Upute za rad s Markovljevim modelima u Matlabu uz diskusiju njihovih primjena u obradbi informacija

Ilustrirano na idetičnom primjeru koji se koristi u knjizi „Speech and Language Processing“, 3. izdanje,
autora: Dan Jurafsky i James H. Martin, sveučilište Stanford, SAD.

prof.dr.sc. Davor Petrinović
Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva
Zagreb

Inačica 0.9

Zagreb, studeni 2020.

Sadržaj

Upute za rad s Markovljevim modelima u Matlabu uz diskusiju njihovih primjena u obradbi informacija

Uvod

Radi lakšeg razumijevanja rada s HMM modelima i korištenja odgovarajućih funkcija iz pripadajućih programski biblioteka, potrebno je ipak objasniti razlike između tipova HMM modela. Dakle, pored HMM modela s diskretnim opservacijama koje smo obradili na predavanjima iz Obrade informacija, postoje i HMM modeli s kontinuiranim ili polu-kontinuiranim opservacijama. Programski paketi će tipično podržavati sve ove HMM modele, pa ćemo u par narednih poglavlja opisati razlike između pojedinih tipova HMM modela. Također, da bi prepoznali značaj ovih modela u brojnim inženjerskim primjenama, kao motivaciju za njihovo usvajanje, u ovom ćemo uvodu prikazati i način kako se ovi modeli koriste u sustavima za automatsko (ili strojno) prepoznavanje izgovorene riječi iz akustičkog zapisa, što se na engleskom jeziku naziva *Automatic Speech Recognition, ASR*. Primjeri takvih komercijalnih sustava koje većina vas danas koristi na brojnim platformama su *Siri* na Apple telefonima, *OK Google* i *Google Assistant* na *Android* platformama, *Alexa* od firme *Amazon* na brojnim ugradbenim računalnim sustavima koji su „*Alexa*-omogućeni“, te *Cortana* za Microsoft programskog okruženje kao i brojna druga manje poznata rješenja. Valja naglasiti da se ovi sustavi nisu pojavili „preko noći“, jer je u njihov razvoj uložen ogroman napor stotina svjetskih znanstvenika kroz period od barem zadnjih 30 godina. Više stotina znanstvenika je doktoriralo u ovom području te su objavljeni brojni znanstveni radovi u časopisima koji su posvećeni upravo ovoj užoj tematici primjene strojnog učenja u obradi prirodnog govora. Može se čak tvrditi da je jedan od najznačajnijih pokretača razvoja teorije strojnog učenja i podatkovne znanosti bilo upravo područje obrade prirodnog govora i prirodnog jezika. Ovi su teorijski pristupi kasnije primijenjeni i u drugim znanstvenim disciplinama, kao što je to uobičajeno kod širenja i preuzimanja znanstvenih spoznaja.

Tipovi osmotrenih podataka i njihova statistička reprezentacija u HMM modelima

U okviru uvodnog izlaganja na predmetu Obrada informacija bavili smo se isključivo modelima s diskretnim opservacijama, npr. boje bombona izvučene iz posuda, ili cijeli broj pojedenih sladoleda u nekom danu i slično. U oba slučaja vjerojatnosti osmatranja simbola u pojedinim stanjima modela možemo opisati histogramima, tj. prebrojavanjem koliko puta je osmotren svaki mogući simbol iz rječnika u pojedinom stanju modela za zadane nizove opservacija. U stvarnim primjenama, pa tako i u postupcima automatskog prepoznavanja govora, osmotrene veličine jesu i dalje diskretne u vremenu, ali nisu diskretnih vrijednosti. Također se ne radi o samo jednoj veličini koja je osmotrena (npr. boja ili broj), nego o cijelom nizu vrijednosti koje su karakteristične za trenutačna svojstva govornog signala u tom kratkom vremenskom trenutku tipičnog trajanja 10 do 20 milisekundi. Iako na predmetu nećemo stvarno naučiti kako pomoću HMM modela opisivati takve procese, čija su osmatranja iz skupa realnih vrijednosti R^N, u ovim uvodnim poglavljima ćemo ipak dati neke osnovne informacije kako se HMM modeli mogu izravno poopćiti i na takve procese s „kontinuiranim rječnikom“ simbola.

Vektori značajki

Skup veličina koji opisuje vremenski kratkotrajna svojstva govora, se naziva vektorom značajki. Brojni su autori predlagali raznovrsne reprezentacije koje trebaju svojstva govora kako ga mi doživljavamo pretvoriti u kompaktnu numeričku reprezentaciju s čim manjim brojem veličina za opis. Vektor značajki je dobro odabran ako je usklađen s psiho-akustičkim modelom percepcije zvuka. To znači da dva vektora značajki trebaju biti bliska ako pripadajući glasovi

nama zvuče slično i obrnuto, dva vektora trebaju biti udaljeni ako su glasovi nama bitno različiti. Danas je jedan od najčešće korištenih vektora značajki za primjene u obradi govora tzv. MFCC vektor, što je kratica od *Mel-Frequency Cepstral Coefficients*. Bez ulaženja u detalje njegovog izračuna, kažimo da on opisuje kratkotrajna spektralna svojstva govora (logaritam modula amplitudnog spektra), ali izračunatog iznad nelinearne frekvencijske skale, koja se još naziva Mel skalom, za razliku od obične DFT transformacije koja je definirana nad linearnom frekvencijskom skalom. U konačnici je takav log-spektar još i diskretnom kosinusnom transformacijom vraćen nazad u vremensku domenu, pa kao takav predstavlja svojevrstan spektar od log-amplitudnog spektra izračunatog iznad nelinearne frekvencijske skale. Činjenica da se ustvari radi o spektru od spektra je i razlog geneze njegovog izvornog naziva „***Ceps*** *tru*m“, koji nastaje preokretanjem prva četiri slova engleske riječi za spektar „***Spec*** *trum*“, a odatle i odgovarajući pridjev „***Ceps*** *tral*“ umjesto „***Spec*** *tral*“. U konačnici se pokazuje da je svega oko 12 kepstralnih koeficijenata dovoljno da vjerno opišu svojstva kratkotrajnog govornog odsječka trajanja od približno 20 do 30 ms. To predstavlja značajnu redukciju dimenzionalnosti, jer uz frekvenciju otipkavanja od 44.1 KHz, ovaj odsječak je imao skoro tisuću vremenskih uzoraka (točnije, 882), koji su ovom reprezentacijom sažeti u ovih 12 kepstralnih koeficijenata, koji vjerno opisuju naš doživljaj tog kratkog odsječka glasa. Također, ova reprezentacija je pogodna iz dodatnog razloga što je naš percepcijski doživljaj udaljenost između dva različita glasa vrlo dobro predviđen običnom Euklidskom udaljenosti između dva pripadajuća MFCC vektora.

Učestalost analize

Postupak analize se mora ponavljati češće nego što je širina korištenog otvora analize, a to znači da se vektori značajki tipično izračunavaju oko 100 puta u sekundi, dakle uz pomak okvira za analizu od 10 ms. To znači da uvijek postoji namjerno preklapanje susjednih okvira analize, kako bi svi detalji vremenske evolucije ovih značajki bili vjerno sačuvani. Konkretno, pri tipičnoj brzini izgovora potrebna je jedna sekunda da se izgovori broj „dvadeset i jedan“. Dakle ove dvije riječi povezane veznikom su sastavljene od 14 glasova i dvije pauze koji će u konačnici MFCC analizom biti opisani s približno 100 vektora značajki dimenzije 12, dakle s ukupno 1200 realnih brojeva. Ako bi broj vektora podijelili s brojem glasova (i pauzi), tada vidimo da u prosjeku svakom glasu pripada oko 6 uzastopnih vektora. Međutim neki glasovi, poput samoglasnika ipak traju natprosječno duže, pa će zapravo broj vektora koji opisuju pojedini glas biti ovisan o glasu: od jedno 3-4 vektora za kratke eksplozivne suglasnike, pa do čak desetak ili više vektora za samoglasnike produljenog izgovora.

Modeliranje vremenske evolucije vektora značajki

Dakle, sada kada imamo vektor značajki govora, moramo nekako opisati vjerojatnost osmatranja točno određene kombinacije ovih 12 realnih vrijednosti u svakom stanju modela. Skrivena stanja HMM modela će modelirati pojedinačne glasove koji se nadovezuju prilikom izgovora pojedine riječi ili pak prilikom izgovora samo jednog dijela riječi poput sloga. Potrebni broj stanja modela je ovisan o broju glasova u nizu, pa će modeliranje samo kraćih dijelova riječi smanjiti potrebni broj stanja modela, što će dati jednostavnije modele. U nastavku ćemo za opis takvog kratkog niza glasova koristiti pojam slog, iako se u stvarnim primjenama koriste parovi susjednih glasova, tzv. difoni, ili u složenijoj izvedbi, segmenti od tri glasa u nizu koje zovemo trifonima. Riječi prirodno nastaju nadovezivanjem glasova, pa ako svako stanje modelira jedan glas, ili tzv. fonem u nizu, cijeli HMM model će tada opisivati izgovor sloga (difona ili trifona) ili možda u složenijim modelima s većim brojem stanja čak i cijelu riječ.

Sada kada smo opisali sam model, moramo još opisati način modeliranja realnih vektorskih opservacija u pojedinim stanjima ovog modela. Jedan pristup rješavanju problema kako diskretne HMM modele ipak iskoristiti za reprezentaciju vremenske evolucije vektora značajki govora temelji se na pretvorbi ovog realnog vektora u indeks, čime se kontinuirana reprezentacija pretvara nazad u željenu diskretnu koju znamo modelirati, pa ćemo takav postupak opisati u narednom poglavlju.

Vektorska kvantizacija u svrhu diskretizacije realnog vektorskog procesa

Kao jednostavniji primjer zamislimo da našim HMM modelom želimo opisati vjerojatnost pripadanja muške osobe određenoj antropometrijskoj skupini, pri čemu će skrivena stanja modelirati vremenski tijek njegovog tjelesnog razvoja od dječje do odrasle dobi. Ako bi odabrali tri standardne varijable antropometrijskih karakteristika definiranih kao mjere: longitudinalnosti skeleta, mase tijela i voluminoznosti tijela, koje pak mjerimo kao visinu tijela u cm, težinu tijela u kg, te kao opseg podlaktice u cm, tada za svaku osobu možemo mjerenjem ovih veličina utvrditi njegovu poziciju u R^3 prostoru u svakom stanju ovog modela. Za očekivati je da se ovi vektori grupiraju u određenim regijama ovog prostora, jer su odabrane varijable međusobno zavisne. Također se može očekivati da su opisana grupiranja drugačija ovisno o stanju modela (dobi) i tipu osobe. Dakle, mogli bi sve osmotrene vektore podijeliti u konačan broj skupina. Kao pogodnu metriku za klasifikaciju mogli bi koristiti Euklidsku metriku u tom R^3 prostoru, ili alternativno težinsku Euklidsku metriku, ako pojedine komponente ovog vektora drugačije doprinose klasifikaciji. U oba slučaja, odluka da li pojedini vektor pripada pojedinoj klasi određena je s minimalnom (težinskom) udaljenosti tog vektora do reprezentanta svake pojedine klase.

Dva su temeljna problema koja treba ovdje riješiti: (1) kako za zadane reprezentante klasificirati nepoznati vektor i (2) kako odabrati broj klasa i kako naći najbolje reprezentante svake od tih klasa. Odgovor na ova pitanja daje teorija vektorske kvantizacije (engl. *Vector Quantization, VQ*) koja opisuje način rješavanja oba problema. Vektorska kvantizacija je zapravo poopćenje algoritma „*K-srednjih vrijednosti*“, (engl. *K-means*) koji rješava isti problem za skalarni slučaj tj. za slučaj da se klasifikacija provodi u R^1 prostoru. Teorija vektorske kvantizacije razvijena je u ranim 80-tim godinama prošlog stoljeća i to prvenstveno u kontekstu digitalne obrade signala, a specifično u kontekstu kodiranja signala u svrhu kompresije s gubitcima. Svi današnji koderi medijskih signala (govora, audio-signala, slike, video-signala itd.) su temeljeni upravo na postupcima vektorske kvantizacije, zbog njene visoke učinkovitosti sažimanja, ali i zbog numeričke jednostavnosti njene izvedbe. Razvojem podatkovne znanosti, danas prepoznajemo da je vektorska kvantizacija temeljena na paradigmi *kompetitivnog učenja*, pa je stoga blisko vezana s modelima *samo-organizirajućih mapa* (engl. *self-organizing maps*) i modelima *rijetkog kodiranja* (engl. *sparse coding*) koji se koriste u području „*dubokog učenja*“ (engl. *deep learning*), a specifično u *autoenkoderima*.

Zaključno, uz izgrađenu vektorsku kodnu knjigu, tj. uz optimalno određen skup reprezentirajućih vektora, svaki nepoznati vektor se temeljem kriterija najmanje udaljenosti može označiti indeksom klase kojoj pripada, čime smo dobili diskretne simbole iz konačnog rječnika. Odabrani broj reprezentanata jest upravo ta dimenzija rječnika (broj mogućih izlaznih simbola). Omeđene regije prostora koje pripadaju pojedinom reprezentantu imaju i posebno ime te se nazivaju Voroni regije, po ruskom matematičaru Гео́ргий Феодо́сьевич Воронóй; 1868 –1908 koji je izučavao ovo područje u devetnaestom stoljeću.

Mane diskretiziranih modela za opis opservacija

Mana prethodno opisanog postupka diskretizacije realnog vektora osmatranja je prvenstveno u činjenici da se dio informacije u tom postupku nepovratno gubi, jer vektorska kvantizacija, jednako kao i obična skalarna kvantizacija nisu invertibilne operacije. Ako se ulazni vektor nalazi točno na rubu između dvije klase, on će na kraju pripasti samo jednoj od te dvije klase, iako je zapravo podjednako udaljen i od jednog i od drugog reprezentanta te dvije klase, tj. zapravo ne pripada niti jednoj. Da smo odabrali veći broj klasa, pogreška bi možda bila manja, jer bi reprezentanti bili gušće raspoređeni po prostoru, ali će pogreška reprezentacije uvijek postojati, tj. iz indeksa pripadnosti nije moguće rekonstruirati izvornu realnu vrijednost ulaznog vektora, jer je ona kvantizacijom nepovratno zamijenjena s vrijednosti reprezentanta klase kojoj je vektor dodijeljen. Ovakvu klasifikaciju možemo smatrati „krutom“ odlukom, jer smo time zapravo definirali da je 100% vjerojatno da je vektor u toj klasi i 0% u svim drugim klasama, pod uvjetom da vektor pripada Voroni regiji te najbliže klase. Međutim nemamo informaciju u kojem dijelu Voronoi regije se vektor stvarno nalazi, da li je u sredini, ili možda na rubovima te ćelije. Za slučaj ulaznog vektora koji se nalazi u blizini poopćene simetrale između dvije klase, logičnija bi bila pretpostavka da su vjerojatnost pripadanja ovim klasama 50% i 50%, a ne 100% samo u jednoj, jer se vektor nalazio samo infinitezimalno bliže središtu baš te jedne odabrane.

HMM modeli s kontinuiranim opservacijama

Opisani problem koji je neizbježan kod diskretiziranih HMM modela rješava se korištenjem HMM modela koji imaju kontinuirane opservacije, čije su vjerojatnosti osmatranja realnog vektora opisane odabranim parametarskim modelima gustoća vjerojatnosti. Najpoznatiji takav parametarski model je *Gaussova razdioba definirana u N-dimenzionalnom prostoru*, jer je kompaktna u opisu, a istovremeno dobro prilagođena brojnim prirodnim procesima. Npr. za ranije opisan problem modeliranja antropometrijskih klasa, mogli bi iz cijelog skupa izmjerenih podataka izgraditi model Gaussove razdiobe u 3-dimenzionalnom prostoru. Takav model je opisan s relativno malim brojem parametara, a riječ je o vektoru očekivane vrijednosti (točka u R^3 prostoru u kojoj je ishodište ove razdiobe) i matrici kovarijance dimenzije 3x3 koja opisuje raspršenja u svim smjerovima, tj. oblik distribucije podataka. S obzirom da je ova matrica kovarijance simetrična, ukupno sadrži samo 6 unikatnih vrijednosti što zajedno s tri očekivane vrijednosti središta čini ukupno 9 slobodnih parametara ove gustoće.

Takav model nazivamo *Gaussovim modelom gustoće vjerojatnosti*, te se evaluacijom ove funkcije gustoće zadanih parametara lako može za svaki nepoznati vektor odrediti izvjesnost njegovog pripadanja baš toj gustoći, što će predstavljati emisijsku vjerojatnost HMM modela u pojedinom stanju. Dakle, mogli bi u HMM modelu svakom stanju pridružiti jednu takvu parametarski opisanu gustoću, čiji su parametri prilagođeni statistici te realne vektorske opservacije baš u tom stanju.

Analogno postupku treniranja diskretnih HMM modela, gdje smo iterativno korigirali matricu diskretnih opservacija B za svako stanje modela, u postupku treniranja kontinuiranih HMM modela treniranje se svodi na iterativno nalaženje optimalnih parametara Gaussovih gustoća u svakom od stanja modela. Ovakav parametarski opis rješava problem „krute“ odluke diskretiziranih opservacija, jer se udaljenost ulaznog vektora od središta gustoće izravno odražava u iznosu vjerojatnosti tog vektora. Što je bliži središtu, argument eksponencijalne funkcije će biti manji (bliži nuli) i njegova vjerojatnost veća, pa stoga kontinuirani HMM modeli mogu preciznije opisivati statistiku opservacija u pojedinim stanjima.

HMM modeli s Gaussovim mješavinama

Konačno, umjesto pridruživanja nezavisnih zasebno treniranih Gaussovih gustoća svakom stanju HMM modela, što osigurava najveću točnost modeliranja, postoji još jedna alternativna mogućnost, koja se naziva *polu-kontinuiranim HMM modelima* (engl. *semi-continuous HMMs*). Temeljena je na sličnoj pretpostavci na kojoj se temelji postupak diskretizacije opservacija pomoću vektorske kvantizacije. Dakle, mogli bismo ulazni proces kojeg modeliramo HMM-om podijeliti u M klasa, ali koje nisu opisane isključivo vektorom reprezentacije svake klase kao kod vektorske kvantizacije, nego su opisane kompletnom Gaussovom gustoćom. Dimenzionalnost prostora u kojem je definirana ta Gaussova gustoća određena je duljinom vektora opservacije. Dakle, opet za isti antropometrijski primjer, mogli bi se zapitati možemo li cjelokupni skup svih izmjerenih podataka (svaki podatak je vektor od tri broja) statistički opisati pomoću konačnog broja Gaussovih razdiobi u R^3 prostoru, gdje u ovom trenutku nećemo pokušavati ove klase povezivati sa stanjima modela (vremenskim tijekom razvoja osobe). Ovaj model ćemo izgraditi isključivo temeljem kriterija čim bolje statističke reprezentacije cjelokupnog podatkovnog skupa od strane modela.

Takav statistički model gdje se skup podataka ne opisuje pomoću jedne Gaussove gustoće nego pomoću kombinacije više takvih gustoća naziva se *modelom s Gaussovim mješavinama* (engl. *Gaussian Mixture Model, GMM*). Uz odabran broj komponenti M, moguće je postupkom učenja odrediti skup od M Gaussovih gustoća koje će najbolje reprezentirati cjelokupni podatkovni skup vektora. Ako se kao kriterij kvalitete GMM modela koristi log-izvjesnost osmatranja svih vektora podatkovnog skupa uz zadan model koji se sastoji od mješavine Gaussovih razdiobi, tada se postupak učenja optimalnih parametara komponenti te mješavine može ostvariti algoritmom *maksimizacije očekivanja* (engl. *Expectation Maximization, EM*). On je temeljen na istom teorijskom principu kao i iterativni postupak kojeg koristimo za gradnju optimalnih parametara HMM modela. Kod Gaussovih mješavina potrebno je definirati još jedan skalarni težinski faktor za svaku komponentu mješavine, jer je GMM model po definiciji obična težinska suma Gaussovih komponenti koje ga sačinjavaju. Da bi GMM i dalje bio funkcija gustoće, njegov integral preko cijele domene mora biti jednak jedinici, a to će biti ostvareno ako se ove težine pojedinih komponenti zbrajaju u jedinicu, jer svaka komponenta mješavine već ispunjava uvjet jediničnog integrala.

Odabirom ovih skalarnih težina dajemo odgovarajući značaj svakoj pojedinoj komponenti mješavine. Npr., ako prva komponenta ima najveću težinu, bitno veću od svih ostalih, tada će karakteristika tog GMM modela bili jako bliska svojstvima prve gustoće koja ju sačinjava. Ako su prve dvije težine podjednake, a ostale blizu nule tada takvom mješavinom zapravo opisujemo podjednaku kombinaciju prve dvije komponente i analogno tome za druge kombinacije težina. Opisano svojstvo nam omogućava, da u polu-kontinuiranim HMM modelima, umjesto povezivanja stanja s parametrima Gaussovih gustoća koje su zasebno optimirane za svako stanje, da zapravo povežemo stanja HMM modela samo s težinama pojedinih komponentni mješavine, a komponente ćemo izgraditi unaprijed temeljem cijelog podatkovnog skupa neovisno od stanja u kojem su osmotrene. Opisano pojednostavljenje bitno reducira broj parametara HMM modela, jer sada umjesto opisa cijele Gaussove gustoće u svakom stanju modela imamo samo skup težina (značaja ili važnosti) s kojima su pojedine gustoće osmotrene u tom stanju, a sva stanja dijele isti skup komponenti koji je unaprijed treniran nad cijelim podatkovnim skupom. Odale proizlazi i naziv ovog modela, „*polu-kontinuiran model*“ jer se radi o diskretnom broju mogućih Gaussovih komponenti, koje se samo težinski kombiniraju u pojedinim stanjima HMM modela. Ove težine mogu biti bilo koji

skup pozitivnih realnih brojeva koji zadovoljava uvjet da se zbrajaju u jedinicu. Odabirom težina mi možemo kompaktno opisivati emisijske vjerojatnosti pojedinih stanja, pa se u postupku učenja HMM modela isključivo ažuriraju ove težine za svako stanje modela kako bi ukupna izvjesnost osmatranja tog niza realnih vektora značajki bila najveća moguća. Parametri samih Gaussovih komponenti se prilikom treniranja HMM-a ne diraju, jer su oni već unaprijed određeni pri gradnji GMM modela nad cijelim podatkovnim skupom opservacijskih vektora.

Treniranje HMM modela u stvarnim primjenama

U stvarnosti se HMM modeli treniraju ne samo s jednim takvim nizom vektora značajki, nego s višestrukim nizovima osmatranja potencijalno različitih duljina, koji svi pripadaju istoj klasi modela, temeljem „ručno označenih“ ulaznih podataka. Taj se postupak ručnog označavanja još naziva anotacijom i često traži iznimno velik ljudski napor u pripremi takvih referentnih oznaka. Često se isti materijal paralelno anotira od stane većeg broja nezavisnih anotatora kako bi se međusobnom usporedbom moglo nedvosmisleno potvrditi konačna valjana oznaka (engl. *ground truth*). U svim velikim IT korporacijama koje nude usluge sustava za automatsko prepoznavanje govora, ovu zadaću ručne anotacije provode čitave „vojske“ educiranih ljudskih slušača, koji moraju preslušati i označiti sate i sate takvih govornih materijala u svrhu proširenja podatkovnog skupa koji će se koristiti pri retreniranju modela. Upravo se u tim postupcima u zadnje vrijeme naglašava problem nedovoljne zaštite osobnih podataka, jer često osobe čiji se govorni materijali (snimke) koriste nisu bile svjesne da ih se snima, ili nisu dale izravnu suglasnost da se njihovi govorni zapisi koriste u tu svrhu gdje će ih preslušavati „žive osobe“, a ne samo računalo (robot).

U kontekstu prepoznavanja sloga u svrhu automatskog prepoznavanja prirodnog vezanog govora, gdje jedan HMM model opisuje jedan slog (u stvarnosti se radi o difonu ili trifonu), skup za treniranje modela se sastoji od višestrukih nizova MFCC vektora prikupljenih pri izgovoru istog sloga od jedne ili više različitih osoba, u svim mogućim riječima gdje se taj isti slog javlja. U slučaju višestrukih nizova opservacija, kriterij korišten za treniranje modela jest da umnožak izvjesnosti osmatranja svih ovih nizova bude maksimalan za tako optimalno određene koeficijente modela. Pri formiranju skupa za učenje potrebno je osigurati dovoljnu reprezentativnost, jer će izgrađen model biti optimiran upravo na ovaj korišten skup. Ako se pri stvarnom korištenju modela u svrhu određivanja klase nepoznatog niza vektora značajki pojavi neki novi niz koji prema „ručnoj oznaci“ i dalje pripada ovoj klasi, ali koji se bitno razlikuje od nizova koji su korišteni za trening modela, tada se može očekivati da će izvjesnost te opservacije biti bitno manja od izvjesnosti nizova koji su korišteni za učenje modela, što može uzrokovati pogrešnu odluku. Kao primjer, ako su za treniranje modela korišteni izgovori samo muških odraslih govornika, a u stvarnom se korištenju pri evaluaciji modela koriste i izgovori ženskih ili dječjih govornika, tada se mogu očekivati bitno lošiji rezultati prepoznavanja, jer model nije izgrađen na reprezentativnom uzorku. Pri stvarnim postupcima gradnje ovih HMM modela, primjenjuju se slični postupci kros-validacije kao i u drugim postupcima strojnog učenja (razdvajanjem ukupne baze na skup za učenje i posebni skup za evaluaciju, te uzastopnim ponavljanjem ovog razdvajanja slučajnim odabirom dijela baze za učenje i dijela za evaluaciju uz definiran omjer, tipično 90% za učenje i 10% za evaluaciju). Postupcima kros-validacije moguće je verificirati je li skup za učenje dovoljno reprezentativan, jer ako bilo koja slučajna podjela dade podjednaku točnost na skupu za evaluaciju, to ukazuje da je skup za trening dovoljno reprezentativan. Međutim takav zaključak je valjan samo u slučaju da je taj cijeli podatkovni skup uistinu reprezentativan za očekivanu primjenu, jer ako

se u primjeni jave novi nizovi osmatranja koji ni u kojem obliku nisu bili sadržani u ovom skupu, ponašanje sustava će biti potpuno nepredvidivo.

Prepoznavanje sloga i povezivanje modela slogova u modele riječi i rečenica

U konačnici moramo objasniti i način kako da računalo razlikuje različite slogove, tj. kako se donosi konačna odluka o nepoznatom osmotrenom nizu vektora značajki. Princip se temelji na izgradnji niza paralelnih HMM modela za sve slogove koje je potrebno prepoznati za zadan skup riječi koje će sustav biti sposoban prepoznavati. Nepoznat niz se istovremeno evaluira sa svim tim unaprijed treniranim modelima, te se odabire onaj model koji daje najveću izvjesnost osmatranja tog nepoznatog niza. Ovi se modeli kontinuirano i paralelno evaluiraju te se prate njihove „aktivacije“, tj. koji model u kojem trenutku daje najveću izvjesnost. Slijed aktivacija ovih modela ukazuje na moguć niz izgovorenih slogova, koji onda formiraju riječi, pa zatim riječi formiraju rečenice.

Za modeliranje ovih viših hijerarhijskih struktura opet se mogu koristiti skriveni Markovljevi modeli, samo što oni tada modeliraju uvjetne vjerojatnosti nizova slogova unutar riječi ili uvjetne vjerojatnosti nizova riječi unutar rečenice. To je posebno važno u slučaju kada je točnost ove najniže akustičke razine prepoznavanja niza glasova relativno niska. Nažalost, to najčešće i jest slučaj, upravo zbog jako velike akustičke varijabilnosti izgovora, pa je iznimno teško s visokom sigurnosti utvrditi koji model je onaj pravi, a pogotovo kada je broj takvih paralelnih HMM modela iznimno velik. Konkretno, ako kao primjer uzmemo hrvatski jezik s približno 30 fonema, tada bi ukupan broj različitih trifona bio 27000 = 30*30*30. Međutim mnogi od ovih trifona ne postoje u stvarnom jeziku, najčešće iz razloga što nisu izgovorljivi, pa je u stvarnosti broj takvih paralelnih HMM modela trifona hrvatskog jezika oko 9000. Dakle, računalo treba za osmotreni niz evaluirati izvjesnost tog nepoznatog osmatranja uz svih 9000 izgrađenih modela i odabrati onaj model koji daje najveću izvjesnost. Pošto je vjerojatnost pogrešnog odabira s tolikim brojem modela nezanemariva, umjesto samo jednog najboljeg kandidata, tj. umjesto samo jednog najboljeg HMM modela prepoznatog sloga koji se trenutačno aktivirao, u više razine se prosljeđuje lista s više mogućih kandidata zajedno s njihovim evaluiranim akustičkim izvjesnostima osmatranja. Konačna odluka o „pravom“ prepoznatom slogu se prepušta višoj razini modela, koji opisuje uvjetne vjerojatnosti nastavljanja slogova jednog na drugi, te se u konačnici odabire ona kombinacija iz nižih slojeva koja s najvećom izvjesnosti formira valjanu riječ iz korištenog rječnika svih mogućih riječi.
Isti princip se zatim koristi i za najvišu hijerarhijsku razinu koja od takvih nizova višestrukih kandidata valjanih riječi bira one kandidate čija je izvjesnost osmatranja najviša, a pri tome uvažavajući model valjane leksičke strukture rečenice, što se još naziva i *modelom jezika*. Kombinacijom ovih višestrukih razina prepoznavanja ostvaruje se ukupna točnost rada koja je danas već iznimno visoka. Komercijalni sustavi za prepoznavanje govora danas imaju točnosti od barem 95% ili više, što znači da od 20 izgovorenih riječi u nizu možda pogriješe samo u jednoj, što ih čini upotrebljivim u brojnim svakodnevnim primjenama. Pri tome točnost prepoznavanja na najnižoj akustičkoj razini ne mora nužno biti jednako visoka, jer upravo dodavanje ovih viših hijerarhijskih slojeva osigurava konačnu točnost rada sustava. Tipično je točnost prepoznavanja na akustičkoj razini oko 60-70%, što znači da se samo u dva od tri slučaja u konačnici odabire upravo model sloga koji je imao najveću izvjesnost osmatranja akustičke razine, dok se u preostaloj trećini slučaja odabiru alternativni kandidati koji su niže na sortiranoj listi modela prema izvjesnosti akustičkog osmatranja.

Algoritamska latencija sustava za prepoznavanje prirodnog govora

Opisana hijerarhijska struktura odlučivanja ASR sustava za prepoznavanje prirodnog povezanog govora ima i jednu neželjenu posljedicu, a to je neizbježno kašnjenje (engl. *latency*) sustava. Konačna odluka o prepoznatom prvom slogu u prvoj riječi u izgovorenoj rečenici bit će možda donesena tek kada računalo „sasluša“, prikupi i obradi cijelu rečenicu, jer svi hijerarhijski slojevi moraju evaluirati uvjetne vjerojatnosti lijevog i desnog konteksta te od svih višestrukih kandidata s proslijeđenih lista mogućih kandidata odabrati one kombinacije koje će proizvesti najveću izvjesnost osmatranja na razini cijele izgovorene rečenice. Na sličan način i ljudi u mozgu obrađuju govornu poruku koju su čuli. Iako možda zbog okolne buke nisu razumjeli baš svaku izgovorenu riječ sugovornika, ipak će kada saslušaju cijelu rečenicu uspjeti rekonstruirati izvornu izgovorenu poruku. Često će i ljudi u takvim uvjetima narušene komunikacije trebati kratko „promisliti“ što je sugovornik mogao izgovoriti, na sličan način kombinirajući uvjetne vjerojatnosti lijevog i desnog konteksta za pojedinačne riječi koje nisu mogli razumjeti zbog okolne buke, ali i uvažavajući kontekst započetog razgovora. Pri korištenju sustava za automatsko prepoznavanje govora, korisnik ne smije biti iznenađen takvim neminovnim algoritamskim kašnjenjem sustava, te u nedoumici zašto odmah nisu prikazane prepoznate izgovorene riječi, odmah nestrpljivo višestruko ponavljati upit ili dijelove upita, jer će takvo ponavljanje samo otežati ili čak onemogućiti ASR sustavu uspješno prepoznavanje rečenice.

Kontekstno prilagođeni sustavi prepoznavanja govora

U nekim primjenama moguće je značajno smanjiti veličinu rječnika svih mogućih valjanih riječi. Konkretno ako se radi o užem kontekstu primjene ASR sustava (npr. automatska transkripcija financijskog izvještaja o stanju na burzama), tada je zapravo ukupan broj korištenih različitih riječi relativno skroman (od nekoliko stotina do najviše nekoliko tisuća). Time se automatski smanjuje i broj potrebnih modela slogova (trifona), jer će možda u tom podskupu valjanih riječi pola modela biti nepotrebno, jer se nikada ne javljaju kao dio bilo koje valjane riječi u korištenom rječniku. Time se smanjuje takozvana *perpleksnost* problema (engl. *perplexity*), što će automatski osigurati veću točnost prepoznavanja, jer je broj mogućih ishoda između kojih treba odabrati pravi bitno manji. Također ovisno o kontekstu primjene, model jezika, odnosno model strukture rečenice može biti vrlo specifičan. Točno određene konstrukcije uobičajenih rečenica se koriste za meteorološki izvještaj i vremensku prognozu u odnosu na rečenice koje će politički komentator koristiti u svojem prilogu. Zbog toga najmoderniji sustavi za automatsko prepoznavanje govora pokušavaju samostalno odrediti kontekst trenutačnog korištenja, pa zatim bez potrebe za intervencijom korisnika biraju modele koji su najprimjereniji i posebno izgrađeni upravo za taj automatski identificiran kontekst (npr. usluga narudžbe za termin u frizerskom salonu, ili npr. rezervacija stola u restoranu).

Iz svega opisanog je vidljivo da se u stvarnoj primjeni zapravo radi o brojnim HMM modelima koji svi moraju biti relevantno utrenirani, kako bi njihova odluka generalizirala i za potpuno nova i nepoznata osmatranja koja se mogu javiti pri korištenju sustava. Iz tog razloga je razumljiva i prednost polu-kontinuiranih HMM modela, jer imaju bitno manji broj „slobodnih parametara“ modela, što automatski pospješuje i postupke učenja. Za relevantno učenje modela, broj korištenih opservacija uvijek mora biti nekoliko redova veličine veći od ukupnog broja parametara modela, pa je to glavni razlog zašto se toliki ljudski i financijski napor ulaže u kontinuiranu nadogradnju podatkovnih skupova koji se koriste za ponovno učenje ovih

modela. Iz izloženog, možemo potvrditi da je sustav za automatsko prepoznavanje govora uistinu potpuno izgrađen i definiran podatcima (engl. *data driven*), jer algoritam sam za sebe je kao prazna knjiga s „bijelim stranicama “, koja tek temeljem stvarnog sadržaja naučenog iz ulaznih podataka postaje „uspješnica“.

Alat korišten za vježbe s HMM modelima

S obzirom da skriveni Markovljevi modeli pripadaju području strojnog učenja u klasu algoritama za nenadzirano učenje, a u Matlabu je strojno učenje podržano u posebnom paketu „*Statistics Toolbox*“, unutar njega je specifično za potrebe rada s HMM modelima uključeno pet temeljnih funkcija:

hmmdecode	Hidden Markov model posterior state probabilities
hmmestimate	Hidden Markov model parameter estimates from emissions and states
hmmgenerate	Hidden Markov model states and emissions
hmmtrain	Hidden Markov model parameter estimates from emissions
hmmviterbi	Hidden Markov model most probable state path

Međutim, umjesto ove „vlastite“ Matlab izvedbe HMM funkcija, za potrebe laboratorijske vježbe iz Obradbe informacija koristit će se jedan drugi Toolbox koji je sveobuhvatniji i koji je specifično napisan baš za podršku HMM modela u okviru Matlab-a, čak i prije nego što je Matlab razvio vlastitu podršku. Riječ je o paketu pod nazivom:

Hidden Markov Model (HMM) Toolbox for Matlab
Written by Kevin Murphy, 1998.
Last updated: 8 June 2005.
Distributed under the MIT License

Paket je dostupan na ovoj poveznici
The University of British Columbia, UBC, Vancouver Campus

https://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html

U kratkom opisu ovog paketa navode se njegove ključne funkcionalnosti, što bi vam temeljem uvodnog poglavlja u kojem smo opisali raznovrsne tipove HMM modela trebalo biti jasno:

„This toolbox supports inference and learning for HMMs with discrete outputs (dhmm's), Gaussian outputs (ghmm's), or mixtures of Gaussians output (mhmm's). The Gaussians can be full, diagonal, or spherical (isotropic). It also supports discrete inputs, as in a POMDP. The inference routines support filtering, smoothing, and fixed-lag smoothing.“

Ovaj paket podržava puno širi skup funkcionalnost za rad s HMM modelima u odnosnu na uvodni opis koji je prezentiran na predavanjima, pa ćemo u sklopu ove vježbe istražiti rad samo s par temeljnih funkcija ovog paketa u svrhu modeliranja HMM modela s diskretnim opservacijama.

Preuzimanje i instalacija programskog paketa „HMM Toolbox“

Ovaj Toolbox se može preuzeti kao jedna ZIP datoteka na sljedećoj poveznici:

https://www.cs.ubc.ca/~murphyk/Software/HMM/hmm_download.html

Number of hits since 23 October 2002:

Download
Click here. Unziping creates a directory called HMMall, which contains 4 subdirectories.

Puna poveznica (iza teksta 'here') za preuzimanje ZIP datoteke je:
https://www.cs.ubc.ca/~murphyk/Software/HMM.zip

Instalacija HMM Toolbox-au Matlab

Ista ova ZIP datoteka učitana je i u materijale za 6. i 7. predavanje na web stranici predmeta OI. U svrhu instalacije potrebno je raspakirati ovu ZIP arhivu u folder naziva HMMall koji se može smjestiti bilo gdje u datotečni sustavu vašeg računala, ali je najbolje da ga stavite u radni folder Matlaba u kojem će biti i vaše preostale izvorne datoteke. Matlab mora imati pristup u folder gdje se nalaze svi programi koji su sastavni dio ovog Toolbox-a. Dodavanje tog novog foldera u putanju pretraživanja datoteka korištenih od strane Matlab-a možete ostvariti ovom naredbom niže (u primjeru je pretpostavljeno da se mapa HMMall nalazi u korijenu C diska (C:/), no ona može biti bilo gdje u stablu, samo tada morate pravilno navesti put do tog foldera kao argument funkcije genpath).

Assuming you unzip it to C:/HMMall...
>> addpath(genpath('C:/HMMall'))

Za brzu provjeru je li Toolbox pravilno instaliran izvršite ovu skriptu:
>> testHMM

Sada možemo demonstrirati najvažnije funkcije iz ovog Toolboxa na identičnom primjeru modeliranja vremenskih prilika temeljenog na zapisima broja pojedenih sladoleda u pojedinom danu kojeg smo koristili kao podlogu na predavanjima. Sva tri temeljena zadatka vezana uz HMM modele (izračun izvjesnosti, dekodiranje i učenje modela) bit će demonstrirana pozivima odgovarajućih funkcija iz ovog Toolboxa.

Primjer rada s programski paketom „HMM Toolbox“ u Matlabu

Pokušat ćemo ovim narednim primjerima točno pratiti izlaganje u dokumentu kojeg smo koristili na predavanjima iz OI i to točno za identičan primjer koji je korišten za ilustraciju rada diskretnih HMM modela. Ostvareni rezultati izvođenja ovih ključnih funkcija će zatim biti uspoređeni s očekivanim rezultatima koji su prikazani u materijalima za predavanja.

Izračun izvjesnosti opservacije za zadani model

Krenimo prvo od definiranja parametara modela. Iako su u modelu vremenskih prilika skrivena stanja označena slovima C za hladno (Cold) i H za vruće (Hot), u Matlabu stanja moramo indeksirati brojevima, pa odabiremo:

```
% Oznacavanje stanja
% Stanje C(old) cemo oznaciti kao prvo stanje
% Stanje H(ot) kao drugo stanje
%
% Vektor inicijalne vjerojatnosti stanja
%
prior0=[
    0.2  % Stanje C
    0.8  % Stanje H
];
```

Na sličan način definiramo i tranzicijsku matricu A koja definira vjerojatnosti promjene stanja, gdje prvi redak odgovara stanju C i njegovim vjerojatnostima prijelaza u stanje C i H, dok drugi redak odgovara stanju H i njegovim prijelazima u stanje C i H:

```
% Matrica vjerojatnosti promjena stanja
%
% a11 a12
% a21 a22
%
transmat0=[
0.5 0.5      % P(C|C) P(H|C)
0.4 0.6      % P(C|H) P(H|H)
];
```

Suma svakog retka mora biti jednaka jedinici, jer se radi o razdiobi vjerojatnosti. Uvodimo i varijablu Q koja će označavati broj stanja modela, jer će biti korištena kao argument u nekim funkcijama:

```
Q=size(prior0,1);
```

Konačno definiramo i matricu emisijskih vjerojatnosti B, tj. za svako stanje definiramo vjerojatnost osmatranja svih izlaznih simbola iz rječnika. Naš izlazni rječnik ima samo tri simbola „1“, „2“ i „3“ koji označavaju broj pojedenih sladoleda u danu. Da su izlazni simboli bile boje izvučenih bombona iz bijele ili zelene posude, kao što smo ilustrirali u živom pokusu na početku prvog predavanja, tada bi svakoj boji morali pridružiti jedan numerički indeks jednako kao što smo indeksirali i skrivena stanja H i C, jer funkcije u Matlabu podržavaju samo indeksirane (numeričke) simbole rječnika.

```
% Matrica emisijskih vjerojatnosti
% svaki redak odgovara jednom stanju, a
% svaki stupac jednoj mogucoj opservaciji
obsmat0=[
0.5 0.4 0.1   % P(1|C) P(2|C) P(3|C)
0.2 0.4 0.4   % P(1|H) P(2|H) P(3|H)
];
O=size(obsmat0,2);
```

Varijabla O će predstavljati ukupni broj simbola rječnika, što u našem primjeru iznosi 3 (jedan, dva ili tri pojedena sladoleda).

Sada definiramo i jedan konkretni niz opservacija, za kojeg ćemo pokušati odrediti njegovu izvjesnost osmatranja uz zadani model korištenjem algoritma „Unaprijed“. Odabrat ćemo identični niz tri osmotrena simbola „1 3 1“ u vremenskim trenutcima t=1, t=2 i t=3, koji je korišten i u materijalima za predavanja.

```
data=[3 1 3];
```

Duljina ovog opservacijskog niza T može biti proizvoljna, a kao što je opisano u uvodnim poglavljima, općenito možemo imati i višestruke nizove opservacija različitih duljina koji svi pripadaju istom HMM modelu pa korištenje višedimenzionalnih polja u Matlabu za njihovu pohranu nije pogodno. Takva podatkovna struktura očekuje fiksnu (identičnu) dimenziju svih nizova osmatranja po svim „osima“. Umjesto višedimenzionalnih polja može se u Matlabu koristiti „cell“, jer ova podatkovna struktura u svakoj ćeliji može sadržavati numerički element proizvoljne dimenzionalnosti (i tipa). Npr. u prvoj ćeliji može biti redak sa 10 elemenata, u drugoj ćeliji stupac od 17 elemenata i konačno u trećoj ćeliji matrica 3x7, itd. Ova „cell“ podatkovna struktura je stoga puno fleksibilnija i omogućava da u svakoj ćeliji bude redak s nizom opservacija proizvoljne duljine T. Ovaj jedan definirani redak 'data' stoga pretvaramo u takvu cell strukturu ovim nizom naredbi:

```
if ~iscell(data)
  data = num2cell(data, 2);
end
ncases = length(data);
```

Varijabla 'ncases' će sadržavati broj opservacijskih nizova u novoj cell strukturi 'data', a za naš jednostavni primjer to će biti 1, jer smo zadali samo jedan opservacijski niz.

Sada ćemo za ove zadane opservacijske simbole i za zadan model izvršili naredni Matlab isječak koji u petlji obrađuje jedan po jedan opservacijski niz i izračunava njegovu izvjesnost osmatranja za zadani model. Ova se izvjesnost $P(O|\lambda)$ uobičajeno logaritmira s prirodnim logaritmom (baze e), te se u tom obliku naziva log-izvjesnost opservacije O (engl. log-likelihood). Korištenje logaritma je značajno, jer povećanjem duljine osmotrenog niza T izvjesnost $P(O|\lambda)$ u pravilu poprima vrlo male vrijednosti. U algoritmu *Unaprijed* se množe vjerojatnosti koje su sve brojevi manji od jedinice, te se takvi mali umnošci zbrajaju i opet prosljeđuju u naredni vremenski trenutak u skladu s indukcijom algoritma *Unaprijed*, što opet rezultira njihovim dodatnim umanjenjem. Ako umjesto jednog niza O imamo dva niza O1 i O2 za koje pretpostavljamo da pripadaju istom HMM modelu λ tada možemo njihovu izvjesnost istovremenog osmatranja definirati kao umnožak $P(O1|\lambda)*P(O2|\lambda)$, što u slučaju reprezentacije korištenjem log-izvjesnosti odgovara sumi dvije log-izvjesnosti

$$\log(P(O1|\lambda) * P(O2|\lambda)) = \log(P(O1|\lambda)) + \log(P(O2|\lambda))$$

Odsječak u nastavku će upravo izračunavati (akumulirati) takvu log-izvjesnost kako bi odredio ukupnu log-izvjesnost cijelog skupa opservacijskih nizova proizvoljnih duljina koji su pohranjeni u „cell“ strukturi 'data'.

```
loglik = 0;
errors = [];
```

```
for m=1:ncases
  obslik0 = multinomial_prob(data{m}, obsmat0);
  [alpha, beta, gamma, ll] = ...
      fwdback(prior0, transmat0, obslik0, 'scaled', 0);
  if ll==-inf
    errors = [errors m];
  end
  loglik = loglik + ll;
end
```

Kako bi lakše razumjeli način izračuna log-izvjesnosti promotrimo prvi (i jedini u ovom slučaju) prolaz ove petlje za prvi niz opservacija duljine tri simbola. Ako fiksiramo indeks petlje na m=1:

```
>> m=1

m =

   1
```

Ovo je sadržaj te prve (i jedine) ćelije strukture 'data':

```
>> data{m}

ans =

   3   1   3
```

Sada izvodimo samo prvu naredbu iz gornje petlje:

```
>> obslik0 = multinomial_prob(data{m}, obsmat0)

obslik0 =

  0.1000   0.5000   0.1000
  0.4000   0.2000   0.4000
```

Vidimo da matrica obslik0 ima T (=3) stupca i Q (=2) retka i da sadrži vjerojatnosti osmatranja odabranog niza „3 1 3“ i to u prvom stanju modela (u prvom retku matrice za stanje C) i u drugom stanju modela (u drugom retku matrice za stanje H). To su ove uvjetne vjerojatnosti:

```
% P(3|C) P(1|C) P(3|C)
% P(3|H) P(1|H) P(3|H)
```

Kako se radi o HMM modelu s diskretnim izlaznim vjerojatnostima (jer imamo konačni broj izlaznih simbola), ova funkcija ('multinomial_prob') jednostavno proziva vrijednosti izlazne matrice vjerojatnosti B (koju smo nazvali 'obsmat0') temeljem indeksa osmotrenih simbola za konkretni zadani niz opservacija u polju 'data{1}'. Time smo numeričke indekse izlaznih simbola pretvorili u pripadajuće vjerojatnosti osmatranja za svako stanje modela i za sve

vremenske trenutke t=1 do t=T=3. Ova pomoćna matrica 'obslik0' će zatim biti korištena kao ulazni argument u funkciju '*fwdback*' koja će implementirati algoritam „*Unaprijed*“.

Algoritam „*Unaprijed*“

```
[alpha, beta, gamma, ll] = ...
    fwdback(prior0, transmat0, obslik0, 'scaled', 0);
```

Ova funkcija u izlaznoj matrici 'alpha' vraća izračunate unaprijedne vjerojatnosti za svako skriveno stanje modela α_t(stanje) za svaki trenutak t=1 do T, gdje redci opet odgovaraju stanjima, dok stupci odgovaraju vremenskim trenutcima osmatranja. Uočite da ista funkcija može temeljem istog skupa ulaznih argumenata izračunati i unazadne vjerojatnosti HMM modela β_t(stanje) za t=T, T-1, … do 1, te ih vratiti u izlaznoj matrici 'beta'. Konačno vraća i matricu 'gamma' koja se izračunava iz unaprijedne i unazadne vjerojatnosti svakog čvora rešetke, a koja se koristi u algoritmu učenja parametara HMM modela u tzv. trećem HMM zadatku.
Pogledajmo što smo dobili u izlaznoj matrici 'alpha':

```
>> format long
>> alpha

alpha =

   0.020000000000000   0.069000000000000   0.005066000000000
   0.320000000000000   0.040400000000000   0.023496000000000
```

Usporedimo ove vrijednosti s vjerojatnostima koje su prikazane na slici A.5 u materijalima za predavanja koja ilustrira primjenu algoritma „*Unaprijed*“ na identičnom primjeru. Prepoznajemo da prvi redak matrice odgovara skrivenom stanju C(old) koje je indeksirano kao stanje 1, dok drugi redak odgovara stanju H(ot) koje je indeksiran kao drugo stanje. Svaki stupac ove matrice odgovara pojedinom vremenskom trenutku t=1, t=2 i t=T=3. Vidimo da u drugom stupcu ove matrice dobivamo unaprijedne vjerojatnosti $\alpha_2(1)$=0.069 i $\alpha_2(2)$=0.0404 koje su ručno izračunate i označene na slici A.5. Da je opservacijski niz bio ograničen samo na prva dva simbola, tada bi ukupna izvjesnost osmatranja te skraćene sekvence bila upravo jednaka sumi ove dvije vjerojatnosti P(„3 1“|λ) = $\alpha_2(1)+\alpha_2(2)$ = 0.069 + 0.0404 = 0.1094

Zbrajanjem unaprijednih vjerojatnosti preko svih skrivenih stanja (tj. preko svih stupaca matrice) dobivamo izvjesnosti osmatranja svih skraćenih nizova observacija: „3“, „3 1“ i konačno „3 1 3“ za cijelu sekvencu.

```
>> sum(alpha)

ans =

   0.340000000000000   0.109400000000000   0.028562000000000
```

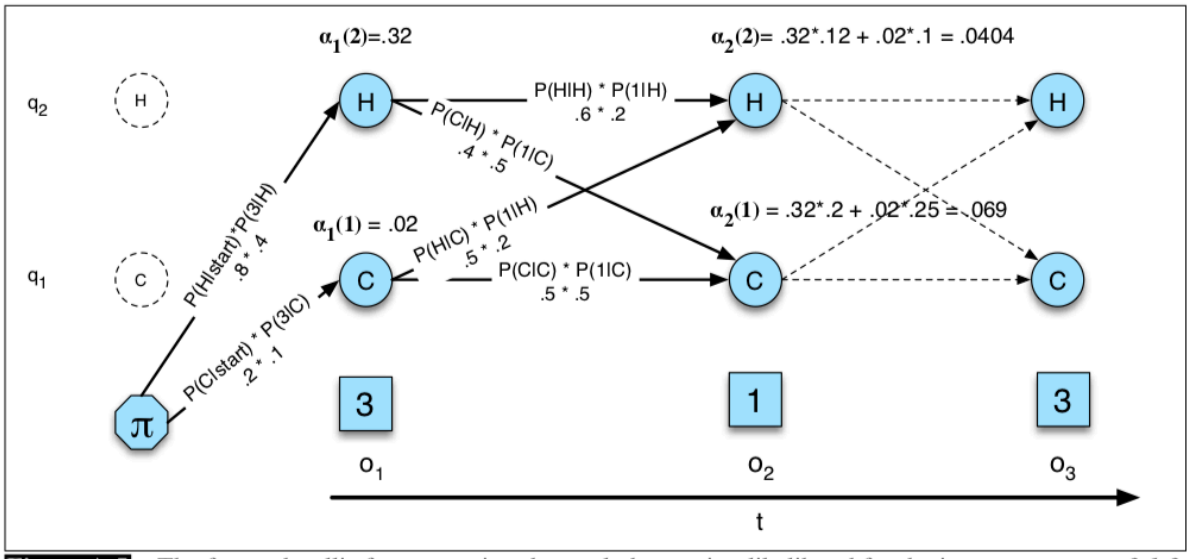


Figure A.5 The forward trellis for computing the total observation likelihood for the ice-cream events *3 1 3*. Hidden states are in circles, observations in squares. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.12: $\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.11: $\alpha_t(j) = P(o_1, o_2 \ldots o_t, q_t = j|\lambda)$.

Dakle završna suma 0.028562 u trećem koraku predstavlja izvjesnost osmatranja cijelog niza P(„3 1 3“|λ) pri čemu ovaj završni korak nije izračunat na slici A.5 nego je samo naznačen crtkanim linijama za prijelaz iz trenutka t=2 u trenutka t=3 rešetke (izračunate su unaprijedne vjerojatnosti samo do koraka t=2). Sve opisano je konzistentno s definicijom rekurzivnog algoritma „Unaprijed“:

1. Initialization:

$$\alpha_1(j) = \pi_j b_j(o_1) \quad 1 \le j \le N$$

2. Recursion:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Algoritam „*Unazad*“

Pogledajmo što sadrži drugi izlazni argument funkcije 'beta'. Kao što smo najavili, ova matrica sadrži unazadne vjerojatnosti:

```
>> beta

beta =

   0.090500000000000   0.250000000000000   1.000000000000000
   0.083600000000000   0.280000000000000   1.000000000000000
```

Ona se u istoj funkciji izračunava od zadnjeg stupca t=T=3 unazad sve do prvog stupca t=1 u skladu s definicijom algoritma *„Unazad“*:

1. **Initialization:**

$$\beta_T(i) \;=\; 1, \quad 1 \le i \le N$$

2. **Recursion**

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\, \beta_{t+1}(j), \quad 1 \le i \le N, 1 \le t < T$$

3. **Termination:**

$$P(O|\lambda) = \sum_{j=1}^{N} \pi_j\, b_j(o_1)\, \beta_1(j)$$

Treći (zadnji) stupac matrice 'beta' odgovara inicijalizacijskom koraku, dok se drugi pa zatim i prvi stupac izračunavaju u rekurzijskom koraku algoritma, konkretno za t=2, pa zatim za t=1. Izvjesnost osmatranja zadanog niza „3 1 3“ možemo odrediti i temeljem ovih unazadnih vjerojatnosti β_t(stanje) u skladu s izrazom za završni korak (Termination). Prema tom izrazu treba pomnožiti vektor inicijalne razdiobe vjerojatnosti, vektor izlazih vjerojatnosti konkretnog osmotrenog simbola u prvom vremenskom koraku (simbola „3“) i izračunate unazadne vjerojatnosti u prvom stupcu matrice beta koji sadrži $\beta_1(1)$ i $\beta_1(2)$.

```
>> beta(:,1).*prior0.*obslik0(:,1)

ans =

   0.001810000000000
   0.026752000000000
```

U Matlabu sumu ova dva umnoška sva tri faktora možemo naći kao:

```
>> sum(beta(:,1).*prior0.*obslik0(:,1))

ans =

   0.028562000000000
```

Očekivano, vidimo da dobivamo identično rješenje za izvjesnost cijele sekvence P(„3 1 3“|λ) kao i primjenom algoritma *„Unaprijed“*.

Identičnu vrijednost izvjesnosti P(O|λ) moramo dobiti ako pomnožimo unaprijedne i unazadne vjerojatnosti 'alpha' i 'beta' te ih zbrojimo preko svih skrivenih stanja (istu vrijednost dobivamo u svakom vremenskom trenutku za t=1 to T):

```
>> sum(alpha.*beta)

ans =

   0.028562000000000   0.028562000000000   0.028562000000000
```

To odgovara izrazu A.21 kojeg smo izveli u poglavlju koje se odnosi na treniranje HMM modela:

The probability of the observation given the model is simply the forward probability of the whole utterance (or alternatively, the backward probability of the whole utterance):

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_t(j)\beta_t(j) \tag{A.21}$$

Izračunavanje pomoćne vjerojatnosti gama, za re-estimaciju parametara HMM modela

Konačno, treći izlazni argument kojeg također vraća funkcija 'fwdback' je matrica 'gamma' koja je definirana izrazom A.27 u poglavlju A.5, a koja se koristi pri treniranju HMM modela za zadane opservacije pri re-estimaciji izlaznih vjerojatnosti HMM modela u matrici B.

As Fig. A.13 shows, the numerator of Eq. A.26 is just the product of the forward probability and the backward probability:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \tag{A.27}$$

Provjerimo odgovara li izračunata izlazna matrica 'gamma' gornjem analitičkom izrazu:

```
>> (alpha.*beta)/sum(alpha(:,end)) - gamma

ans =

   1.0e-16 *

                  0                        0                        0
                  0   0.555111512312578                        0
```

Vidimo da se radi o identičnim vrijednostima (do unutar numeričke pogreške reprezentacije u double float zapisu).

Izračun log-izvjesnosti opservacije za zadani model

Konačno, zadnji argument kojeg vraća funkcija 'fwdback' je 'll' (log-likelihood) odnosno log-izvjesnost osmatranja O, a to je jednostavno ln(P(O|λ)), tj. prirodni logaritam izvjesnosti, što lako možemo potvrditi:

```
>> ll

ll =

  -3.555678115951395

>> log(sum(alpha(:,end)))

ans =

  -3.555678115951395
```

Funkcija za izračun log-izvjesnosti osmotrenih nizova za zadani HMM model

Opisani detaljno analizirani odsječak Matlab koda kojim smo ilustrirali način izračunavanja log-izvjesnosti konkretne opservacije O za zadani model λ, i to na više različitih načina, jest zapravo sastavni dio funkcije 'dhmm_logprob.m' ovog paketa, koja izračunava upravo željenu log-izvjesnost za ulaznu matricu opservacija data i za zadan model. Ta matrica opservacija može biti zadana ručno kao u našem primjeru, ali također može biti generirana automatski temeljem zadanog modela. U nastavku je cjelovit Matlab kod ove kratke funkcije:

```
function [loglik, errors] = dhmm_logprob(data, prior, transmat,
obsmat)
% LOG_LIK_DHMM Compute the log-likelihood of a dataset using a
discrete HMM
% [loglik, errors] = log_lik_dhmm(data, prior, transmat, obsmat)
%
% data{m} or data(m,:) is the m'th sequence
% errors  is a list of the cases which received a loglik of -
infinity

if ~iscell(data)
  data = num2cell(data, 2);
end
ncases = length(data);

loglik = 0;
errors = [];
for m=1:ncases
  obslik = multinomial_prob(data{m}, obsmat);
  [alpha, beta, gamma, ll] = fwdback(prior, transmat, obslik,
'fwd_only', 1);
  if ll==-inf
    errors = [errors m];
  end
  loglik = loglik + ll;
end
```

Ovu smo funkciju mogli odmah pozvati sa zadanim ulaznim argumentima pa bi odmah dobili željenu log-izvjesnost, bez da detaljno analiziramo sve elemente izračuna:

```
data=[3 1 3];
>> dhmm_logprob(data, prior0, transmat0, obsmat0)

ans =

  -3.555678115951395
```

Valja naglasiti da u stvarnoj izvedbi funkcije 'dhmm_logprob.m', da se funkcija 'fwdback' poziva uz jednu dodatnu opciju „`'fwd_only', 1`" koja određuje da se umjesto izračuna svih izlaznih argumenata izračunava samo unaprijedna vjerojatnost 'alpha' i log-izvjesnost 'll', dok se umjesto 'beta' i 'gamma' vraćaju prazni vektori [ ]. Također, u usporedbi s ranijim analiziranim odsječcima, vidimo da pri pozivu funkcije 'fwdback' ne koristimo jednu drugu opciju koju smo ranije koristili, konkretno: „`'scaled', 0`". Ova druga opcija zapravo traži da se izračun unaprijedne i unazadne vjerojatnosti provodi prema izvornim analitičkim izrazima koji su opisani u materijalima za predavanja, bez skaliranja, dok je izostavljanje ove

opcije zapravo identično postavljanju na default vrijednost „`'scaled', 1`“. Takvom default opcijom se zapravo aktivira automatsko skaliranje vjerojatnosti alpha i beta čime se postiže veća numerička stabilnost njihovog izračuna. Ovo skaliranje nema utjecaj na konačnu log-izvjesnost (jer se krati) osim što ju čini numerički stabilnijom kod dugačkih opservacijskih sekvenci.

Zadatak dekodiranja skrivenih stanja Viterbi algoritmom

Pokažimo sada kojom funkcijom ćemo utvrditi najizvjesniji niz skrivenih stanja modela za zadani model i opservacije, što je još poznato kao zadatak dekodiranja. U tu svrhu koristimo algoritam 'Viterbi', a pripadajuća funkcija za nalaženje optimalnog puta u ovom programskom paketu se zove 'viterbi_path.m'. Za svaku funkciju iz paketa možete dobiti kratke upute za korištenje primjenom Matlab naredbe 'help' iza koje upišete naziv funkcije (ili skripte) koja se nalazi unutar aktualne putanje pretraživanja. Tako npr. za ovu funkciju 'viterbi_path.m' dobivamo:

```
>> help Viterbi_path
  VITERBI Find the most-probable (Viterbi) path through the HMM
state trellis.
  path = viterbi(prior, transmat, obslik)

  Inputs:
  prior(i) = Pr(Q(1) = i)
  transmat(i,j) = Pr(Q(t+1)=j | Q(t)=i)
  obslik(i,t) = Pr(y(t) | Q(t)=i)

  Outputs:
  path(t) = q(t), where q1 ... qT is the argmax of the above
expression.
```

Pozovimo sada ovu funkciju za zadane parametre modela (uoči da smo kao zadnji argument proslijedili matricu 'obslik0' koja sadrži vjerojatnosti osmatranja zadanog niza izlaznih simbola O = '3', '1', '3' u stanjima 1 i 2):

```
% Najizvjesniji put
>> vpath = viterbi_path(prior0, transmat0, obslik0)

vpath =

     2     1     2
```

Dakle, najizvjesniji niz skrivenih stanja modela u koracima t=1, 2 i t=T=3 za opservaciju '3', '1', '3' jest 2, 1, pa 2, odnosno H(ot), C(old) pa H(ot) uz korištenje izvornih simboličkih oznaka stanja.

Sada kada imamo utvrđen najizvjesniji put, možemo odrediti i kolika je izvjesnost osmatranja zadanog niza, ali baš samo za taj najizvjesniji (ili Viterbijev) put stanja, a ne za sve moguće puteve kao što smo računali u algoritmima 'Unaprijed' i 'Unazad'. U tu svrhu ćemo koristiti funkciju 'dhmm_logprob_path.m' koja se poziva s istim ulaznim argumentima kao i prethodna funkcija, ali joj se kao zadnji argument još dodaje i redak 'vpath' s željenim putem:

```
% Vjerojatnost uzduz najizvjesnijeg puta
[ll, p] = dhmm_logprob_path(prior0, transmat0, obslik0, vpath)

cp=cumprod(p)

ll =

  -4.358310108056565

p =

   0.320000000000000   0.200000000000000   0.200000000000000

cp =

   0.320000000000000   0.064000000000000   0.012800000000000

>>
>> log(cp(end))

ans =

  -4.358310108056565
```

Ova funkcija vraća log-izvjesnost osmatranja cijele opservacije u izlaznom argumentu 'll', ali vraća i redak 'p' u kojem su inkrementalne vjerojatnosti uzduž zadanog puta 'vpath'. Da bi dobili tražene vjerojatnosti svakog čvora uzduž Viterbi puta, potrebno je izračunati kumulativne umnoške ovih inkrementalnih vjerojatnosti iz retka 'p' što ostvarujemo funkcijom 'cumprod'. Time u cp(1) dobivamo p(1), u cp(2) dobivamo p(1)*p(2), odnosno u cp(3) dobivamo umnožak sve tri inkrementalne vjerojatnosti p(1)*p(2)*p(3). Ako izračunamo logaritam ove zadnje cp(3), tada dobivamo upravo log-izvjesnost cijele opservacije 'll'. Usporedimo ova rješenja sa slikom A.10 iz materijala za predavanja koja prikazuje upravo ovaj identični primjer:

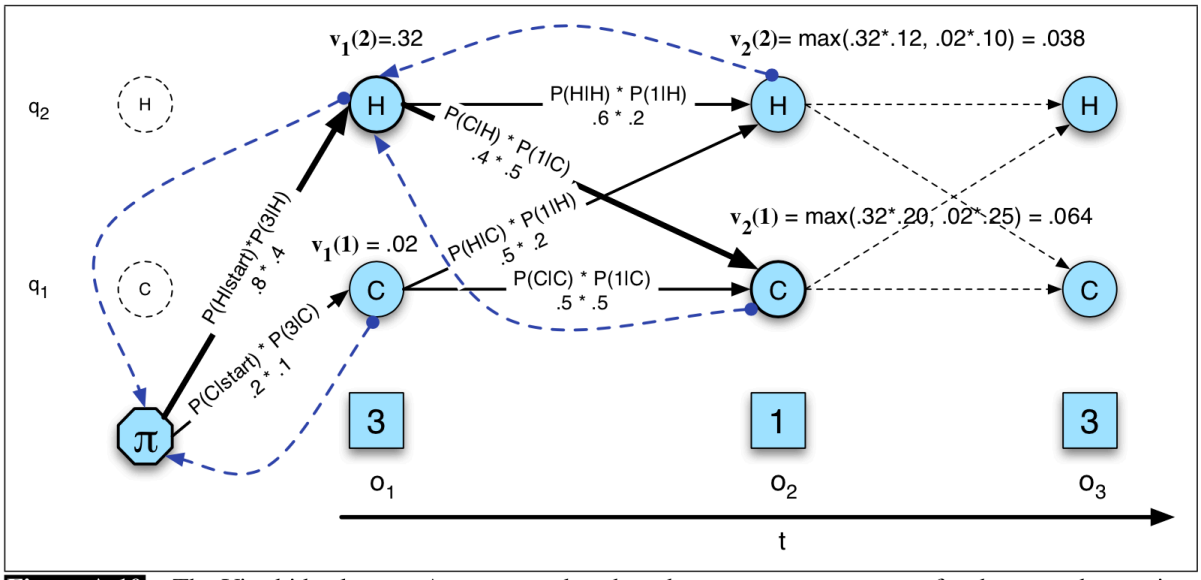


Figure A.10 The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

Možemo prepoznati da je prva vrijednost p(1)=cp(1) jednaka varijabli $v_1(2)=0.32$, a da je druga inkrementalna vjerojatnosti p(2) jednaka umnošku na dijagonalnom putu koji povezuje stanje 2 u koraku t=1 i stanje 1 u koraku t=2, tj. umnošku P(C|H)*P(1|C)=0.4*0.5=0.2, te se množenjem cp(1) i vjerojatnosti duž ovog dijagonalnog puta dobiva $v_2(1)$ koji se nalazi u cp(2)=0.064. Analogno, u koraku t=3 se vraćamo u završno stanje 2, pa je inkrementalna vjerojatnost p(3) uzduž ovog (crtkano označenog dijagonalnog) puta jednaka P(H|C)*P(3|H)=0.5*0.4=0.2. Množenjem s prethodnom vjerojatnosti $v_2(1)$ (tj. sa cp(2)) dobivamo konačni $v_3(2)$ koji se nalazi u cp(3).

Nalaženje izvjesnosti osmatranja duž svih mogućih puteva skrivenih stanja

Za kraj pokušajmo za isti ovaj jednostavni primjer odrediti log-izvjesnosti osmatranja iste opservacije O='3','1','3', ali pojedinačno uzduž svih mogućih puteva rešetke stanja, kojih ima N^T, odnosno 2^3=8. To možemo učiniti narednim kratkim odsječkom Matlab koda, gdje ćemo istu funkciju 'dhmm_logprob_path.m' u petlji pozvati za sve moguće puteve:

```
% Matrica svih moguci puteva
mpath=[
1 1 1
1 1 2
1 2 1
1 2 2
2 1 1
2 1 2
2 2 1
2 2 2];

llm=zeros(8,1);  % Stupac za log-izvjesnosti
for i=1:8,
  [llm(i), p] = dhmm_logprob_path(prior0, transmat0, obslik0,
mpath(i,:));
end;
```

Pogledajmo dobivenu log-izvjesnosti osmatranja za svih 8 mogućih puteva stanja:

```
>> llm

llm =

  -8.294049640102028
  -6.907755278982137
  -9.433483923290392
  -7.641724454062337
  -5.744604469176456
  -4.358310108056565
  -6.478573644256656
  -4.686814175028601
```

Vidimo da se uistinu najveća izvjesnost dobiva za šesti slučaj, a to je upravo ona kombinacija koja odgovara Viterbi putu 2, 1, 2, te iznosi -4.35831. Ovo rješenje 'llm' nam omogućava i dodatni način provjere algoritma 'Unaprijed' i 'Unazad' koji su izračunali ukupnu izvjesnost opservacije O preko svih mogućih puteva stanja. To znači da bi zbrajanjem eksponenciranih vrijednosti stupca 'llm' morali dobiti P(„3 1 3“|λ):

```
>> sum(exp(llm))

ans =

   0.028562000000000
```

Vidimo da smo stvarno dobili identičnu vrijednost izvjesnosti kao i ranije opisanim algoritmima, s tom razlikom da je numerička složenost ovakvog direktnog izračuna proporcionalna sa N^T, dok su algoritmi „Unaprijed“ i „Unazad“ bitno manje složenosti zbog njihove rekurzivne realizacije.

Ovo zadnje pomoćno rješenje 'llm' nam također omogućava određivanje alternativnih puteva dekodiranih skrivenih stanja, a ne samo najboljeg Viterbi puta. Ako bi log-izvjesnosti prvo sortirali, pa tek onda eksponencirali, te odredili kumulativnu sumu izvjesnosti osmatranja od najizvjesnijeg, do najmanje izvjesnog puta stanja tada bi za ovaj primjer dobili:

```
% Sortiraj puteve prema izvjesnosti osmatranja od najizvjesnijeg do
% najmanje izvjesnog
[sllm,illm]=sort(-llm);

% Kumulativno zbrojih izvjesnosti od samo jednog najizvjesnijeg
% puta, pa prva dva puta stanja po izvjesnosti, pa prva tri, ... i
% tako sve do sume izvjesnosti po svim mogucim putevima stanja
cumsum(exp(-sllm))

ans =

   0.012800000000000
   0.022016000000000
   0.025216000000000
   0.026752000000000
   0.027752000000000
   0.028232000000000
   0.028482000000000
   0.028562000000000
```

Konkretne puteve stanja od najizvjesnijeg do najmanje izvjesnog možemo pročitati korištenjem indeksa 'illm' kojeg je vratila funkcija za sortiranje negativnih log-izvjesnosti.

```
>> mpath(illm,:)

ans =

     2     1     2  % Najizvjesniji
     2     2     2
     2     1     1
     2     2     1
     1     1     2
     1     2     2
     1     1     1
     1     2     1  % Najmanje izvjesni put
```

Možemo se također zapitati koliko takvih najboljih puteva trebamo odabrati da ostvarimo određen postotak ukupne izvjesnosti osmatranja. To možemo lako provjeriti na način da prethodnu kumulativnu sumu podijelimo s njenim zadnjim članom koji je suma izvjesnosti svih puteva:

```
>> cumsum(exp(-sllm))/sum(exp(llm))

ans =

   0.448147888803305
   0.770814368741685
   0.882851340942511
   0.936629087598908
   0.971640641411666
   0.988446187241790
   0.997199075694979
   1.000000000000000
```

Vidimo da Viterbijev put pokriva 44.8% ukupne izvjesnosti osmatranja. Ako mu dodamo i naredni put po izvjesnosti, a to je put 2, 2, 2, tada ta dva puta zajedno osiguravaju već 77.1% izvjesnosti, dok još i uz treći put stanja, 2, 1, 1, dobivamo čak 88.3%. Preostalih 5 kombinacija malo doprinose ukupnoj izvjesnosti, pa ih možemo ignorirati kao moguće kandidate za dekodirana stanja modela.

Zadatak generiranja opservacija za zadani model

Prije završnog zadatka, koji će biti zadatak treniranja parametara HMM modela iz višestrukih nizova osmatranja, moramo prvo naučiti kako automatski generirati takve nizove. U tu svrhu ovaj Toolbox posjeduje funkciju 'dhmm_sample.m' koja se poziva prema ovom predlošku:

```
% generiraj visestruki opservacijski niz:
T = 15;  % duljina svakog niza
nex = 10;  % broj opservacijskih nizova
data = dhmm_sample(prior0, transmat0, obsmat0, nex, T);
```

čijim pozivom dobivamo 10 različitih nizova duljine 15 simbola (prikazan je samo jedan primjer generiranih podataka, jer će svaki novi poziv ove funkcije generirati novi slučajni skup, čija je statistika određena zadanim parametrima modela):

```
data

data =

  2   2   2   3   1   3   2   3   1   2   2   3   1   1   2
  2   3   3   2   1   1   2   2   3   2   3   3   2   2   2
  2   2   1   3   2   2   3   1   2   1   2   2   2   2   2
  3   3   1   3   3   2   2   3   3   2   2   3   2   1   2
  2   3   1   3   2   1   3   1   1   3   3   2   2   1   2
  3   1   3   1   1   2   1   1   3   2   1   2   3   2   3
  1   2   1   1   1   3   2   2   3   1   2   2   2   3   1
  3   1   3   2   1   3   3   3   1   2   3   3   1   1   1
  1   3   1   1   1   1   2   2   1   1   1   2   3   2   2
  2   3   2   2   2   2   2   1   3   3   2   2   1   3   2
```

Napomenimo još jednom da su ovi opservacijski nizovi konstruirani točno u skladu sa zadanim parametrima modela, istog kojeg koristimo u cjelokupnom dosadašnjem primjeru, dakle temeljem vektora inicijalne razdiobe vjerojatnosti stanja 'prior0', temeljem matrice prijelaznih vjerojatnosti 'transmat0' i konačno matrice vjerojatnosti opservacije izlaznih simbola u oba stanja modela, 'obsmat0'. Možemo zamisliti kao da je funkcija 'dhmm_sample.m' provodila niz slučajnih eksperimenata, te je sukladno zadanim parametrima modela u svakom trenutku vremena: (1) temeljem 'obsmat0' birala jedan od tri moguća izlazna simbola u trenutačno zatečenom stanju, i (2) određivala novo stanje temeljem trenutačnog stanja, sukladno matrici 'transmat0'.

Dugotrajna statistika generiranih nizova

Postavlja se pitanje, kakvo smo rješenje uopće trebali očekivali, tj. kako možemo potvrditi da su generirani podatci uopće sukladni zadanom modelu. Jednostavna provjera može se provesti na način da odredimo učestalost pojave svih mogućih izlaznih simbola za svaki generirani opservacijski niz, te da ovu empirijsku učestalost usporedimo s očekivanim dugotrajnim vjerojatnostima izlaznih simbola za zadani model. Konkretno za izlazni simbol '1' trebali bi uvjetnu vjerojatnost osmatranja tog simbola '1' u stanju q=1 pomnožiti s vjerojatnosti da se HMM model nalazi u stanju q=1, te na to dodati uvjetnu vjerojatnost osmatranja istog simbola '1' u stanju q=2 pomnoženu s vjerojatnosti da je model u stanju q=2, odnosno prema izrazu:

$$P(1)=P(1|q=1)*P(q=1) + P(1|q=2)*P(q=2)$$

Analogno i za preostala dva izlazna simbola '2' i '3'. Sve uvjetne vjerojatnosti osmatranja izlaznih simbola imamo u matrici 'obsmat0', ali ono što nemamo su vjerojatnosti stanja P(q=1) i P(q=2). Valja naglasiti da su promjene stanja HMM modela potpuno određene prijelaznom matricom A ('transmat0' u primjeru), tj. ne ovise o opservacijama, pa se može očekivati da se potrebne vjerojatnosti stanja mogu naći direktno iz nje. Taj dio HMM modela (kada bi stanja bila otkrivena), nije ništa drugo nego vremenski diskretni i vremenski-homogen Markovljev lanac s konačnim brojem stanja. Uvjet vremenske homogenosti je ispunjen ako je matrica prijelaznih vjerojatnosti stalna, tj. ako se ne mijenja s vremenom, što je kod nas slučaj. Za takvu klasu Markovljevih lanaca, prijelazna vjerojatnost u k-tom vremenskom koraku se može odrediti uzastopnim množenjem matrice prijelaznih vjerojatnosti k puta, dakle kao k-ta potencija matrice A: A^k. U slučajevima kada je Markovljev lanac ireducibilan i aperiodički, tada postoji jedinstvena stacionarna distribucija π_{stac}, koja definira upravo tražene vjerojatnosti stanja. Prema Perron-Frobeniusovom teoremu, za takav slučaj k-ta potencija matrice, A^k, konvergira u matricu jediničnog ranga (s identičnim retcima), gdje svaki njen redak sadrži upravo ovu stacionarnu distribuciju stanja π_{stac}. Bez ulaženja u dokazivanje uvjeta ove konvergencije, (jer postoji mnogo specijalnih slučaja koje bi bilo potrebno razmotriti), pokažimo na našem primjeru što bi dobili uzastopnim množenjem matrice A, koja se kod nas zove 'transmat0'.

```
>> a0=transmat0; for i=1:100, a0=a0*transmat0; end;
>> a0

a0 =
```

```
   0.444444444444445   0.555555555555556
   0.444444444444445   0.555555555555556
```

Očigledno smo dobili vjerojatnosti P(q=1)=4/9 i P(q=2)=5/9. Sada možemo izračunati i očekivane učestalosti pojave izlaznih simbola te dobivamo:

$$P(1)=P(1|q=1)*P(q=1) + P(1|q=2)*P(q=2) = (5/10)*(4/9)+(2/10)*(5/9)=(5/15)$$
$$P(2)=P(2|q=1)*P(q=1) + P(2|q=2)*P(q=2) = (4/10)*(4/9)+(4/10)*(5/9)=(6/15)$$
$$P(3)=P(3|q=1)*P(q=1) + P(3|q=2)*P(q=2) = (1/10)*(4/9)+(4/10)*(5/9)=(4/15)$$

To znači da bi u idealnom slučaju u opservacijskoj sekvenci duljine 15 simbola trebalo biti sadržano 5 simbola '1', 6 simbola '2' i konačno 4 simbola '3'. Empirijske učestalosti izlaznih simbola određene iz opservacijskih sekvenci konačnog trajanja težiti će ovim idealnim učestalostima u slučaju kada duljina tih sekvenci T teži u beskonačno, jer je to upravo uvjet pod kojim je izračunata stacionarna distribucija π_{stac}.

Kako smo mogli odmah u Matlabu dobiti ove dugotrajne statistike izlaznih simbola? Prepoznajemo da gornji izrazi zapravo odgovaraju množenju retka stacionarne distribucije stanja π_{stac} s matricom emisijskih vjerojatnosti B ('obsmat0' u našem primjeru). Pomnožimo još s 15 (zajedničkim nazivnikom sve tri vjerojatnosti) da dobijemo učestalosti simbola u nizu duljine 15:

```
>> a0(1,:)*obsmat0*15

ans =

   5.000000000000001   6.000000000000000   4.000000000000001
```

Provjerimo sada je li vremenski dugotrajna statistika automatski generiranih podataka sukladna zadanom modelu. Konkretno, prebrojimo za svaku od 10 generiranih sekvenci u matrici 'data' koliko puta se pojavio svaki mogući izlazni simbol, za što ćemo koristiti funkciju 'hist', kojoj kao drugi argument navodimo 'binove' podataka koje prebrojavamo (Napomena: funkcija 'hist' operira po stupcima, pa joj serviramo transponiranu matricu data).

```
>> hm=hist(data',[1 2 3])

hm =

     4     2     3     2     5     6     6     6     8     2
     7     8    10     6     5     4     6     2     5     9
     4     5     2     7     5     5     3     7     2     4
```

Dakle, proizlazi da u prvoj sekvenci imamo 4 jedinice, 7 dvojki i 4 trojke, u drugoj sekvenci imamo 2 jedinice, 8 dvojki i 5 trojki, .. i tako sve do desete sekvence koja ima 2 jedinice, 9 dvojki i 4 trojke. Ako nađemo prosječan broj pojava svih izlaznih simbola preko ovih 10 različitih sekvenci te ako ovaj prosjek podijelimo s duljinom sekvence T=15, trebali bi dobiti empirijske vjerojatnosti bliske očekivanima (pomnožimo ih još sa 15 radi lakše usporedbe s analitički izvedenima):

```
>> mean(hm')/T*15
```

```
ans =

   4.400000000000000   6.200000000000000   4.400000000000000
```

Prema ovome proizlazi da od 15 izlaznih simbola u prosjeku imamo 4.4 jedinica, 6.2 dvojki i 4.4 trojki, što ima određenu sličnost s očekivanih 5+6+4, ali baš i nije jednako. Očigledno su ove sekvence prekratke da bi njihovi histogrami odgovarali vremenski dugotrajnim statistikama. Ponovimo stoga automatsko generiranje podataka ali uz 100 puta dulje opservacijske sekvence, tj. uz T=1500, te izračunajmo ponovno empirijske statistike ovakvih dugačkih nizova:

```
% generiraj visestruki opservacijski niz:
T = 1500;    % duljina svakog niza
nex = 10;  % broj opservacijskih nizova
data_1500 = dhmm_sample(prior0, transmat0, obsmat0, nex, T);
% Prebroji simbole u svakoj sekvenci
hm_1500=hist(data_1500',[1 2 3]);
% Usporedi s ocekivanim ucestalostima
mean(hm_1500')/T*15
ans =

   4.968000000000000   6.064000000000000   3.968000000000000
```

Vidimo da statistike duljih opservacijskih sekvenci 'data_1500' puno bolje prate analitički izvedene očekivane izlazne učestalosti simbola, što potvrđuje da funkcija za generiranje podataka uistinu radi sukladno zadanim parametrima modela.

Kakav bi rezultat dobili ako umjesto produljenja sekvenci povećamo njihov broj, tj. ako vratimo duljinu T nazad na 15, ali 100 puta povećamo broj nizova na nex=1000?

```
% generiraj visestruki opservacijski niz:
T = 15;    % duljina svakog niza
nex = 1000;  % broj opservacijskih nizova
data_1000 = dhmm_sample(prior0, transmat0, obsmat0, nex, T);
% Prebroji simbole u svakoj sekvenci
hm_1000=hist(data_1000',[1 2 3]);
% Usporedi s ocekivanim ucestalostima
mean(hm_1000')/T*15
ans =

   4.916000000000000   5.927000000000000   4.157000000000000
```

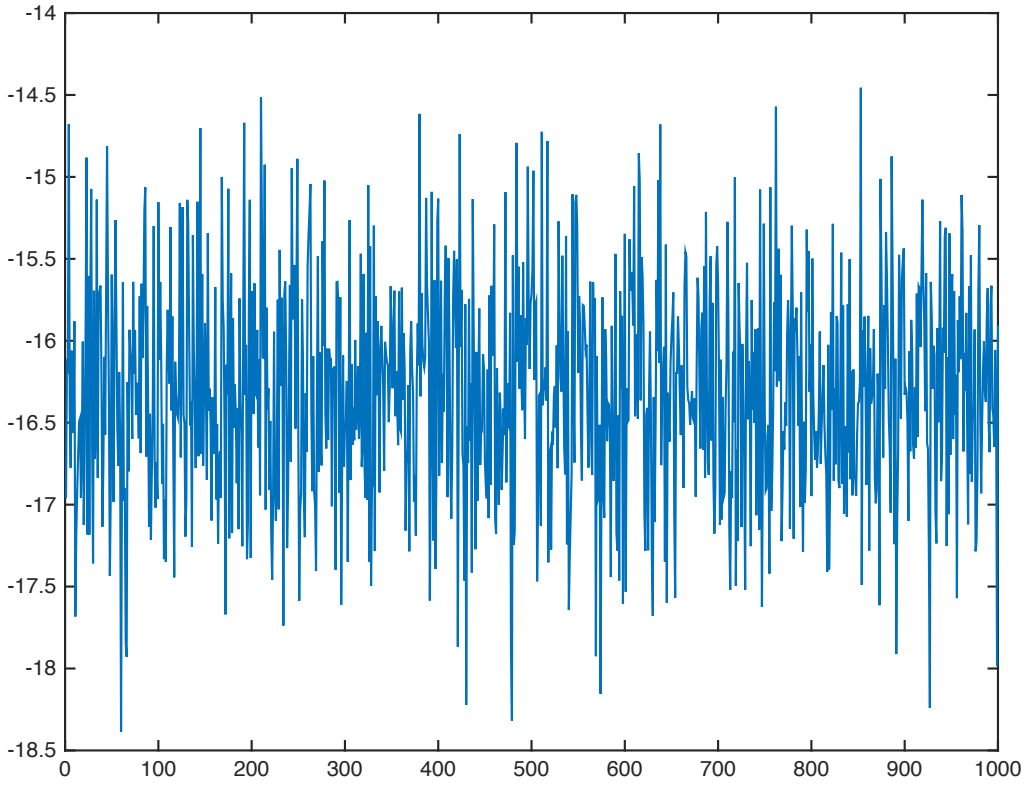
Iako na prvi pogled izgleda kao da oba pristupa daju sličan efekt, to ipak nije slučaj. Kod vrlo dugačkih sekvenci, inicijalna razdioba vjerojatnosti stanja iz vektora π (koji se zove 'prior0' u našem primjeru) neće imati utjecaja (zbog stacionaranosti k-te potencije matrice A za veliki k), dok u slučaju kraćih sekvenci, npr. T=15, utjecaj inicijalnih vjerojatnosti će se ipak odražavati u rezultatima eksperimenta. Opisano ponašanje je logično, jer se kod dugačkih sekvenci početno stanje modela relativno brzo 'istitralo', te je ponašanje dominantno određeno prijelaznom matricom A, a samo na početku sekvence i početnim stupcem π.

Provjera izvjesnosti osmatranja generiranih podataka

Za ova tri skupa automatski generiranih opservacijskih nizova možemo izračunati izvjesnosti njihovog osmatranja za zadani model (isti koji je korišten za njihovo generiranje). Ilustrirajmo to za skup 'data_1000', pomoću narednog odsječka gdje računamo log-izvjesnost svakog od 1000 kratkih nizova.

```
% Izracunaj u petlji log-izvjesnosti svakog niza
nex=size(data_1000,1);  % Broj eksperimenata
llm=zeros(nex,1);       % Stupac log-izvjesnosti
for i=1:nex,
    llm(i)=dhmm_logprob(data_1000(i,:), prior0, transmat0, obsmat0);
end;
```

Rezultat možemo prikazati i slikom, naredbom plot(llm)



Vidimo da je raspon log-izvjesnosti prilično velik, od -18.5 (najmanje izvjesni niz) do -14.5 (najizvjesniji niz). Možemo odrediti i omjer najizvjesnijeg niza u odnosu na najmanje izvjestan generirani niz:

```
>> exp((max(llm)-min(llm)))

ans =

  50.944825651065138
```

Da smo funkciji 'dhmm_logprob' kao prvi argument proslijedili cijelu matricu 'data_1000', funkcija bi vratila sumu log-izvjesnosti svih nizova (što odgovara umnošku njihovih izvjesnosti,

odnosno izvjesnosti istovremenog osmatranja svih 1000 nizova), no tada ne bi imali uvid u razlike izvjesnosti između pojedinih opservacija.

```
>> ll=dhmm_logprob(data_1000, prior0, transmat0, obsmat0);
>> disp(sum(llm)-ll)
  0
```

Generalno će raspršenje izvjesnosti biti veće na kratkim nizovima, jer se izlazna statistika opisana modelom „nije stigla“ u cijelosti demonstrirati. Kao mjeru ovog relativnog raspršenja možemo izračunati kvocijent standardne devijacije i srednje vrijednosti stupca log-izvjesnosti svih opservacija:

```
>> std(llm)/mean(llm)

ans =

  -0.040006421311045
```

iz čega vidimo da je za sekvence trajanja T=15 to raspršenje iznosi oko 4% očekivane log-izvjesnosti.

Ponovite isti postupak za dugačke opservacijske nizove 'data_1500' te diskutirajte razliku u odnosu na opisani slučaj. Kako iznos log-izvjesnost ovisi o duljini sekvence? Koliko je relativno raspršenje log-izvjesnosti dugačkih sekvenci? Kako to objašnjavate?

Degenerirani HMM model s identičnom dugotrajnom statistikom

Za kraj promislite postoje li možda i drugačiji HMM modeli koji imaju identičnu vremenski dugotrajnu statistiku izlaznih simbola, kao i model kojeg koristimo za primjer. Što ako HMM model degenerira, te ima samo jedno stanje, a upravo u tom jedinom stanju su njegove emisijske vjerojatnosti identične dugotrajnim vjerojatnostima osmatranja za naš analiziran primjer. Da li se taj model uopće razlikuje od našega i bi li log-izvjesnost osmatranja generiranih nizova s našim modelom ('data_1500' i 'data_1000') bila jednaka za oba modela? Kao polazište promisli kakav model opisuju ovi parametri:

```
priorL=[
    0.5  % Stanje C
    0.5  % Stanje H
];
% Matrica vjerojatnosti promjena stanja
%
transmatL=[
0.5 0.5      % P(C|C) P(H|C)
0.5 0.5      % P(C|H) P(H|H)
];
 % Matrica emisijskih vjerojatnosti
% svaki redan odgovara jednom stanju, a
% svaki stupac jednoj mogucoj observaciji
obsmatL=[
5 6 4   % P(1|C) P(2|C) P(3|C)
5 6 4   % P(1|H) P(2|H) P(3|H)
]/15;
```

Zadatak treniranja parametara modela temeljem opservacija

Sada kada znademo generirati sintetičke opservacijske podatke za zadane parametre modela, možemo pokazati i zadnju funkciju ovog paketa koja je važna za diskretne HMM modele, a to je funkcija 'dhmm_em' koja se koristi za učenje parametara modela temeljeno na algoritmu maksimizacije očekivanja (engl. Expectation Maximization, EM), koji je opisan u materijalima za predavanja.
Ulaz u ovu funkciju su višestruki nizovi opservacija koji svi pripadaju istoj klasi za koju želimo izgraditi HMM model. Funkcija se poziva na ovaj način:

```
% improve guess of parameters using EM
[LL, prior2, transmat2, obsmat2] = dhmm_em(data_1000, prior1,
transmat1, obsmat1, 'max_iter', 10);
```

Kao izlaz vraća ukupnu log-izvjesnost 'LL' zadanih opservacija ('data_1000') uz optimalno određene koeficijente modela, a ti su koeficijenti: 'prior2' (redak inicijalne vjerojatnosti), 'transmat2' (matrica prijelaznih vjerojatnosti stanja) i 'obsmat2' (matrica emisijskih vjerojatnosti). Međutim, ova funkcija pored podataka za učenje očekuje i početne parametre modela, jer kao što je opisano na predavanjima, EM algoritam je iterativni algoritam koji u svakom koraku temeljem postojećeg modela računa nove parametre modela koji će povećati izvjesnost osmatranja zadanog niza. Takav iterativni algoritam stoga za svoj prvi korak traži određen inicijalni model, od kojeg će krenuti. Za model s Q=2 stanja i rječnik s O=3 simbola, potpuno slučajne parametre modela možemo izračunati narednim odsječkom:

```
% initial guess of parameters
prior1 = normalise(rand(Q,1));
transmat1 = mk_stochastic(rand(Q,Q));
obsmat1 = mk_stochastic(rand(Q,O))
```

Svaki naredni poziv istog odsječka će svaki puta generirati novi skup slučajnih parametara modela, koji svi zadovoljavaju uvjete na sume vjerojatnosti koje moraju biti jednake 1. Takav potpuno slučajan model će biti dovoljan za inicijalizaciju postupka treniranja, a sam EM algoritam garantirano konvergira, tj. u svakom koraku uvijek nužno povećava izvjesnost osmatranja. Funkcija 'dhmm_em' će ispisivati log-izvjesnost u svakoj iteraciji EM algoritma, te će zaustaviti izračun kada relativna promjena log-izvjesnosti bude dovoljno mala, ili kada bude dosegnut maksimalni broj iteracija koji je definirna sa zadnja dva ulazna argumenta: „, 'max_iter', 10);”

Primjer izvršavanja ove funkcije na 1000 opservacijskih nizova duljine 15 koje smo generirali u prošlom poglavju u matrici 'data_1000' i to uz slučajnu inicijalizaciju modela daje:

```
% improve guess of parameters using EM
[LL, prior2, transmat2, obsmat2] = dhmm_em(data_1000, prior1,
transmat1, obsmat1, 'max_iter', 10);

iteration 1, loglik = -17321.546728
iteration 2, loglik = -16404.111541
iteration 3, loglik = -16374.989855
iteration 4, loglik = -16358.670188
iteration 5, loglik = -16348.640820
```

```
iteration 6, loglik = -16342.056279
iteration 7, loglik = -16337.514173
iteration 8, loglik = -16334.258950
iteration 9, loglik = -16331.854393
iteration 10, loglik = -16330.034303
```

Iterativni EM algoritam pokazuje tipično ponašanje ove klase algoritama, tj. u prvim koracima se log-izvjesnost značajnije mijenja, dok kasnije sve sporije i sporije, jer rješenje relativno brzo konvergira optimalnom.

Evaluacija treniranog modela

Kao što je opisano u uvodnim poglavljima, evaluacija izgrađenog modela se mora provoditi na podatkovnom skupu koji nije korišten u postupku učenja, jer jedino tako možemo potvrditi da je model „generalizirao“. Pored podataka korištenih za učenje 'data_1000', u prošlom smo poglavlju generirali i dodatni opservacijski skup 'data', koji ima istu duljinu T=15, ali samo 10 takvih opservacijskih nizova. Ove podatke možemo iskoristiti kao „evaluacijsku bazu“, te izračunati njihovu log-izvjesnost osmatranja uz sve modele:

- zadani model temeljem kojeg smo generirali sve podatke (prior0, transmat0, obsmat0),
- inicijalni model sa slučajno odabranim parametrima (prior1, transmat1, obsmat1),
- optimalni model izgrađen EM algoritmom (prior2, transmat2, obsmat2).

```
>> ll0=dhmm_logprob(data, prior0, transmat0, obsmat0)

ll0 =

    -1.631252692507668e+02

>> ll1=dhmm_logprob(data, prior1, transmat1, obsmat1)

ll1 =

    -1.735761577055685e+02

>> ll2=dhmm_logprob(data, prior2, transmat2, obsmat2)

ll2 =

    -1.631280680263178e+02
```

Vidimo za su izvjesnosti za zadani i optimalni model gotovo identične, dok je izvjesnost za slučajni model manja.

Usporedite zadane i optimalno određene parametre HMM modela. Jesu li jednaki? Kako to objašnjavate?

Pokušajte ponoviti gradnju modela ali uz nove slučajne inicijalizacije. Istražite koliko se razlikuju konačne log-izvjesnosti takvih modela na test skupu 'data' ovisno o slučajnoj inicijalizaciji.