Mogućnosti prepoznavanja govora korištenjem biblioteke SpeechRecognition

Tolja, Margarita

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Margarita Tolja

MOGUĆNOSTI PREPOZNAVANJA GOVORA KORIŠTENJEM BIBLIOTEKE SPEECHRECOGNITION

Diplomski rad

Voditelj rada:
dr. sc. Goran Igaly, v. pred.

Zagreb, veljača, 2022.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Mami za svaki biskupski kruh, tati za svaku dobru vibraciju, baki za svaku pileću jetricu, nonu za svaki aleksandrin, ekipi za svaku foru i Petri za sve između; Hvala.

Sadržaj

Uvod

Prosječnom čovjeku je interakcija s računalom ili nekim drugim elektroničkim uređajem danas postala dio svakodnevnice. Takva interakcija se i dalje većinom odvija posredno; korištenjem tipkovnice, upravljačkih konzola te raznih drugih sučelja. Iako su ljudi naučili efikasno navigirati navedenim sučeljima, takva interakcija im nije prirodna. Čovjekovo najvažnije sredstvo komunikacije je jezik, a primarni medij govor. Uzevši da prosječan čovjek može izgovoriti 150 riječi u minuti, a natipkati samo 40, jasna je potreba za implementacijom govorne interakcije čovjeka i stroja. Osim što se govorom brže prenose informacije, rad uz slobodne ruke (*handsfree*) otvara potpuno nove mogućnosti korištenja tehnologije.

U kontekstu govorne interakcije radi se distinkcija među sljedećim poljima računarstva:

- Prepoznavanje govora (*Speech Recognition*) se bavi pretvorbom govora u tekst.
- Sinteza govora (*Speech Synthesis*) se bavi pretvorbom teksta u umjetno generiran govor.
- Obrada prirodnog jezika (*Natural Language Processing*) se bavi *razumijevanjem* jezika.
- Prepoznavanje glasa (*Voice Recognition*) se bavi identifikacijom osobe na temelju glasa.

Ovaj rad obrađuje područje prepoznavanja govora, i to iz perspektive korisnika Python biblioteke SpeechRecognition. Riječ je o složenom, multidisciplinarnom području koje se ubrzano razvija. Kao rezultat su nastala brojna službena i rješenja otvorenog koda, od kojih su najvažnija dostupna preko istaknute biblioteke.

U prvom poglavlju se daje pregled razvoja sustava za automatsko prepoznavanje govora. Posebno se opisuju i uspoređuju konvencionalni, te noviji *end-to-end* sustavi. Sljedeće poglavlje opisuje sve mogućnosti (i nemogućnosti) biblioteke SpeechRecognition. Na kraju, u trećem poglavlju je demonstrirana upotreba biblioteke na vlastitoj aplikaciji za kreiranje transkripata.

Poglavlje 1

Automatsko prepoznavanje govora

Kao što je spomenuto u uvodu, prepoznavanje govora predstavlja sposobnost računala da prepozna izraze izgovorene od strane različitih govornika te ih pretvori u tekst. Sustavi koji to omogućuju se nazivaju sustavima za automatsko prepoznavanje govora (*Automatic Speech Recognition*, skraćeno ASR).

Automatsko prepoznavanje govora predstavlja zahtjevan problem iz više razloga. Za početak, ono uključuje hvatanje i digitalizaciju zvučnih valova, njihovo pretvaranje u osnovne jezične jedinice ili foneme, izgradnju riječi od fonema, i napokon kontekstualne analize riječi. Svaki od navedenih koraka se može zasebno proučavati kao netrivijalan problem strojnog učenja, a ASR sustavi se dodatno bave i njihovom integracijom.

Nadalje, pri razvoju ASR sustava je nužno obratiti pozornost na brojne izvore varijabilnosti. Ulazni govor varira ovisno o karakteristikama govornika – specifičnom naglasku, brzini govora, spolu i starosti. Također, razlikuju se mogući stilovi govora – je li govor opušten ili služben, radi li se o kontinuiranom govoru ili izoliranim riječima te kako prethodna riječ utječe na sljedeću (koartikulacija). Poseban izazov predstavljaju različita okruženja govornika – razina pozadinske buke, postojanje jeke, te međusobno uplitanje sugovornika. Na kraju, treba uzeti u obzir i specifičnost zadatka ASR sustava – koliko je opširan vokabular, odnosno stavlja li se fokus na određenu gramatiku i naredbe ili se prepoznaje slobodan govor.

U nastavku se opisuje kako se kroz povijest mijenjao pristup problemu automatskog prepoznavanja govora, te na koji način rade današnji najnapredniji ASR sustavi.

1.1 ASR sustavi kroz povijest

Dizajniranje stroja koji oponaša ljudsko ponašanje, posebice sposobnost pravilnog reagiranja na govorni jezik, je intrigiralo znanstvenike i inžinjere stoljećima. Tehnologija prepoz-

navanja govora je postala popularna tema i u neznanstvenim sferama nakon pojave Hal-a iz „2001: A Space Odyssey" (1968.) te C3PO-a iz „Star Wars" sage (1977.). Slijedi kratak kronološki pregled najvažnijih dostignuća iz tog područja, a iscrpniji pregled je dostupan u [14].

Intenzivniji razvoj sustava za prepoznavanje govora započinje tek u drugoj polovici dvadesetog stoljeća, prateći općeniti razvoj tehnologije. Početna istraživanja su bila najviše inspirirana načinom na koji ljudi govore i čuju, a rješenja su se bazirala na prepoznavanju određenih jezičnih uzoraka. Prvi sustavi za prepoznavanje govora bili su usmjereni na brojeve, a ne na riječi. Godine 1952. Bell Laboratories dizajnirao je sustav „Audrey", koji je mogao prepoznati izgovor znamenaka za jednog govornika. Deset godina kasnije, IBM je predstavio sustav „Shoebox". „Shoebox" je prepoznavao brojeve od 0 do 9 te šest različitih aritmetičkih operacija. Oba sustava su radila isključivo s izoliranim riječima, odnosno uz velike pauze između izgovorenih riječi.

Tijekom 60-ih godina, diljem svijeta se razvijaju ASR sustavi, i dalje skromnih mogućnosti. Krajem desetljeća, Atal i Ikatura neovisno oblikuju osnovne koncepte linearno-prediktivnog kodiranja (*linear predictive coding*, skraćeno LPC), koji pojednostavljuju reprezentaciju zvučnog signala [14]. Do kraja 60-ih godina, ASR sustavi su mogli prepoznati riječi s do četiri samoglasnika i do devet suglasnika.

U 70-im godinama prošlog stoljeća ministarstvo obrane SAD-a pokreće Speech Understanding Research program. To je bio jedan od najvećih programa u ovom području koji je na kraju iznjedrio „Harpy" – ASR sustav razvijen pod vodstvom sveučilišta Carnegie Mellon. „Harpy" je imao vokabular od 1011 riječi te mogućnost prepoznavanja kontinuiranog govora. Važnim doprinosom ovog sustava se smatra novi grafovski prikaz jezika koji je koristio leksičku reprezentaciju riječi.

Paralelno s naporima američkog programa, oblikuju se dva nova smjera u istraživanju prepoznavanja govora, predvođena timovima iz IBM-a i AT&T Bell Laboratories-a. Istraživači iz IBM-a su za cilj imali stvoriti pisaći stroj koji se aktivira govorom, tj. stroj za transkripciju. Tehnički cilj je bio podržati prepoznavanje što većeg vokabulara, s naglaskom na domenu poslovne korespondencije. Upravo zato je u sklopu rada na ovom sustavu najveći napredak napravljen u načinu shvaćanja jezičnih modela. Kao rezultat njihovih napora je predstavljen sustav „Tangora" koji je postizao dobre rezultate, ali se morao trenirati posebno za svakog govornika. S druge strane, AT&T Bell Laboratories program je za cilj imao razviti automatizirane telekomunikacijske usluge, kao što su glasovno biranje, te upravljanje usmjeravanjem poziva. Najvažniji zahtjev je bio da sustav dobro funkcionira za veliku populaciju govornika, bez potrebe za dodatnim treniranjem. Zato je primarni fokus ovog programa bio poboljšanje akustičnog modela, s naglaskom na bolju kontrolu akustičnih varijabilnosti.

U 80-im godinama dolazi do svojevrsne konvergencije navedenih dvaju pristupa, uzrokovane ubrzanim razvojem statističkih metoda za prepoznavanje govora. Daleko najvažnija je metoda skrivenih Markovljevih modela (*Hidden Markov Models*, skraćeno HMM) koja je u upotrebi još i danas. Riječ je o potpunom zaokretu; dotada se prepoznavanje baziralo na traženju zvučnih obrazaca u riječima, dok se upotrebom HMM-a može procijeniti vjerojatnost da nepoznati zvukovi zapravo tvore riječ. Istraživači iz IBM-a, predvođeni Frederickom Jelinekom su odigrali ključnu ulogu u razvoju statističkih metoda. Jelinek je već u 70-ima zagovarao odmak od imitacije ljudskog razumijevanja govora, ističući: „Zrakoplovi ne mašu krilima." [17]

Daljnjim razvojem statističkih metoda za prepoznavanje govora se postižu sve bolji rezultati. Tako u 90-im godinama već nastaju sustavi koji prepoznaju vokabular od deset tisuća riječi. Također, u to doba su objavljeni prvi komercijalni sustavi za prepoznavanje govora. Najpoznatiji od njih su sustavi „Dragon Dictate" – sustav namijenjen osobnim računalima koji je prepoznavao 30 riječi po minuti (uz prethodno treniranje), te „VAL" – interaktivni portal koji je davao tražene informacije preko telefona. Iako su ovi sustavi svakako predstavljali napredak, još su bili daleko od idealnog.

Do 2000-tih godina sustavi za prepoznavanje govora dostižu točnost od 80%. Međutim, dobri rezultati su se postizali samo za relativno ograničene vokabulare. U sljedećem desetljeću dolazi do stagnacije napretka u ovom području, sve do 2010. godine kada Google prezentira „Voice Search" aplikaciju za iPhone. Budući da se radi o mobilnoj aplikaciji, prepoznavanje govora je postalo dostupno milijunima korisnika. Ovo je značajno jer su se prikupljeni podaci od preko milijardu pretraživanja mogli koristiti za daljnje pospješivanje modela. [28]

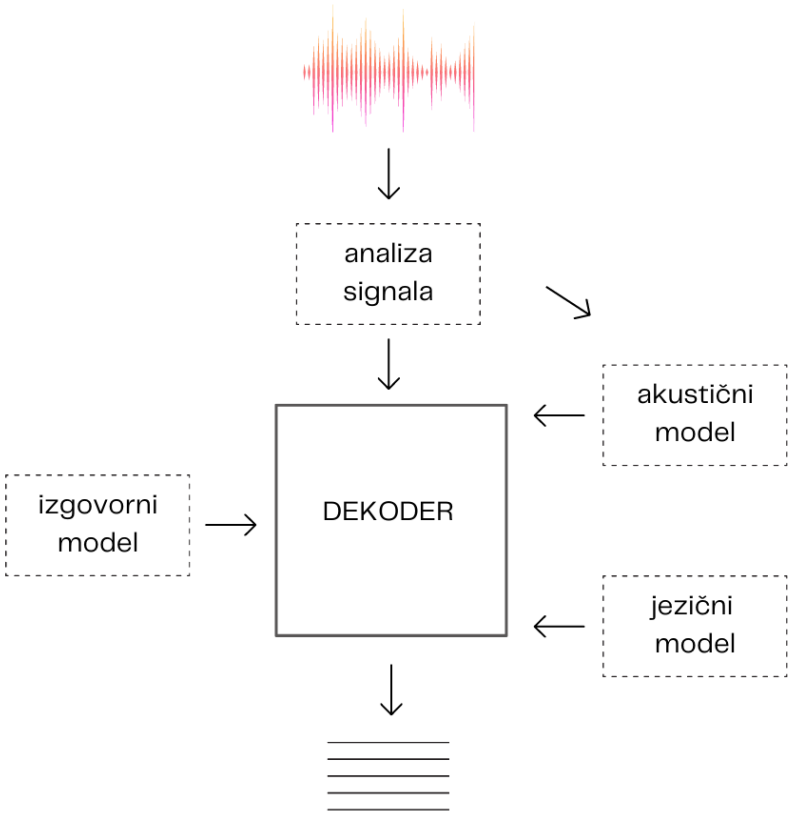
Od 2010. godine do danas, ponovno se bilježi veliki pomak u kvaliteti sustava za prepoznavanje govora, ponajviše zbog uključivanja dubokih neuronskih mreža (*deep neural networks*, skraćeno DNN) u proces prepoznavanja. Osnovna ideja korištenja DNN-a je bila poznata još od prije, ali se počela primjenjivati tek kada su postali dostupni potrebni računalni resursi. Iz istog razloga, sustavi temeljeni na dubinskim neuronskim mrežama su većinom implementirani kao „Cloud" rješenja. Kao primjeri ovakvih sustava se mogu izdvojiti svi današnji popularni virtualni asistenti: „Voice Search" (Google), „Siri" (Apple), „Cortana" (Microsoft), „Alexa" (Amazon) i drugi. Navedeni sustavi uspješno prepoznaju vokabulare od više od milijun riječi.

Iako se današnji sustavi mogu pohvaliti visokom točnošću (oko 85%), ovaj problem je još daleko od toga da se smatra riješenim. Neki od izazova današnjice su poznati od prije, a tiču se prepoznavanja različitih sugovornika, te prepoznavanja govornika s poteškoćama u izgovoru. Također, izostaje kvalitetna podrška za mnoge afričke i azijske jezike. Nadalje, novi izazovi se odnose na razvoj ASR sustava sa smanjenim računalnim resursima te postavke privatnosti i sigurnosti pri korištenju istih.

1.2 Konvencionalni ASR sustavi

U ovom i sljedećem odlomku se daje pregled načina rada najsuvremenijih ASR sustava danas. Preciznije, opisuju se dvije vrste sustava: konvencionalni i *end-to-end* sustavi. Prva od navedenih predstavlja trenutno najzastupljenije sustave u komercijalnoj upotrebi, dok druga predstavlja središte današnjih akademskih istraživanja. *End-to-end* sustavi se konceptualno i strukturalno znatno odmiču od uvaženog principa rada ASR sustava, te su zato dosadašnji sustavi dobili atribut „konvencionalni", „tradicionalni" ili „standardni".

Konvencionalni ASR sustavi prepoznaju ulazni govor kroz faze (*pipeline*). Na slici 1.1 je prikazana osnovna struktura tipičnog ASR sustava. Kao što je prikazano, kako bi se od ulaznog zvuka došlo do krajnjeg transkripta, integrirano je pet podsustava od kojih svaki ima važnu i točno definiranu ulogu. [15]



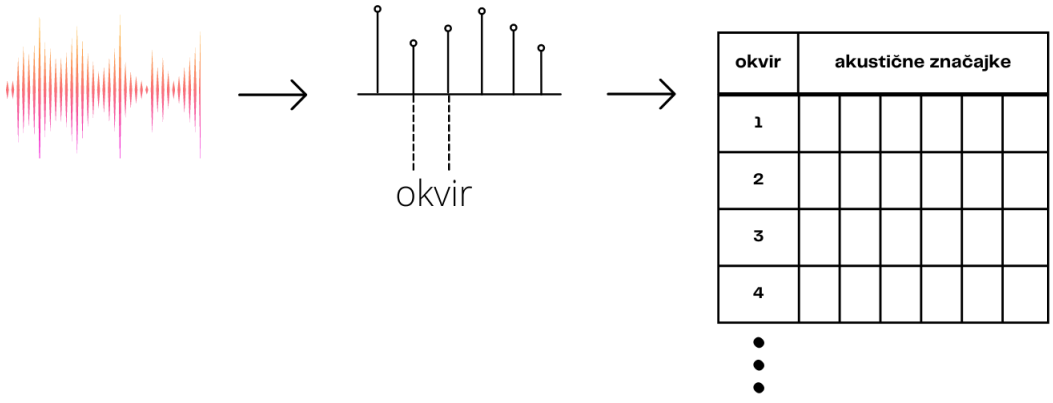
Slika 1.1: Struktura konvencionalnog sustava

1.2.1 Analiza zvučnog signala

Početna komponenta svakog sustava za prepoznavanje govora je komponenta za analizu zvučnog signala. Njezina zadaća je pretvoriti ulazni zvučni signal u digitalan oblik *razumljiv* računalima.

Proces digitalizacije zvučnog signala se sastoji od dva koraka. Prvo, valni oblik audio signala se diskretizira u okvire (*frames*) od 10 do 40 milisekundi. Ovakvim pojednostavljenjem ne dolazi do značajnog gubitka informacija zbog činjenice da je govor vremenski kvazi-stacionaran signal [33]. To znači da se na kratkim vremenskim intervalima od 5 do 100 milisekundi svojstva zvučnog signala značajno ne mijenjaju. Zato je moguće npr. interval od 40 milisekundi svesti na samo jedan okvir koji odgovara stanju u određenoj sekundi tog intervala.

Sljedeći korak ima zadatak na temelju izabranih zvučnih okvira odrediti vrijednosti akustičnih značajki (*feature extraction*). Postoje razne metode za određivanje vrijednosti značajki, a najpoznatija je metoda mel-frekvencijskih kepstralnih koeficijenata (*Mel Frequency Cepstral Coefficients*, skraćeno MFCC) koja je inspirirana načinom na koji radi ljudsko uho [33].



Slika 1.2: Analiza zvučnog signala

Kao što je prikazano na slici 1.2, rezultat analize signala je niz vektora vrijednosti značajki, od kojih svaki vektor opisuje jedan okvir.

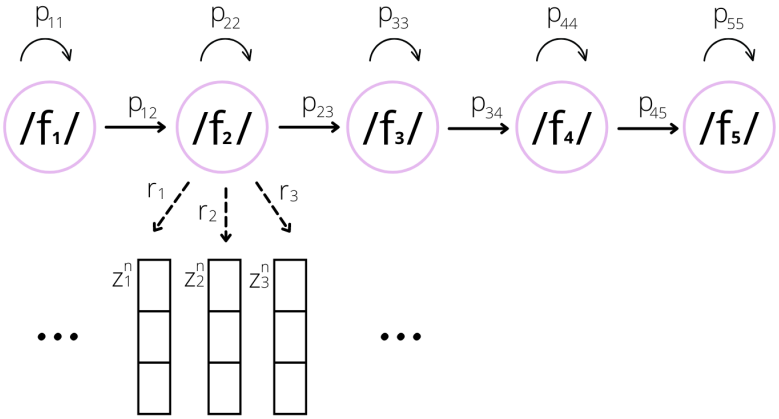
1.2.2 Akustični model

Sljedeća komponenta se često izdvaja kao najsloženiji dio sustava. Njena uloga je na osnovu predanih vrijednosti akustičkih značajki odrediti odgovarajući niz fonema.

Fonem je najmanja glasovna jedinica koja ima razlikovnu ulogu u riječi [21]. Primjerice, razlika između riječi „drag" i „trag" osigurava fonemski status glasovima [d] i [t], koji se kao fonemi bilježe /d/ i /t/. Svaki jezik ima specifičan skup fonema (obično od 20 do 60 fonema), kojima se može opisati svaka riječ tog jezika. Preslikavanja `riječ` → `niz_fonema` definiraju stručnjaci za pojedini jezik. Oblikovanje fonološkog rječnika je zahtjevan proces te mnogi jezici nemaju dovoljno kvalitetne rječnike tog tipa.

Na prvi pogled može biti nejasno zašto su za osnovnu jedinicu izabrani fonemi, a ne riječi. Razlog tome je što upotreba fonema donosi veću mogućnost generaliziranja. Naime, modeli bazirani na fonemima imaju šansu prepoznati nove, modelu nepoznate riječi jer su te riječi sagrađene od poznatih fonema. Suprotno tome, modeli koji rade s riječima nemaju mogućnost prepoznati riječi koje nisu bile među podacima za treniranje. Akustični modeli koji kao osnovnu jedinicu koriste riječi mogu imati dobre rezultate jedino ako se radi o jako malom ciljanom vokabularu. [15]

Akustični model se može prikazati kao usmjereni težinski graf, u kojem fonemi čine čvorove. Pripadna težina na bridu iz čvora A u B će onda biti vjerojatnost da fonem označen s A prethodi fonemu označenom s B u govoru. Popularna paradigma koja se koristi za učenje tih *tranzicijskih* vjerojatnosti su skriveni Markovljevi modeli (*Hidden Markov Models*, skraćeno HMM). [15]

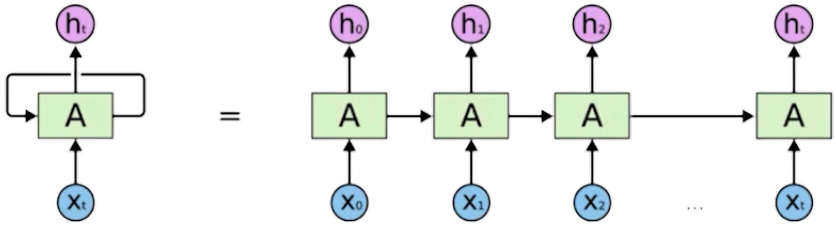


Slika 1.3: Akustični model – HMM

Na slici 1.3 je pojednostavljeni prikaz takvog akustičnog modela. Za svaki i, s $/f_i/$ je označen pripadajući fonem čvora, koji u HMM-u predstavlja jedno od skrivenih stanja. U stvarnim sustavima, stanja mogu odgovarati i nekim drugim jedinicama, ali ipak najčešće se radi s fonemima. Sa z_i je označen vektor akustičnih značajki koji odgovara jednom mogućem okviru. Dalje, s npr. r_1 je označena vjerojatnost da se u stanju $/f_2/$ generirao vektor značajki z_1^n. Vjerojatnosti označene s r se standardno dobivaju iz podataka za treniranje

korištenjem Gaussovih miješanih modela (*Gaussian Mixture Models*, skraćeno GMM). Te vrijednosti se onda koriste za dobivanje *tranzicijskih* vjerojatnosti preko HMM-a, označene s p_i.

Ovakav akustični model je bio standardan sve dok se nisu počele koristiti duboke neuronske mreže (*Deep Neural Networks*, skraćeno DNN). Postoje dva osnovna načina kako se DNN koriste u sklopu akustičnog modela. Tako u nekim ASR sustavima DNN mijenjaju dosadašnju ulogu Gaussovih miješanih modela, te se koriste za dobivanje vjerojatnosti r_i. Takvi sustavi se još nazivaju hibridnim ASR sustavima. Drugi i češći oblik upotrebe je sveobuhvatniji – DNN čine cijeli akustični model (HMM-i se onda ne koriste). Posebice, akustični modeli s povratnom neuronskom mrežom (*Recurrent Neural Network*, skraćeno RNN) su pokazali izrazito dobre rezultate [7].



Slika 1.4: Akustični model – RNN

Slika 1.4 prikazuje rad RNN modela, gdje su s x_i označeni ulazni vektori značajki, a s h_i izlazni vektor s vjerojatnostima za svaki fonem. Može se vidjeti kako se isti model (A) koristi višestruko. To je važno jer se na taj način modelira ovisnost trenutnog fonema o kontekstu u kojem je izgovoren, tj. fonemima koji mu prethode. Postoje razni načini na koje se ovaj model može ostvariti, od kojih je model duge kratkotrajne memorije (*Long Short-Term Memory*, skraćeno LSTM) najzastupljeniji u praksi.

Kao što je već navedeno, rezultat akustičnog modela se intuitivno shvaća kao niz fonema koji odgovaraju ulaznom zvuku, tj. pripadnim značajkama. Međutim, treba naglasiti da se ovdje radi o vjerojatnosnom modelu te da je *pravi* rezultat zapravo niz vektora oblika:

$$\begin{bmatrix} p_{1/f_1/}, & \ldots, & p_{1/f_m/} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} p_{n/f_1/}, & \ldots, & p_{n/f_m/} \end{bmatrix} .$$

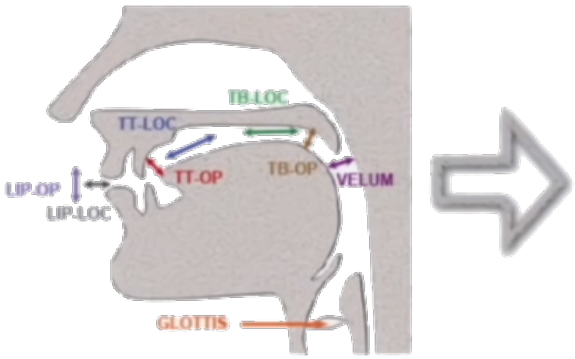
Analogno vrijedi za rezultat jezičnog modela te krajnji rezultat ASR sustava.

1.2.3 Izgovorni model

Sljedeća komponenta ima zadaću *sastaviti* riječi na osnovu manjih jedinica od kojih se sastoje, najčešće fonema. Ovo je jedini model u sklopu ASR sustava koji se ne uči iz podataka za treniranje, već se definira na osnovu fonoloških rječnika. Zato se izgovorni model (*Pronunciation Model*) ponekad u literaturi jednostavno naziva – rječnik (*Lexicon*).

Problematični aspekt ovog dijela je što se stvaran izgovor riječi jako često ne poklapa s izgovorom definiranim u rječniku. Izgovor određene riječi može varirati ovisno o naglasku govornika, brzini govora (neki fonemi se gutaju) ili kontekstu (prethodna i sljedeća riječ utječu na onu u sredini). Primjerice, u engleskom jeziku za svaku riječ postoji prosječno pet mogućih načina izgovora (rastava na foneme).

Budući da ne postoje fonološki rječnici sa svim mogućim varijantama izgovora riječi, ovaj problem se zaobilazi na druge načine. Često se problemu pristupa tako da se naglasak s analize glasa prebaci na analizu samog nastanka tog glasa. Preciznije, analiziraju se stanja artikulatora – pomičnih organa vokalnog trakta koje čovjek nesvjesno koristi kako bi formirao određeni glas. Za svaki od artikulatora je analizom ustanovljen skup od konačno mnogo njegovih stanja. Također, na temelju pozicija/stanja svih artikulatora je točno određen glas koji se tako može proizvesti. Dakle, svaka riječ se može na gotovo jedinstven način opisati preko stanja artikulatora tijekom njena izgovora. Ovo je ključna karakterizacija koja motivira korištenje artikulatura za značajke u izgovornom modelu (slika 1.5). Takvim modelom se elegantnije prikazuju varijabilnosti i nepravilnosti u izgovoru, ali se pokazao složenijim za konstruirati. [15]



Feature	Values
LIP-LOC	protruded, labial, dental
LIP-OP	closed, critical, narrow,
TT-LOC	dental, alveolar, ...
TB-LOC	palatal, velar, uvular, ...
TT-OP, TB-OP	closed, critical, narrow, ...
GLOTTIS	closed, critical, open
VELUM	closed, open

Slika 1.5: Artikulatori glasa

1.2.4 Jezični model

Jezična komponenta dalje analizira rezultate prethodne komponente – riječi. Intuitivno, njena uloga je da od više mogućih sekvenci riječi koje slično zvuče, nađe onu koja semantički i gramatički ima najviše *smisla*. Preciznije, jezični model (*Language Model*) dodjeljuje vjerojatnost za svaku moguću sekvencu riječi. Da bi to mogao, model se uči na velikom skupu podataka za treniranje (tekstovi na određenom jeziku). Primjerice, jezični model treniran za engleski jezik će znati odrediti kako je rečenica „*I will be back soonish*" vjerojatnija od rečenice „*I will be bassoon dish*".

Fundamentalna vrsta jezičnih modela su n-gram modeli. Naime, n-gramom se naziva bilo koja sekvenca od n riječi. N-gram modeli se koriste za procjenu vjerojatnosti sljedeće riječi na temelju $n-1$ prethodnih, ali i za određivanje vjerojatnosti cijele sekvence. Pri odabiru broja n se mora naći kompromis između kvalitete rezultata (veći n) i jednostavnosti modela (manji n). U pravilu se za n biraju vrijednosti ne veće od 5. Osnovna ideja modela je jednostavna: vjerojatnosti se računaju *brojenjem* sekvenci riječi iz velikog jezičnog korpusa. Na uvodnom primjeru, u slučaju četiri-gram modela ($n = 4$) bi to izgledalo ovako:

$$\mathbb{P}(\text{"}back\text{"} \mid \text{"}I\ will\ be\text{"}) \approx \frac{\#(\text{"}I\ will\ be\ back\text{"})}{\#(\text{"}I\ will\ be\text{"})}\,, \tag{1.1}$$

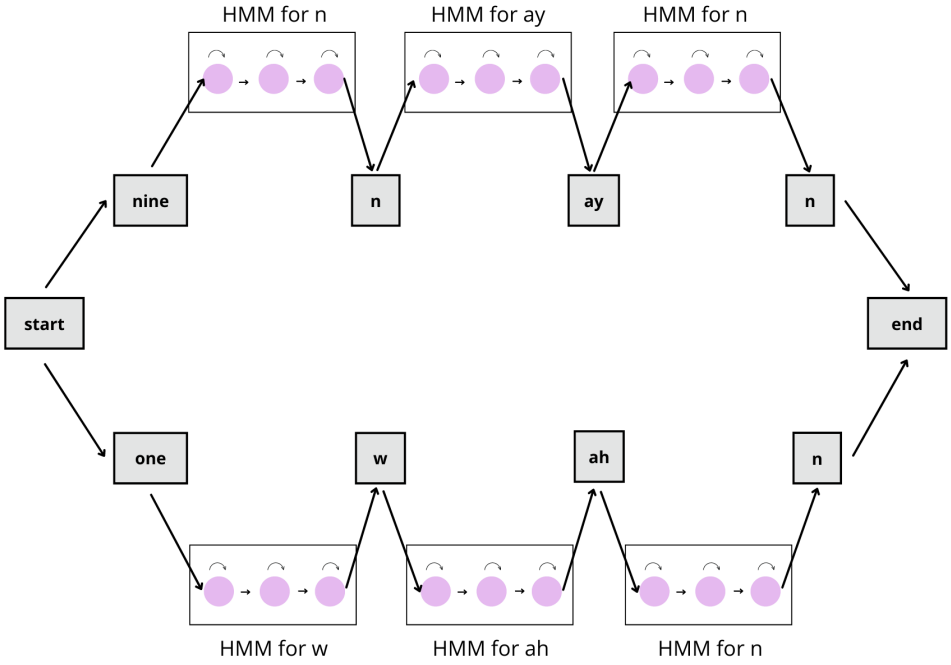
gdje $\#(\textit{izraz})$ označava koliko puta se taj izraz pojavljuje u danom korpusu. Dakle, uvjetna vjerojatnost za sljedeću riječ se aproksimira kao omjer broja sekvenci s tom riječi na kraju te broja sekvenci konteksta (duljine $n-1$) koji joj prethodi. Sada se vjerojatnost cijele sekvence može dobiti kao produkt uvjetnih vjerojatnosti njenih riječi s danim kontekstima. [9]

Ovakva aproksimacija na temelju nekoliko prethodnih riječi se pokazala dovoljnom u praksi. Međutim, problem s n-gram modelima je što ni najveći korpusi teksta i dalje ne sadrže dovoljno podataka. Svaki jezik je promjenjiv te čak i neovisno o tome uvijek će postojati neki n-gram koji nije bio u podacima za treniranje. Za takav n-gram $w_1, \ldots, w_n$ će se pogrešno dobiti $\mathbb{P}(w_n \mid w_1, \ldots, w_{n-1}) = 0$, budući da je brojnik (iz izraza kao u primjeru 1.1) jednak nula. Ovaj problem se rješava tzv. funkcijama za zaglađivanje (*smoothing methods*). Dotične funkcije rezerviraju određenu vjerojatnost za n-grame koji se ne pojavljuju u korpusu. Postoje razne metode koje određuju na koji način se ta rezervirana vrijednost distribuira na nepoznate n-grame (više u [30]).

N-gram modeli se i dalje koriste u nekim ASR sustavima. Uz njih, jezični modeli se ponekad implementiraju kao povratne neuronske mreže (RNN). Takvi modeli su se doduše pokazali sporijima, tako da se u produkciji još uvijek više koriste n-gram modeli, uz eventualno krajnje vrednovanje rezultata (*rescoring*) s RNN modelima. [15]

1.2.5 Dekoder

Dekoder predstavlja zadnju komponentu svakog konvencionalnog ASR sustava. Ona je sabirnog tipa (vidi sliku 1.1) – koristi akustični, izgovorni i jezični model. Naime, na osnovu spomenutih modela se kreira težinski graf koji ih sve povezuje (težine su vjerojatnosti). Problem prepoznavanja govora se tako napokon svodi na problem pretraživanja grafa.



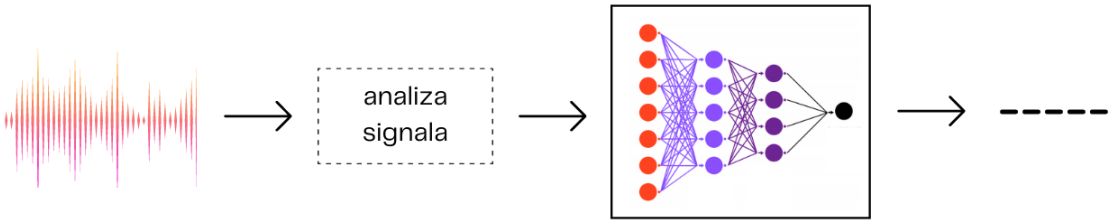
Slika 1.6: Pretraživanje grafa

Na slici 1.6 ilustriran je primjer grafa za sustav koji prepoznaje samo dvije riječi: *nine* i *one*. Svaka od riječi (jezični model) povezana je s pripadnim fonemima (izgovorni model), a svaki fonem je dalje povezan s pripadnim HMM-om (akustični model). Već za ovakav minimalan sustav graf pretraživanja ima nemali broj čvorova. Za ASR sustave s vokabularom od oko 40000 riječi taj broj raste na oko deset milijuna čvorova. Upravo zato se za određivanje najvjerojatnije sekvence riječi moraju koristiti aproksimacijske tehnike za pretraživanje. [15]

1.3 *End-to-End* ASR sustavi

U ovom potpoglavlju se daje uvod u tzv. *End-to-End* sustave za prepoznavanje, te se navode njihovi dosadašnji dosezi u usporedbi s konvencionalnim sustavima. *End-to-End* ili

kraće E2E sustavi predstavljaju najnoviju generaciju sustava za prepoznavanje govora te su trenutno predmetom velikog interesa struke. Glavna ideja E2E sustava je, kao što im i ime kaže, da sekvencu zvučnog ulaza direktno preslikaju u sekvencu odgovarajućih grafema (slova). Na slici 1.7 je prikazana vanjska struktura E2E sustava.



Slika 1.7: Struktura E2E sustava

Ako se gornji prikaz usporedi sa strukturom tipičnog konvencionalnog sustava (slika 1.1), odmah je vidljivo kako se ovdje radi o kompaktnijem sustavu. Naime, E2E sustav prepoznaje govor upotrebom jedinstvenog modela s dubokim neuronskim mrežama, bez odvojenih podsustava. Preciznije, DNN model iz vektora vrijednosti akustičnih značajki (dobivenog klasičnom analizom signala) nalazi odgovarajuću sekvencu grafema. Takav rezultat onda prolazi krajnje vrednovanje (*rescoring*) u kojem se grafemi grupiraju u riječi (obično pojednostavljeni LM).

DNN model unutar E2E sustava zahtijeva ogromnu količinu podataka za treniranje. Ti podaci moraju biti *spareni* – jednim podatkom se smatra određeni zvuk uz pripadni tekstualni transkript. Jezični model u konvencionalnim sustavima je kao podatke za treniranje koristio čisti tekst, koji se lako može prikupiti s interneta. S druge strane, *sparene* podatke je znatno teže prikupiti u potrebnoj količini pa se oni na razne načine umjetno generiraju. Iako se navedeno može smatrati manom E2E sustava, njihova velika prednost nad konvencionalnim sustavima je što ne koriste izgovorni model. Stoga, E2E sustavi ne ovise o kvaliteti i uopće postojanju fonološkog rječnika za određeni jezik. Dodatno poboljšanje vezano za treniranje modela je što E2E modeli pri treniranju koriste kriterij optimalnosti koji se poklapa s krajnjim kriterijem za vrednovanje sustava (WER[1]). Kao što je ranije opisano, u konvencionalnim sustavima to nije slučaj – svaki sastavni model se trenira za-

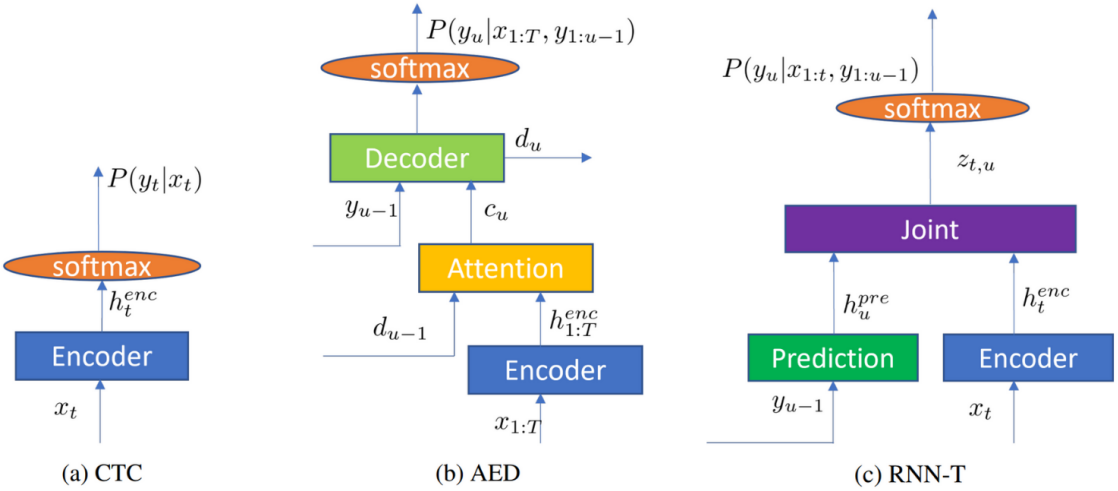
[1] WER je oznaka za mjeru netočno prepoznatih riječi (*word error rate*). Razlikuju se tri vrste grešaka koje sustav može napraviti: greške supstitucije, greške brisanja te greške umetanja. WER se računa tako da uzima sve vrste grešaka u obzir:

$$WER = \frac{N_{sub} + N_{ins} + N_{del}}{N_{ref}},$$

gdje je s N_{ref} označen broj riječi u referentnom, pravom transkriptu.

sebno i to po specijaliziranom kriteriju. Nema garancije da ti svi kriteriji zajedno u nekim posebnim slučajevima neće odudarati od krajnjeg kriterija optimalnosti.

DNN modeli za E2E sustave se ubrzano razvijaju i nadograđuju. Među puno različitih rješenja, izdvajaju se tri najznačajnija tipa modela. Njihova okvirna struktura te izlazne vrijednosti prikazane su na slici 1.8. Detaljna analiza ovih modela nadilazi okvir ovog rada, ali se može naći u [19].



(a) CTC (b) AED (c) RNN-T

Slika 1.8: Tipovi DNN modela u E2E sustavima

Svaki od modela započinje komponentom *Encoder*. Ta komponenta je zadužena za reprezentaciju akustičnih značajki na većoj razini apstrakcije, s kojom radi sam model.

Connectionist Temporal Classification ili CTC je model dizajniran da daje preslikavanje: ulazni zvučni okvir → izlazni znak (*label*). Uz znakove abecede, u skup mogućih izlaznih znakova je dodan znak <b> koji označava prazninu (*blank*). Time je omogućeno da se za svaki zvučni okvir dobije jedan izlazni znak (slovo ili <b>), iako je broj ulaznih okvira uvijek veći od broja izlaznih slova.

CTC modeli su bili prvi koji su se proučavali te su zadovoljavajućim rezultatima motivirali daljnji razvoj E2E sustava. Međutim, takvi modeli se danas kritiziraju zbog pretpostavke o uvjetnoj nezavisnosti susjednih zvučnih okvira, na kojoj počiva račun za dobivanje izlazne vjerojatnosti (slika 1.8).

Attention-based Encoder-Decoder ili AED predstavlja proširenje CTC modela. Spomenuti problem nezavisnosti zvučnih okvira se rješava dodavanjem dvaju međukomponenata.

Komponenta *Attention* identificira okvire koji su relevantni za trenutni izlaz, dok komponenta *Decoder* daje procjenu izlaza kao funkciju koja prima i prethodno prepoznate znakove.

Oba navedena tipa modela imaju veliki nedostatak. Naime, budući da za računanje izlaznih vjerojatnosti sa slike 1.8 obje tehnike koriste podatke o cijeloj ulaznoj sekvenci, takvi modeli nisu pogodni za prepoznavanje govora uživo (*live-stream*). Taj nedostatak se često pokušava izbjeći umjetnom podjelom ulaznog zvuka na fragmente. Postoje razne složene strategije za rastav ulaznog zvuka, ali u praksi se ipak nisu pokazale kao dovoljno dobra rješenja.

RNN Transducer ili RNN-T je model u kojem se izlazna vjerojatnost računa isključivo pomoću podataka o okvirima koji prethode trenutnom. Stoga, RNN-T pruža prirodan način za prepoznavanje govora uživo, zbog čega zasada spada među najpopularnije E2E modele.

Sljedeća tablica ([19]) daje uvid u performanse najnovijih E2E sustava. Za svaki sustav navedena je vrsta DNN modela koju koristi. Usto, navedena je i vrsta modela koja se koristi za generiranje akustičnih značajki visoke razine apstrakcije (više o načinima realizacije komponente *Encoder* u [19]). Iz tablice se može vidjeti znatan napredak u zadnjih nekoliko godina, kako na *čistim*, prethodno obrađenim podacima, tako i na *sirovim* podacima za testiranje. Također, mogu se uočiti i dva jasna trenda; prelazak s CTC na AED/RNN-T vrste modela, te prelazak s LSTM na *Transformer* vrstu komponente *Encoder*. Još treba istaknuti kako se u donjoj tablici ne navodi veličina i propusnost modela. Budući da ti čimbenici posebno utječu na konačnu kvalitetu, vrednovanje modela isključivo na temelju mjere *WER* treba uzeti s rezervom.

Tablica 1.1: Rezultati reprezentativnih *non-streaming* E2E sustava za korpus Librispeech[2]

work with key technologies	year	model	encoder	test-clean/other WER
Deep Speech 2: more labeled data, curriculum learning [325]	2016	CTC	bi-RNN	5.3/13.2
policy learning, joint training [326]	2018	CTC	CNN+bi-GRU	5.4/14.7
Shallow fusion, BPE, and pre-training [33]	2018	AED	BLSTM	3.8/12.8
ESPRESSO recipe: lookahead word LM, EOS thresholding [258]	2019	AED	CNN+BLSTM	2.8/8.7
SpecAugment [278]	2019	AED	CNN+BLSTM	2.5/5.8
ESPnet recipe: SpecAugment, dropout [86]	2019	AED	Transformer	2.6/5.7
Semantic mask [327]	2019	AED	Transformer	2.1/4.9
Transformer-T, SpecAugment [89]	2020	RNN-T	Transformer	2.0/4.6
Conformer-T, SpecAugment [97]	2020	RNN-T	Conformer	1.9/3.9
wav2vec 2.0: SSL with unlabeled data, DataAugment [28]	2020	CTC	Transformer	1.8/3.3
internal LM prior correction, EOS modeling[328]	2021	RNN-T	BLSTM	2.2/5.6
w2v-BERT: SSL with unlabeled data, SpecAugment [329]	2021	RNN-T	Conformer	1.4/2.5

[2] Librispeech [23] je korpus koji sadrži 1000 sati govora sakupljenog iz javno dostupnih audio knjiga.

Na kraju, treba istaknuti kako su E2E sustavi nedavno nadmašili točnost konvencionalnih sustava, uz daleko *manje* modele. Međutim, koliko god su se E2E sustavi pokazali lakšima za optimizirati, toliko su i teži za interpretirati. Postoji nekolicina problema koje je u starim sustavima bilo jednostavno za locirati i otkloniti, a kojima se u sklopu E2E sustava treba pristupiti na potpuno drugi način. Dakle, iako E2E sustavi imaju bolje rezultate u smislu osnovne mjere točnosti, dok se ne nadoknade sve ostale manjkavosti oni neće zamijeniti konvencionalne sustave u produkciji.

U sljedećem poglavlju se detaljno opisuje kako se preko biblioteke SpeechRecognition mogu koristiti neki od poznatih sustava za prepoznavanje govora.

Poglavlje 2

Biblioteka SpeechRecognition

Prilikom započinjanja Python projekta koji uključuje prepoznavanje govora u nekom obliku, programeri se susreću sa sljedećim izborom biblioteka: apiai, assemblyai, google-cloud-speech, pocketsphinx, SpeechRecognition, watson-developer-cloud i wit. Većina ovih biblioteka je specijalizirana za prepoznavanje govora korištenjem točno određenog aplikacijskog programskog sučelja (API-ja), po kojemu nose i nazive. Biblioteka SpeechRecognition se među navedenima ističe kao jedina koja nudi podršku za nekoliko (čak sedam) API-ja. Navedeno nije utjecalo na složenost sučelja, dapače, ova biblioteka se ističe i u smislu jednostavnosti korištenja.

SpeechRecognition [35] je najpopularnija Python biblioteka za prepoznavanje govora. Biblioteka je nastala kao projekt otvorenog koda (*open source*) i dostupna je za korištenje besplatno. U nastavku se daje detaljan opis svih funkcionalnosti ove biblioteke te osvrt na njen daljnji razvoj.

2.1 Instalacija i početak

Biblioteka SpeechRecognition je kompatibilna s Python verzijama 2.6, 2.7, 3.3 i novijima. Posljednja stabilna verzija biblioteke je SpeechRecognition 3.8.1 iz 2017. godine. Biblioteka je dostupna na PyPI-u (*Python Package Index*) [36], glavnom repozitoriju za Python pakete. Prije instalacije potrebno je instalirati sve pakete o kojima ova biblioteka ovisi, a oni su navedeni u datoteci *requirements.txt*. Sama instalacija se može provesti na dva načina. Najlakše je biblioteku instalirati korištenjem pip-a (*Package Installer for Python*), programa za instalaciju Python paketa, naredbom `# pip install SpeechRecognition`. Alternativno, izvorni kod biblioteke se može preuzeti s PyPI-a te se onda instalacija pokreće naredbom `# python setup.py install`.

Dokumentacija biblioteke je sažeta, ali kvalitetna. Glavnu dokumentaciju predstavlja datoteka *reference.rst* koja je generirana iz docstring[1] dokumentacije izvornog koda. Dokumentacija je napisana u obliku liste definicija s opisima svih klasa i funkcija biblioteke. Za složenije objekte je uz opis dan i isječak koda koji demonstrira korištenje. Detaljniji primjeri korištenja svih važnijih funkcionalnosti paketa se mogu naći u direktoriju *examples*. Nadalje, najčešći problemi i pitanja korisnika su opširno komentirani u naslovnom opisu paketa. Ako se korisnik nađe pred problemom koji nije referenciran u službenoj dokumentaciji, velika je vjerojatnost da je isti bio predmetom rasprave u nekom od otvorenih problema (*issues*) na GitHub-u, pa se tamo mogu naći alternativna, ili barem kratkoročna, rješenja. Budući da ova biblioteka ima veliku zajednicu korisnika, svaki novi problem ili prijedlog rješenja je obično obrađen u kratkom roku.

2.2 Funkcionalnost

Osnovna funkcionalnost biblioteke SpeechRecognition je prepoznavanje govora, tj. pretvaranje govora u tekst. Dodatno, ova biblioteka sadrži tipove specificirane za jednostavnu manipulaciju izvorom govora – bila to audio datoteka ili mikrofon. Za minimalan radni primjer transkribiranja audio datoteke dovoljno je samo sedam linija koda.

```
1 import speech_recognition as sr
2
3 recognizer = sr.Recognizer()
4 audioFile = sr.AudioFile('example_file.wav')
5
6 with audioFile as source:
7     audio = recognizer.record(source)
8
9 result = recognizer.recognize_google(audio)
```

Primjer 2.1: Prepoznavanje govora iz audio datoteke

Po uspješnom izvršavanju gornjeg koda, varijabla `result` će sadržavati transkribirani tekst iz datoteke *example_file.wav*. Na ovom jednostavnom primjeru mogu se upoznati sve glavne klase biblioteke: `AudioFile`, `AudioData` i `Recognizer`. Instanca klase

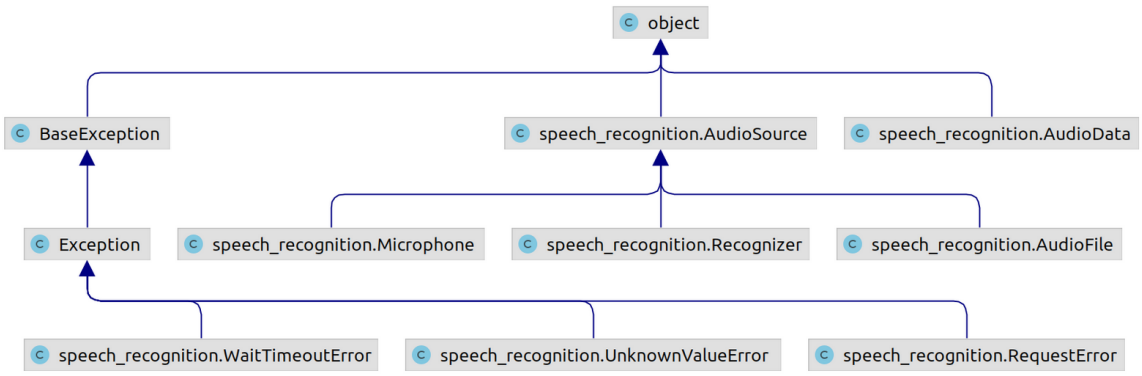
[1] Python docstring su string literali koji se pojavljuju odmah nakon definicije funkcije, metode, klase ili modula. Docstring paketa se nalazi u datoteci *__init__.py* i dodatno opisuje sve sadržane module i potpakete. Na temelju docstring-a se pomoću raznih alata može generirati dokumentacija u nekom od tekstualnih formata.

`AudioFile` upravlja datotekom *example_file.wav*, odnosno omogućuje otvaranje i čitanje sadržaja ove datoteke. U liniji 7 se iz datoteke čita *sirovi* sadržaj tipa `AudioData` iz kojeg se napokon vrši prepoznavanje u liniji 9. Za sve postavke i samo prepoznavanje govora koristi se klasa `Recognizer`.

Iako se ovdje radi o minimalnom primjeru, struktura koda i odnosi među klasama se ne mijenjaju ni kada je riječ o naprednijim upotrebama. Dakle, svako prepoznavanje govora s bibliotekom SpeechRecognition provodi se u tri osnovna koraka:

1. Definiranje izvora zvuka.
2. Dohvaćanje *sirovih* audio podataka iz izvora zvuka.
3. Prepoznavanje govora na temelju audio podataka.

U prvom koraku se radi o instanciranju klase `AudioFile` ili `Microphone`, dok u daljnjim koracima glavnu ulogu obavlja instanca klase `Recognizer`.

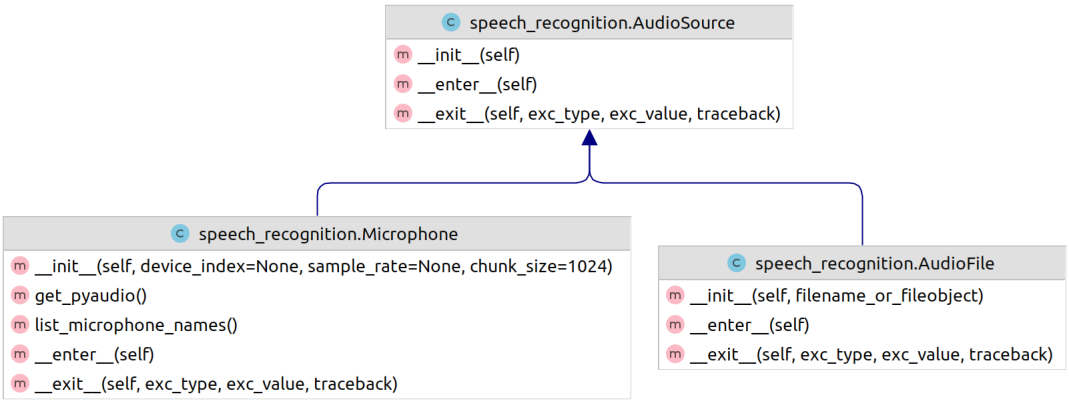


Dijagram 2.1: Klase biblioteke SpeechRecognition

Na slici 2.1 je prikazan pojednostavljeni dijagram klasa biblioteke SpeechRecognition. Osim već spomenutih klasa, na dijagramu su označene; klasa `AudioSource` – apstraktna nadklasa svih klasa koje predstavljaju ili rade s nekim izvorom zvuka, te klasa `Microphone` – analogon klase `AudioFile`, zadužena za upravljanje zvukom koji dolazi s mikrofona. Biblioteka još sadrži klase koje predstavljaju specifične iznimke do kojih može doći prilikom prepoznavanja, a to su: `WaitTimeoutError`, `UnknownValueError` i `RequestError`.

2.2.1 Izvor zvuka

Početna točka pri transkripciji govora je određeni zvuk. Biblioteka SpeechRecognition kao ulaz podržava zvuk iz audio datoteke ili mikrofona. Ulazne jedinice se u kodu definiraju instanciranjem neke od `AudioSource` potklasa.



Dijagram 2.2: AudioSource klasa s potklasama

Na dijagramu 2.2 su prikazane klase `AudioFile` i `Microphone` sa svim pripadajućim funkcijama. Sve tri klase su klase tipa *Context Manager*[2] pa se njihovim instancama pristupa uz ključnu riječ `with`.

```
1 with audio_source_instance as source:
2     pass
```

Primjer 2.2: *Context Manager*

2.2.1.1 Audio datoteka

Kao što je demonstrirano u primjeru 2.1, upravljanje audio datotekom kao izvorom zvuka se obavlja preko klase `AudioFile`. Iz dijagrama 2.2 je vidljivo da je za instanciranje ove klase dovoljno specificirati put do audio datoteke ili samu datoteku u objektu tipa `io.BytesIO`. Nakon instanciranja klase `AudioFile`, moguće je dohvatiti trajanje audio

[2] *Context Manager* su Python klase koje omogućuju pojednostavljeno upravljanje resursima unutar bloka. Sva potrebna logika se definira u `__enter__` i `__exit__` funkcijama. Instancama ovakve klase se onda pristupa uz ključne riječi `with` i `as`, a odgovarajuća logika se provodi automatski. [16]

datoteke preko svojstva `DURATION`. Svojstvo se odnosi na duljinu datoteke u sekundama i dostupno je samo unutar konteksta, odnosno nakon ključne riječi `with`.

Važno je naglasiti kako audio datoteka mora biti u jednom od sljedećih formata:

- WAV (*Waveform Audio File Format*) – isključivo PCM i LPCM [3],
- AIFF (*Audio Interchange File Format*) – dopušta se i AIFFC [4],
- FLAC (*Free Lossless Audio Codec*) – isključivo nativni FLAC [5].

Svaki pristup objektu `AudioFile` nepodržanog formata rezultira iznimkom `ValueError`.

2.2.1.2 Mikrofon

Upravljanje zvukom koji dolazi iz mikrofona se obavlja preko klase `Microphone`. Za rad s mikrofonom postoji dodatan zahtjev – potrebno je instalirati biblioteku PyAudio minimalne verzije 0.2.11. Biblioteka je također dostupna za instaliranje preko pip-a [24], a koristi se isključivo interno u implementaciji biblioteke SpeechRecognition. Ukoliko biblioteka PyAudio nije ispravno instalirana, pokušaj rada s klasom `Microphone` će rezultirati iznimkom `AttributeError`.

Iz dijagrama 2.2 se vidi da je za instanciranje klase `Microphone` potrebno predati sljedeća tri argumenta.

- Indeks uređaja (`device_index`) je broj koji određuje koji mikrofon će se koristiti za slušanje. Lista svih dostupnih uređaja se može dobiti pozivom statičke funkcije `list_microphones()`.

```
>>> speech_recognition.Microphone.list_microphone_names()
['HDA Intel PCH: ALC272 Analog (hw:0,0)', 'front',
 'sysdefault', 'surround40', 'hdmi', 'Gaming Headset']
```

Primjer 2.3: Dostupni mikrofoni

[3] PCM (*Pulse-code modulation*) je metoda reprezentacije analognog signala. U uniformnim vremenskim intervalima, bilježi se jedna od diskretnih vrijednosti koja je najbliža trenutnoj amplitudi signala. Audio signal je onda predstavljen kao niz takvih vrijednosti. LPCM (*Linear pulse-code modulation*) je podvrsta PCM metode u kojoj su moguće vrijednosti niza linearno uniformne (suprotno od PCM metode, gdje se vrijednosti određuju kroz funkciju amplitude). [3]

[4] AIFFC se odnosi na komprimiranu varijantu AIFF formata. [2]

[5] Ne dopušta se FLAC komprimiran Ogg omotačem.

Indeks uređaja se odnosi na redni broj u listi `list_microphones()`. Sada se primjer 2.4 može proširiti tako da se definira objekt koji će koristiti npr. „sysdefault" mikrofon. „sysdefault" je drugi uređaj u listi pa se kao argument predaje njegov indeks koji iznosi 1.

```
>>> microphone = speech_recognition.Microphone(device_index=1)
```

Primjer 2.4: *Default* mikrofon

- Frekvencija (`sample_rate`) je broj uzoraka audio signala u sekundi (Hertz). Više vrijednosti donose bolju kvalitetu, ali i veću propusnost audio snimke, pa posljedično prepoznavanje govora iz takvih snimki traje duže.

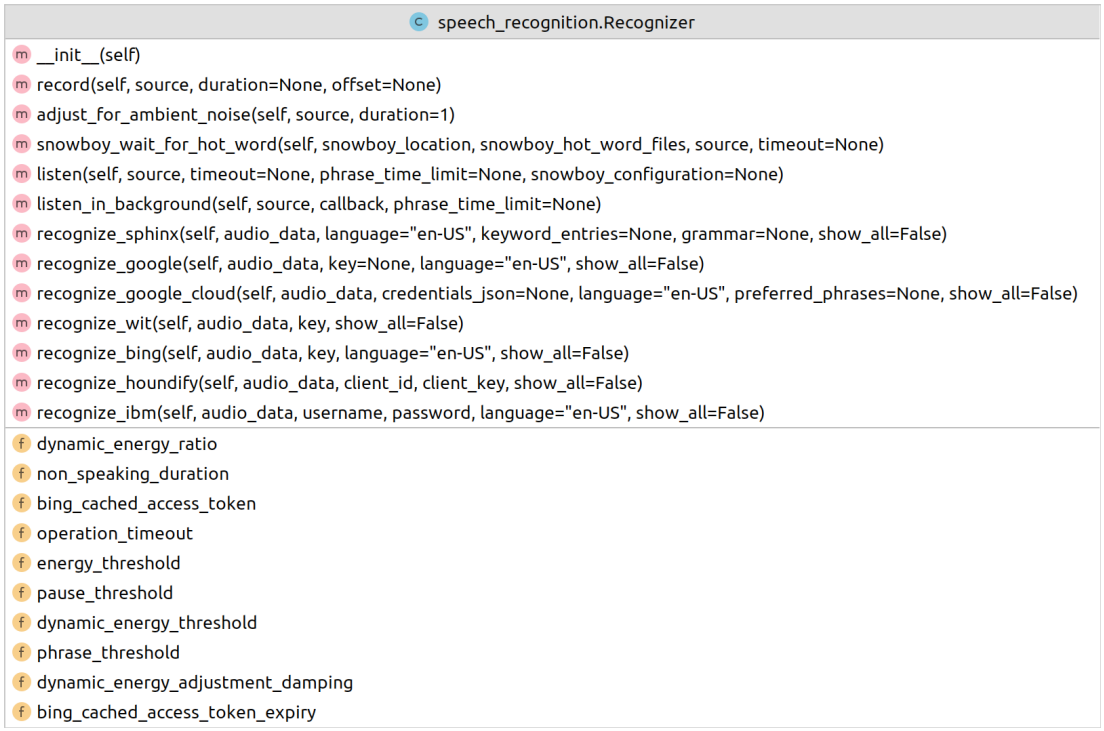
- Veličina uzorka (`chunk_size`) se odnosi na broj okvira u koje se dijele potencijalno vrlo dugi signali. Veće vrijednosti pomažu u izbjegavanju poremećaja uzrokovanih pozadinskom bukom, ali smanjuju osjetljivost pri detekciji stvarnog govora.

SpeechRecognition dokumentacija preporuča da se za posljednja dva argumenta, koji se tiču frekvencije i veličine uzorka, ostave inicijalne vrijednosti.

2.2.2 Obrada zvuka

Nakon što je definiran točan izvor zvuka, zvuk iz mikrofona ili audio datoteke je potrebno na neki način obraditi da bismo dohvatili *sirove* audio podatke. Takvi podaci predstavljaju stvarnu početnu točku za prepoznavanje govora. Obrada zvuka se zapravo sastoji od dva koraka. Prije samog početka, moguće je odrediti način na koji će se rukovati pozadinskom bukom. Ovaj korak nije obavezan, ali može uvelike doprinijeti kvaliteti krajnjeg rezultata. Biblioteka SpeechRecognition nudi jedinstven skup opcija za upravljanje pozadinskom bukom neovisno o izvoru zvuka. Drugi korak se odnosi na samu obradu izvora zvuka – slušanje ulaza s mikrofona ili snimanje sadržaja postojeće audio datoteke. Iako se ovaj korak razlikuje ovisno o izvoru zvuka, ključna logika te čak definicije odgovarajućih funkcija u oba slučaja prate isti kalup.

Funkcionalnost vezana za sve korake obrade zvuka, te naknadno i za prepoznavanje govora, je sadržana u klasi `Recognizer`. Dijagram 2.3 prikazuje sve funkcije (*methods*) i svojstva (*fields*) klase `Recognizer`. Neki od njih se koriste isključivo interno u implementaciji paketa, dok ih je većina javno dostupna.



Dijagram 2.3: Klasa Recognizer

2.2.2.1 Pozadinska buka

Ispravno razlikovanje stvarnog govora od pozadinske buke je jedna od glavnih kritičnih točaka u procesu prepoznavanja govora. Biblioteka SpeechRecognition kroz klasu `Recognizer` nudi nekoliko opcija koje određuju način upravljanja, tj. detekcije pozadinske buke pri obradi zvuka. Usto, klasa `Recognizer` ima specificirane inicijalne vrijednosti svih svojstava vezanih za pozadinsku buku. Te vrijednosti su određene tako da daju dobre rezultate u uvjetima s beznačajnom pozadinskom bukom, stoga je ponekad moguće preskočiti njihovu dodatnu optimizaciju. Ipak, u većini slučajeva, postavljanjem ovih svojstava na vrijednosti prilagođene trenutnoj upotrebi će poboljšati krajnju kvalitetu transkripta.

Postoje četiri svojstva klase `Recognizer` koja se izravno tiču pozadinske buke.

- `energy_threshold` je svojstvo tipa `float` koje predstavlja energetski prag te se odnosi na percipiranu glasnoću zvuka. Vrijednosti ispod ovog praga smatraju se tišinom, a vrijednosti iznad govorom. Stvarna vrijednost energetskog praga dodatno ovisi o osjetljivosti mikrofona, odnosno kvaliteti audio datoteke. Tipične vrijednosti

za tiho okruženje su od 0 do 100, dok tipične vrijednosti za govor iznose između 150 i 3500. Inicijalna vrijednost ovog svojstva je postavljena na 300.

Logika za namještanje ove vrijednosti najviše ovisi o okruženju govornika. Ako su primijećeni pokušaji prepoznavanja riječi čak i kad nema govora, vrijednost treba povećati. Na primjer, osjetljiviji mikrofoni ili mikrofoni u glasnijim sobama mogu imati energentski prag i do 4000. U suprotnom, ako se izgovorene riječi ne prepoznaju dobro ili uopće, vrijednost ovog svojstva treba smanjiti.

- `dynamic_energy_threshold` je svojstvo tipa `bool` koje određuje hoće li se energetski prag dinamički prilagođavati tijekom slušanja. Vrijednost je inicijalno `True`. Ovakva automatska prilagodba je pogodna za okruženja u kojima je razina pozadinske buke nepredvidiva, što zapravo čini većinu slučajeva. Iznimno, ako se radi o izrazito kontroliranim uvjetima, moguće je postići bolje rezultate ukoliko se dinamička opcija isključi, postavljanjem ove vrijednosti na `False`. U tom slučaju, potrebno je svojstvo `energy_threshold` postaviti na točno očekivanu vrijednost. Općenito, dobra početna vrijednost svojstva `energy_threshold` omogućuje da se dinamičkim podešavanjem brže dostigne optimalna vrijednost.

Preostala dva svojstva imaju utjecaja samo ako je uključena opcija dinamičkog podešavanja energetskog praga, tj. `dynamic_energy_threshold = True`.

- `dynamic_energy_adjustment_ratio` je svojstvo tipa `float` koje određuje za koliko minimalno puta govor mora biti glasniji od pozadinske buke da bi bio detektiran. Inicijalna vrijednost je 1.5 – dakle, uzima se da govor mora biti barem 1.5 puta glasniji od pozadinske buke. Preniske vrijednosti dovode do više *false positive*[6] prepoznavanja, dok previsoke vrijednosti dovode do gubitka dijela govora. Ovu vrijednost se zato ne preporuča znatno mijenjati.

- `dynamic_energy_adjustment_damping` je svojstvo tipa `float` koje određuje koliki će postotak energetskog praga biti utvrđen nakon jedne sekunde dinamičke prilagodbe. Inicijalna vrijednost ovog svojstva je 0.15. Niže vrijednosti će omogućiti brže određivanje energetskog praga, ali će postojati veća vjerojatnost da određene fraze budu propuštene. Navedeno se najviše odnosi na fraze kojima se tijekom izgovora polako mijenja glasnoća. Ovu vrijednost se zato također ne preporuča mijenjati.

[6] *False positive* prepoznavanja se odnose na riječi/fraze koje su prepoznate iz dijela snimke u kojem uopće nema govora. Dakle, te riječi u finalnom transkriptu ne bi trebalo zamijeniti drugim riječima, već u potpunosti izbrisati.

Među navedenim svojstvima, svojstvo `energy_threshold`, tj. energetski prag, igra najveću ulogu pri detekciji pozadinske buke. Ostala svojstva su zapravo samo pomoćne vrijednosti koje izravno ili neizravno određuju način na koji će se energetski prag kalibrirati. Kada se napokon vrijednost praga stabilizira, ista postaje stvarno mjerilo na temelju kojeg se razlikuje govor od pozadinske buke.

Određivanje optimalne vrijednosti energetskog praga općenito predstavlja složen problem. Naime, ta vrijednost ovisi o puno faktora i iako je većina tih ovisnosti poznata, izostaje način za manualno određivanje optimalne vrijednost bez suvišnog testiranja. Na primjer, moguće je da će stvarni energetski prag za govor s izrazito glasnim okruženjem ispasti niži od prosječno glasnog govora koji je snimljen izrazito osjetljivim mikrofonom. Iz navedenih razloga, kada je riječ o energetskom pragu, najbolje se osloniti na dinamičko podešavanje koje biblioteka nudi.

Kao što je već spomenuto, opcija dinamičkog podešavanja je inicijalno uključena. Iako će se uz ovu opciju energetski prag automatski podešavati, za potpunu prilagodbu je ipak potrebno određeno vrijeme, tako da je još uvijek moguće dobiti *false positive* vrijednosti u periodu prilagodbe. Ovakve greške se mogu minimizirati određenim manipulacijama početne vrijednosti energetskog praga. Posebno se izlažu dva pristupa.

Ponekad je dovoljno postaviti visoku početnu vrijednost energetskog praga, tako da je prag uvijek iznad razine pozadinske buke. S vremenom se onda prag automatski smanji na povoljniju vrijednost. Postavljanje energetskog praga na 4000 se u praksi pokazala kao dobra početna točka [35].

Drugi i preporučeni način je korištenje funkcije `adjust_for_ambient_noise` (vidi dijagram 2.3) za kalibriranje praga na dobru početnu vrijednost. Funkcija kao argumente prima instancu `AudioSource` te `float` vrijednost. Prvi argument se očekivano odnosi na objekt koji predstavlja izvor zvuka. Drugi argument određuje maksimalnu duljinu intervala u sekundama na temelju kojeg će se odrediti energetski prag. Ovaj argument je inicijalno postavljen na 1 sekundu, a vrijednost bi trebala biti najmanje 0.5 da bi interval predstavljao reprezentativan uzorak. Način rada ove funkcije je sljedeći: ona uzima interval definirane duljine s početka danog audio izvora te na temelju njega određuje energetski prag pozadinske buke. Važno je istaknuti da se ova funkcija treba koristiti na dijelovima zvuka u kojima nema govora. U suprotnom, pokušaj kalibracije se zaustavlja čim je detektiran bilo kakav govor. Također, treba imati na umu da se interval zvuka koji se koristi u sklopu funkcije `adjust_for_ambient_noise` smatra *obrađenim*, tj. taj dio *streama* se neće uzeti u obzir pri naknadnom dohvaćanju audio podataka [6]. To je jedan od glavnih razloga zašto je ponekad poželjno duljinu intervala skratiti na vrijeme kraće od inicijalne jedne sekunde, iako se time smanjuju šanse za pronalazak dobre početne vrijednosti.

2.2.2.2 Dohvaćanje audio podataka

Dohvaćanje audio podataka iz izvora zvuka je drugi nužan korak u procesu prepoznavanja govora. Audio podacima se smatra *stream sirovih* podataka koji opisuje zvuk. Kao što je već naglašeno, u ovom koraku važnu ulogu igra točan tip izvora koji se obrađuje; pa se tako u slučaju mikrofona govori o *slušanju* govora, dok je u slučaju audio datoteke riječ o *snimanju* govora.

Snimanje audio datoteke

U ovom odlomku je pobliže opisano kako se audio podaci mogu dohvatiti iz audio datoteke. Prije svega, pretpostavlja se da postoji objekt `AudioFile` (definiran kao u odlomku 2.2.1.1), koji predstavlja neku audio datoteku. Također, definiran je i objekt `Recognizer` koji će se koristiti za dohvaćanje audio podataka. Pripadna funkcija zadužena za dohvaćanje audio podataka je funkcija `record`, čija je definicija navedena ispod.

```
record(self, source, duration=None, offset=None)
```

Funkcija prima tri argumenta, `source`, `duration` i `offset`, a vraća audio podatke kao instancu `AudioData` (vidi dijagram 2.1). Samo je prvi među argumentima obavezan. Funkcija `record` snima sadržaj danog `source` objekta od početka do kraja te vraća obrađene audio podatke. Ako su dani argumenti `offset` i `duration`, funkcija `record` će obraditi točan interval koji je njima određen. Argument `offset` označava početak intervala od interesa, dok argument `duration` definira koliko isti traje. Oba argumenta su iskazana u sekundama. Sada je moguće proširiti uvodni primjer 2.1.

```
1  import speech_recognition as sr
2
3  recognizer = sr.Recognizer()
4  audioFile = sr.AudioFile('example_file.wav')
5
6  with audioFile as source:
7      recognizer.adjust_for_ambient_noise(source, 2)
8      audio = recognizer.record(source, 5, 10)
9
10 result = recognizer.recognize_google(audio)
```

Primjer 2.5: Snimanje zvuka iz audio datoteke

Promjene u kodu su napravljene samo unutar *konteksta* audio datoteke. U liniji 7 objekt `recognizer` čita prve dvije sekunde audio datoteke te kalibrira razinu pozadinske buke. U sljedećoj liniji objekt `recognizer` čita audio datoteku počevši od pete sekunde, u trajanju od deset sekundi, te napokon vraća pripadne audio podatke intervala. Važno je primijetiti da se ovdje čita od pete sekunde do sada nepročitanog dijela datoteke. Dakle, ovim isječkom koda se zapravo dohvaćaju audio podaci od sedme do sedamnaeste sekunde originalnog audio zapisa.

Slušanje s mikrofona

Način dohvaćanja audio podataka s mikrofona se idejno ne razlikuje od prethodno opisanog postupka za obradu audio datoteke. Međutim, budući da se radi o audio zapisu u nastajanju, postoje određeni dodatni parametri koji se mogu postaviti.

Prvo se pretpostavlja da su definirani objekti `Microphone` i `Recognizer` koji redom predstavljaju izvor zvuka te objekt za njegovu obradu. Pripadna funkcija za slušanje govora s mikrofona je funkcija `listen`. Radi se o analogonu funkcije `record`.

```
listen(self, source, timeout=None, phrase_time_limit=None, snowboy_configuration=None)
```

Funkcija `listen` snima govor s mikrofona (`source`) te vraća audio podatke u obliku objekta `AudioData`. Snimanje započinje čim je registriran neki govor, točnije čim mikrofon registrira zvuk čija energetska razina prelazi energetski prag (`energy_threshold`). Snimanje se zaustavlja ako duže vremena nema registriranog govora ili je ulaz s mikrofona potpuno prekinut. svojstvom `pause_threshold` klase `Recognizer` je određeno nakon koliko tišine se snimanje zaustavlja. Ova vrijednost je inicijalno postavljena na 0.8 sekundi te se može mijenjati. Niže vrijednosti će umanjiti vrijeme nepotrebnog slušanja, ali isto tako mogu dovesti do preranog prekida sporijih govornika. Završni period tišine se automatski isključuje iz povratnih audio podataka.

Funkcija `listen` nudi mogućnost postavljanja dodatnih uvjeta na zaustavljanje snimanja. Tako se parametrom `timeout` može odrediti maksimalan period čekanja – od poziva funkcije do početka govora. Ako u tom roku nije registriran nikakav govor, funkcija `listen` će završiti s iznimkom `WaitTimeoutError`. Sljedećim parametrom, `phrase_time_limit`, određuje se maksimalno trajanje govora. Snimanje govora će se nakon tog vremena prekinuti te će funkcija vratiti audio podatke za period do prekida. Ovdje se ne uzima u obzir period prije početka govora. Na kraju se može primijetiti da ukoliko su obje ove vrijednosti zadane, cijeli proces snimanja će trajati maksimalno `timeout` + `phrase_time_limit` sekundi.

Preko posljednjeg parametra `snowboy_configuration` omogućena je integracija s alatom Snowboy [18], popularnim izvanmrežnim alatom za detekciju riječi okidača (*hotwords*). Ako je taj parametar zadan, funkcija `listen` će pauzirati sve dok Snowboy ne detektira riječ okidač, nakon čega će normalno nastaviti s radom.

Kako točno uključiti Snowboy alat? Budući da se radi o vanjskom alatu, potrebno ga je prvo instalirati i dodati u projekt. Najnovija verzija (1.3.0) zasad nije dostupna na PyPI-u pa se preporuča alat instalirati iz izvornog koda, i to po sljedećim koracima:

1. Klonirati projekt s GitHub-a [18].
2. Otvoriti projekt u terminalu te pokrenuti naredbu `# python3 setup.py build` .
3. U istom terminalnu pokrenuti naredbu `# python3 setup.py install` .

Funkcija `listen` očekuje Snowboy parametar tipa `Tuple[str, Iterable[str]]` . Kroz prvi element se predaje put do Snowboy direktorija. Drugim elementom se predaje lista puteva do datoteka koje predstavljaju modele. Svaki model se odnosi na jednu riječ/frazu koja će biti okidač. Modeli mogu biti univerzalni ili privatni. Univerzalni modeli su datoteke s ekstenzijom `.umdl` i oni rade jednako dobro neovisno o govorniku. S druge strane, privatni modeli imaju ekstenziju `.pmdl` te su istrenirani da prepoznaju samo jednog govornika.

Još treba napomenuti kako je s 2020. godinom ukinuta službena podrška za Snowboy. U sklopu toga je ugašena i pripadna mrežna stranica, koja je između ostalog nudila jednostavno sučelje za izradu vlastitog privatnog modela (usluga izrade univerzalnog modela se naplaćivala). Usprkos tome, zbog visoke pouzdanosti, ovaj alat se i dalje često koristi te je projekt nastavio *živjeti* kroz odvojeni ogranak (*forkani* projekt) [27]. Novi projekt sadrži i proširenje koje omogućuje programsko kreiranje vlastitog privatnog modela. Nažalost, ovaj dio je podržan samo za sustave Ubuntu 16.04 i macOS.

U nastavku se sada daje cjelovit primjer za slučaj prepoznavanja govora s mikrofona. Primjer 2.6 se može podijeliti na tri dijela. Prvi dio (do linije 9) se odnosi na definiranje mikrofona. Tu je demonstrirano kako definirati mikrofon određen nazivom. U drugom dijelu (do linije 15) se obrađuje zvuk s mikrofona. Tijekom prve 2 sekunde slušanja, objekt `recognizer` će kalibrirati razinu pozadinske buke. Nakon toga će krenuti slušati govor (linija 13), sve dok se ne detektira riječ okidač „Siri”. Ako se to ne dogodi u roku od 5 sekundi, program će završiti s iznimkom. U suprotnom, nakon „Siri”, govor se sluša sve dok nema stanke duže od 1.5 sekundi, maksimalno 10 sekundi. Na kraju, u `audio` su pohranjeni snimljeni audio podaci, na temelju kojih se u zadnjem dijelu (linija 16) govor pretvara u tekst.

```
1  import speech_recognition as sr
2
3  recognizer = sr.Recognizer()
4  microphone = None
5
6  for i, name in enumerate(sr.Microphone.list_microphone_names()):
7      if name == "HDA Intel PCH: ALC272 Analog (hw:0,0)":
8          microphone = sr.Microphone(i)
9
10 with microphone as source:
11     recognizer.adjust_for_ambient_noise(source, 2)
12     recognizer.pause_threshold = 1.5
13     audio = recognizer.listen(source, 5, 10, ['home/project/snowboy',
14                               'home/models/siri.umdl'])
15
16 result = recognizer.recognize_google(audio)
```

Primjer 2.6: Slušanje zvuka s mikrofona

Izvršavanje funkcije `listen` može trajati proizvoljno dugo, što često zna predstavljati problem ukoliko program za to vrijeme ne bi smio biti *zamrznut*. Upravo zbog toga, ovu funkciju je u većini slučajeva poželjno pozivati u dretvi odvojenoj od glavne. Biblioteka SpeechRecognition za ovo nudi *prečac* u vidu funkcije `listen_in_background`.

```
m listen_in_background(self, source, callback, phrase_time_limit=None)
```

Gornja funkcija pokreće pozadinsku (*deamon*) dretvu koja opetovano snima po jednu sekundu zvuka s mikrofona (`source`). Nakon obrade svake sekunde, iz pozadinske dretve se automatski poziva `callback` funkcija. Ta funkcija preko parametara dobiva pripadni objekt `Recognizer`, te objekt `AudioData` s audio podacima za tu sekundu. Povratna vrijednost funkcije `listen_in_background` je tzv. zaustavna funkcija. Pozivom te funkcije se zahtijeva prekid rada pozadinske dretve. Zaustavna funkcija prihvaća jedan `bool` argument, kojim se određuje hoće li funkcija prvo pričekati da pozadinska dretva završi s radom.

Uvjet na maksimalnu duljinu cijelog govora se daje argumentom `phrase_time_limit`, potpuno jednako kao kod funkcije `listen`. Međutim, pozadinsko slušanje ne podržava nikakve uvjete na početak govora.

Glavna ideja korištenja pozadinskog slušanja je demonstrirana u sljedećem primjeru.

```
1  import time
2  import speech_recognition as sr
3
4
5  # poziva se iz pozadinske dretve
6  def callback(recognizer, audio):
7      print(recognizer.recognize_google(audio))
8
9
10 recognizer = sr.Recognizer()
11 mic = sr.Microphone()
12
13 # započinje pozadinsko slušanje
14 stop_listening = recognizer.listen_in_background(mic, callback)
15
16 # nevezano računanje (5 sekundi)
17 for _ in range(50): time.sleep(0.1)
18
19 # prekid slušanja
20 stop_listening(wait_for_stop=False)
```

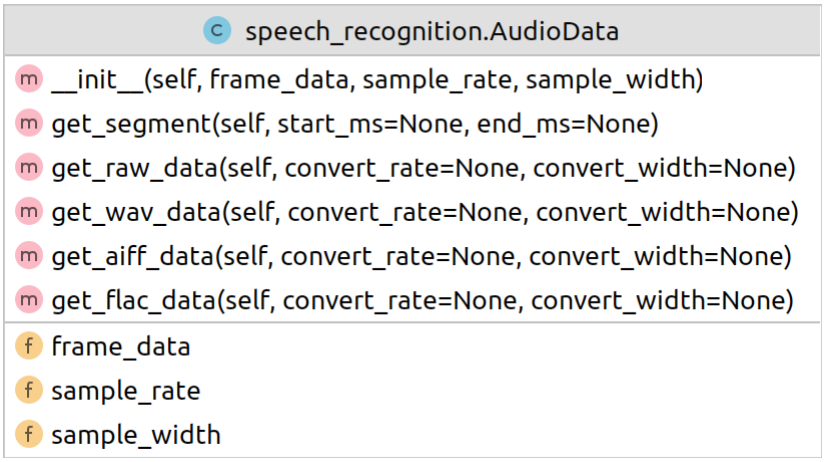
Primjer 2.7: Pozadinsko slušanje s mikrofona

Prvo se pokreće pozadinsko slušanje (primijetiti da nije potreban `with` blok). Nakon toga, program neometano nastavlja s radom (linija 17), istovremeno slušajući govor u pozadini. Poslije svake sekunde govora, pozadinska dretva poziva `callback` funkciju te ista ispisuje pripadni dio teksta. Ovakvim pristupom se uspješno stvara dojam stvaranja transkripta u stvarnom vremenu. U ovom primjeru su radi jednostavnosti izostavljeni dijelovi vezani za kalibriranje pozadinske buke te rukovanje iznimkama u `callback` funkciji.

2.2.2.3 Rezultat obrade zvuka

Zaključno, iz navedenih primjera obrade zvuka je lako uočiti kako je sučelje biblioteke gotovo isto za zvuk iz datoteke i mikrofona. Zapravo sve složenije razlike i stvarno barata-nje datotekama/mikrofonima su *sakrivene* u samoj implementaciji biblioteke. Jedina bitna razlika pri kodiranju je što se ovisno o izvoru zvuka koristi funkcija `record` ili `listen`, uz ponovno minimalno različite argumente. Važno je primijetiti kako navedene funkcije kao rezultat vraćaju objekte istog tipa. Na taj način se osigurava da daljnji rad s audio

podacima bude potpuno neovisan o izvoru zvuka. U oba slučaja se radi o instanci klase `AudioData`.



Dijagram 2.4: Klasa AudioData

Dijagramom 2.4 su prikazana sva svojstva i funkcije članice klase `AudioData`. Kao što je spomenuto ranije, instanca ove klase predstavlja *sirove*, *mono* audio podatke. Svaka `AudioData` instanca je određena trima svojstvima. Sami audio podaci su sadržani u svojstvu `frame_data`, i to kao slijed bajtova koji predstavljaju audio uzorke. Veličina audio uzoraka je određena svojstvom `sample_width`. Dakle, svaki uzorak je specificiran sa `sample_width` brojem bajtova. Dodatno, pretpostavlja se da audio podaci imaju stopu uzorkovanja od `sample_rate` uzoraka po sekundi.

Sve funkcije članice klase `AudioData` služe za dohvat podataka (*getter*). Funkcija `get_segment_data` vraća audio podatke za interval određen parametrima (u milisekundama). Ostale funkcije imaju međusobno jednake definicije oblika `get_X_data` (vidi dijagram 2.4), gdje se „X" odnosi na određeni audio format. Sve takve funkcije vraćaju X-kodiran niz bajtova koji reprezentira audio podatke. Upisivanje ovih bajtova direktno u datoteku rezultira validnom datotekom „X" formata. U primjeru 2.8 je prikazano kako govor s mikrofona spremiti u WAV datoteku.

Još treba istaknuti kako se instance klase `AudioData` gotovo uvijek dobivaju preko funkcija `record`, `listen` i `listen_in_background`, te se u pravilu ne instanciraju izravno. Dobivene instance se predaju kao parametri funkcijama zaduženim za pretvorbu u tekst. Više riječi o samoj pretvorbi bit će u sljedećem potpoglavlju.

```
# ...

with microphone as source:
    audio = recognizer.listen(source)

# prijepis bajtova u novu datoteku
with open('/home/Novi_govor.wav', 'wb') as file:
    file.write(audio.get_wav_data())

# ...
```

Primjer 2.8: Spremanje govora u audio datoteku

2.2.3 Prepoznavanje govora

U ovom odlomku je detaljnije opisana pretvorba: audio podaci → tekst, koja ujedno predstavlja zadnji i najvažniji korak u procesu prepoznavanja. Bitno je naglasiti kako biblioteka SpeechRecognition u svojoj implementaciji ne koristi nikakve modele strojnog učenja, već radi kao omotač (*wrapper*) za vanjske API-je. Dakle, biblioteka SpeechRecognition interno samo obavlja pozive (*requests*) na određeni API, koji će odraditi *teži* dio posla i vratiti transkript. Biblioteka zasad podržava sedam različitih API-ja, čiji su pozivi enkapsulirani kroz odgovarajuće funkcije članice klase `Recognizer`.

```
recognize_sphinx(self, audio_data, language="en-US", keyword_entries=None, grammar=None, show_all=False)
recognize_google(self, audio_data, key=None, language="en-US", show_all=False)
recognize_google_cloud(self, audio_data, credentials_json=None, language="en-US", preferred_phrases=None, show_all=False)
recognize_wit(self, audio_data, key, show_all=False)
recognize_bing(self, audio_data, key, language="en-US", show_all=False)
recognize_houndify(self, audio_data, client_id, client_key, show_all=False)
recognize_ibm(self, audio_data, username, password, language="en-US", show_all=False)
```

Dijagram 2.5: Funkcije za pretvorbu audio podataka

Svaka od navedenih funkcija transkribira govor definiran audio podacima, koristeći API po kojemu je nazvana. Rad ovih funkcija se iz perspektive korisnika biblioteke gotovo ne razlikuje. Sve funkcije imaju povratne vrijednosti istog tipa, primaju većinom iste argumente te bacaju iznimke istog tipa. Zajednički parametri svih `recognize_*` funkcija su sljedeći:

- `audio_data` je `AudioData` objekt koji predstavlja govor koji treba transkribirati.

- Autorizacijski parametri omogućuju pristup API-ju. Upute za registraciju i dohvaćanje pristupnih ključeva se mogu naći u dokumentaciji biblioteke i na službenim stranicama svakog API-ja.

- `show_all` je parametar tipa `bool` koji određuje oblik povratne vrijednosti funkcije. Ukoliko je taj parametar `False`, funkcija vraća najvjerojatniji transkript u obliku stringa. U suprotnom, funkcija vraća sirovi JSON odgovor zaprimljen od strane API-ja.

```
>>> r.recognize_google(audio)
'a time squirrel makes a nice pet'
>>> r.recognize_google(audio, show_all=True)
{
        'alternative': [{
                'transcript': 'a time squirrel makes a nice pet',
                'confidence': 0.94333524
        }, {
                'transcript': 'daytime squirrel makes a nice pet'
        }, {
                'transcript': 'squirrel makes a nice pet'
        }, {
                'transcript': '8 time squirrel makes a nice pet'
        }, {
                'transcript': 'a time squirrel makes a noise pet'
        }],
        'final': True
}
```

Primjer 2.9: Parametar `show_all`

U primjeru 2.9 se može vidjeti razlika između rezultata funkcije ovisno o vrijednosti parametra `show_all`. Ako parametar nije naveden, uzima se vrijednost `False` (vidi definiciju funkcije na slici 2.5). Navedeni transkript je nastao iz audio snimke jedne od rečenica iz Harvard govornog korpusa[7]. Rečenica glasi: „A tame squirrel makes a nice pet".

[7] Harvard govorni korpus [1] je skup izgovorenih rečenica koje se koriste za standardizirano testiranje raznih audio sustava. Rečenice su osmišljene tako da koriste foneme po istoj frekvenciji kako se pojavljuju u standardnom engleskom jeziku. U primjeru 2.9 je korištena jedna od pripadnih audio datoteka s izvornim govornikom [10].

Proces pretvorbe govora u tekst može naići na razne probleme kao što su primjerice nerazumljiv govor, nekvalitetna internetska veza, neuspješna API autorizacija, nedostupan API i slični. Usprkos tome, biblioteka SpeechRecognition nudi jednostavno sučelje kada je riječ o spomenutim, ali i problemima specifičnima za neki od API-ja. Svi problemi se svode na dva tipa iznimki: `UnknownValueError` i `RequestError`. Tako će `recognize_*` funkcija završiti s iznimkom `UnknownValueError` ako se radi o nerazumljivom govoru, a s iznimkom `RequestError` ako je došlo do greške prilikom interakcije s API-jem. Dodatno, trajanje svakog API poziva se može ograničiti preko svojstva `operation_timeout` objekta `Recognizer` (vidi 2.3). Ako interakcija s API-jem premaši ovo ograničenje, funkcija također završava s iznimkom `RequestError`.

2.2.3.1 API-ji za pretvorbu govora u tekst

U prethodnom dijelu je naglašeno kako je sučelje biblioteke prema svim API-jima gotovo isto. Međutim, kvaliteta transkripta i podržane opcije se poprilično razlikuju od API-ja do API-ja. U nastavku su opisane specifičnosti pojedinih API-ja koje treba imati na umu pri njihovom odabiru. Na početku se daje pregled svih API-ja uz neke osnovne karakteristike.

Tablica 2.1: Pregled podržanih API-ja

API *	Način rada	Plaćanje	Ograničenja	Jezici	Hrvatski jezik	Dodatne opcije **
Google	*Online*	Besplatno	50 poziva dnevno	137 jezika	Da	Ne
Google Cloud	*Online*	0.16 - 0.24 kn/min	900 poziva po minuti	137 jezika	Da	Da
Sphinx	*Offline*	Besplatno	Nije navedeno	4 jezika	Ne	Da
Houndify	*Online*	Besplatno	Nije navedeno	22 jezika ***	Ne	Ne
Wit	*Online*	Besplatno	60 poziva po minuti	131 jezika	Da	Ne

* Navedeni su skraćeni nazivi API-ja kao u `recognize_*` funkcijama.
** Dodatne opcije u sklopu biblioteke SpeechRecognition.
*** Biblioteka SpeechRecognition za Houndify nudi samo engleski jezik.

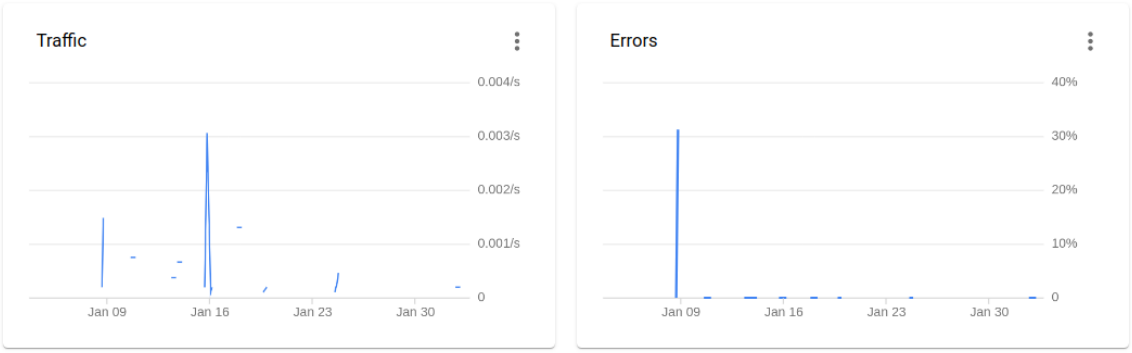
U nastavku se navode detaljniji opisi svakog od API-ja, prateći redoslijed iz tablice 2.1.

Google Speech API [11]

Ovaj API je jednostavnija verzija Google Cloud API-ja, namijenjena uglavnom za nekomercijalno razvijanje i testiranje. Dotični API očekivano zaostaje po kvaliteti za službenim API-jem, ali predstavlja dovoljno dobru alternativu za jednostavne upotrebe. API je besplatan, no postoji dosta strogo ograničenje na broj poziva (vidi tablicu 2.1). U svim dosadašnjim primjerima je korišten baš ovaj API, iz razloga što isti može raditi bez predanog autorizacijskog parametra. Biblioteka SpeechRecognition tada koristi Google-ov zadani ključ. Ovakav pristup se preporuča samo za testiranje, budući da Google može opozvati ovaj ključ bilo kada.

Google Cloud Speech-to-Text API [12]

Google Cloud API se ističe kao jedan od najboljih *speech to text* API-ja, što ga čini čestim izborom u komercijalnim softverima. Za uspješnu autorizaciju je prvo potrebno otvoriti račun i uspostaviti projekt, za koji se onda mogu dohvatiti potrebni ključevi. U svakom trenutku, dostupan je pregled cjelokupne upotrebe API-ja u sklopu projekta.



Slika 2.6: Promet prema Speech-to-Text API-ju

Google Cloud API nudi razne mogućnosti za podešavanje modela specifičnim zahtjevima projekta. Biblioteka SpeechRecognition podržava jednu od tih opcija: specifikaciju preferiranih fraza. Navedene fraze će imati prednost pri prepoznavanju nad frazama koje slično zvuče. Ova opcija se može pokazati korisnom za prepoznavanje ključnih riječi/naredbi ili za dodavanje riječi koje ne postoje u Google-ovom vokabularu. Međutim, treba napomenuti kako postoji ograničenje od 5000 fraza po pozivu. U sklopu biblioteke SpeechRecognition, preferirane fraze se predaju kao niz stringova funkciji `recognize_google_cloud` (vidi definiciju na slici 2.5).

Alat Pocketsphinx [25]

Pocketsphinx je perolaki (*lightweight*) izvanmrežni alat za prepoznavanje govora razvijen kao projekt otvorenog koda, u sklopu projekta CMUSphinx[8]. Alat je posebno prilagođen za mobilne uređaje, iako jednako dobro radi i na računalima. Pocketsphinx predstavlja iznimku među API-jima, budući da se zapravo radi o alatu, a ne API-ju. Ovaj alat nije integriran u biblioteci SpeechRecognition, no u sklopu biblioteke se mogu naći posebne upute za instalaciju i podešavanje. Instalirani paket inicijalno podržava samo engleski jezik, ali je naknadno moguće instalirati ekstenzije za talijanski, francuski te mandarinski jezik. Za postavljanje takvih ekstenzija, budući da uključuje stvaranje modela iz izvornog koda, su potrebni nešto jači računalni resursi (bar 20GB RAM-a).

Alat Pocketsphinx uz regularno prepoznavanje govora, nudi i dvije dodatne opcije. Prva opcija se odnosi na prepoznavanje točno određenih ključnih riječi ili fraza. Ako su ključne riječi zadane, prepoznavatelj će bilježiti samo njih (ako se pojavljuju), dok će sve ostalo ignorirati. Za svaku ključnu riječ se još može specificirati koliko će prepoznavatelj biti na nju osjetljiv. Ključne riječi se predaju funkciji `recognize_sphinx` kroz parametar `keyword_entries` (vidi def. na slici 2.5). Parametar prima listu elemenata oblika `(ključna_riječ, osjetljivost)`, gdje je `osjetljivost` vrijednost od 0 (više *false negative* rezultata) do 1 (više *false positive* rezultata).

Druga dodatna opcija se odnosi na prepoznavanje govora određenog FSG[9] ili JSGF[10] gramatikom. Prepoznavatelj će u tom slučaju bilježiti samo izraze koji odgovaraju pravilima gramatike. Opcija se uključuje predavanjem parametra `grammar`, koji očekuje put do datoteke s definiranom gramatikom. Ako je predana gramatika JSGF tipa, pripadna FSG gramatika će se spremiti na istoj lokaciji kako bi se ubrzao sljedeći upit. Opcija ključnih riječi ima prednost nad ovom opcijom, odnosno ako je zadan parametar `keyword_entries`, eventualna gramatička pravila će se zanemariti.

Na slici 2.8 su dani primjeri JSGF (lijevo) i FSG gramatike (desno). Obje gramatike generiraju isti jezik. Svaka rečenica jezika je neki niz brojeva od jedan do tri (na engleskom jeziku).

[8] Projekt CMUSphinx [32] objedinjuje nekoliko alata za prepoznavanje govora, razvijenih na sveučilištu Carnegie Mellon. Razvoj ovog projekta stagnira od kraja 2019. godine, kada je objavljeno da je fokus istraživanja prebačen na novi alat za prepoznavanje (Kaldi).

[9] FSG (*Finite State Grammar*) je konačan skup pravila, gdje svako pravilo specificira stanje sustava u kojem se može primijeniti, simbol koji se generira i stanje sustava nakon primjene pravila. [8]

[10] JSGF (*Java Speech Grammar Format*) je vrsta tekstualnog prikaza gramatike. Pri definiranju pravila se osim tradicionalnih zapisa, usvajaju određeni stilovi i konvencije programskog jezika Java. JSGF zapisi se najčešće koriste baš u područjima vezanim za prepoznavanju govora. Više o pravilima se može naći u [31].

counting.gram

```
#JSGF V1.0;

/**
 * JSGF Grammar for English counting example
 */

grammar counting;

public <counting> = ( <digit> ) +;

<digit> = one | two | three;
```

counting.fsg

```
FSG_BEGIN <counting.counting>
NUM_STATES 3
START_STATE 0
FINAL_STATE 1
TRANSITION 0 1 1.000000 one
TRANSITION 0 1 1.000000 two
TRANSITION 0 1 1.000000 three
TRANSITION 0 2 1.000000 one
TRANSITION 0 2 1.000000 two
TRANSITION 0 2 1.000000 three
TRANSITION 2 0 1.000000
FSG_END
```

Slika 2.7: JSGF i FSG gramatike

Houndify Speech To Text Only API [29]

Houndify je bogata platforma za rad s govorom, razvijena od strane tvrtke SoundHound. Platforma je jedinstvena po tome što nudi cjelokupna rješenja za bilo kakvu vrstu govorne funkcionalnosti uključujući prepoznavanje govora, *razumijevanje* govora te ponekad i odgovore na pitanja. Prilikom izrade klijenta za novi projekt, mogu se specificirati točne domene kojih se projekt tiče, kao npr. sport, glazba ili vrijeme. Rezultati prepoznavanja govora onda na pitanja iz tih domena sadrže moguće odgovore te dodatne informacije.

Budući da se biblioteka SpeechRecognition bavi isključivo prepoznavanjem govora, ona ne nudi sučelje za dodatne mogućnosti platforme Houndify. Biblioteka koristi isključivo Houndify Speech to Text Only API. Navedeni API daje običan transkript govora lišen dodatnih podataka o domeni.

Wit.ai API [34]

Wit.ai je platforma za prepoznavanje govora primarno namijenjena za korištenje u aplikacijama za pametne uređaje. Ova platforma također nudi dodatni skup opcija iz područja obrade prirodnog jezika. Za razliku od Houndify-ja, Wit.ai ne nudi toliko široke rezultate, ali omogućuje razumijevanje generalne namjere (*intent*) izrečenoga. Govor rastavljen kao u donjem primjeru je implementacijski lako povezati s odgovarajućom akcijom.

Slika 2.8: Wit.ai – razumijevanje namjere govora

Kako bi dohvatili ključeve za autorizaciju, potrebno je prvo otići na službenu stranicu API-ja te stvoriti aplikaciju. Nakon toga, aplikaciju treba istrenirati preko interaktivnog sučelja. Aplikacija se trenira unošenjem primjera mogućih izjava koje se vežu s odgovarajućom namjerom. Na temelju tih primjera se stvara personalizirani model koji se koristi pri pozivu API-ja.

Kao što je i slučaj s Houndify-jem, navedene naprednije opcije Wit-ovog API-ja nisu podržane u sklopu biblioteke SpeechRecognition. Pripadna funkcija `recognize_wit` će vratiti samo transkript govora.

Ostali API-ji

Iz tablice 2.1 su namjerno izostavljena dva API-ja za koje biblioteka SpeechRecognition nudi pripadne funkcije: `recognize_bing` i `recognize_ibm`.

Prva navedena funkcija se odnosi na Microsoft Bing Speech API. Radi se o zastarjelom API-ju koji je ukinut u korist novog Microsoft Azure Speech API-ja [20]. Novi Microsoftov API spada među najkvalitetnije API-je za prepoznavanje govora. API po svemu predstavlja glavnu, i nešto jeftiniju, alternativu Google Cloud API-ja. Glavna prednost nad Google-ovim API-jem su bolje postavke privatnosti i sigurnosti.

Druga funkcija se odnosi na IBM Watson Speech to Text API [13]. Ovaj API nije zastario, već je zastarjela implementacija same funkcije. Naime, krajem 2019. godine, IBM je ažurirao način upravljanja korisničkim računima i način pristupa svim svojim uslugama (više u [22]). Biblioteka SpeechRecognition se i dalje oslanja na zastarjeli oblik autorizacije, zbog čega se bez intervencije u izvornom kodu ova funkcija ne može koristiti.

2.3 Daljnji razvoj

U prvom poglavlju dan je detaljan opis korisničkog sučelja biblioteke SpeechRecognition, s popratnim primjerima. Iz svega opisanog, može se zaključiti kako je glavna prednost ove biblioteke upravo njena jednostavnost. Biblioteka nudi širok i složen skup funkcionalnosti, a sve to preko sučelja od svega 3 *radne* klase, koje sve skupa nemaju više od 25 funkcija članica. Dodatno, sučelje koje se nudi korisniku je intuitivno i na visokoj razini apstrakcije, ali i dalje nudi razne mogućnosti prilagodbe.

Treba se još osvrnuti i na neke slabije točke ove biblioteke. Općenito, svi nedostaci biblioteke imaju korijen u tome što od 2017. godine do danas nije objavljena nijedna nova verzija. Budući da se područje prepoznavanja govora brzo razvija, puno toga se otada promijenilo. Naveden je primjer dvaju kvalitetnih API-ja koji se više ne mogu koristiti jer implementacija biblioteke ne prati njihov razvoj. Unatoč tome, ova biblioteka i dalje ima veliku zajednicu korisnika, čemu svjedoči i broj nedavno otvorenih prijedloga za

poboljšanje na GitHub-u. Također, kao odgovor na izostanak podrške, krajem 2020. godine jedan korisnik je uspostavio ogranak projekta (*fork*), koji se otada aktivno održava. U sklopu njega su prihvaćeni svi popravci vezani za postojeću funkcionalnost biblioteke te su dodane i neke nove opcije. Tako primjerice ovaj projekt nudi podršku i za neke API-je novije generacije kao sto su: Microsoft Azure, Vosk [5] i Tensorflow [4]. Ostaje za vidjeti hoće li originalan projekt *oživjeti* ili će ovaj ogranak postati novi standard.

Nakon većinom teorijskog pregleda biblioteke, u nastavku je opisano iskustvo korištenja iste kroz izradu vlastite aplikacije.

Poglavlje 3

Desktop aplikacija Transkripta

Kako bi se zaokružila analiza biblioteke SpeechRecognition, ostalo je na temelju teorijskih znanja iz prethodnih poglavlja, isprobati rad biblioteke u praksi. Glavna motivacija pri osmišljavanju aplikacije je bila što sveobuhvatnija upotreba biblioteke. Ideja se onda nametnula sama te je u sklopu rada razvijena Transkripta, desktop aplikacija koja služi za stvaranje transkripata. Aplikacija je alatnog tipa – iz ulaznih podataka koji mogu biti govor prenesen mikrofonom ili gotova audio datoteka, stvaraju se izlazni podaci u obliku odgovarajućeg teksta. Primarno je namijenjena za brzo i jednostavno stvaranje bilješki/skripti u svrhu pregleda određenog predavanja, intervjua, dnevničkog zapisa ili slično.

Slika 3.1: Početni zaslon

3.1 Razvojno okruženje i instalacija

Aplikacija je razvijena u programskom jeziku Python 3.8.10 kroz integrirano razvojno okruženje PyCharm tvrtke JetBrains. Grafičko korisničko sučelje (*graphical user interface*, skraćeno GUI) je ostvareno pomoću biblioteke PyQt 6.1.0 [1]. Izvorni kod aplikacije se može naći na GitHub-u (`https://github.com/tomarga/Skripta`), gdje postoje i upute za pokretanje.

Aplikacija se može pokrenuti iz izvršnog direktorija ili iz izvornog koda. Budući da aplikacija tijekom rada pristupa vanjskim API-jima, u oba slučaja je prvo potrebno stvoriti datoteku *env.json* te u njoj specificirati vlastite pristupne ključeve.

```
{
    "GOOGLE_API_KEY": "google_key",
    "GOOGLE_CLOUD_CREDS": "google_cloud_creds",
    "HOUNDIFY_CLIENT_ID":  "houndify_client_id",
    "HOUNDIFY_CLIENT_KEY":  "houndify_client_key"
}
```

Primjer 3.1: Struktura datoteke *env.json*

Upute za stvaranje vlastitih ključeva se mogu naći u dokumentaciji pripadnog API-ja. Također, u dokumentaciji biblioteke SpeechRecognition kao i na GitHub-u aplikacije se nalaze potrebne poveznice.

Za pokretanje aplikacije iz izvršnog direktorija treba pratiti sljedeće korake:

1. Preuzeti zapakirani direktorij *skripta.tar.gz* – poveznica je u opisu GitHub-a repozitorija (*README.md*).

2. Raspakirati direktorij na željeno mjesto.

3. Premjestiti vlastitu datoteku *env.json* u raspakirani direktorij.

4. Otvoriti direktorij u terminalu i pokrenuti izvršnu datoteku: `# ./transkripta` .

[1] PyQt [26] je skup Python modula koji čine omotač za višeplatformsku C++ biblioteku Qt. Osim klasičnih GUI elemenata, Qt pa tako i PyQt, sadrži apstrakcije mrežnih utičnica, programskih dretvi, regularnih izraza, SQL baza podataka te potpuno funkcionalan internetski preglednik. Qt klase koriste mehanizam signala i utora za komunikaciju između objekata. Također, Qt uključuje i Qt Designer, jednostavan alat za izradu GUI-a. PyQt može generirati Python kod iz `.ui` datoteka kreiranih pomoću Qt Designer-a.

Izvršni direktorij je napravljen pomoću alata PyInstaller[2]. Bitno je napomenuti kako navedeni alat generira izvršni direktorij isključivo za trenutni operacijski sustav. Iz tog razloga se Transkripta na ovaj način zasad može pokrenuti samo na Linux Ubuntu sustavima, s verzijom ne manjom od 20.04.

Alternativno, na Windows, MacOS te ostalim Linux sustavima aplikacija se može pokrenuti iz izvornog koda, a za to je potrebno:

1. Klonirati projekt s GitHub-a.
2. Instalirati Python3.
3. Instalirati sve ovisnosti navedene u datoteci *requirements.txt*.
4. Premjestiti vlastitu datoteku *env.json* u korijenski direktorij *Skripta*.
5. Pokrenuti skriptu *main.py*: `# Python3 main.py` .

3.2 Grafičko sučelje i funkcionalnost

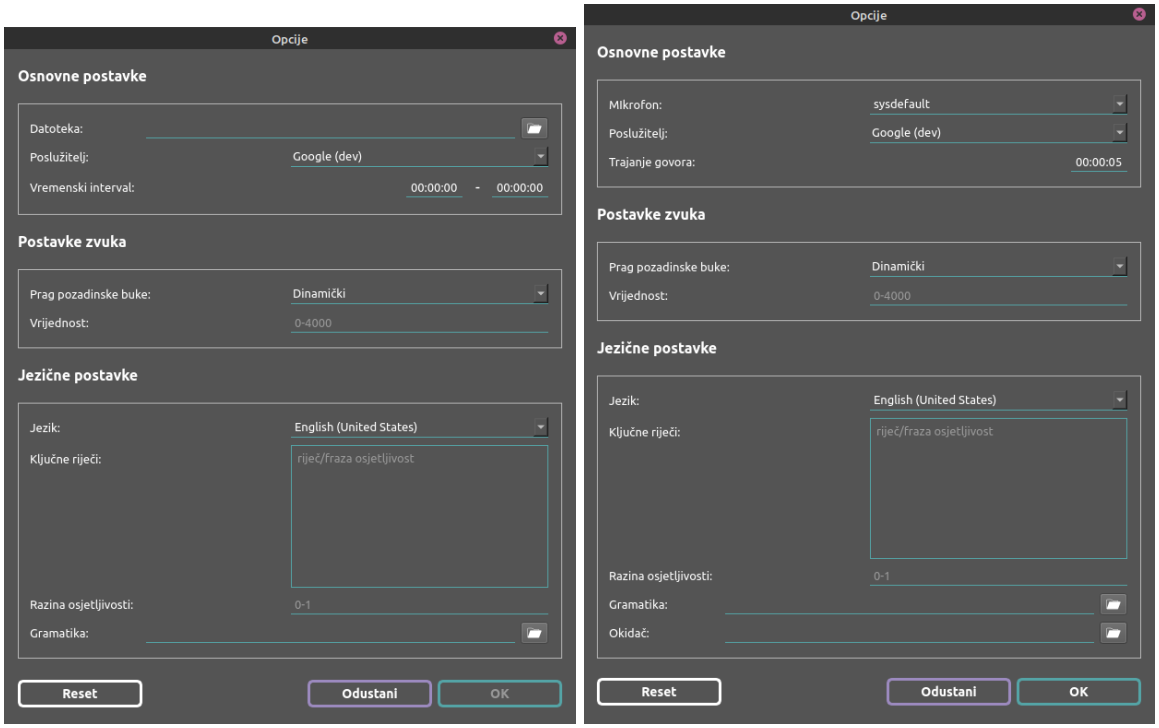
Aplikacija nudi jednostavno i intuitivno korisničko sučelje. Budući da se radi o aplikaciji alatnog tipa, sučelje je bilo prirodno bazirati na dijalozima, s naslovnim dijalogom trajno prikazanim u pozadini.

3.2.1 Postavke transkripcije

Po pokretanju aplikacije, otvara se naslovni dijalog kao na slici 3.1. Na njemu korisnik bira između dvije glavne opcije: transkripcija audio datoteke (gumb „Učitaj") i transkripcija govora s mikrofona (gumb „Snimi"). Odabirom pojedine opcije, otvara se pripadni dijalog za postavke opcija transkripcije. Postavke su osmišljene tako da pokrivaju gotovo svu funkcionalnost biblioteke SpeechRecognition iz odlomka 2.2. Naravno, postoje male razlike između dijaloga s postavkama za mikrofon te dijaloga s postavkama za audio datoteku. Zajednička im je okvirna podjela na tri skupa postavki: osnovne postavke, postavke zvuka te postavke jezika.

Na dnu dijaloga se nalaze gumbi za početak transkripcije (gumb „OK"), povratak na naslovni dijalog (gumb „Odustani") te za resetiranje izabranih opcija na inicijalne vrijednosti (gumb „Reset"). Na slici 3.2 su prikazana oba dijaloga s opcijama postavljenim na inicijalne vrijednosti.

[2] PyInstaller pakira Python aplikaciju i sve njezine ovisnosti u jedan direktorij ili datoteku. Korisnik onda može pokrenuti aplikaciju bez postavljanja Python interpretera ili bilo kojih modula.



(a) Audio datoteka (b) Mikrofon

Slika 3.2: Opcije transkripcije

Osnovne postavke

Grupa pod nazivom „Osnovne postavke" predstavlja minimalan skup opcija koje treba odrediti da bi se mogla pokrenuti transkripcija.

Prva opcija iz ove grupe se odnosi na definiranje izvora zvuka. U slučaju (a), potrebno je specificirati točan put do audio datoteke. Put se može unijeti manualno ili automatski, odabirom datoteke iz zasebnog dijaloga. Dijalog za odabir datoteke se otvara klikom na ikonu „direktorij" te je inicijalno pozicioniran u korisničkom direktoriju (*home*). Dijalog prikazuje samo datoteke s validnom audio ekstenzijom. U slučaju (b), potrebno je iz padajućeg izbornika odabrati mikrofon koji će primati govor.

Sljedeća opcija se odnosi na odabir API-ja koji će se koristiti za prepoznavanje govora, i ona se ne razlikuje ovisno o tipu transkripcije. U padajućem izborniku se može birati između sljedeća četiri API-ja: „Google (dev)", „Google", „Sphinx" i „Houndify". API-ji „Bing" i „IBM" su izostavljeni zbog razloga navedenih u odlomku 2.2.3.1, dok je „Wit" API izostavljen zbog problema pri stvaranju korisničkog računa. Naime, Wit nudi prijavu isključivo preko Facebook računa, a svaki pokušaj je rezultirao beskonačnim učitavanjem.

Zadnja nužna opcija definira točan dio govora od interesa. U slučaju (a), potrebno je definirati interval audio datoteke koji se promatra (`min` – `max`). Ako je put do datoteke određen, ovaj interval će automatski biti postavljen na cijelo trajanje snimke (00:00:00 – `trajanje_snimke`). Interval se onda može po volji ažurirati, s time da su onemogućeni *besmisleni* unosi (`max` manji od `min` ili `max` veći od trajanja snimke). U slučaju (b), potrebno je odrediti točno trajanje govora, tj. koliko će se dugo slušati govor s mikrofona. Inicijalno je ova vrijednost postavljena na 5 sekundi, a maksimalno trajanje je 1 sat. Neovisno o ovoj opciji, slušanje s mikrofona će biti prekinuto ako se u bilo kojem trenutku zabilježi interval tišine od 5 minuta.

Postavke zvuka

Ovaj skup opcija definira način na koji će se upravljati pozadinskom bukom. Opcije su iste bez obzira na tip transkripcije. U oba slučaja, ovisno o očekivanom okruženju, korisnik može izabrati jednu od sljedeće tri opcije:

- Dinamičko upravljanje pozadinskom bukom podrazumijeva automatsko kalibriranje energetskog praga tijekom obrade zvuka. Početna vrijednost praga se određuje na temelju prvih 75 stotinki sekunde zvuka.
- Hibridno upravljanje pozadinskom bukom također podrazumijeva automatsko kalibriranje energetskog praga tijekom obrade zvuka. Međutim, u ovom slučaju korisnik sam određuje početnu vrijednost praga.
- Fiksni način upravljanja podrazumijeva konstantnu vrijednost energetskog praga tijekom cijele obrade zvuka. Točnu vrijednost određuje korisnik.

Navedene opcije se direktno tiču svojstava opisanih u odlomku 2.2.2.1.

Jezične postavke

Opcije iz ove grupe su okvirno vezane za tip/sadržaj govora koji se očekuje. Kao osnovnu opciju, prvo je potrebno odrediti jezik kojim se govori. Pripadni padajući izbornik se ažurira ovisno o trenutno odabranom API-ju – za Sphinx i Houndify je ponuđen samo engleski jezik, dok je za Google-ove API-je ponuđeno više od 100 jezika. Točan popis jezika se može naći u sklopu dokumentacije API-ja [12]. Ostale opcije se tiču dodatnih opcija specifičnih za pojedini API, ranije opisanih u odlomku 2.2.3.1. Budući da opcije nisu univerzalne, pripadna polja se po potrebi onemogućuju ovisno o odabranom API-ju.

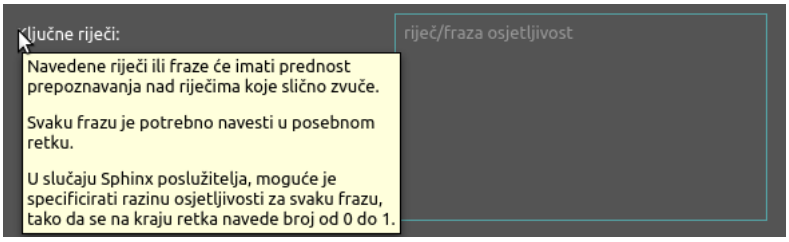
Opcija s oznakom „Ključne riječi" je omogućena ukoliko je odabran Google-ov (službeni) ili Sphinx API. U prvom slučaju, korisnik može unijeti određene preferirane fraze koje će imati prednost pri prepoznavanju. Pritom se svaka fraza mora unijeti u zaseban redak

teksta. U slučaju Sphinx API-ja, korisnik na sličan način može odrediti ključne riječi u govoru. Pri prepoznavanju će se onda registrirati samo te riječi, ako ih uopće ima. Kao što je objašnjeno u odlomku 2.2.3.1, za svaku ključnu riječ je moguće odrediti koliko će prepoznavatelj na nju biti osjetljiv. Ove vrijednosti se mogu navesti nakon svake ključne riječi (u istom retku). Alternativno, razina osjetljivosti se može zadati *globalno* za sve ključne riječi, tako da se unese vrijednost pod oznakom „Razina osjetljivosti". Vrijednosti koje se odnose na osjetljivost ključnih riječi moraju biti realni brojevi od 0 do 1.

Sljedeća opcija „Gramatika" je također omogućena samo ako je odabran Sphinx API. Potrebno je unijeti put do FSG ili JSGF datoteke u kojoj je definirana određena gramatika. Kao i pri odabiru audio datoteke, put je moguće unijeti izravno ili odabirom datoteke u specijaliziranom dijalogu. U dijalogu se pritom prikazuju samo datoteke s ekstenzijama `.fsg` i `.gram`.

Dosad opisane jezične postavke su bile univerzalne za transkripciju bilo kojeg tipa. Preostala opcija s oznakom „Okidač" je ponuđena samo pri transkripciji s mikrofona. Ona omogućuje da se mikrofon *aktivira* tek nakon što je izrečena riječ okidač. Za uključivanje ove opcije, potrebno je unijeti put do datoteke u kojoj je definiran model za neku riječ okidač. U pripadnom dijalogu za odabir se zato filtriraju datoteke s ekstenzijama `.umdl` i `.pmdl`.

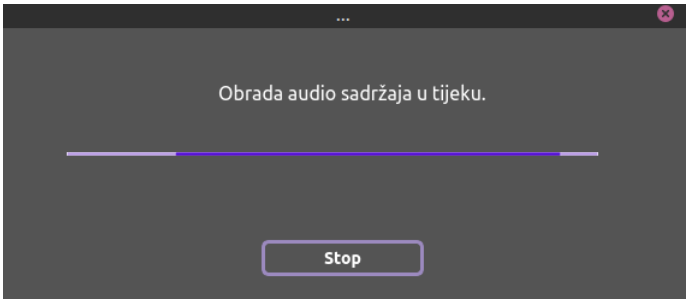
Navedene informacije o svakoj opciji su dostupne i na korisničkom sučelju, u obliku info-oblačića (*tooltip*) koji se otkrije prelaskom miša iznad određene oznake.



Slika 3.3: Info-oblačić

3.2.2 Transkripcija i rezultat

Nakon odabira željenih opcija u dijalogu s postavkama, proces prepoznavanja govora se pokreće klikom na gumb „OK". Budući da se radi o dužem procesu, za vrijeme prepoznavanja govora se prikazuje info-dijalog s prigodnom animacijom za učitavanje (*loading*). U slučaju prepoznavanja govora s mikrofona, poruka u dijalogu se mijenja ovisno o fazi – prvo se sluša govor, a onda se obrađuje. U svakom trenutku, korisnik može obustaviti slušanje/obradu pritiskom na gumb „Stop" ili zatvaranjem dijaloga.



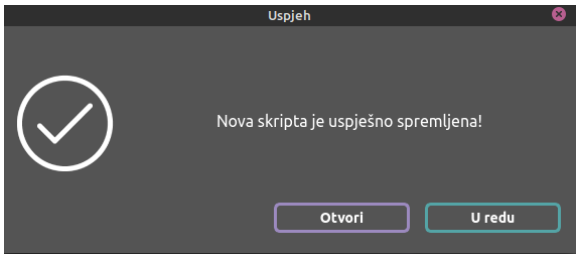
Slika 3.4: Obrada zvuka

Kada prepoznavanje govora završi, dijalog za učitavanje se zatvara, a otvara se novi dijalog prikazan na slici 3.5.

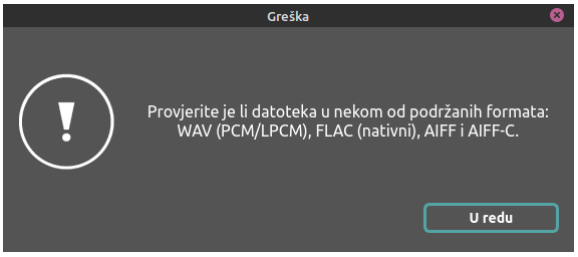
Slika 3.5: Rezultat

Gornji dijalog napokon sadrži tekst nastao kao rezultat prepoznavanja. Tekst je sada dan korisniku na eventualnu doradu prije spremanja. Pritiskom na gumb „Spremi” otvara se

dijalog za odabir lokacije i naziva nove datoteke. Inicijalno, put do datoteke je postavljen na *home/Transkripti/Novi_transkript.txt*. Pritom se kreira novi direktorij *home/Transkripti*, ako već ne postoji. Put do datoteke se naravno može mijenjati, a dopušteni su svi poznatiji tekstualni formati. Po uspješnom spremanju datoteke, korisnik je obaviješten o završetku u novom statusnom dijalogu kao na slici 3.6. Klikom na gumb „Otvori", korisnik može pregledati stvoreni transkript. Datoteka se otvara u prigodnoj aplikaciji ovisno o njenom formatu.



Slika 3.6: Uspjeh



Slika 3.7: Greška

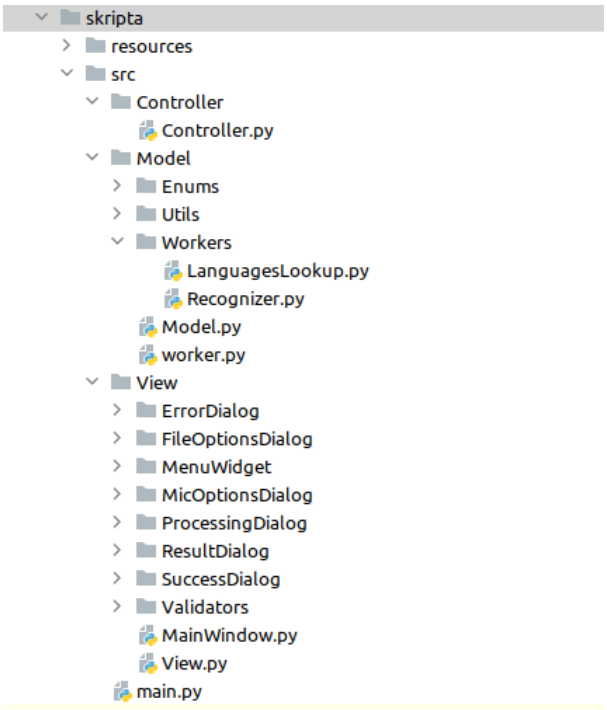
Dosad je opisan tok aplikacije u slučaju uspješnog prepoznavanja. Ukoliko prepoznavanje ipak nije završilo uspješno, prikazuje se statusni dijalog o grešci. Na slici 3.7 je dan primjer jednog takvog dijaloga. Poruka o grešci ovisi o uzroku iste, a uzroci mogu biti sljedeći:

- Nepodržana audio datoteka (slučaj (a)).
- Nepostojeća datoteka (slučaj (a)).
- Nepoznat mikrofon (slučaj (b)).
- Nerazumljiv govor.
- Loša internetska veza.
- Neuspješna autorizacija na API.
- Odgovor API-ja nije zaprimljen u zadanom roku od 30 minuta.
- Početak govora nije registriran u zadanom roku od 5 minuta (slučaj (b)).

3.3 Implementacija

Aplikacija je razvijena tako da prati osnovnu ideju arhitekturalnog obrasca *Model-View-Controller* (kraće MVC). Sukladno tome, izvorni kod (direktorij *src*) je podijeljen u tri

direktorija, što se može vidjeti iz strukture projekta na slici 3.8. U direktoriju *resources* se nalaze potrebni resursi vezani za GUI: slike, ikone i animacije. U nastavku se daje kratak pregled implementacije po MVC komponentama, s naglaskom na upotrebu biblioteke SpeechRecognition. Na kraju, dijagramom 3.10 dan je pregled svih najvažnijih klasa, također grupiranih po MVC komponentama (lijevo *Controller*, desno *View* i dolje *Model*).



Slika 3.8: Struktura projekta

3.3.1 *View*

Komponenta *View* definira izgled aplikacije, a ostvarena je kroz kod u istoimenom direktoriju. Kao što je prikazano na slici 3.8, taj direktorij se sastoji od: GUI elemenata, pripadnih validatora te datoteka *MainWindow.py* i *View.py*.

Izgled pojedinog GUI elementa definiran je unutar zasebnog direktorija, koji sadrži pripadne `.ui` i `.py` datoteke. Svi GUI elementi razvijeni su korištenjem interaktivnog alata QtDesigner, koji na temelju dizajna generira `.ui` datoteke. Takve datoteke mogu se pretvoriti u Python datoteku pomoću PyQt-a, naredbom:

```
# pyuic6 -x element.ui -o element.py .
```

Generirana Python datoteka sadržava klasu koja odgovara dizajniranom GUI elementu. Ta klasa koristi se direktno u daljnjoj implementaciji.

U direktoriju *Validators* definirane su klase `DurationValidator` i `FileInputValidator`. Navedene klase nasljeđuju PyQt klasu `QValidator` koja omogućuje kontroliranje korisničkog unosa. Na ovaj način se kontrolira unos opcija koje se tiču vremenskog intervala i puta do određene datoteke.

Najvažnija klasa ove komponente je klasa `View` (vidi dijagram 3.10). Ona predstavlja sučelje za baratanje svim GUI elementima – sadrži funkcije za njihovu inicijalizaciju, aktivaciju/deaktivaciju te dohvaćanje unešenih podataka. Također, ona sadrži `mainWindow` svojstvo s instancom klase `MainWindow`, čijom aktivacijom se pokreće sama aplikacija.

Još se treba osvrnuti na način korištenja vizualnih resursa u GUI elementima. Qt ima razvijen vlastiti sustav resursa koji se temelji na korištenju `.qrc` datoteka[3]. Nažalost, najnovija verzija PyQt-a ne nudi podršku za ovaj sustav pa se u ovoj aplikaciji resursima pristupa preko relativnih puteva.

3.3.2 *Controller*

Komponenta *Controller* implementirana je kao jedinstvena klasa (vidi dijagram 3.10). Klasa *Controller* u konstruktoru prima instance klasa *View* i *Model* te poziva funkciju članicu `connectSignalsAndSlots`. Povezivanje signala i utora je od ključne važnosti jer se time definira glavni tok rada aplikacije – sve akcije vezane za korisnički unos, te promjene na grafičkom sučelju u skladu s fazom obrade zvuka.

Bitno je uočiti kako klasa *Controller* služi samo za uspostavljanje konekcija između komponenata *Model* i *View*, a ne za izravnu manipulaciju njima. Stoga, iako je izvorni kod organiziran po MVC obrascu, rad aplikacije je zapravo bliži arhitekturalnom obrascu tipičnom za sve Qt aplikacije, a to je obrazac *Model-View*.

3.3.3 *Model*

Budući da je Transkripta aplikacija alatnog tipa, nije bilo potrebe za pohranom podataka u bazu. Dakle, komponenta *Model* je zadužena isključivo za logiku aplikacije. Program *worker.py* te klasa `Model` iz istoimenog direktorija predstavljaju sučelje ove komponente, a sve važnije klase su izdvojene u dijagramu 3.10.

[3] XML datoteka koja sadrži kolekciju resursa aplikacije.

3.3.3.1 Dohvaćanje podataka

U ovom odlomku se navode zadaće modela koje se ne tiču izravno prepoznavanja govora. Riječ je o dohvaćanju ili obradi podataka, koji se po potrebi predaju komponenti *View* za prikaz na sučelju. Uglavnom se radi o podacima potrebnim za prikaz svih opcija u dijalogu s postavkama. Spomenuta funkcionalnost je ostvarena kroz klasu `Model`. Sve njene funkcije su navedene u dijagramu 3.10. Klasa `Model` tako nudi listu svih dostupnih mikrofona te listu svih dostupnih jezika za Google-ove API-je. Ovi podaci se koriste kao opcije u pripadnim padajućim izbornicima. Također, klasa `Model` određuje trajanje pojedine audio datoteke te po potrebi stvara direktorij u kojem će se spremati novi transkripti.

Dohvaćanje liste podržanih jezika za Google-ove API-je je najsloženija od navedenih funkcija klase `Model`. Lista jezika se dohvaća sa stranice *https://cloud.google.com/speech-to-text/docs/languages* uz pomoć biblioteka Pandas i Numpy. Budući da taj postupak traje nekoliko sekundi, lista se dohvaća samo jednom i to pri inicijalizaciji objekta `Model`. Kako to ne bi usporilo pokretanje aplikacije, ovaj postupak se odvija u odvojenoj dretvi. Nova dretva je implementirana kao instanca klase `QThread` koja izvršava funkciju `run` klase `LanguagesLookup`.

3.3.3.2 Prepoznavanje govora

Središnja funkcionalnost aplikacije, prepoznavanje govora, implementirana je tako da se odvija u procesu odvojenom od glavnog. Odvajanje neke vrste je bilo nužno kako bi se spriječilo zamrzavanje korisničkog sučelja. Višeprocesorski pristup je izabran iz nekoliko razloga. Naime, ključan uvjet je bio da se slušanje odnosno obrada govora može prekinuti u bilo kojem trenutku (gumb „Stop" na slici 3.4). Klasa `QThread` ne podržava ovakav tip zaustavljanja, dok se korištenje Pythonovih nativnih dretvi izbjegava jer nemaju definirane pripadne signale i utore.

Novi proces je implementiran kao instanca klase `QProcess` koja izvršava program *worker.py*. Klasa `QProcess` ima definirane signale i utore koji omogućuju jednostavnu komunikaciju glavnog i radnog procesa. Radni program prima sve unešene postavke transkripcije preko argumenata komandne linije. Na temelju tih postavki kreće slušati s mikrofona ili obrađivati audio datoteku, a rezultat se ovisno o uspješnosti *vraća* preko standardnih kanala za izlaz (`stdout` i `stderr`).

```
(venv) margarita@asusich:~/Desktop/DR/PycharmProjects/Skripta/skripta/src/Model$ python3 worker.py -h
usage: worker.py [-h] [-i INPUT] [-f FILE] [-o OFFSET] [-d DURATION] [-m MIC] [-st SPEECH_TIMEOUT] [-hw HOTWORDS]
 [-e {dynamic,mixed,fixed}] [-sv START_VALUE] [-a {google,google_cloud,sphinx,houndify}] [-l LANGUAGE]
                [-p [PHRASES [PHRASES ...]]] [-pv [PHRASES_VALUES [PHRASES_VALUES ...]]] [-g GRAMMAR]
```

Slika 3.9: Argumenti programa *worker.py*

Na slici 3.9 su prikazani argumenti koje prima program *worker.py*. Argumenti očito odgovaraju opcijama koje korisnik unosi na dijalogu s postavkama.

U programu *worker.py* dobiveni argumenti se prvo transformiraju u odgovarajuće objekte – instance klasa `MicOptions`, `FileOptions` te `CommonOptions` (vidi dijagram 3.10). S tim objektima se onda incijalizira klasa `Recognizer`, u sklopu koje se odvija samo prepoznavanje govora.

U nastavku se daju najvažniji dijelovi koda koji se tiču prepoznavanja govora. Implementacija je razvijena na osnovu primjera iz poglavlja 2. Zato se sada ne ponavljaju detaljna objašnjenja, već se komentiraju isključivo neke specifičnosti pri upotrebi biblioteke.

Pozadinska buka

U isječku 3.2 dana je inicijalizacija objekta za prepoznavanje. Linijom 2 određeno je da će se rezultat prepoznavanja čekati maksimalno 30 minuta. Ostatak koda odnosi se na postavljanje opcija pozadinske buke, ovisno o korisničkom odabiru: dinamička, hibridna ili fiksna opcija (vidi odjeljak Postavke zvuka u potpoglavlju 3.2.1). Opcije postavki zvuka određene su enumeracijom `EnergyThresholdOption`.

```
1  recognizer = sr.Recognizer()
2  recognizer.operation_timeout = 1800
3
4  if self.commonOptions.energyOption == EnergyThresholdOption.FIXED:
5      recognizer.energy_threshold = self.commonOptions.energyValue
6      recognizer.dynamic_energy_threshold = False
7
8  if self.commonOptions.energyOption == EnergyThresholdOption.MIXED:
9      recognizer.energy_threshold = self.commonOptions.energyValue
10
11 if self.commonOptions.energyOption == EnergyThresholdOption.DYNAMIC:
12     recognizer.adjust_for_ambient_noise(source, 0.75)
```

Isječak koda 3.2: Funkcija initRecognizer()

Obrada zvuka

Prvo se navodi obrada zvuka u slučaju snimanja iz audio datoteke (isječak 3.3). U liniji 4 se inicijalizira prepoznavatelj kao u prethodnom isječku 3.2. Sama obrada zvuka se provodi na potpuno isti način kao u poglavlju 2. Pritom se koriste parametri svojstva `fileOptions`, koji odgovaraju korisničkom unosu.

```
audioFile = sr.AudioFile(self.fileOptions.file)

with audioFile as source:
    recognizer = self.initRecognizer(source)
    audio = recognizer.record(source, self.fileOptions.duration,
                self.fileOptions.offset)
```

Isječak koda 3.3: Funkcija handleFileInput()

Sljedeći isječak 3.4 prikazuje obradu zvuka u slučaju slušanja s mikrofona. Analogno kao pri obradi govora, korisnički unos je sada dan svojstvom `micOptions`. Na početku se inicijaliziraju mikrofon i prepoznavatelj. Linijom 5 je određeno da se slušanje prekida nakon tišine od 5 minuta. U sljedećem dijelu se definira parametar za riječ okidač, ako je takva riječ definirana unosom. Napokon, u liniji 11 se pokreće slušanje. Slušanje će se prekinuti ako se početak govora ne registrira kroz 5 minuta. Za slušanje se ne koristi funkcija `listen_in_background` jer ona ne nudi podršku za riječ okidač. Također ovaj isječak koda se svakako izvodi u procesu odvojenom od glavnog pa nema potrebe za stvaranjem dodatne pozadinske dretve. Na kraju, u liniji 14 se na standardni izlaz ispisuje poruka o završetku slušanja. Na temelju te poruke se ažurira dijalog učitavanja sa slike 3.4.

```
mic = sr.Microphone(device_index=self.micOptions.mic)

with mic as source:
    recognizer = self.initRecognizer(source)
    recognizer.pause_threshold = 300

    snowboyDirectory = ROOT_DIRECTORY.parent.__str__() + '/snowboy'
    hotwordsConf = (snowboyDirectory, [self.micOptions.hotwords])
        if self.micOptions.hotwords is not None else None

    audio = recognizer.listen(source, 300,
                self.micOptions.speechTimeout, hotwordsConf)

    sys.stdout.write("Done listening")
    sys.stdout.flush()
```

Isječak koda 3.4: Funkcija handleMicInput()

U oba isječka koda je radi jednostavnosti izostavljena logika vezana za obradu iznimki.

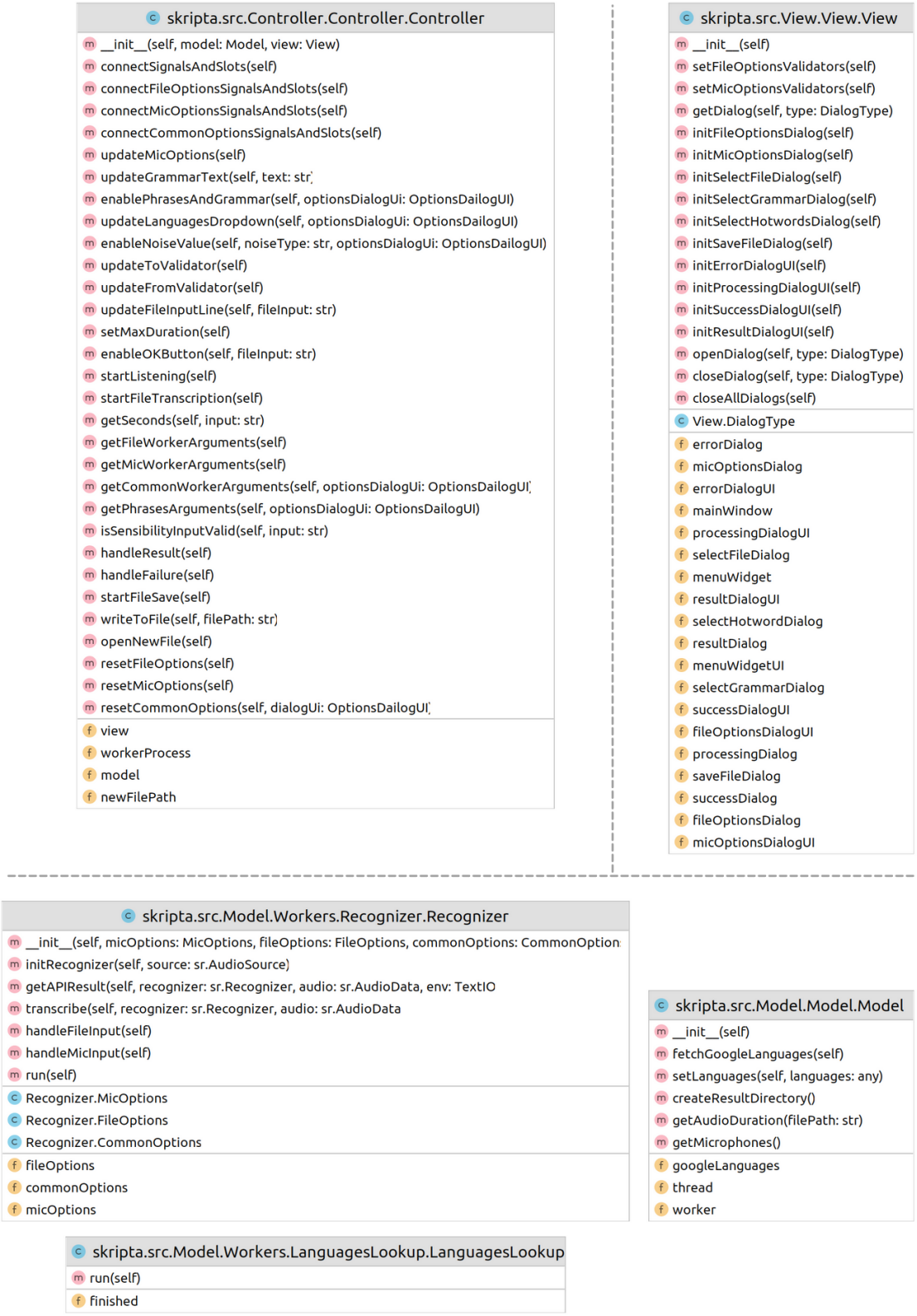
Prepoznavanje govora

Donji isječak 3.5 odnosi se na prepoznavanje govora iz audio podataka. Kao što je opisano u poglavlju 2, za to je dovoljno pozvati odgovarajuću funkciju oblika `recognize_*`. U isječku je prikazano kako se iz datoteke *env.json* dohvaćaju pojedini pristupni ključevi, te kako se isti predaju kao argumenti za svaki od API-ja. Opcije koje je korisnik izabrao dostupne su preko svojstva `commonOptions`. Posebno, tipom API-ja se upravlja preko istoimene enumeracije.

```
env = open(ROOT_DIRECTORY.parent.__str__() + "/env.json", 'r')

with env:
    jsonData = json.load(env)

    if self.commonOptions.api == API.GOOGLE:
        apiKey = jsonData["GOOGLE_API_KEY"]
        return recognizer.recognize_google(audio, apiKey,
                    self.commonOptions.language)

    if self.commonOptions.api == API.GOOGLE_CLOUD:
        creds = json.dumps(jsonData["GOOGLE_CLOUD_CREDS"])
        return recognizer.recognize_google_cloud(
                    audio, creds, self.commonOptions.language,
                    self.commonOptions.phrase)

    if self.commonOptions.api == API.HOUNDIFY:
        clientID = jsonData["HOUNDIFY_CLIENT_ID"]
        clientKey = jsonData["HOUNDIFY_CLIENT_KEY"]
        return recognizer.recognize_houndify(audio, clientID,
                    clientKey)

    if self.commonOptions.api == API.SPHINX:
        return recognizer.recognize_sphinx(
                    audio, self.commonOptions.language,
                    self.commonOptions.phrases,
                    self.commonOptions.grammar)
```

Isječak koda 3.5: Funkcija transcribe()

Na kraju, u glavnom programu *main.py* je potrebno samo stvoriti instance klasa *Model*, *View* i *Controller* te otvoriti glavni prozor s `view.mainWindow.show()`.



Dijagram 3.10: Najvažnije klase

3.4 Primjeri korištenja

Ovim odlomkom želi se dati uvid u kvalitetu transkripcije koja se može očekivati od aplikacije Transkripta. Testiranje se provodi na engleskom jeziku, budući da je to jedini jezik kojeg podržavaju svi API-ji. Za primjer su preuzete rečenice iz Harvard korpusa s britanskom govornicom [10]. Preciznije, izabrano je pet rečenica iz liste broj 14:

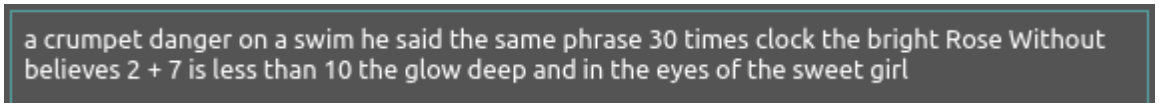
1. A cramp is no small danger on a swim.
2. He said the same phrase thirty times.
3. Pluck the bright rose without leaves.
4. Two plus seven is less than ten.
5. The glow deepened in the eyes of the sweet girl.

3.4.1 Opcija „Učitaj”

Prvo se promatraju rezultati dobiveni prepoznavanjem govora iz audio datoteke. Usput se komentiraju eventualne mogućnosti poboljšanja preko dodatnih korisničkih postavki.

„Google(dev)” API

Na donjoj slici je prikazan rezultat testa za Google API, uz inicijalne postavke.



Slika 3.11: Google API – rezultat testa

Prvo što se može primijetiti je oblik teksta – rezultat je dan kao niz riječi, bez interpunkcijskih znakova. Nažalost, iako neki API-ji nude automatsko postavljanje interpunkcije, ono se ne može uključiti kroz biblioteku SpeechRecognition. Zbog jednostavnije evaluacije nadalje se promatraju formatirani rezultati.

1. A crumpet is no small danger on a swim.
2. He said the same phrase 30 times.
3. Clock the bright Rose Without believes.
4. 2 + 7 is less than 10.
5. The glow deep and in the eyes of the sweet girl.

S 5 krivo prepoznatih (crveno) te 3 propuštene (žuto) riječi od njih 39 ukupno, može se izračunati kako za ovaj primjer vrijedi $WER = 0.21$, tj. transkript je oko 80% ispravan. Bolji rezultat može se očekivati od službenog Google API-ja.

Zanimljivo je još primijetiti da ovaj API brojeve pretvara u odgovarajuće znamenke (4. rečenica). Također, u 3. rečenici je dio s „Rose Without" ispravno prepoznat, ali je netočno zapisan s velikim početnim slovom. Može se pretpostaviti kako se vlastite imenice, ovisno o kontekstu, pri prepoznavanju na neki način favoriziraju.

„Google Cloud" API

Rezultati od službenog Google API-ja su sljedeći:

> 1. A cramp is no small danger on a swim.
> 2. He said the same phrase 30 times.
> 3. Clock the bright Rose Without leaves.
> 4. 2 + 7 is less than 10.
> 5. The glow deepened and the eyes of the sweet girl.

Sa samo 4 greške, ovaj API je očito uspješniji od prethodnog. Dodatno, neke od ovih grešaka moguće je ispraviti uz postavku dodatnih opcija (vidi dijalog na slici 3.3). Prvo, budući da dvije riječi „A" i „leaves" potpuno nedostaju iz transkripta, može se zaključiti kako njihov izgovor *prolazi ispod* energetskog praga. Ako se postavke zvuka ažuriraju tako da se odabere opcija „Hibridni" uz relativno nisku početnu vrijednost 1000, rezultat za 1. i 3. rečenicu je malo bolji:

> 1. A cramp is no small danger on a swim.
> 3. Clock the bright Rose Without believes.

Više nema propuštenih riječi, iako transkript 3. rečenice nije do kraja ispravljen. Ta rečenica se sada može promotriti zasebno, tako da se u postavkama odredi odgovarajući vremenski interval (od 9. do 14. sekunde). Na taj način se dobije:

> 3. Took the bright Rose Without leaves.

Dobiveni rezultat je bolji i bez dodatnih intervencija. Uzrok tome može biti što je prepoznavatelj ranije u obzir uzimao kontekst 4. rečenice, iako su one zapravo nevezane. Tako npr. ako se uzme da je „Rose Without" ime neke osobe, onda riječ „believes" u kontekstu „bright Rose Without believes 2 + 7 is less than 10" ima puno više smisla od riječi „leaves". Napokon, za potpuno ispravan transkript ove rečenice, dovoljno je još u postavkama dodati ključnu riječ „Pluck".

„Houndify" API

Ovaj API daje sljedeće rezultate:

1. A cramp is no small danger on a swim.
2. He said the same phrase thirty times.
3. Pluck the bright rose without leaves.
4. Two plus seven is less than ten.
5. The glow deep and in the eyes of the sweet girl.

Sa samo jednom greškom, ovaj API je postigao najbolji rezultat na ovom primjeru. Još treba napomenuti kako je ovdje za jezik bio postavljen američki engleski (jedini podržan za Houndify) te bi s britanskim engleskim rezultat možda bio i bolji.

Alat „Sphinx"

Ovim alatom su dobiveni sljedeći rezultati:

1. The crowd business small change illness when.
2. He said the same phrase that he times.
3. Camp the bride price without please.
4. Shoot plus seven anthrax and ten.
5. The glare deep into the eyes of the swede fell.

S više od pola netočno prepoznatih riječi ($WER = 0.54$), očito je da Sphinx dosad daje najgori transkript za ovaj primjer. Na istom primjeru se sada testiraju dodatne opcije ovog API-ja: ključne riječi i gramatika.

Očekivani način rada s definiranim ključnim riječima opisan je u potpoglavlju 3.2.1. Testiranjem na ovom primjeru utvrđeno je da ova opcija najbolje radi ako je postavljena razina osjetljivosti što manja. Primjerice, ako se za ključnu riječ definira riječ „girl" uz razinu osjetljivosti 0.1, aplikacija ispravno vraća jednu riječ „girl" (u svim rečenicama ta riječ se koristi samo jednom). Svaka veća razina osjetljivost rezultira s više *false-positive* prepoznavanja; za vrijednost 0.2 riječ „girl" se pronađe 3 puta, dok je za vrijednost 1 to čak 58 puta (iako sve rečenice zajedno imaju samo 39 riječi).

Za testiranje rada s gramatikom koristila se gramatika kao na slici 2.8, koja dopušta nizove brojeva jedan, dva i tri. Ova opcija radila je ispravno za nizove brojeva, ali samo s velikom pauzom između riječi. No, ukoliko je kao ulaz dan govor koji ne prati gramatička pravila (nema brojeva), alat će za svaku riječ pokušati naći najsličniju dopuštenu riječ. Tako je za uvodnih pet rečenica dobiven sljedeći rezultat.

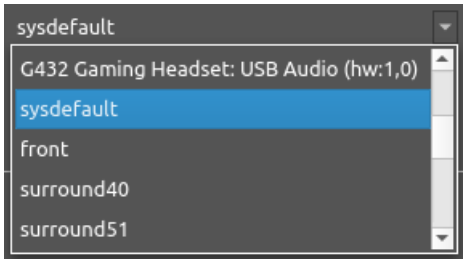
> one three two one two one two one two one two two two three one three three two two one two three two three two one three three two two three one one one three two two one two two one three two two two one three three three two

Na kraju treba spomenuti da je za snimku od pet rečenica koja traje 23 sekunde, prepoznavanje govora u prosjeku trajalo oko 5 sekundi.

3.4.2 Opcija „Snimi”

Većina opcija za slušanje s mikrofona je zajednička već opisanim opcijama za audio datoteku. Zato se ovdje stavlja naglasak na opcije specifične za govor uživo.

Kao što je ranije prikazano, po odabiru opcije „Snimi”, prvo je potrebno iz padajućeg izbornika izabrati mikrofon.



Slika 3.12: Odabir mikrofona

Na slici 3.12 prikazani su neki od ponuđenih mikrofona na testnom sustavu. Sve skupa je ponuđeno oko petnaest *mikrofona*, a samo dva predstavljaju stvarne mikrofone koji se mogu koristiti. Razlog tome je što funkcija `list_microphones` koristi širu definiciju mikrofona. U novom ogranku projekta SpeechRecognition je zato razvijena nova funkcija `list_working_microphones` koja dodatno filtrira dostupne mikrofone.

Preostala opcija specifična za ulaz s mikrofona je riječ okidač. Opcija je testirana za riječ okidač „Alexa”. Pripadni univerzalni model je preuzet sa Snowboy repozitorija [18]. Testiranjem je utvrđeno da ova opcija radi kao što je očekivano, iako nije potpuno pouzdana – riječ okidač je ponekad potrebno izreći više puta. Ovo je neobično, budući da alat Snowboy među korisnicima slovi kao izrazito kvalitetan. U ovom slučaju, mogući uzroci lošijih performansi su upotreba univerzalnog umjesto privatnog modela ili interna implementacija biblioteke SpeechRecognition. Naime, za korišteni model „Alexa”, u dokumentaciji su definirani optimalni parametri, koji se ne poklapaju s parametrima korištenim pri integraciji unutar biblioteke SpeechRecognition.

Naposljetku, slijedi primjer prepoznavanja govora na hrvatskom jeziku. Pritom je korišten Google Cloud API. Iz pomoćnih ispisa sa slike 3.13, mogu se iščitati sve odabrane postavke, dok je na slici 3.14 prikazan krajnji rezultat. Iz transkripta se može lako razumjeti glavna ideja teksta, ali trebat će još neko vrijeme da alati za prepoznavanje sustignu umijeće pisanja Branka Ćopića.

```
margarita@asusich:~/Desktop/skripta$ ./transkripta
['-i', 'mic', '-m', '6', '-st', '15', '-e', 'dynamic', '-a', 'google_cloud', '-l', 'hr-HR', '-p', 'kopalja', 'juriš', 'po šumi']
worker output: Done listening
worker output: staze puta ježurka ježić povazdan luta vam se bavi često ga vide 300 kopanja na juriš ide
```

Slika 3.13: Hrvatski jezik – parametri

staze puta ježurka ježić povazdan luta vam se bavi često ga vide 300 kopanja na juriš ide

Slika 3.14: Hrvatski jezik – rezultat

Zaključak

U sklopu ovog rada kao studijski primjer razvijena je Transkripta, desktop aplikacija za stvaranje tekstualnih zapisa na temelju govora. Aplikacija radi s dva tipa unosa, snimljenim i govorom uživo, a u oba slučaja nudi razne opcije za podešavanje. Spomenute opcije su podijeljene na osnovne, postavke zvuka te postavke jezika. Kao najvažnija opcija izdvaja se odabir vanjskih API-ja za prepoznavanje, od kojih Transkripta podržava čak četiri. Sva opisana funkcionalnost ostvarena je korištenjem biblioteke SpeechRecognition – od upravljanja mikrofonom/datotekom do obrade zvuka i dohvaćanja krajnjih rezultata. Pritom se biblioteka pokazala izrazito fleksibilnom i intuitivnom za korištenje, ponajviše zbog kompaktnog sučelja na visokoj razini apstrakcije. Međutim, ovu biblioteku nisu zaobišle klasične boljke projekata otvorenog koda, pri čemu se prvenstveno misli na kudikamo nekonzistentnu dokumentaciju (neki dijelovi ne odgovaraju trenutnoj verziji izvornog koda) te parcijalno zastarjelu implementaciju (implementacija ne prati razvoj pojedinih API-ja). Stoga se za ozbiljnije primjene ipak preporuča izravno korištenje pouzdanih API-ja, dok se biblioteka SpeechRecognition može smatrati idealnom za jednostavne primjene i testiranje.

Na kraju je demonstriran rad aplikacije na osnovnim primjerima. Testiranjem je pokazano kako kvaliteta transkripta uvelike varira ovisno o korisničkim postavkama. Najveće razlike se odnose na odabir API-ja pa su tako API-ji Google i Houndify dali zadovoljavajuće rezultate, dok se alat Sphinx pokazao dosta lošijim izborom. U sklopu daljnjeg razvoja aplikacije, bilo bi zanimljivo usporediti rezultate dobivene korištenjem novijih API-ja dostupnih preko nedavno postavljenog ogranka projekta SpeechRecognition. Također, samo korisničko iskustvo bi se moglo poboljšati uvođenjem višedretvenosti pri obradi duljih unosa te implementacijom transkripcije u stvarnom vremenu (*live-stream*). Takav pristup se ipak zaobilazi jer bi *lomljenje* govora na dijelove pokvarilo krajnji rezultat.

Bibliografija

[1] *IEEE Recommended Practice for Speech Quality Measurements*. IEEE Transactions on Audio and Electroacoustics, 17:225–246, 1969.

[2] *Audio Interchange File Format*. https://en.wikipedia.org/wiki/Audio_Interchange_File_Format, 2021. (pristupljeno 26. 1. 2022.).

[3] *Pulse-code modulation*. https://en.wikipedia.org/wiki/Pulse-code_modulation, 2021. (pristupljeno 26. 1. 2022.).

[4] Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu i Xiaoqiang Zheng: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015. https://www.tensorflow.org/, Software available from tensorflow.org.

[5] Alpha Cephei, Inc: *Vosk API*. https://github.com/alphacep/vosk-api/, 2019. (pristupljeno 3. 2. 2022.).

[6] Amos, David: *The effect of noise on speech recognition*. https://realpython.com/python-speech-recognition/#the-effect-of-noise-on-speech-recognition, 2018. (pristupljeno 29. 1. 2022.).

[7] Bacchiani, Michiel: *Google Research on End-to-End Models for Speech Recognition*. https://linedevday.linecorp.com/2020/en/sessions/9551, 2020. (pristupljeno 8. 2. 2022.).

[8] Chomsky, Noam i George A. Miller: *Finite state languages*. Information and Control, 1(2):91–112, 1958, ISSN 0019-9958. https://www.sciencedirect.com/science/article/pii/S0019995858900822.

[9] Daniel Jurafsky, James Martin: *N-gram Language Models*. https://web.stanford.edu/~jurafsky/slp3/3.pdf, 2021. (pristupljeno 9. 2. 2022.).

[10] Demonte, Philippa: *HARVARD speech corpus - audio recording collection*. https://doi.org/10.17866/rd.salford.c.4437578.v1, 2019. (pristupljeno 1. 2. 2022.).

[11] Google: *Speech API*. https://console.cloud.google.com/apis/api/speech-json.googleapis.com, 2021. (pristupljeno 2. 2. 2022.).

[12] Google: *Speech-to-Text API*. https://cloud.google.com/speech-to-text, 2021. (pristupljeno 2. 2. 2022.).

[13] IBM, Inc: *Watson Speech to Text*. https://www.ibm.com/cloud/watson-speech-to-text, 2022. (pristupljeno 3. 2. 2022.).

[14] Juang, B.H. i Lawrence R. Rabiner: *Automatic Speech Recognition – A Brief History of the Technologys*. 2004.

[15] Jyothi, Preethi: *Automatic Speech Recognition - An Overview*. https://www.microsoft.com/en-us/research/video/automatic-speech-recognition-overview/, 2017. (pristupljeno 7. 2. 2022.).

[16] Khalid, Muhammad Yasoob Ullah: *Context Managers*. https://book.pythontips.com/en/latest/context_managers.html#context-managers, 2017. (pristupljeno 26. 1. 2022.).

[17] Kincaid, Jason: *A Brief History of ASR: Automatic Speech Recognition*. https://medium.com/descript/a-brief-history-of-asr-automatic-speech-recognition-b8f338d4c0e5, 2018. (pristupljeno 5. 2. 2022.).

[18] KITT.AI: *Snowboy Hotword Detection*. https://github.com/Kitt-AI/snowboy, 2018. (pristupljeno 30. 1. 2022.).

[19] Li, Jinyu: *Recent Advances in End-to-End Automatic Speech Recognition*, 2022.

[20] Microsoft, Inc: *Azure Speech API*. https://azure.microsoft.com/en-us/services/cognitive-services/speech-services/, 2022. (pristupljeno 3. 2. 2022.).

[21] Miroslav Krleža, Leksikografski zavod: *fonologija. Hrvatska enciklopedija*. http://www.enciklopedija.hr/Natuknica.aspx?ID=20069, 2021. (pristupljeno 7. 2. 2022.).

[22] Mohammad, Zia: *Identity and Access Management Updates for Watson Services*. https://ziamohammad.medium.com/identity-and-access-management-updates-for-watson-services-12b6344b9cf, 2019. (pristupljeno 3. 2. 2022.).

[23] Panayotov, Vassil, Guoguo Chen, Daniel Povey i Sanjeev Khudanpur: *Librispeech: An ASR corpus based on public domain audio books*. U *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, stranice 5206–5210, 2015.

[24] Pham, Hubert: *PyAudio 0.2.11*. https://pypi.org/project/PyAudio/, 2017. (pristupljeno 27. 1. 2022.).

[25] Prazdnichnov, Dmitry: *Pocketsphinx*. https://github.com/cmusphinx/pocketsphinx, 2018. (pristupljeno 2. 2. 2022.).

[26] Riverbank Computing, Inc: *PyQt*. https://riverbankcomputing.com/software/pyqt/intro, 2021. (pristupljeno 12. 2. 2022.).

[27] seasalt.ai: *Snowboy Hotword Detection - Forked*. https://github.com/seasalt-ai/snowboy, 2018. (pristupljeno 30. 1. 2022.).

[28] sonix.ai, Inc: *A short history of speech recognition*. https://sonix.ai/history-of-speech-recognition. (pristupljeno 5. 2. 2022.).

[29] SoundHound, Inc: *Houndify*. https://www.houndify.com/dashboard, 2021. (pristupljeno 2. 2. 2022.).

[30] Stanley F. Chen, Joshua Goodman: *An Empirical Study of Smoothing Techniques for Language Modeling*. Harvard Computer Science Group Technical Report TR-10-98, 1998.

[31] Sun Microsystems, Inc: *JSpeech Grammar Format*. https://www.w3.org/TR/2000/NOTE-jsgf-20000605/, 2000. (pristupljeno 2. 2. 2022.).

[32] University, Carnegie Mellon: *CMUSphinx*. https://cmusphinx.github.io, 2019. (pristupljeno 2. 2. 2022.).

[33] Urmila Shrawankar, Vilas Thakar: *Techniques for Feature Extraction in Speech Recognition System: A Comparative Study*. https://arxiv.org/ftp/arxiv/papers/1305/1305.1145.pdf. (pristupljeno 7. 2. 2022.).

[34] Wit.ai, Inc: *wit.ai*. https://wit.ai/, 2022. (pristupljeno 3. 2. 2022.).

[35] Zhang, Anthony: *SpeechRecognition*. https://github.com/Uberi/speech_recognition, 2017. (pristupljeno 22. 1. 2022.).

[36] Zhang, Anthony: *SpeechRecognition Package 3.8.1*. https://pypi.org/project/SpeechRecognition, 2017. (pristupljeno 22. 1. 2022.).

Sažetak

U ovom radu predstavljena je biblioteka SpeechRecognition – popularna Python biblioteka za prepoznavanje govora. Na početku je dan povijesni pregled razvoja sustava za automatsko prepoznavanje govora, a posebno su istaknuta dva tipa najnovijih sustava. Nadalje, detaljno je opisana funkcionalnost biblioteke te njeno trenutno stanje razvoja. Integracija biblioteke je demonstrirana na primjeru vlastite desktop aplikacije za kreiranje glasovnih bilješki. Biblioteka se pokazala izrazito fleksibilnom i jednostavnom za korištenje, ali sa sobom nosi i određena ograničenja.

Summary

This thesis presents the SpeechRecognition library – a popular Python library for speech to text conversion. By way of an introduction, an historical overview of development of automatic speech recognition systems is given, with two types of the latest systems highlighted. Furthermore, the functionality of the library and its current state of development are described in detail. The integration of the library is demonstrated with a new desktop application for creating voice notes. The library proved to be extremely flexible and easy to use, but it also has certain limitations.

Životopis

Rođena sam 24. listopada 1996. godine u Dubrovniku, gdje sam pohađala osnovnu i srednju školu. Godine 2015. preselila sam se u Zagreb kako bih studirala na preddiplomskom sveučilišnom studiju Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu. Preddiplomski studij završila sam 2018. godine. Iste godine upisala sam diplomski sveučilišni studij Računarstvo i matematika.