# Nadzirano učenje u strojnom provjerniku pravopisa

**Brala, Nikša**

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 1322

# NADZIRANO UČENJE U STROJNOM PROVJERNIKU PRAVOPISA

Nikša Brala

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 1322

# NADZIRANO UČENJE U STROJNOM PROVJERNIKU PRAVOPISA

Nikša Brala

Zagreb, lipanj 2024.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 4. ožujka 2024.

# ZAVRŠNI ZADATAK br. 1322

| | |
|---|---|
| Pristupnik: | **Nikša Brala (0036539221)** |
| Studij: | Elektrotehnika i informacijska tehnologija i Računarstvo |
| Modul: | Računarstvo |
| Mentor: | prof. dr. sc. Gordan Gledec |

Zadatak: **Nadzirano učenje u strojnom provjerniku pravopisa**

Opis zadatka:

Usluga strojne provjere pravopisa "ispravi.me - hrvatski akademski spelling checker" kontinuirano se unapređuje funkcionalnostima koje poboljšavaju iskustvo upotrebe. Usluga počiva na rječničkoj bazi i n-gramskom sustavu stvorenom na temelju tekstova prispjelih na obradu. Rječnička baza sastavljena je od općejezičnog i posebnojezičnog fonda, koje treba trajno održavati svježima i aktualnima. Cilj je završnog rada proučiti dosadašnji razvoj usluge strojne provjere pravopisa te unaprijediti postojeći sustav nadziranog učenja novih različnica iz tekstova koje na obradu šalju korisnici i nadograditi ga metodama umjetne inteligencije. Radu treba priložiti izvorni i izvršni kod razvijenog sustava te potrebnu dokumentaciju.

Rok za predaju rada: 14. lipnja 2024.

## Sadržaj

# Uvod

Strojni provjernik pravopisa Hascheck, danas poznatiji po svom web-sjedištu Ispravi.me, svake godine broji sve više posjeta i obrađenih tekstova. Provjernik pritom neprestano svim novim i nepoznatim, ali smislenim, riječima koje obradi širi i razvija svoj rječnik. Ulazak svake riječi u rječnik mora odobriti čovjek, a riječi koje mu dolaze na ručnu provjeru poželjno je što bolje *pročistiti* od nesuvislih i pogrešnih riječi tako da se ljudski rad svede na minimum. Ovaj će rad pokušati stvoriti programsko rješenje za prepoznavanje i izdvajanje glagolskih oblika iz korpusa prijavljenih pogrešaka. Programski će se izvesti i ispitatiklasični programerski pristup i pristup korištenjem metode nadziranog strojnog učenja, tj. stabla odluke.

U drugom će se poglavlju opisati složenosti postojećih klasifikacija glagola u hrvatskom jeziku i predstavit će se klasifikacija koja je prihvatljivije složenosti i čija je implementacija programski izvedivija.

Treće poglavlje opisuje programsko baratanje riječima koje se obrađuju i objašnjava pogramsko rješenje izvedeno klasičnim programerskim pristupom.

Četvrto će poglavlje dati iscrpni uvid u temelje i sve preduvjete za izradu stabla odluke. Objašnjeno je učenje stabla odluke, svi izazovi koji se najčešće pojavljuju usput i primjena stabla odluke na konkretan problem.

Peto će poglavlje detaljno prikazati rezultate primjene programskog rješenja na dva različita ulazna korpusa podataka uz opis varijabilnog ulaznog parametara koji je moguće jednostavno prilagoditi ulaznom skupu podataka.

# 1. Hrvatski akademski spelling checker - Hascheck

## 1.1. Općenito o strojnom provjerniku pravopisa

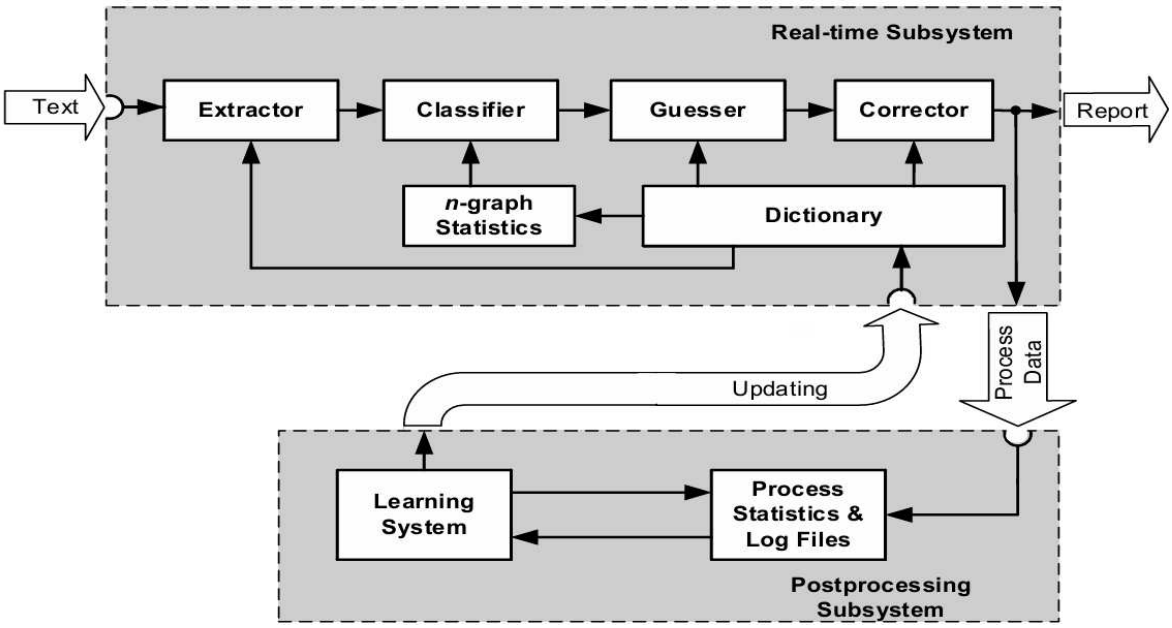
*Hrvatski akademski spelling checker* [1], tj. Hascheck [Hašek], jedna je od najstarijih internetskih usluga u Hrvatskoj koja od proljeća 1994. godine do danas u različitim oblicima djeluje kao javna i besplatna usluga pravopisne provjere teksta pisanog hrvatskim jezikom. Korisnički dio usluge od srpnja 2016. mijenja naziv (i adresu) u Ispravi.me, a Hašek ostaje pogonski mehanizam koji u pozadini obrađuje tekst [1]. Minimalističko i intuitivno sučelje zajedno s visokom razinom funkcionalnosti usluge osiguravaju visoku posjećenost web-sjedišta Ispravi.me čija je posjećenost u razdoblju od 2009. godine do 2023. godine s nekoliko tisuća korisničkih sesija mjesečno porasla na preko 200 000 korisničkih sesija mjesečno [1]. Korisnici usluzi pristupaju izravnim pristupom i organskim pretraživanjem, a među rezultatima organskog pretraživanja ključne riječi *provjera pravopisa* usluga Ispravi.me nalazi se na prvoj poziciji [2]. O upoznatosti korisnika s postojanjem i radom ove usluge svjedoči postotak od 64,59 % posjeta izravnim pristupom web-sjedištu, dok su korisnici koji su na ovo web-sjedište došli organskim pretraživanjem u većini slučajeva pretražili ključne riječi *ispravi me*, *hašek* i *hašek ispravi me* [3]. Korisnici usluzi pristupaju iz više od 160 zemalja [1], a najviše se posjeta ostvari iz Republike Hrvatske (83,56 %), BiH (8,1 %), Srbije (3,47 %), Slovenije (2,52 %) i Njemačke (1,65 %) [3].

## 1.2. Arhitektura sustava

Konvencionalni pravopisni provjernici, kakav je npr. Microsoftov pravopisni provjernik za hrvatski jezik, u pravilu rade sa samo tri komponente [4]:

1. rječnik;
2. ekstraktor, koji iz teksta za daljnju obradu izdvaja sve ono što u rječniku nije pronađeno;
3. korektor, koji nudi moguće ispravke za različnice koje nisu u rječniku.

Haschekova se arhitektura (Slika 1.1) bitno razlikuje od takve arhitekture. Hascheckov stvarnovremenski podsustav, koji reagira na tekstove zaprimljene na obradu, uz navedene tri komponente sadrži još klasifikator i morfologizator. Te dvije komponente daju dodatnu *inteligenciju* sustavu [4].




Slika 1.1 Arhitektura jezgrenog izvornog sustava Haschecka [4]

## 1.3. Učenje sustava

Neizostavan je element učinkovite, točne i brze pravopisne provjere bogata baza riječi (rječnik) s kojom je provjernik u neprekidnoj vezi i koja se stalno širi. Ekspertni sustav za učenje u pozadini temeljne obrade tekstova neprestano uči nove riječi [1], a učenje je nadgledano kako bi se osigurala i očuvala visoka čistoća rječničke baze. Veliki je nedostatak takvog pristupa njegovo održavanje koje iziskuje iznimno veliki ljudski angažman, ali osigurava očuvanje i poboljšavanje kvalitete usluge [1]. Stvarnovremenski podsustav (Slika 1.1) opslužuje postprocesni podsustav s brojnim podatcima potrebnim za učenje, a Hascheckov je sustav za učenje onaj dio koji ga po *inteligenciji* bitno izdvaja od drugih pravopisnih provjernika, bez obzira na to o kojemu je jeziku riječ [5].

Isprva su se različnice u rječnik dodavale kombiniranjem ručne provjere, konzultacija rječnika, pravopisa, gramatike i jezičnog priručnika uz dodatnu provjeru konteksta pojavljivanja riječi tražilicom *Google* i frekvencijom pojavljivanja [6]. Prosjek brzine takve obrade različnica[1] iznosio je malo više od 100 riječi po satu [6]. Takav je pristup najprecizniji i najpouzdaniji, ali i najsporiji, a ljudska su sredstva ograničena.

[1] Različnica - svaki različiti i jedinstveni oblik nastao od jedne ili više riječi u ili izvan određenog teksta [6]

Hašek je dosad obradio korpus od preko 16 milijardi pojavnica[2] i raspolaže s bazom koja broji više od 2 milijuna različnica, koje sve imaju potvrdu u tekstovima pisanim hrvatskim jezikom [1]. Ključna je činjenica da rječnički korpus sadrži i nastoji prikupiti sve dosad obrađene suvisle različnice, a ne samo osnovni oblik svih riječi (npr. infinitiv za glagole i nominativ za imenice), što uvelike povećava opseg rječnika.

## 1.4. Podatci

Ljudski su resursi ograničeni, a broj tekstova pristiglih na obradu sve je veći. Kao što je objašnjeno u [6], tu je enormnu količinu riječi moguće programski filtrirati i sortirati kako bi se maksimalno umanjio nepotrebni utrošak ljudske energije. Sadašnji sustav izdvajanja nepoznatih pojavnica svakoj nepoznatoj pojavnici dodijeli oznaku klase, zapiše broj njezinih pojavljivanja, još je jednom obradi pravopisnim provjernikom i zapiše ispravke koje pravopisni provjernik predloži pa je jednostavno provjeriti radi li se o nekoj uobičajenoj pogreški čiji je ispravni oblik među ponuđenim ispravcima. Među prijavljenim se greškama često nalaze oblici suvislih riječi koji još nisu popisani u rječniku, a u rječniku se nalazi/e neki drugi oblik/ci te riječi. Takve riječi sigurno pripadaju nekoj vrsti riječi hrvatskog jezika (npr. imenice, glagoli, pridjevi, zamjenice…), a svaka od tih vrsta riječi ima definirana i popisana pravila po kojima se riječi te vrste, uglavnom, ponašaju i mijenjaju kroz oblike.

U okviru istraživanja ovog projekta zadatak je bio iz dobivenog korpusa prijavljenih grešaka izdvojiti potencijalne glagolske oblike. Korpus na kojem se provodilo testiranje obuhvaća skup prikupljenih grešaka iz razdoblja od početka 2024. godine do sredine travnja iste godine. Korpus se ukupno sastoji od 3 334 463 prijavljenih pojavnica, a svaka se linija sastoji od sljedećih parametara:

1. Oznaka klase kojoj riječ pripada (-klasa-)
2. Riječ prijavljena kao pogreška
3. Broj pojavljivanja te riječi
4. Ponuđeni ispravci ili oznaka pripadnosti skupini *riječ*, *tuđica* ili *ime*

Oznaka klase predstavlja jednu od klasa iz tablice Tablica 1.1 [6] kojom je prikazana i zastupljenost pojedine klase u korpusu.

[2] Pojavnica - svaka riječ koja se pojavljuje u određenom tekstu [6]

Tablica 1.1 Klase različnica

| Oznaka klase | Opis klase | Zastupljenost klase |
|---|---|---|
| -kk- | kombinacija slova i brojeva | 6,35 % |
| -cc- | različnica ne dulja od 3 slova | 2,32 % |
| -xx- | ekstremno sumnjiva različnica | 20,15 % |
| -ll- | vrlo sumnjiva različnica | 17,59 % |
| -mm- | umjereno sumnjiva različnica | 22,67 % |
| -ss- | gotovo nesumnjiva različnica | 26,85 % |
| -gg- | neuobičajeno pisanje velikih i malih slova | 4,07 % |

Neke su riječi unaprijed označene kao potencijalne riječi, tuđice ili imena (oznake !RIJEČ!, !TUĐICA! i !IME!). Uz većinu riječi nakon oznake => stoje ponuđeni ispravci, ali nekim riječima pravopisni provjernik uopće nije pronašao potencijalne ispravke. Zastupljenosti tih obilježja prikazane su tablicom Tablica 1.2.

Tablica 1.2 Zastupljenosti obilježja

| Obilježje | Zastupljenost obilježja |
|---|---|
| Ponuđeni ispravci | 77,06 % |
| Dodijeljena oznaka !RIJEČ! | 2,57 % |
| Dodijeljena oznaka !TUĐICA! | 0,51 % |
| Dodijeljena oznaka !IME! | 1,86 % |
| Bez ponuđenih ispravaka | 18,00 % |

# 2. Prepoznavanje glagola

## 2.1. Glagoli u hrvatskom jeziku

Prvi izazov kod prepoznavanja pripadnosti riječi nekom glagolskom obliku predstavlja sama klasifikacija glagola u hrvatskom jeziku, tj. pronalazak kriterija i obilježja po kojima bi se dobivena riječ jasno mogla razlučiti kao potencijalni oblik nekog glagola. Hrvatska školska gramatika [7] glagole u hrvatskom jeziku definira kao promjenjive riječi kojima se izriče radnja, stanje i zbivanje. Oblici su glagola određeni gramatičkom kategorijom vremena (sadašnje – prezent; prošlo – aorist, imperfekt, perfekt, pluskvamperfekt; buduće – futur I., futur II.), lica (prvo, drugo, treće), broja (jednina, množina), stanja (aktiv, pasiv), načina (zapovjednost – izriče se imperativom, uvjetnost radnje – izriče se kondicionalom I. i kondicionalom II.), vida (svršeni, nesvršeni, dvovidni (oni koji su i svršeni i nesvršeni)), prijelaznosti (prijelazni, neprijelazni, povratni) [7].

Klasifikacija glagola u hrvatskom jeziku detaljno je obrađena u [8]. O kompleksnosti tvorbe i kategorizacije glagola u hrvatskom jeziku govori nam činjenica da ne postoji konsenzus o podjeli i klasifikaciji glagola na vrste i razrede u suvremenom hrvatskom jeziku. Šošić u [8] detaljno analizira tri različite podjele glagola na vrste i razrede u suvremenim hrvatskim gramatikama: podjela Mije Lončarića (1979.), podjela Stjepana Babića (1986.) te podjela Josipa Silića i Ive Pranjkovića (2005.). Lončarić glagole dijeli na šest vrsta koje, grananjem na razrede, ukupno čine sedamnaest skupina glagola. Babićeva se podjela glagola svodi na sedam vrsta koje s razredima čine šesnaest skupina. Razdioba Silić-Pranjković ukupno broji najviše podjela – čak šest vrsta koje, nakon podjele na razrede, ukupno daju dvadeset devet skupina, a objavljena je i u *Gramatici hrvatskog jezika* (2005.) [8].

Nakon detaljne analize tih triju podjela utvrđeno je da bi odabir klasifikacije Silić-Pranjković za izvedbu programskog rješenja bio optimalan jer glagole ukupno dijeli na najveći broj skupina, a to bi, u konačnici, osiguralo najprecizniju klasifikaciju i najdetaljnije definirana obilježja. Međutim, pokušaj programskog definiranja obilježja po kojima bi programsko rješenje prepoznalo glagolski oblik neke vrste ili razreda glagola pokazao se prekompleksnim i nedovoljno efikasnim. Spomenute su klasifikacije glagola pogodnije za samu tvorbu glagola, a ne i za prepoznavanje glagola, pa je umjesto jedne od njih odabrana jednostavnija, ali praktičnija, klasifikacija glagolskih oblika po glagolskim vremenima.

## 2.2. Podjela glagola po glagolskim oblicima i nastavcima

Klasifikacija glagola po glagolskim vremenima određuje nastavak, tj. sufiks ili dometak, riječi, tj. potencijalnog glagola, kao najznačajnije obilježje riječi po kojem se tu riječ može svrstati u potencijalne glagole. U hrvatskom bismo jeziku vjerojatno za svaki nastavak svakog glagolskog vremena mogli pronaći riječ koja nije glagol, a završava ekvivalentnim nastavkom (Tablica 2.1) pa je poseban naglasak nužno staviti na riječ *potencijalne* jer nastavak sam po sebi ne može biti dovoljan uvjet za označavanje pripadnosti riječi nekom od glagolskih oblika. Takva bi klasifikacija riječi kao glagolskih oblika garantirala izrazito nizak stupanj točnosti.

Tablica 2.1 Primjeri riječi (za svako glagolsko vrijeme) koje završavaju nekim glagolskim nastavkom, a ne pripadaju nekom od glagolskih oblika

| **Glagolski nastavak** | **Glagolsko vrijeme** | **Riječ** |
|---|---|---|
| -aš | prezent | šaš |
| -eh | aorist | grijeh |
| -ah | imperfekt | uzdah |
| -la | gl. pridjev radni | zdjela |
| -ta | gl. pridjev trpni | cesta |
| -vši | gl. prilog prošli | bivši [pridjev] |
| -ći | gl. prilog sadašnji | ptići |
| -j | imperativ | čaj |

Stupanj se točnosti te klasifikacije znatno može povećati dodavanjem dodatne provjere koja uključuje komunikaciju s rječnikom pravopisnog provjernika. Rječnik sadrži popis svih riječi koje su prolaskom kroz pravopisni provjernik označene kao suvisli oblik neke riječi. Rječnik tako sadrži i većinu oblika većine glagola u hrvatskom jeziku, pa se na temelju tih već popisanih oblika nekog glagola može, uz provjeru nastavaka, odrediti je li riječ koju program trenutno obrađuje zapravo suvisli oblik nekog već viđenog glagola. Takav je način jednostavnog uklanjanja glagolskog nastavka iz riječi, tj. iz potencijalnog glagola,

vjerojatno najjednostavniji način izvođenja kakve-takve glagolske osnove iz glagola. To je ujedno i najpraktičnije rješenje za programsku implementaciju jer pokriva većinu pravilnih glagola, a nezanemariv ostatak ovom metodom ostaje nepokriven zbog brojnih nepravilnih glagola, brojnih glasovnih promjena i sl.

Za što kvalitetniju provjeru ključno je precizno definirati glagolska vremena koja sadrže nastavke karakteristične za dotično glagolsko vrijeme. Nastavke unutar jednog glagolskog vremena zatim treba organizirati u skupine nastavaka tako da svaka skupina nastavaka odgovara istim oblicima pravilnih glagola na čiju će se osnovu stavljati ti nastavci. Nema smisla npr. glagolu *radim* nastavak promijeniti iz -im u -emo da dobijemo 1. l. mn. *rademo*, nego oblik početnog glagola za 1. l. mn. treba prebaciti u *radimo* – nastavci -im i -imo trebaju biti u istoj skupini nastavaka unutar skupine *prezent*. Također nema smisla mijenjati nastavke riječi koje završavaju na -vši i -avši jer su to jedini nastavci glagolskog priloga prošlog, pa se može pretpostaviti da ta riječ pripada tom glagolskom obliku.

Tako ostvarena podjela glagola ukupno broji osam glagolskih oblika, tj. osam skupina skupinâ glagolskih nastavaka [7], a to su:

1. Prezent (Tablica 2.2) [7]
2. Aorist (Tablica 2.3) [7]
3. Imperfekt (Tablica 2.4) [7]
4. Glagolski pridjev trpni (Tablica 2.5) [7]
5. Glagolski pridjev radni (Tablica 2.6) [7]
6. Glagolski prilog prošli (Tablica 2.7) [7]
7. Glagolski prilog sadašnji (Tablica 2.8) [7]
8. Imperativ (Tablica 2.9) [7]

Tablica 2.2 Skupine nastavaka za prezent

| **Prezent** | | | |
|---|---|---|---|
| Lice | 1. skupina | 2. skupina | 3. skupina |
| 1. l. jd. | am | im | em |

| | | | |
|---|---|---|---|
| 2. l. jd. | aš | iš | eš |
| 3. l. jd. | a | i | e |
| 1. l. mn. | amo | imo | emo |
| 2. l. mn. | ate | ite | ete |
| 3. l. mn. | aju | e | u |

Tablica 2.3 Skupine nastavaka za aorist

| **Aorist** | | | |
|---|---|---|---|
| Lice | 1. skupina | 2. skupina | 3. skupina |
| 1. l. jd. | oh | eh | h |
| 2. l. jd. | e | e | / |
| 3. l. jd. | e | e | / |
| 1. l. mn. | osmo | esmo | smo |
| 2. l. mn. | oste | este | ste |
| 3. l. mn. | oše | eše | še |

Tablica 2.4 Skupina nastavaka za imperfekt

| **Imperfekt** | |
|---|---|
| Lice | 1. skupina |
| 1. l. jd. | ah |
| 2. l. jd. | aše |
| 3. l. jd. | aše |

| | |
|---|---|
| 1. l. mn. | asmo |
| 2. l. mn. | aste |
| 3. l. mn. | ahu |

Tablica 2.5 Skupine nastavaka za glagolski pridjev trpni

| **Glagolski pridjev trpni** | | |
|---|---|---|
| Lice | 1. skupina | 2. skupina |
| 1. l. jd. | n | t |
| 2. l. jd. | na | ta |
| 3. l. jd. | no | to |
| 1. l. mn. | ni | ti |
| 2. l. mn. | ne | te |
| 3. l. mn. | na | ta |

Tablica 2.6 Nastavci za glagolski pridjev radni

| **Glagolski pridjev radni** | |
|---|---|
| Lice | 1. skupina |
| 1. l. jd. | o / ao |
| 2. l. jd. | la |
| 3. l. jd. | lo |
| 1. l. mn. | li |
| 2. l. mn. | le |

| | |
|---|---|
| 3. l. mn. | la |

Tablica 2.7 Nastavci za glagolski prilog prošli

| **Glagolski prilog prošli** |
|---|
| vši |

Tablica 2.8 Nastavci za glagolski prilog sadašnji

| **Glagolski prilog sadašnji** |
|---|
| ći |

Tablica 2.9 Skupine nastavaka za imperativ

| **Imperativ** | | | | |
|---|---|---|---|---|
| Lice | 1. skupina | 2. skupina | 3. skupina | 4. skupina |
| 2. l. jd. | / | j | i | ji |
| 1. l. mn | mo | jmo | imo | jimo |
| 2. l. mn. | te | jte | ite | jite |

Izostavljeno je nekoliko skupina i nastavaka jer su, u kontekstu programskog rješenja, sadržani u nekoj od susjednih skupina. Izostavljeni su:

- Skupina nastavaka prezenta (-jem, -ješ, -je, -jemo, -jete, -ju) jer je obuhvaćena trećom skupinom nastavaka prezenta
- Dvije skupine nastavaka imperfekta (-jah, -jaše, -jaše, -jasmo, -jaste, -jahu i -ijah, -ijaše, -ijaše, -ijasmo, -ijaste, -ijahu) jer su obuhvaćene jedinom uključenom skupinom nastavaka

- Dvije skupine nastavaka glagolskog pridjeva trpnog (-en, -ena, -eno, -eni, -ene, -ena i -jen, -jena, -jeno, -jeni, -jene, -jena) jer su obuhvaćeni prvom skupinom nastavaka
- Nastavak -avši glagolskog priloga prošlog jer je obuhvaćen nastavkom -vši.

Infinitiv je izuzet s popisa glagolskih vremena zato što većina glagola u infinitivu završava nastavkom -ti (iznimke koje završavaju na -ći program će svrstati pod glagolski prilog sadašnji) i nema drugih nastavaka kojima bi se mogla provjeriti pojavnost u rječniku. Nastavak -ti često se pojavljuje kod imenica (npr. *kosti*, *mati*, *fosfati*, *ugljikohidrati*, *nokti*), pa bi označavanje svih riječi koje završavaju na -ti kao infinitiv bilo neprecizno. Uklanjanjem nastavka -ti od infinitiva dobije se infinitivna osnova od koje se tvore aorist, imperfekt, glagolski pridjev radni, glagolski pridjev trpni i glagolski prilog prošli [7], pa se za provjeru infinitiva koriste sve skupine glagolskih nastavaka (ukupno njih osam) tih glagolskih vremena.

### 2.2.1. Nedostatci

Ova je klasifikacija glagola itekako praktična, ali je daleko od savršenog rješenja.

Kriteriji za programsko svrstavanje neke riječi u skupinu glagola ovom su klasifikacijom ograničeni na provjeru ostalih nastavaka iste skupine istog glagolskog vremena. Moguće je da je neki glagol u rječniku prisutan u relativno malom broju oblika unutar jedne skupine nastavaka, pa će program odbaciti glagolski oblik koji pripada toj skupini zbog nedovoljnog broja podudarnosti kod konzultacije s rječnikom. Šošić u [8] navodi kako je u većini slučajeva od pojedinog glagolskog oblika moguće izvesti njegovu infinitivnu ili prezentsku osnovu koje su polazišna točka za sva glagolska vremena. Izvođenje tih osnova iz glagolskih oblika zahtijeva programski znatno složeniju klasifikaciju glagola na vrste i razrede [8], ali bi implementacija takve klasifikacije omogućila konzultaciju s rječnikom u kojoj program, nakon izvođenja infinitivne ili prezentske osnove, provjerava oblike tog potencijalnog glagola i izvan skupine nastavaka kojoj taj potencijalni glagol pripada.

Dodatan problem predstavljaju imenice i pridjevi jer imaju velik broj izvedenih oblika koji završavaju nastavcima neke skupine glagolskih nastavaka, i to uglavnom iz skupina nastavaka glagolskog pridjeva trpnog. Tako će program, primjerice, riječ *stopedeset* označiti kao glagolski oblik čak i ako tražimo tri podudarnosti u rječniku jer rječnik sadrži oblike *stopedeseti*, *stopedeseta* i *stopedesete*, a svi su ti nastavci unutar iste skupine nastavaka za glagolski pridjev trpni. Ako tražimo manji broj podudarnosti u rječniku, npr. dvije, tada i

ostala glagolska vremena predstavljaju problem jer program, primjerice, riječ *trokutasti* svrstava među glagole zato što u rječniku pronađe oblike *trokutastim* i *trokutaste* (čiji su nastavci u istoj skupini nastavaka prezenta kao nastavak i).

# 3. Programsko rješenje

## 3.1. Odabir programskog jezika i struktura podataka

Za što uspješniju i jednostavniju manipulaciju riječima (objekti tipa *string*) ključno je bilo odabrati programski jezik koji je dovoljno *visoke razine* kako bi se, radi jednostavnosti i jasnoće, izbjeglo korištenje regularnih izraza (*regex*). Iz tog je razloga programsko rješenje izvedeno u programskom jeziku Java, verzije 21.

Uz velik broj rastavljanja i sastavljanja sadržaja objekata razreda *String*, veliku prepreku razvoju brzog i učinkovitog programskog rješenja čini i rukovanje s izrazito velikim tekstualnim datotekama. Java u tom pogledu kombiniranom uporabom razreda *FileReader* i razreda *BufferedReader* nudi brzo i učinkovito rješenje za čitanje velikih tekstualnih datoteka jer obavlja relativno mali broj U/I operacija i ugrađenom metodom *readLine* osigurava vrlo jednostavno čitanje redak po redak, a svi su ulazni podatci tako i zapisani. Tekstualne se datoteke s riječima koje idu na obradu čitaju cijele – radi se o jednostavnoj iteraciji po sadržaju cijele datoteke – ali jako česta provjeravanja pojavnosti riječi u tekstualnoj datoteci rječnika zahtijeva bolje rješenje od običnog pretraživanja cijele tekstualne datoteke. Obično pretraživanje jamči vremensku složenost O(n) gdje je n broj redaka u rječniku, a u ovom slučaju n iznosi preko 3 milijuna. Za rješenje se ovog problema u Javi tekstualna datoteka rječnika jednostavno *hashira* pohranjivanjem svih riječi (redaka) rječnika u strukturu podataka *HashSet*. Pretraživanje postojanja neke riječi unutar tako organiziranog rječnika jamči složenost O(1) čime je pretraga rječnika optimizirana.

Glagolski su oblici prikazani kao objekti razreda *GlagolskiOblik*. Razred *GlagolskiOblik* sadrži atribute *naziv* (predstavlja naziv glagolskog oblika) i listu *nastavci* koja je tipa *ArrayList<ArrayList<String>>* i sadrži onoliko lista tipa *ArrayList<String>* koliko taj glagolski oblik ima skupina nastavaka. Program će u obradi riječi često iterirati po tim nastavcima, tj. listama, pa se razred *ArrayList* nameće kao najpraktičniji odabir jer osigurava jednostavnu iteraciju, dok za jednostavnu iteraciju po objektu razreda tipa npr. *HashSet* taj objekt prvo treba pretvoriti u listu. Razred *GlagolskiOblik* sadrži i konstruktor i funkcije za postavljanje (metode *set*) i dohvat (metode *get*) spomenutih atributa.

Na početku se izvođenja programa, a nakon učitavanja ulaznih datoteka, inicijaliziraju svi glagolski oblici pozivom funkcije *inicijalizacijaGlOblika* koja za sve

prethodno navedene glagolske oblike stvori novi objekt razreda *GlagolskiOblik* i doda mu sve pripadajuće skupine nastavaka. Zatim se svi popisani glagolski oblici dodaju u strukturu *glagoli* koja je objekt tipa *HashMap<GlagolskiOblik, HashSet<String[]>>*. Ta struktura podataka funkcionira po principu ključ-vrijednost gdje su ključevi svi glagolski oblici, a njihove su vrijednosti sve riječi izdvojene kao potencijalni pripadnici tom glagolskom obliku. Sve su obrađene riječi (i one koje program obradom svrsta pod potencijalne glagolske oblike i one koje program obradom odbaci) razvrstane u strukture podataka tipa *HashSet<String[]>* jer razred *HashSet* osigurava pohranu jedinstvenih riječi, pa ne dolazi do redundancije, i dodavanje elemenata s optimalnom vremenskom složenosti O(1). Svrha i upotreba strukture *String[]* opisane su sljedećem potpoglavlju.

## 3.2. Priprema riječi za obradu

Podatci koji dolaze na obradu i format ulaznih redaka opisani su u potpoglavlju 1.4 Podatci, a primjeri redaka koji dođu na analizu dani su tablicom Tablica 3.1.

Tablica 3.1 Primjeri redaka za analizu

| **Redak za analizu** |
|---|
| -ss- 'tito 2 |
| -k- 16h 2 |
| -ll- znantsvenog 1 => znanstvenog? |
| -ss- Amigovim 3 => !IME!? |
| -mm- stedjeti 1 => štedjeti? sjedjeti? stidjeti? studjeti? |

Svaki redak treba rastaviti na dijelove tako da iz retka izdvojimo riječ i ponuđene potencijalne ispravke (ako postoje). Program liniju prvo dijeli na dvije strane po separatoru " => ". Prvi će član novodobivene liste sigurno sadržavati oznaku klase i riječ, neovisno o tome postoji li uopće separator " => ", pa taj dio još dijelimo po separatoru " " da bismo s pozicije s indeksom 0 dobili oznaku klase, a s pozicije s indeksom 1 riječ. U riječi odmah, radi jednostavnosti, ugrađenom metodom *toLowerCase* sva velika slova prebacujemo u odgovarajuća mala slova. Retkom se nadalje upravlja u obliku strukture *String[]* naziva

*parRijecDioDesnoOdStrelice* koja predstavlja jednodimenzionalnu listu podataka tipa *String* i uvijek sadrži jedina dva elementa koja su relevantna za daljnju analizu: riječ i dio retka koji dolazi nakon strelice => (to su ili ponuđeni ispravci ili neka od triju ranije spomenutih oznaka). U slučajevima u kojima redak nema strelicu, element liste *parRijecDioDesnoOdStrelice* na poziciji s indeksom 1 je prazni objekt razreda *String*, tj. "".

Program za svaku riječ koja mu dođe na obradu prvo, provjerom podudarnosti riječi s odgovarajućim regularnim izrazom, odbacuje riječi koje sadrže neke znakove koji nisu dio abecede, npr. *!*, #, @ i sl.

Program zatim eliminira riječi za koje pravopisni provjernik nije ponudio ispravke (za te slučajeve se može pretpostaviti da se ne radi o suvisloj riječi) i riječi koje pripadaju nekoj od klasa koje nisu kandidati za daljnju provjeru, a to su klase: -kk-, -cc-, -xx- i -gg-; čija su tumačenja i zastupljenosti prikazane tablicom Tablica 1.1.

Jasno je da riječi koje pripadaju klasi -kk- sigurno nisu neki od glagolskih oblika jer nema smisla u daljnje razmatranje uzimati riječi koje sadrže brojeve.

Riječi klase -cc- se također mogu odbaciti jer je mali postotak glagola duljine do 3 slova.

U korpusu prijavljenih pogrešaka klasa -xx- ukupno broji 671 845 riječi, a klasa -gg- 135 570 riječi. Za analizu tih dviju klasa nasumično je odabran uzorak od po 500 riječi iz svake klase, a među njima je u klasi -xx- pronađeno 7 riječi koje su pogrešno napisani glagolski oblici. Riječi iz uzorka te klase uglavnom su bili zatipci i nesuvisle riječi. U uzorku iz klase -gg- nije pronađen nijedan glagolski oblik, a riječi su uglavnom deklinacije kratica i zatipci.

Navedene klase u korpusu prijavljenih grešaka čine veliki postotak od ukupnih riječi (čak 32,89 %), a njihova daljnja analiza nema smisla, pa je isplativije rješenje u startu ih izuzeti iz daljnje provjere.

Za riječi koje je pravopisni provjernik označio kao imena pretpostavljamo da je bio u pravu i izuzimamo ih iz daljne analize.

Odbačeni se retci, radi detaljne obrade i jednostavnog uvida u odbačene retke, ne zanemaruju u potpunosti, nego ih program pohranjuje u strukturu *smece* klase *HashSet* u obliku liste *parRijecDioDesnoOdStrelice*. Retci koji su prošli dosadašnje provjere i još nisu odbačeni idu na daljnju detaljnu analizu, radi preglednosti i sistematičnosti, organiziranu u sklopu metode *analizaRijeci*.

## 3.3. Analiza riječi

Program će riječ vjerojatno trebati analizirati po nastavcima ako je došla do ove razine, ali prije toga preostaje opcija provjeriti ponuđene ispravke i, iz riječi u obradi, pokušati dobiti neku od ponuđenih riječi. Potprogram *ijeJe* redom zamijeni svaki *ije* s *je*, *je* s *ije* te *e* s *ije* i *je* (zbog česte pojave riječi iz ekavskog govora) i takav oblik riječi usporedi sa svakom ponuđenom riječi. Dodatno se obavi i zamjena svih slova koja imaju svoju inačicu s *kvačicom* u hrvatskom jeziku. Potprogram *kvacice* redom mijenja *dj* u *đ* i *dž*, *dz* u *đ* i *dž*, *c* u *č* i *ć*, *č* u *ć*, *ć* u *č*, *s* u *š* te *z* u *ž*. Nisu pokrivene sve kombinacije jer iznimno rijetko npr. *ž* bude napisan umjesto *z*, a obrnuti je slučaj relativno čest. Također je nepotrebno provoditi dvostruke zamjene jer pravopisni provjernik ne detektira dvostruku grešku, tj. kao ponudu za unesenu riječ kao što je npr. *cekic* pravopisni provjernik ponudi *cekin*, a ne *čekić*, pa zamjene ovog tipa smisla ima provoditi samo jednu po jednu.

Ako je primjenom neke od zamjena dobivena neka od ponuđenih riječi, možemo pretpostaviti da će se u većini slučajeva raditi o pogreški takvog tipa i program odbacuje riječ dodavanjem strukture *parRijecDioDesnoOdStrelice* u strukturu *ijeJeKvaciceSmece* tipa *HashSet* kako bi, u fazi testiranja, bilo jednostavno odrediti uspješnost i korisnost ovih provjera.

Nakon svih prethodnih provjera na red dolazi provjera popisanih nastavaka glagolskih oblika. Potprogram *jeLiGlagol* za svaki glagolski oblik i za svaku skupinu nastavaka unutar tog glagolskog oblika uzima svaki nastavak i, ako nastavak nije prazni objekt razreda *String*, tj. “”, ugrađenom metodom *endsWith* provjerava završava li riječ koja je u obradi s dotičnim nastavkom. Ako te dvije provjere (nastavak nije “” i riječ završava tim nastavkom) budu zadovoljene na red dolazi potprogram *provjeriOstaleNastavke* koji iz riječi uklanja nastavak čime taj dio riječi bez nastavka postaje *osnova* na koju potprogram onda dodaje ostale nastavke iz te skupine nastavaka. Taj potprogram za svaki dobiveni oblik *osnova* + *nastavak_iz_skupine* provjerava nalazi li se on u rječniku, tj. ispituje pojavnost te konkretne riječi u rječniku.

Program, primjerice, u riječi *pričam* uočava glagolski nastavak -am prve skupine nastavaka prezenta. Program zatim uklanjanjem uočenog nastavka -am od te riječi tvori osnovu *prič*, na koju onda dodaje ostale nastavke iz te skupine (-aš, -a, -amo, -ate, -aju) kako bi dobio oblike *pričaš*, *priča, pričamo*, *pričate* i *pričaju*.

U slučaju da za neki izvedeni oblik dođe do podudarnosti s nekom riječi u rječniku, mogli bismo pretpostaviti da se radi o nekom glagolskom obliku jer oblik te riječi s nekim drugim nastavkom iz iste skupine nastavaka postoji u rječniku. Problem je što bi npr. riječ *kući* tada završila među glagolima jer završava prezentskim nastavkom -i, a zamjenom tog nastavka nastavkom -e, dobijemo riječ *kuće* koja je suvisli oblik imenice *kuća* i nalazi se u rječniku. Naravno da je tako česta imenica već zapisana u rječniku, ali neka bi se nezapisana riječ koja nije glagolski oblik tako mogla provući među glagole. Rješenje tog problema predloženo je u poglavlju 5.1.

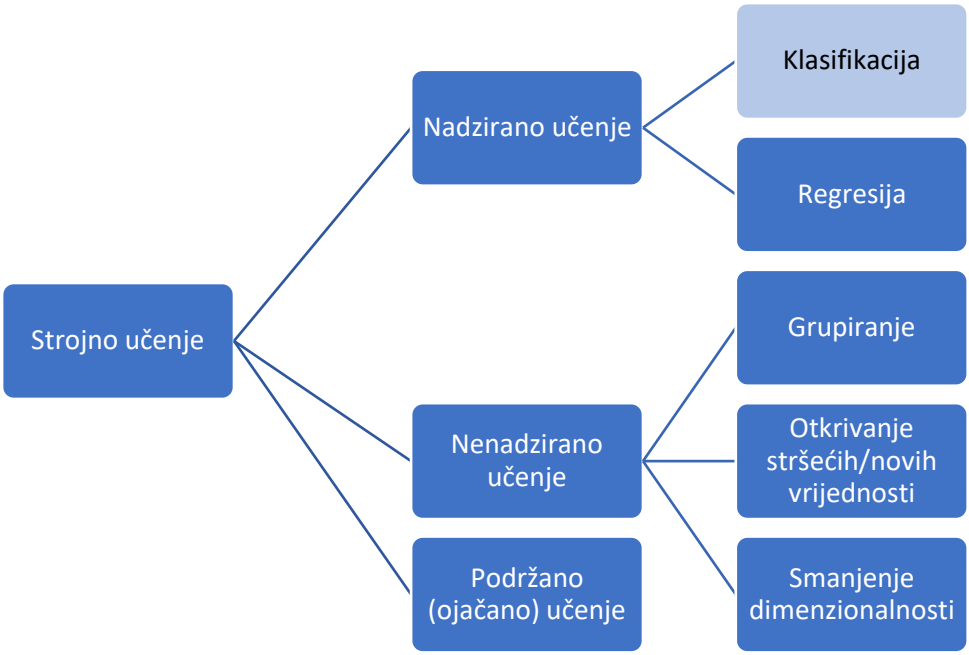
Ako nakon provjere svih nastavaka nije pronađena podudarnost u bazi, dodatno se provjeravaju 3. skupina nastavaka aorista i 1. skupina nastavaka imperativa koje sadrže nastavak “”. Ovaj je nastavak nužno provjeriti zasebno (i to na kraju) jer metoda *endsWith* za predani parametar “” uvijek vraća *true*, pa je nužno prvo pregledati sve ostale nastavke, a ove dvije skupine tek na kraju.

# 4. Programsko rješenje sa stablom odluke

Govornik hrvatskog jezika lako bi, na temelju iskustva, prepoznao je li neka riječ glagolski oblik ili nije. Kako bi računalo za istu riječ programom moglo obaviti isti taj zadatak potrebno je definirati velik broj uvjeta i pravila koje riječ mora zadovoljiti da bi je program svrstao pod (potencijalne) glagole. Ti su uvjeti i pravila implementirani kroz prethodna poglavlja i njihova je točnost ograničena, a ne obuhvaćaju velik broj glagola koji se ne ponašaju po definiranim pravilima (tzv. nepravilni glagoli).

Druga je opcija ispitati je li korisniji i učinkovitiji pristup istrenirati računalo, tj. program, da, kao i govornik hrvatskog jezika, prepozna (potencijalni) glagolski oblik kada ga vidi. Kretanje u tom smjeru zahtijeva traženje rješenja unutar područja umjetne inteligencije čije su primjene u svijetu računarstva sve šire i sve češće.

Strojno učenje grana je umjetne inteligencije koja se temelji na učenju nad skupom podataka i bavi se prepoznavanjem uzoraka i donošenjem odluka s ciljem što manje potrebe za intervencijom čovjeka [9]. Podjela strojnog učenja prikazana je dijagramom Dijagram 4.1, a ostatak će se ovog poglavlja baviti nadziranim strojnim učenjem, tj. klasifikacijom.

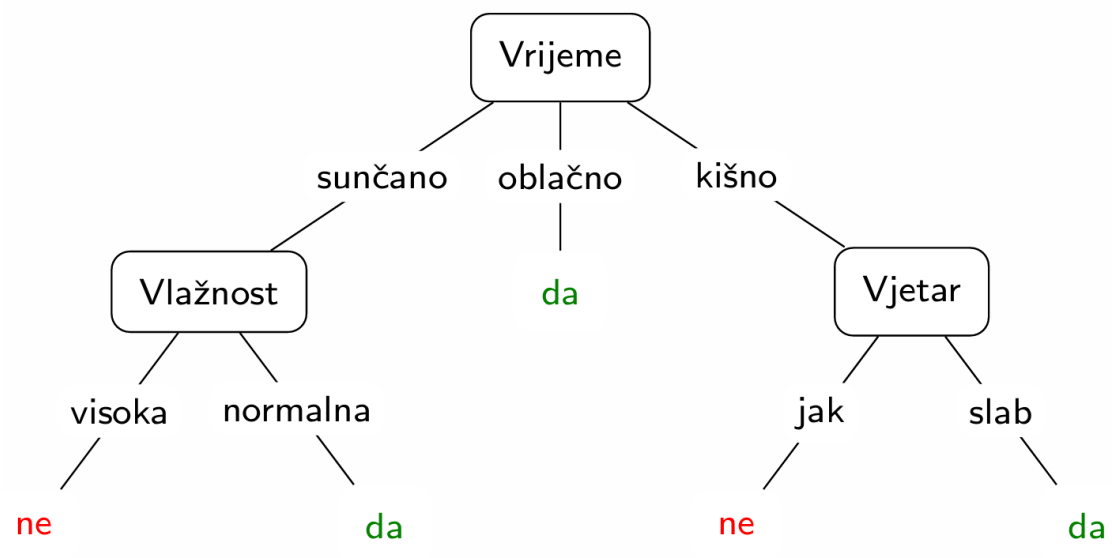



Dijagram 4.1 Podjela strojnog učenja [9]

## 4.1. Nadzirano strojno učenje – stabla odluke

Nadzirano strojno učenje [9] (engl. *supervised learning*) zasniva se na podatcima koji su u obliku *(ulaz, izlaz) = (**x**, y)*, a zadatak je pronaći preslikavanje $y = f(\mathbf{x})$. Nadzirani model na označenim primjerima prvo treba trenirati (naučiti), nakon čega se može koristiti za predikciju na njemu do tada neviđenim primjerima. Vrste predikcije su klasifikacija ($y$ je diskretna/nebrojčana vrijednost) i regresija ($y$ je kontinuirana/brojčana vrijednost) [9]. S obzirom na to da je cilj svaku riječ izdvojiti u jedno od kategorija *glagolski oblik* i *nije glagolski oblik*, klasifikacija je jasan odabir vrste predikcije.

Jedan od načina implementacije klasifikacije jest klasifikacija stablom odluke. Stablo odluke [9] grade unutarnji čvorovi koji predstavljaju značajke (atribute), grane ispod svakog čvora koje odgovaraju vrijednostima te značajke, a na završnim se čvorovima stabla odluke nalaze listovi koji odgovaraju klasifikacijskim odlukama (oznakama klase). Primjer je takvog stabla prikazan slikom Slika 4.1. Stablo odluke svaki primjer koji mu dođe na ulaz klasificira slijednim ispitivanjem vrijednosti značajki, počevši od korijena (vrha) stabla prema dnu (listovima), a kada dođe do lista primjer se klasificira oznakom lista. Primjer prikazan slikom Slika 4.1 prikazuje u kojim je kombinacijama vremenskih uvjeta dan pogodan za igranje odbojke. Svaka staza od korijena od korijena do lista predstavlja jedno pravila u obliku konjunkcije uvjeta nad vrijednostima značajki. Tako na primjer konjunkcija uvjeta (Vrijeme = kišno ∧ Vjetar = slab) daju oznaku klase *da* (dan je pogodan za igranje odbojke). Neki od algoritama za izgradnju stabla odlučivanja su ID3, C4.5 i CART [9], a ostatak će se ovog poglavlja baviti primjenom algoritma ID3 koji je pogodan upravo za klasifikaciju.




Slika 4.1 Primjer stabla odluke [9]

## 4.2. Algoritam ID3

Ross Quinlan je 1986. razvio algoritam ID3 (*Iterative Dichotomiser 3*) koji rekurzivno gradi stablo odlučivanja iterativnim odabirom atributa na temelju kojeg je, u tom trenutku razvoja stabla, najpovoljnije dalje podijeliti podatke [9]. Odabir tog atributa vrši se na temelju vrijednosti informacijske dobiti za čiji se izračun koristi vrijednost entropije tog atributa.

### 4.2.1. Entropija

Po [9], entropija (*fiz.* mjera nereda sustava; *inform.* mjera za gubitak informacije) u kontekstu stabla odluke iskazuje mjeru u kojoj su vrijednosti neke značajke raspršene po oznakama klase. Uzmemo li da skup D sadrži označene primjere iz K klasa, onda je izraz za entropiju skupa D prikazan sljedećom formulom:

$$E(D) = -\sum_{i=1}^{K} P(y = i) \log_2 P(y = i)$$

gdje je $P(y = i)$ vjerojatnost klase $y = i$. Dogovorno, $0 \log_2 0 = 0$. Savršeni je slučaj podjele onaj u kojem klasa na kraju uvijek bude jedna te ista i tada vrijedi E = 0 (srednji redak na slici Slika 4.2). Najgora je podjela ona za koju vrijedi E = 1, jer ulazni primjer tada u jednakom broju slučajeva završi u svakoj od klasa. To bi bio slučaj u kojem je u trećem i četvrtom stupcu sa slike Slika 4.2 jednaka vrijednost. U programskom je rješenju izračun

entropije za pojedinu vrijednosti svake značajke ostvaren jednostavnom metodom *Entropy* koja za predani skup podataka i određenu značajku računa i vraća vrijednost entropije [9].

| Značajka | Vrijednost | # da | # ne | $E(D)$ |
|---|---|---|---|---|
| **Vrijeme** | sunčano | 2 | 3 | 0.971 |
| | oblačno | 4 | 0 | 0 |
| | kišno | 3 | 2 | 0.971 |

Slika 4.2 Prikaz vrijednosti entropije na primjeru značajke *Vrijeme*

### 4.2.2. Informacijska dobit

Prema [9], svakom je koraku izrade stabla odluke cilj odabrati značajku koja što više smanjuje entropiju, a u tom odabiru ključnu ulogu igra parametar informacijske dobiti. Kriterij informacijske dobiti (engl. *information gain*) mjeri očekivano smanjenje entropije skupa primjera uslijed podjele primjera po vrijednostima neke značajke. Informacijska dobit (IG) značajke *x* na skupu primjera *D* prikazan je sljedećom formulom:

$$IG(D,x) = E(D) - \sum_{v \in V(x)} \frac{|D_{x=v}|}{|D|} E(D_{x=v})$$

gdje je *E(D)* entropija skupa primjera *D*, $D_P$ je podskup primjera koji zadovoljavaju uvjet *P*, a *V(x)* je skup mogućih vrijednosti značajke *x*. U programskom je rješenju informacijska dobit prikazana jednostavnom metodom *InformationGain* koja za predani skup podataka, određenu značajku i uz višestruke pozive funkcije *Entropy* opisane u prethodnom potpoglavlju računa i vraća vrijednost informacijske dobiti [9].

### 4.2.3. Programsko rješenje

Stablo je odlučivanja u programskom rješenju organizirano i prikazano klasom *Node* koju opisuju atributi *value* (vrijednost čvora), *label* (oznaka klase čvora) i *subtrees* (skup podataka tipa *HashMap<String, Node>* koji povezuje nazive čvorova na sljedećoj razini). Cijelo se stablo tako može prikazati jednim objektom tipa *Node* jer taj objekt sadrži ostatak stabla u svom atributu *subtrees*.

**function** id3($D, D^{parent}, X, y$)

  **if** $D = \emptyset$ **then**

    $v \leftarrow \text{argmax}_v |D^{parent}_{y=v}|$

```
    return Node(v)
v ← argmax_v |D_{y=v}|
if X = ∅ or D = D_{y=v} then
    return Node(v)
x ← argmax_{x∈X} IG(D, x)
subtrees ← ∅
for v ∈ V(x):
    t ← id3(D_{x=v}, D, X\{x}, y)
    subtrees ← append(subtrees, (v, t))
return Node(x, subtrees)
```

Pseudokod 4.1 Algoritam ID3 (utjelovljen metodom *fit*) [9]

Dvije glavne metode na kojima počiva programsko rješenje su metoda *fit* koja kao argument prima skup podataka i provodi učenje modela te metoda *predict* koja kao argument prima skup podataka i predviđa ciljnu varijablu na temelju već naučenog modela. Metoda *fit* predstavlja srž algoritma ID3 jer je učenje modela algoritmom ID3 implementirano upravo metodom *fit* koja je idejno temeljena na pseudokodu Pseudokod 4.1 ($X$ predstavlja skup svih značajki, $y$ je oznaka klase, a $D$ je skup označenih primjera). Dodatno su implementirane pomoćne metode *findDxv*, *findDyv* i *DEquals* za lakše upravljanje skupom podataka $D$, a implementirana je i pomoćna metoda *argmax* za pronalazak najčešćih oznaka primjera u čvorovima i za pronalazak najdiskriminativnije značajke.

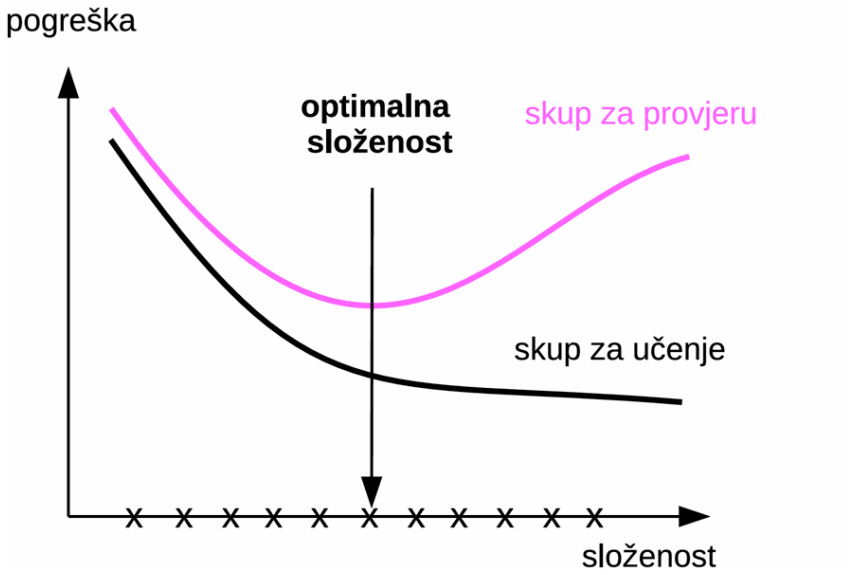
## 4.3. Učenje stabla (*fitting*)

Tijekom treniranja (učenja) modela stabla objekt koji programu dođe na ulaz sadrži niz značajki koje su predstavljene vektorom $\boldsymbol{x}$, a svaka značajka predstavlja jedan aspekt ulaznog objekta (npr. radna memorija, pohrana, tip hlađenja…). Program, uz vektor $\boldsymbol{x}$, na ulaz prima i oznaku $y$ (engl. *label*) koja označava kojoj klasi taj objekt pripada. Ulazni se parametri mogu predstaviti kao par ($\mathbf{x}$, $y$). Nakon učenja na odgovarajućem broju ulaznih slučajeva, isti se model koristi za predikciju. Tijekom predikcije, model na ulazu dobiva isključivo primjere $\mathbf{x}$, a na izlazu daje oznaku $y$. Glavni je cilj treniranog modela da dobro generalizira. Poželjno je da je razvijeni model složen jer to znači da se može vrlo dobro prilagoditi podatcima i za te podatke davati (skoro) savršene predikcije.

### 4.3.1. Prenaučenost (engl. *overfitting*)

Za uspješno je generaliziranje modela nužno izbjeći presloženost [9], tj. prenaučenost (engl. *overfitting*) modela. Pojavljuje se u slučajevima kada je model treniran na prevelikom broju primjera pa se prilagodio isključivo tim primjerima. Ključno je osigurati i da se model ne uči na skupu podataka koji će se koristiti za evaluaciju. Učenje je modela na svim dostupnim podatcima potpuno pogrešan pristup jer će takav model za ulaze iz tog skupa savršeno generalizirati, ali će za neviđene slučajeve dati pogrešne predikcije, što ga čini beskorisnim. Do prenaučenosti dolazi i ako primjeri za učenje sadrže *šum*, tj. sadrže ulaze s greškom (pogrešne vrijednosti značajki ili pogrešna oznaka klase).

Metoda unakrsne provjere (engl. *cross-validation*) osigurava zaštitu od prenaučenosti jer ispituje kako model radi na neviđenim podatcima. Neviđeni primjeri, naravno, nisu dostupni i zato je nužno da dio poznatih primjera u ovoj provjeri glumi te neviđene primjere. Cijeli je skup podataka koji je na raspolaganju potrebno razdvojiti na skup podataka za učenje i skup podataka za testiranje. Uobičajeno 75 % ulaznog skupa podataka otpada na skup za učenje, a preostalih 25 % na skup za testiranje. Postoji i pristup s dodatnim skupom podataka za provjeru (engl. *validation set*), ali je takav pristup vjerojatno presložen za projekt ovog obujma. Model prvo provodi učenje na skupu za učenje (75 % podataka), a naučeni model zatim provodi testiranje na skupu za testiranje (25 % podataka) s čijim se podatcima susreće po prvi puta. Nakon testiranja parametar *točnost* (engl. *accuracy*) govori o točnosti, tj. sposobnosti generalizacije modela [9]. Optimalna složenost kao savršeni omjer između podnaučenosti i prenaučenosti prikazana je dijagramom Dijagram 4.2.




Dijagram 4.2 Dijagram optimalne složenosti [9]

## 4.4. Priprema riječi za obradu

Učenje stabla odluke provodit će se na kombinaciji riječi iz rječnika i pogrešnih riječi jer je stablo odluke ključno upoznati sa slučajevima u kojima je predana riječ jedan od glagolskih oblika, sa slučajevima u kojima predana riječ nije jedan od glagolskih oblika i sa slučajevima u kojima riječ uopće nije suvisla riječ, nego pogreška. Kao što je opisano u prethodnom potpoglavlju, za ulazne je objekte nužno definirati pripadajuće značajke.

Stvarni će ulazni objekti ovog sustava biti u formatu s 4 parametra opisanom u potpoglavlju 1.4 Podatci pa je kao jednu od značajki razumno odabrati parametar *–klasa--*. Sve će riječi iz rječnika biti označene oznakom klase *-ss-* (najmanja razina sumnjivosti), a sve će preostale klase biti zastupljene primjerima riječi iz skupa pogrešaka. Druga značajka može biti sama riječ, a treća značajka može biti 4. parametar iz ulaznog formata definiranog u potpoglavlju 1.4.

Četvrtom bi značajkom stablo odluke na neki način trebalo provjeriti pripada li ta riječ nekom od glagolskih oblika. Jedno od rješenja može biti izdvajanje nastavka riječi i provjera njegove pojavnosti među glagolskim nastavcima, ali je takva provjera upitne isplativosti jer su glagolski nastavci kao npr. -a, -e, -la, -as i sl. prisutni u velikom broju riječi koje ne pripadaju nekom od glagolskih oblika. Stablu bi odlučivanja kroz učenje tako trebalo biti predstavljeno da je npr. riječ *rekla* (nastavak -a) glagolski oblik, a npr. riječ *kuća* nije glagolski oblik iako također završava na -a. Stablo bi te dvije riječi s istim nastavkom moglo

razlikovati isključivo prema osnovi što znači da bi stablo trebalo poznavati osnove svih glagolskih oblika iz rječnika kako bi moglo razlučiti da je *rekla* glagolski oblik, a *kuća* nije, što je upitne izvedivosti i neupitno je neefikasno. Isti bi se problem pojavio i kod riječi kao što su *munje* (glagolski nastavak -e), *škola* (glagolski nastavak -la), *pas* (glagolski nastavak -as) i sl.

Za razlučivanje je glagolskih oblika od ostalih riječi neminovna konzultacija s rječnikom kako bi program ustanovio pripada li ta riječ nekom glagolskom obliku nekog već postojećeg glagolskog oblika u rječniku. Tako bi stablo odlučivanja, kao i programsko rješenje bez stabla odlučivanja opisano u 3. poglavljuProgramsko rješenje, moralo provesti provjeru metodom *jeLiGlagol* (čija je funkcionalnost opisana u potpoglavlju 3.3 Analiza riječi) i rezultat provjere zapisati kao jednu od značajki. S obzirom na to da važnost značajke *riječ* za izgradnju stabla sada predstavlja i opisuje značajka o zadovoljenosti minimalne pojavnosti, nadalje nema smisla držati parametar *riječ* kao jednu od značajki bitnih za izgradnju stabla. Opisani će ulazni objekt stabla odluke, nakon ove provedene inicijalne obrade, sadržavati tri značajke i oznaku klase, tj. sadržavat će:

1. [značajka] Oznaka klase kojoj riječ pripada
2. [značajka] Ponuđeni ispravci ili oznaka pripadnosti skupini *riječ*, *tuđica* ili *ime*
3. [značajka] Zadovoljena minimalna pojavnost (parametar *prag*) riječi s nastavcim iz iste skupine glagolskih nastavaka (poprima vrijednosti *true* ili *false*, a vrijednost računa programsko rješenje kada pročita svaki ulaz)
4. [oznaka klase] Potencijalni glagol (poprima vrijednosti *true* ili *false*, a za izračun su vrijednosti potrebne sve četiri značajke)

## 4.5. Provedba učenja

Za što ispravniju i bolje nadziranu provedbu učenja, učenje valja krenuti provoditi nad malim skupom ulaznih podataka i postupno ga povećavati. Na prva tri skupa podataka za učenje (veličine 10, 20 i 30 ulaznih primjera) primijećen je uzorak koji se pojavljuje u naučenom stablu odluke kod sva tri skupa podataka. Naime, svako je stablo izgrađeno tako da je treća značajka, tj. značajka o zadovoljenosti minimalne pojavnosti, korijenski čvor iz kojeg se, pri ispitivanju ispitnih primjera nad naučenim stablom, svi primjeri odmah mogu razvrstati u klasu *da* (ovo je potencijalni glagolski oblik) i *ne* (ovo nije potencijalni glagolski

oblik). To znači da su ostale značajke redundantne jer nikako ne pridonose izgradnji stabla, pa ne utječu na klasifikaciju nepoznatih ispitnih primjera prilikom ispitivanja stabla.

Tablica 4.1 Analiza skupa podataka za učenje

| **Klasa riječi** | **Riječ** | **Ponuda** | **Zadovoljen *prag*** | **Oznaka klase** |
|---|---|---|---|---|
| -ss- | brijem | RIJEČ | da | da |
| -mm- | nagalsila | naglasila | ne | ne |
| -mm- | najdraze | najdraže | ne | ne |
| -ss- | naljeganje | nalijeganje | ne | ne |
| -ss- | isperete | RIJEČ | da | da |
| -ll- | Aldeburghu | IME | ne | ne |
| -ll- | Amstronga | Armstronga | ne | ne |
| -ss- | pišemo | RIJEČ | da | da |
| -kk- | TorinO | Torino | ne | ne |
| -kk- | f78b | f78bsfef | ne | ne |

Obrazloženje ovog fenomena može se prikazati na najmanjem skupu podataka za učenje (prikazan tablicom Tablica 4.1), a primjenjiv je i na bilo koje veće skupove podataka za učenje. Početni se čvor, kao i svaki novi čvor u stablu, odabire na temelju vrijednosti informacijske dobiti (objašnjene u potpoglavlju 4.2.2) koja predstavlja stupanj diskriminativnosti svake značajke. Pri svakoj izgradnji novog čvora stabla, algoritam ID3 odabire čvor s najvećom informacijskom dobiti, tj. čvor koji je najdiskriminativniji (pravi najveću razliku). Na primjeru iz tablice Tablica 4.1 jasno je da *zadovoljen_prag = da* uvijek daje oznaku klase *da*, a *zadovoljen_prag = ne* uvijek daje oznaku klase *ne*. Uvođenje značajke o zadovoljenosti praga znači da će uvijek postojati značajka s optimalnom informacijskom dobiti i algoritam će uvijek baš tu značajku odabrati za korijenski čvor, a stablo će odmah tada stati s daljnjim učenjem.

Oznaka klase koja govori o tome je li riječ potencijalni glagolski oblik potpuno je istovjetna sa značajkom o zadovoljenosti praga jer je ta značajka, na ovoj razini, jedini

izvedivi pokazatelj mogućnosti da bi riječ mogla pripadati nekom glagolskom obliku. Iz ovoga je jasno da stablo odluke izvedeno algoritmom ID3 nije adekvatan pristup za rješavanje ovog problema.

# 5. Ispitivanje programskog rješenja

## 5.1. Ispitivanje nad skupom riječi iz rječnika

Rječnik nad kojim se provodi ispitivanje broji 1 111 504 riječi. Za ispitivanje nad skupom riječi iz rječnika, programu je, prije pokretanja, kao drugi parametar potrebno predati Booleovu vrijednost *false*. Program prije početka svih provjera obavezno učita, tj. *hashira*, rječnik. Ugrađenom je metodom *toArray* u Javi jednostavno iterirati po svim riječima rječnika pohranjenog kao objekt tipa *HashSet*. S obzirom na to da se u ovom slučaju preskaču sve inicijalne provjere (jer su sve riječi rječnika ispravne), za svaku se riječ poziva jedino metoda *jeLiGlagol* koja obavlja provjeru riječi po nastavcima.

Da bi metoda *jeLiGlagol* riječ smatrala potencijalnim glagolskim oblikom, riječ mora zadovoljiti definirani minimalni broj pojavnosti riječi u rječniku s nekim drugim nastavkom iz iste skupine. Taj minimalni broj pojavnosti nadalje zovemo *prag*. Njegova se vrijednost definira trećim ulaznim parametrom programa koji se predaje programu prije pokretanja. U nastavku su prikazani rezultati za ispitivanje programskog rješenja za različite vrijednosti parametra *prag*.

Ispitamo li program za slučaj *prag* = 1, program će pod potencijalne glagolske oblike pogrešno klasificirati poprilično velik broj riječi iz rječnika kao potencijalne glagole. Primjeri takvih slučajeva prikazani su u tablici Tablica 5.1. i iz njih je jasno zašto se to događa. Raspodjela riječi po glagolskim vremenima i skupu podataka *smece* prikazana je dijagramom Dijagram 5.1. Najveći problem predstavljaju prezent (u koji program svrsta 33,44 % riječi iz rječnika) i aorist (u koji program svrsta 49,64 % riječi iz rječnika), a skup *smece* (u koji bi trebale ići sve riječi koje nisu glagolski oblici) ostaje prazan. Kod prezenta su problematični nastavci *i* i *e* iz druge skupine nastavaka te *e* i *u* iz treće skupine nastavaka jer se ti nastavci često pojavljuju u oblicima riječi koje nisu glagoli.

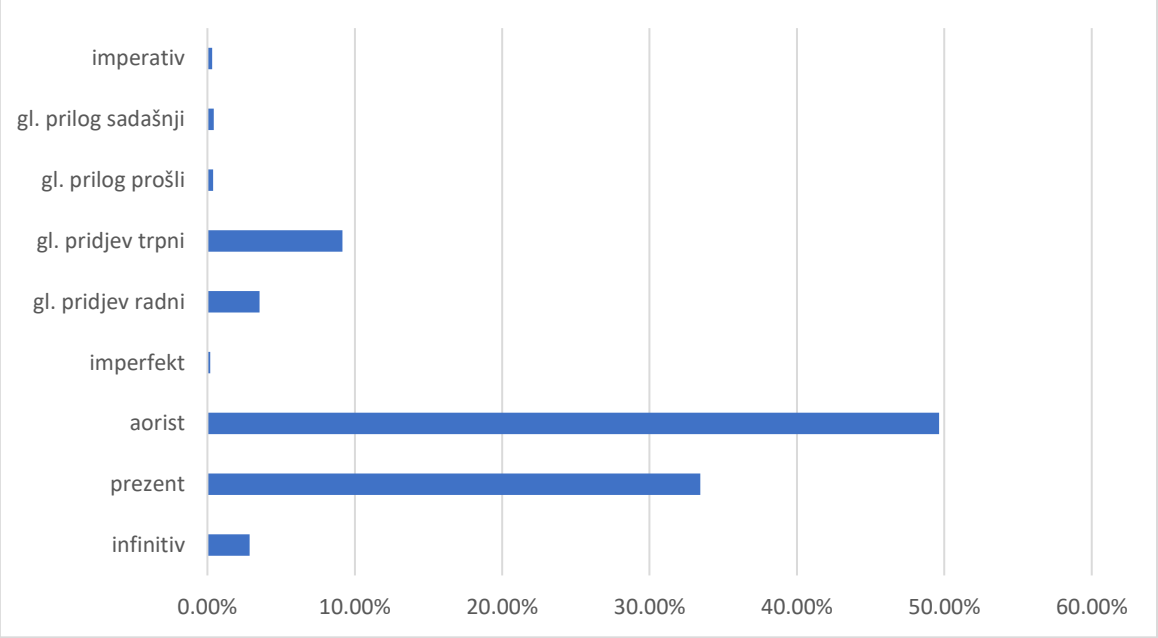
Tablica 5.1 Primjeri pogrešno klasificiranih riječi kada je *prag* = 1

| **Riječ** | **Oblik riječi koji spada u istu skupinu nastavaka** | **Dodijeljeni razred** |
|---|---|---|
| aviopromet | avioprometa | gl. pridjev trpni |
| kuhinja | kuhinja | aorist |

| | | |
|---|---|---|
| otkazano | otkazani | gl. pridjev trpni |
| pohanim | pohane | prezent |
| crvenkasti | crvenkasto | infinitiv |

Ako program, primjerice, obrađuje riječ *kuhinje* i zamjenom nastavaka dobije *kuhinji*, ta riječ postoji u rječniku i program riječ *kuhinje* svrstava pod prezent. Treća skupina nastavaka aorista također je problematična jer u 2. i 3. l. jd. sadrži nastavak “”, pa će program riječ, ako ne završava nijednim nastavkom nekog od ostalih glagolskih oblika, svrstati pod aorist jer svaka riječ završava nastavkom “” i program će zapravo, kada na red dođe zamjena s drugim nastavkom “”, u rječniku tražiti identičan oblik početne riječi što uvijek rezultira pronalaskom i svrstavanjem u aorist.

Jasno je da ovakav pristup nema smisla – parametar *prag* nužno je povećati.

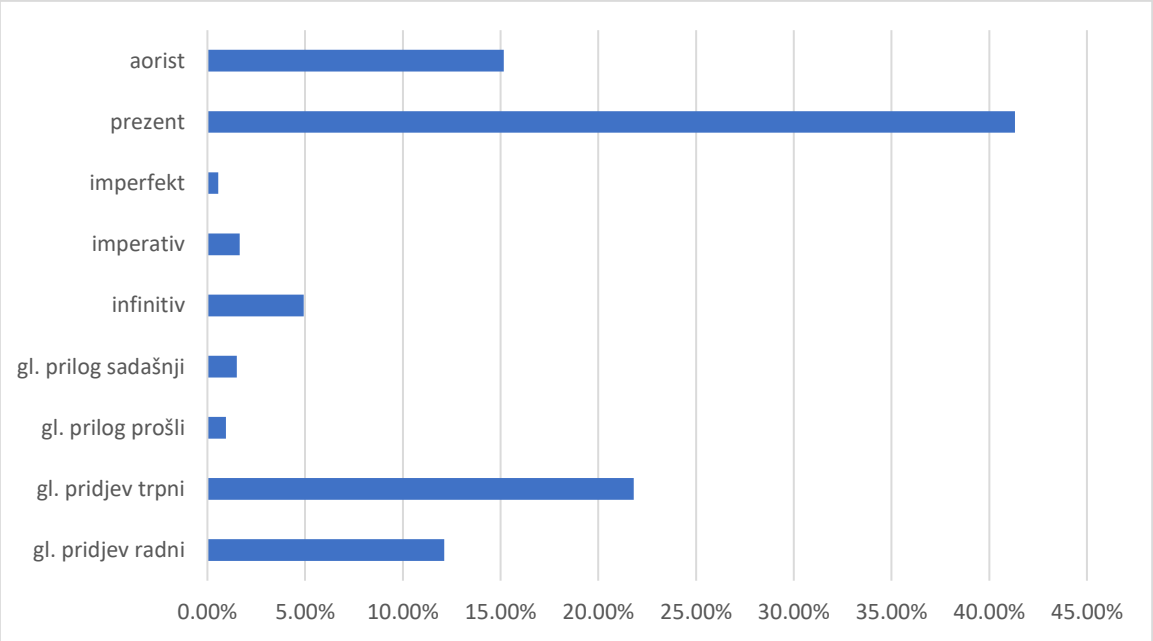



Dijagram 5.1 Riječi klasificirane po glagolskim oblicima za *prag* = 1

Povećanjem parametra *prag* na vrijednost *prag* = 2 rezultati se znatno mijenjaju. Prvi je, i vjerojatno najveći, pomak to što skup podataka *smece* ne ostane prazan. Program postavljenim uvjetom sigurnije i točnije grupira glagole, a stupanj pogreške znatno pada jer ranije spomenute problematične skupine nastavaka ne sadrže više od dva nastavka karakteristična za oblike riječi koje nisu glagolski oblici. Tako bi program, primjerice, za riječ *kuhinje* i *prag* = 1 u rječniku pronašao još i *kuhinji* što bi bio dovoljan uvjet da riječ svrsta pod prezent. Za istu riječ i *prag* = 2 program pronalazi *kuhinji*, ali ne pronalazi inačice

riječi s nekim od ostalih nastavaka koji su dio iste skupine (-im, -iš, -imo, -ite) jer se oni, uglavnom, ne pojavljuju u oblicima riječi koje ne pripadaju glagolskim oblicima. Broj riječi koje ne pripadaju nekom glagolskom obliku određen je veličinom skupa podataka *smece* koji broji 663 731 riječi, tj. 59,71 % od svih riječi rječnika. Preostalih 447 773 riječi, tj. 40,29 % od svih riječi rječnika, program je razvrstao pod glagolske oblike, a raspodjela je riječi po glagolskim vremenima unutar tog skupa prikazana dijagramom Dijagram 5.2 i znatno je stvarnija i točnija od raspodjele riječi iz Dijagram 5.1 za *prag* = 1.

Za analizu točnosti klasifikacije riječi u skup glagola odabran je slučajni uzorak od 384 riječi iz skupa svih riječi koje je program svrstao pod glagole. Za računanje veličine uzorka korišten je parametar veličine populacije glagola (447 773), stupanj pouzdanosti[3] od 95 %, granica pogreške[4] od 5 % i proporcija populacije[5] od 50 % [10]. Točnost klasifikacije na tom slučajnom uzorku iznosi 47,14 %.



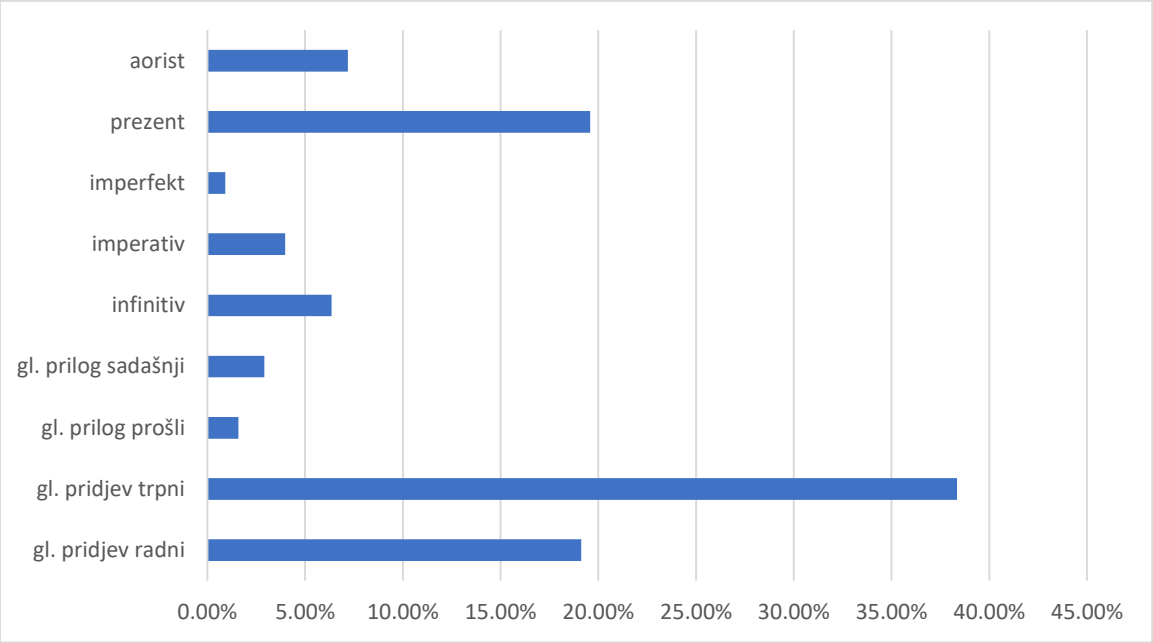

Dijagram 5.2 Riječi klasificirane po glagolskim oblicima za *prag* = 2

U slučaju ispitivanja programskog rješenja za vrijednost *prag* = 3 skup glagola ukupno broji 267 923 riječi, tj. 24,10 % od svih riječi, a raspored glagola po glagolskim vremenima prikazan je dijagramom Dijagram 5.3. Vidljiva je prevlast glagolskog pridjeva

[3] Stupanj pouzdanosti – vjerojatnost da stvarna vrijednost koja se proučava bude unutar vrijednosti izračunate na temelju uzorka [10]
[4] Granica pogreške – mjera za iskazivanje greške prilikom uzorkovanja [10]
[5] Proporcija populacije – procjena udjela svojstva od interesa unutar cijele populacije [10]

trpnog, a to je zato što su njegovi nastavci prisutni u izvedenim oblicima velikog broja imenica i pridjeva (npr. rječnik za pridjev *mesnat* sadrži i oblike *mesnata*, *mesnato* i *mesnati*), pa te imenice, čak i za *prag* = 3, prolaze kao glagoli.

Za analizu točnosti klasifikacije riječi u skup glagola odabran je slučajni uzorak od 384 riječi iz skupa svih riječi koje je program svrstao pod glagole. Za računanje veličine uzorka korišten je parametar veličine populacije glagola (267 923), stupanj pouzdanosti od 95 %, granica pogreške od 5 % i proporcija populacije od 50 % [10]. Točnost klasifikacije na tom slučajnom uzorku iznosi 63,02 %.




Dijagram 5.3 Riječi klasificirane po glagolskim oblicima za prag = 3

Za ispitivanje programa s parametrom *prag* = 4 skup riječi svrstanih u glagole broji 233 760 riječi, a za analizu točnosti klasifikacije riječi u skup glagola odabran je slučajni uzorak od 384 riječi iz skupa svih riječi koje je program svrstao pod glagole. Za računanje veličine uzorka korišten je parametar veličine populacije glagola (233 760), stupanj pouzdanosti od 95 %, granica pogreške od 5 % i proporcija populacije od 50 % [10]. Točnost klasifikacije na tom slučajnom uzorku iznosi 73,44 %.

Klasifikacija riječi uz postavke *prag* = 5 pokriva isključivo glagolske oblike za koje rječnik sadrži sve ostale oblike iz te skupine glagolskih nastavaka. Veličina skupa označenih glagola iznosi 172 832, a za računanje točnosti koristi se slučajni uzorak od 384 riječi [10] (veličina je uzorka izračunata istim parametrima kao i za slučaj *prag* = 4 uz pilagođen parametar veličine populacije). Točnost iznosi 93,75 %.

## 5.2. Ispitivanje nad skupom prikupljenih pogrešaka

Prethodnim je potpoglavljem ispitana točnost rada programskog rješenja uz različite postavke parametra *prag* za ispitivanje nad rječnikom, tj. riječima koje smatramo potpuno ispravnima. Stvarni je skup podataka, tj. skup podataka s prijavljenim pogreškama, gotovo tri puta veći od rječnika i manje je uredan, pa prije provjere nastavaka prolazi filtre opisane u potpoglavlju 3.2, a slične postavke ispitivanja programskog rješenja utvrđene na ispitivanju nad rječnikom možemo primijeniti i na ovaj ispitni slučaj. Broj je odbačenih riječi, očekivano, znatno veći jer se radi o skupu podataka u kojem su potencijalne suvisle riječi u velikoj manjini, a tek u tom podskupu program može tražiti potencijalne glagolske oblike.

Tablica 5.2 prikazuje rezultate ispitivanja programskog rješenja za različite vrijednosti parametra *prag*. Za riječi je iz rječnika bilo jasno koje su ispravno klasificirane kao glagoli, a koje ne, ali na ovom je skupu podataka točnost teško definirati jer su neki glagoli ispravno klasificirani, ali su pogrešno napisani, pa je često teško odrediti o kojoj se riječi uopće radi. Zbog tih je prevelikih odstupanja i nepravilnosti računanje točnosti na ovom skupu podataka preskočeno, ali su iz uzorka skupa glagola izdvojeni neki pronađeni smisleni glagolski oblici koji ne postoje u rječniku, a to su: *tapšem, osposobih, presulo, suosjećajmo, zbrisaše, proređemo, pritekni, zakašljemo, otpočnimo, prisluškujte, uspuhale, prizemljuješ, prozračuješ, koristivši, okretavši, rasplamsajmo, namjeravajte, dojahaše, versificirani, otpušemo, krivudale, zarotirajmo, premišljale, unaprjeđuješ,*

Tablica 5.2 Statistika skupa glagola za različite vrijednosti parametra *prag*

| ***prag*** | **Veličina skupa glagola** | **Udio glagola** |
|---|---|---|
| 1 | 66 388 | 1,99 % |
| 2 | 38 519 | 1,16 % |
| 3 | 20 167 | 0,6 % |
| 4 | 16 222 | 0,49 % |
| 5 | 13 120 | 0,39 % |

# Zaključak

Hrvatski je jezik iznimno složen za programsku analizu i oko kompleksnih dijelova, kao što je klasifikacija glagolskih oblika, ne postoji konsenzus i ne postoji ispravni ili jednoznačni način podjele glagola. Programsko rješenje izvedivo u opsegu ovog rada može se zadovoljiti pojednostavljenom podjelom glagola na skupine nastavaka glagolskih vremena. Takav pristup ograničava programsko rješenje na isključivo pravilne glagole i još nam više približava razinu složenosti koju obrada hrvatskog jezika zahtijeva. Za ovu se razinu zadatka programsko rješenje, s običnim algoritmima grananja, pokazalo adekvatno i primjerene složenosti. Programsko rješenje iz ulaznog skupa riječi uspijeva prepoznati glagolske oblike uz (očekivani) šum uglavnom raznih oblika pridjeva i imenica za koje postoji velik broj izvedenih oblika koji završavaju glagolskim nastavcima. Stupanj je nužne minimalne podudarnosti (parametar *prag*) pritom jednostavno prilagoditi ulaznom skupu podataka, a točnost se klasifikacije riječi kao glagola povećava povećanjem vrijednosti parametra *prag*. Jasno je da povećanje vrijednosti parametra *prag* jamči povećanje točnosti klasifikacije riječi kao glagola, ali ostaje pitanje koliko je to isplativo jer bi program tako mogao odbaciti oblike nekih glagola koji su u rječniku relativno mladi i rječnik ne broji više od dva izvedena oblika tog glagola unutar jedne skupine nasavaka.

Stablo se odluke za analizu zadanog skupa pogrešaka pokazalo kao neadekvatno rješenje, a programsko rješenje jezičnih problema ovakvog tipa i ovolikog opsega treba nastaviti tražiti među ostalim granama, pristupima i algoritmima umjetne inteligencije.

# Literatura


[1] *Ispravi.me – powered by Hascheck*. Poveznica: https://ispravi.me/; pristupljeno 3. lipnja 2024.

[2] *Ahrefs – Website Traffic Checker*. Poveznica: https://ahrefs.com/traffic-checker; pristupljeno 7. lipnja 2024.

[3] *SimilarWeb*. Poveznica: https://www.similarweb.com/; pristupljeno 7. lipnja 2024.

[4] Gledec, G., Dembitz, Š. *Razvoj i održavanje n-gramskog sustava hrvatskog jezika*. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2024.

[5] Dembitz, Š. *Funkcionalna leksikografija mrežnog pravopisnog provjernika*. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2012.

[6] Pavlek, J. *Proširenje leksičke baze mrežnog pravopisnog provjernika*. Magistarski rad. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2010.

[7] *Hrvatska školska gramatika*. Poveznica: http://gramatika.hr/; pristupljeno: 16. svibnja 2024.

[8] Šošić, L. *Opis glagolskih vrsta i razreda u hrvatskom jeziku*. Diplomski rad. Sveučilište u Splitu, Filozofski fakultet, 2020.

[9] Dalbelo Bašić, B., Šnajder, J. *Strojno učenje*. Inačica v1.5. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, ak. god. 2019./2020.

[10] *Calculator.net – Sample size calculator*. Poveznica: https://www.calculator.net/sample-size-calculator.html; pristupljeno 29. svibnja 2024.

# Sažetak

Nadzirano učenje u strojnom provjerniku pravopisa

Strojni provjernik pravopisa Hascheck, danas poznatiji po svom web-sjedištu Ispravi.me, svake godine broji sve više posjeta i obrađenih tekstova. Provjernik pritom neprestano svim novim i nepoznatim, ali smislenim, riječima koje obradi nadziranim učenjem širi i razvija svoj rječnik. Nadzor je potreban radi očuvanja točnosti rječnika, pa ulazak svake riječi u rječnik mora odobriti čovjek, a riječi koje mu dolaze na ručnu provjeru poželjno je što bolje *pročistiti* od nesuvislih i pogrešnih riječi tako da se ljudski rad svede na minimum. Ovaj rad nudi programsko rješenje za izdvajanje glagolskih oblika iz korpusa prijavljenih grešaka. Programsko se rješenje temelji na jednostavnoj i programski ostvarivoj klasifikaciji glagola hrvatskog jezika po nastavcima koja je iscrpno opisana i obrađena kroz poglavlja ovog rada. Uz klasični programerski pristup, detaljno je opisan i pristup izrade programskog rješenja temeljenog na metodi nadziranog strojnog učenja - stablu odluke. Na kraju su rada prikazani rezultati i uspješnost primjene ostvarenog programskog rješenja zajedno s opisom varijabilnog ulaznog parametara koji je moguće jednostavno prilagoditi ulaznom skupu podataka.

# Summary

Supervised learning in an online spellchecking service

An online spellchecking service Hascheck, today better known for its new website Ispravi.me, continues its trend of rapid growth thanks to the increasing number of total website visits and total number of processed tasks. In doing so, the spellchecker tends to continuously expand and develop its dictionary with all new and unknown but meaningful words it processes through supervised learning. Supervision is necessary to maintain the accuracy of the dictionary, so the entry of each word into the dictionary must be approved by a human. It is desirable to filter out nonsensical and incorrect words as much as possible before they reach manual verification, thus minimizing human effort. This paper offers a programmatic solution for extracting verb forms from the corpus of reported errors. The solution is based on a simple and programmatically feasible classification of verbs in Croatian language by their suffixes, which is comprehensively described and elaborated upon in the chapters of this paper. Alongside the classical programming approach, the paper also provides details for the development of the programmatic solution based on the supervised machine learning method - decision tree. At the end of the paper, the results and effectiveness of the implemented programmatic solution are presented, along with a description of the variable input parameter that can be easily adjusted to the input data set.