *Python* skripta i pokretati iz konzole. Na ulaz u skriptu se možegu prosleditи četiri pet parameara:

1. --*inputfile* ili –*i*: Ulazni fajl gde je smešten jedan patern kao u tabeli 1, ali bez kolone *Question* ili folder sa više ovakvih *csv* fajlova.
2. --*outputfile* ili –*o*: Izlazni fajl folder ili folder fajl gde će biti smešten *csv* fajl sa rezultatima skripte (*id* paterna, broj klastera kojem pripada patern, kao i vektor atributa za dati patern).
2.3. --*roi ili –r*: Tekstualni fajl sa nazivima regiona od interesa, gde je svaki naziv smešten u novu liniju.
3.4. --*quiet* ili –*q*: Parametar kojim se ispis na konzoli svodi na minimum.
4.5. --*verbose* ili –*v*: Parametar kojim se zahteva detaljan ispis na konzoli.

Ukoliko korisnik želi da definiše sopstvene regione, potrebno je da nazive regiona unese u tekstualni fajl i prosledi putanju do tog fajla uz pomoć –*r* parametra. *Quiet* parametar je zamišljen da se koristi kada ima puno paterna koje treba identifikovati, ali u tom slučaju je obavezan –*o* parametar. Ideja –*v* je da se korisniku omogući detaljno objašnjenje klastera kojem pripada patern sa ulaza. Primer ispisa na konzolu ukoliko nisu dodati –*v* i –*q* parametri je dat ispod:

Patern broj 105 pripada klasteru „Temeljan sa akcentom na koduTemeljito sa naglaskom na kodu“

Ukoliko se doda parametar –*v* ispis postaje sledeći:

Patern broj 105 pripada klasteru „Temeljito sa naglaskom na kodu“. Osobine ovog klastera su: ‘U proseku između 10 i 20 različitih pogleda. Mnogo vremena provedenog na kodu (u proseku najmanje 10 sekundi po pogledu). Prosečno potroše više od 50 sekundi po pitanju.‘

Kraći i duži opisi klastera se mogu definisati u posebnim tekstualnim fajlovima, a primeri u radu su dobijeni analizom prosečnih vrednosti atributa po klasterima. Ukoliko korisnik ne definiše ove opise, ispis na konzolu će (bez obzira na –v parametar) izgledati:

Patern broj 105 pripada klasteru 3

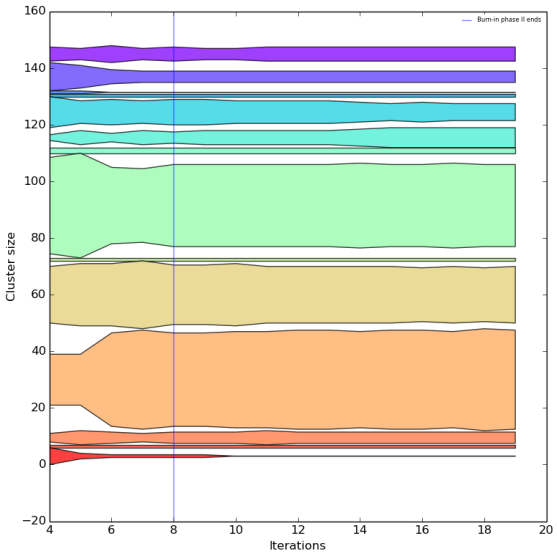
Rezultati ispitivanja brzine skripte pokazuju prosečnu brzinu od 8,26 milisekundi po pitanju bez –*q* i –*v* parametara, a 6,95 milisekundi po pitanju sa –*q* parametrom.

4. REZULTATI

Rezultat rada jeste identifikacija šablona gledanja. Da bi se šabloni uspešno identifikovali, prvo je potrebno labelirati podatke u određene klase. Ove klase su dobijene klasterovanjem uz pomoć *DP_GP_cluster* algoritma. Rezultati klasterovanja se nalaze u poglavlju 4.1, dok se rezultati klasifikacionih algoritama nalaze u poglavlju 4.2. Ovi rezultati su dobijeni kada je broj regiona od interesa bio 7 i razlikovaće se od rezultata dobijenih kada se broj regiona promeni.

4.1 Rezultati DP_GP_cluster algoritama

DP_GP_cluster algoritam je pokazao postojanje 12 nezavisnih klastera u koje se mogu grupisati paterni pogleda. Optimalan broj klastera je dobijen *Gibbs*-ovim uzorkovanjem nakon osam epoha što je prikazano na slici 5.

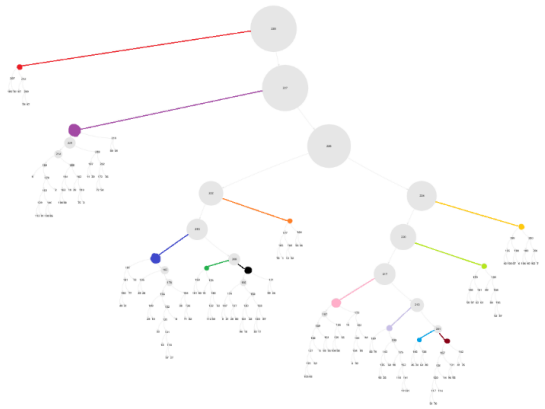


Slika 4. Promena veličine klastera pri *Gibbs*-ovom uzorkovanju tokom vremena

Kada se skup podataka propusti kroz algoritam za *hijerarhijsku aglomerativnu klasterizaciju* dobija se dendrogram sa slike 5.

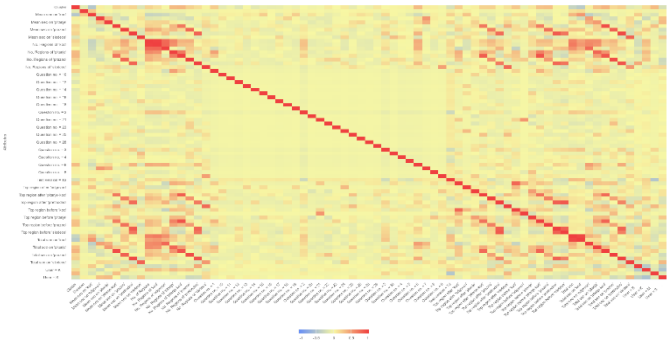
Slika 5. Dendrogram hijerarhijskog klasterovanja paterna posmatranja regija od interesa

Na dendrogramu se ne primećuje koliko ima različitih klastera jer se linija razdvajanja može staviti na bilo koju visinu dendrograma. Ali kada se model prikaže na stablu (slika 6), pojavljuju se distinktivna razdvajanja klastera koja idu u prilog *DP_GP_cluster* algoritmu i postojanju 12 klastera.



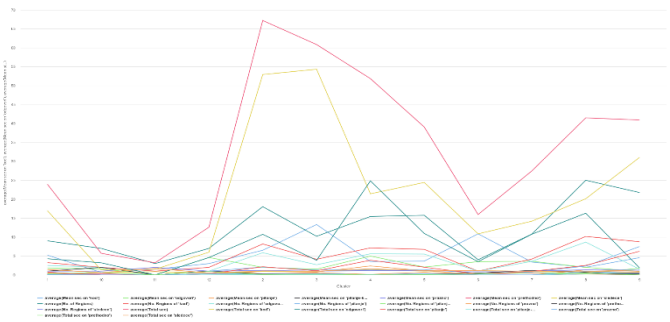
Slika 6. Stablo hijerarhijskog klasterovanja paterna

Matrica korelacije atributa vektora (slika 7) na osnovu kojih su dobijeni klasteri pokazuje vrlo jaku korelaciju (p>0.9) između ukupnog broja regija i broja pojavljivanja u paternu (p=0.984), kao i između vremena provedenog na pitanju i ukupnog vremena provedenog gledajući regiju '*kod*' (p=0.957). Kada se posmatra uticaj atributa na klastere, pronađena je samo slaba korelacija sa prosečnim (p=-0.458) i ukupnim (p=-0.422) vremenom provedenim gledajući regiju '*kod*'.



Slika 7. Matrica korelacije atributa

Zaključak o značaju pojedinih atributa u klasterima i, još bitnije, identifikacija paterna za krajnjeg korisnika sistema dobijena je na osnovu analize prosečnih vrednosti atributa po klasterima (slika 8).



Slika 8. Prosečne vrednosti atributa po klasterima

Kada se analiziraju pojedinačno atributi i njihove vrednosti za svaki klaster pojedinačno, dobije se tabela 2 iz koje su izvučeni zaključci o svakom klasteru, tj. o opisu klastera.

Klaster	Opis vrednosti atributa
1	U proseku manje od 10 različitih pogleda
	U proseku potroše manje od 30 sekundi po pitanju
2	U proseku između 10 i 20 različitih pogleda
	Česti pregledi odgovora (u proseku više od 3 puta po pitanju)
	Česti pogledi na kod (u proseku više od 5 puta po pitanju)
	U proseku potroše više od 50 sekundi na pitanje
3	U proseku između 10 i 20 različitih pogleda
	Mnogo vremena provedenog na kodu (u proseku najmanje 10 sekundi po pogledu)
	U proseku potroše više od 50 sekundi na pitanje
4	U proseku između 10 i 20 različitih pogleda
	Česti pregledi odgovora (u proseku više od 3 puta po pitanju)
	Česti pogledi na kod (u proseku više od 5 puta po pitanju)
	Gledaju odgovore duže (u proseku najmanje 4 sekunde po pogledu)
	Neko vreme provedeno gledajući test pitanje
	U proseku potroše više od 50 sekundi na pitanje
5	U proseku između 10 i 20 različitih pogleda
	Česti pregledi odgovora (u proseku više od 3 puta po pitanju)
	Česti pogledi na kod (u proseku više od 5 puta po pitanju)
	U proseku potroše oko 40 sekundi na pitanje
6	U proseku manje od 10 različitih pogleda
	Gledaju odgovore duže (u proseku najmanje 4 sekunde po pogledu)
	Mnogo vremena provedenog na kodu (u proseku najmanje 10 sekundi po pogledu)
	U proseku potroše manje od 20 sekundi na pitanje
7	U proseku između 10 i 20 različitih pogleda
	Česti pregledi odgovora (u proseku više od 3 puta po pitanju)
	Gledaju odgovore duže (u proseku najmanje 4 sekunde po pogledu)
	U proseku potroše manje od 30 sekundi po pitanju
8	Česti pregledi odgovora (u proseku više od 3 puta po pitanju)
	Česti pogledi na kod (u proseku više od 5 puta po pitanju)
	U proseku više od 20 različitih pogleda
	Neko vreme provedeno gledajući test pitanje
	U proseku potroše oko 40 sekundi na pitanje
9	Česti pogledi na kod (u proseku više od 5 puta po pitanju)
	Česti pogledi na pitanje (u proseku više od 5 puta po pitanju)
	U proseku više od 20 različitih pogleda
	Neko vreme provedeno gledajući test pitanje
	U proseku potroše oko 40 sekundi na pitanje
10	Čest pogled na dugme za povratak
	U proseku manje od 10 različitih pogleda
	U proseku potroše manje od 10 sekundi na pitanje

11	U proseku manje od 10 različitih pogleda
	U proseku potroše manje od 10 sekundi na pitanje
12	U proseku manje od 10 različitih pogleda
	Gledaju odgovore duže (u proseku najmanje 4 sekunde po pogledu)
	U proseku potroše manje od 20 sekundi na pitanje

Tabela 2. Tekstualni opis vrednosti atributa po klasterima

Da bi se smanjio ispis svih tekstualnih vrednosti klastera i ograničio ispis na kratku rečenicu izvršena je indukcija tipova paterna. Ovim su paterni učenika podeljeni na tipove, čiji kratak opis tipa predstavlja labelu klastera (klase) kojem pripada taj patern. Poboljšan je i ispis na konzoli, gde se umesto beznačajnog broja klastera, korisniku ispisuje kraći ili duži opis tipa paterna, tj. atributa klastera. Tipovi paterna su prikazani u tabeli ispod.

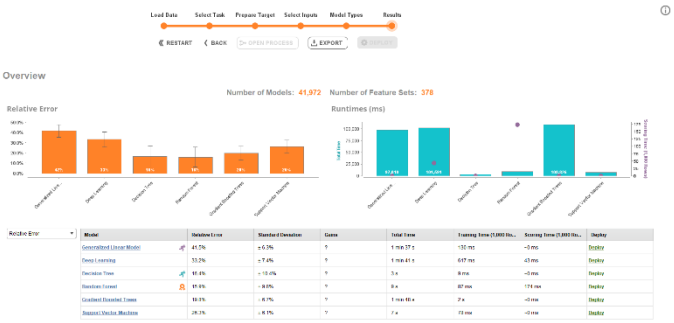
Klaster	Tip (paterna) učenika
1	Brz i fokusiran
2	Temeljan, gde su kod i odgovori od suštinske važnosti
3	Temeljan sa akcentom na kodu
4	Temeljan, gde su pitanje i odgovori od suštinske važnosti
5	Brz i temeljan, gde su šifra i odgovori od suštinske važnosti
6	Brz i fokusiran samo na pitanje i odgovore
7	Fokusiran na odgovorima
8	Površan, gde su kod i odgovori od suštinske važnosti
9	Površan, gde su kod i pitanja od suštinske važnosti
10	Površan i brz, sa pogledima na prethodna pitanja.~~Nije učio, pokušava da pronađe odgovore u prethodnim pitanja~~
11	Površan i brz~~Nije učio, nije ga briga~~
12	Površan, fokusiran samo na odgovore~~Nije učio, pokušava da pogodi koristeći odgovore~~

Tabela 3. Tip paterna učenika

4.2 Rezultati klasifikacionih algoritama

Pored klasterizacije paterna i identifikacije klastera, kreiran je i model za predikciju budućih paterna posmatranja. Ovaj model na ulaz prima patern gledanja jednog pitanja, a na izlazu daje predikciju klastera. Isprobano je više klasifikacionih modela, a najbolji se koristi za klasifikaciju u softverskoj biblioteci.

Na početku klasifikacije bilo je potrebno odabrati vrstu klasifikacionog modela. Za to je iskorišćen alat [10] koji omogućava automatizaciju kreiranja klasifikacionog modela. Testirano je 6 različitih modela sa automatskom optimizacijom parametara: *Generalized Linear Model*, *Deep Learning*, *Decision Tree*, *Random Forest*, *Gradient Boosted Trees* i *Support Vector Machine*. Rezultati automatskog testiranja (slika 9) pokazuju da *Random Forest* algoritam ima najmanju relativnu grešku (15,9%). Svrha automatskog testiranja jeste da ukaže na kvalitet skupa podataka, kao i da pomogne pri odabiru između klasičnih modela mašinskog učenja i neuronskih mreža.



Slika 9. Rezultati automatskog treniranja klasifikacionog modela

Rezultati sa slike 9 pokazuju superiornost klasičnih modela mašinskog učenja nad dubokim učenjem za ovaj skup podataka. Zbog ovoga, a za potrebe softverske biblioteke, istrenirana su 4 klasifikaciona modela bazirana na mašinskom učenju: *Logistic Regression*, *Random Forest Classifier*, *MLP Classifier* i *SVM*. Najbolju validacionu tačnost (73,91%) je imao *Logistic Regression*, gde je skup podataka podeljen na trening i test skup u razmeri 4:1. Ovo je za oko 10% manja tačnost nego sa automatskim treniranjem.

Vrednosti težina atributa dobijenih nakon treniranja modela, ukazuju da najveći uticaj na klasifikaciju imaju atributi: *Ukupan broj sekundi provedenih na regiji 'kod'* (w=0,429), *Frekvencija/broj pogleda regije 'sledece'* (w=0,409), *Ukupna dužina trajanja pitanja u sekundama* (w=0,375) i *Najčešće posmatrana regija pre regije 'kod'* (w=0,355).

5. ZAKLJUČAK

Edukacija je značajna grana razvoja društva i kao takva unapređenja u obrazovanju su imperativ. Tehnologija praćenja pokreta oka je u poslednjih par decenija uznapredovala dovoljno da su se pojavila komercijalno dostupni hardver i softver za praćenje pokreta oka. Primena ove tehnologije u obrazovanju bi mogla da pruži novi uvid u kognitivne sposobnosti učenika, a samim tim i donese poboljšanja u edukaciji. Klasifikovanje učenika u različite klase na osnovu načina polaganja testa, moglo bi da pomogne nastavnicima u pripremi nastave, testova i učenika za nova gradiva. Klasifikacija učenika na 12 klasa, u ovom radu, je primer jedne od mogućih primena ove tehnologije u obrazovanju.

Problemu se pristupilo prvo prikupljanjem podataka pokreta očiju tokom polaganja e-testova. Prikupljeni su pogledi 5 učenika kroz 115 pitanja koji su transformisani u veliki skup podataka. Ovaj skup je činio 115 *csv* fajlova, gde je svaki fajl imao podatke o sekvencijalnom pogledu na *ROI* zajedno sa vremenom pogleda na iste. Regije od interesa su za potrebe rada podeljene na: *pitanje*, *kod*, *odgovori*, *prethodno*, *sledece*, *prazno* i *pitanje-kod*. Ovaj skup podataka je iskorišćen za dalju obradu iz koje su dobijeni vektori atributa pogleda. Uloga atributa je da prenesu dovoljno informacija iz vremenski zavisnih podataka u podatke fiksne dužine.

Klasterizacija informacionih vektora vršena je uz pomoć *DP_GP_cluster* algoritma. Ovaj algoritam vođen *Gibbs*-ovim uzorkovanjem pokazao je postojanje 12 klastera, tj. paterna pogleda. Analizom prosečnih vrednosti atributa po klasterima

dobijeni su opisi klastera (tabela 2). Iz opštih opisa klastera izvučeni su zaključci o tipu paterna koji predstavlja labelu klastera (tabela 3).

Automatskim testiranjem izvršeno je treniranje i validacija 6 algoritama za klasifikaciju. Najbolji rezultat dao je *Random Forest* sa relativnom greškom od 15,9%. Za potrebe softverske biblioteke, istrenirana su 4 klasifikaciona algoritma od kojih je najbolju validacionu tačnost imao *Logistic Regression* algoritam (73,91%). Pokazalo se da je najuticajnija regija na klasifikaciju regija *'kod'* sa težinom atributa *w*=0,429.

Kreirana softverska biblioteka je napisana u *Python* programskom jeziku. Biblioteka se oslanja na istreniran *Logistic Regression* algoritam za klasifikaciju paterna pogleda. Biblioteka dozvoljava podešavanje 45 parametra, od kojih dva služe za podešavanje ulaznog/izlaznog fajla ili foldera, jedan služi za podešavanje naziva *ROI*, dok preostala dva kontrolišu količinu ispisa na konzolu iz koje se biblioteka pokreće. Drugi način pokretanja ove softverske biblioteke je iz neke druge *Python* skripte ili programa. Biblioteka dozvoljava definisanje proizvoljnih regija, pod uslovom da se mora ponoviti *pipeline* rada – prikupljanje podataka, definisanje *ROI* i dobijanje skupa podataka, klasterizacija skupa podataka i konačno treniranje klasifikacionog modela sa novim skupom podataka. U ovom radu akcenat nije bio na prikupljanju podataka, već na kreiranju softverske biblioteke za klasifikaciju paterna.

Potencijalni nedostaci rada su mali skup prikupljenih podataka nad većom grupom ispitanika. Sam kvalitet podataka je na zadovoljavajućem nivou, iako je bilo fluktuacija u tačnosti aparature za praćenje pogleda. Ovo je rešeno uz pomoć ~~test~~ softvera za testiranje sa velikim razmakom između regija od interesa.

Postoji nekoliko tačaka mogućeg unapređenja rada koje obuhvataju: nove varijacije klasterizacionih i klasifikacionih algoritama, drugačiji odabir atributa za formiranje informacionog vektora, pa čak i preskakanje informacionog vektora i korišćenje direktno prikupljenih podataka za treniranje klasterizacionih i klasifikacionih algoritama. Unapređenje rada bi mogla da predstavlja izrada celokupnog alata ili sistema alata koji će vršiti testiranje učenika, prikupljanje pogleda učenika i njihovo klasifikovanje u skoro realnom vremenu i prikaz rezultata nastavniku.

REFERENCE

[1] Wikipedia - Eye tracking: https://en.wikipedia.org/wiki/Eye_tracking [Online]

[2] West, Julia M., et al. "eyePatterns: software for identifying patterns and similarities across fixation sequences." Proceedings of the 2006 symposium on Eye tracking research & applications. ACM, 2006.

[3] Wang, Yiren, and Fei Tian. "Recurrent residual learning for sequence classification." Proceedings of the 2016 conference on empirical methods in natural language processing. 2016.

[4] Göbel, Fabian, and Henry Martin. "Unsupervised Clustering of Eye Tracking Data." Spatial Big Data and Machine Learning in GIScience, Workshop at GIScience 2018. 2018.

[5] Kiefer, Peter, Ioannis Giannopoulos, and Martin Raubal. "Using eye movements to recognize activities on cartographic maps." Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2013.

[6] Blascheck, Tanja, et al. "Visual comparison of eye movement patterns." Computer Graphics Forum. Vol. 36. No. 3. 2017.

[7] Gazepoint - Analysis Professional Eye-Tracking Software: https://www.gazept.com/product/gazepoint-analysis-professional-edition-software/ [Online]

[8] Nieto-Barajas, L.E. and Contreras-Cristán, A., 2014. A Bayesian nonparametric approach for time series clustering. Bayesian Analysis, 9(1), pp.147-170.

[9] McDowell, I.C., Manandhar, D., Vockley, C.M., Schmid, A.K., Reddy, T.E. and Engelhardt, B.E., 2018. Clustering gene expression time series data using an infinite Gaussian process mixture model. PLoS computational biology, 14(1), p.e1005896.

[10] RapidMiner alat - dokumentacija: https://docs.rapidminer.com/ [Online]