МАТЕМАТИЧКА ГИМНАЗИЈА

МАТУРСКИ РАД

из предмета

Програмирање и програмски језици

на тему

Примена машинског учења и скривених Марковљевих модела на препознавање топологије трансмембранских протеина

Ученик:
Марина Ивановић, $\mathrm{IV_d}$

Ментор:
Петар Величковић

Београд, мај 2016.

Садржај

Глава 1

Увод

Идеја учења од природе је стара више хиљада година. Данас су развијени разни алати за изучавање природе и циклуса у њој. Обрадом истражених биолошких феномена и коришћењем стечених знања зарад даљих истраживања, али и ради свеукупне примењености, *биоинформатика* [11] постаје значајна информатичка вештина која манипулише стеченим знањима и допуњује их. Постала је неизоставни део биолошких истраживања, али има велику примену и у осталим областима. Користи знања биолошких система, математичке моделе и информатичке концепте. Управо због јединственог приступа и брзом начину доласка до одговора, биоинформатика је рутина која постаје саставни део свих истраживања живог света.

Предвиђање гена, откривање лекова, одређивање структуре протеина и моделовање еволуције само су неке од бројних истраживачких поља ове информатичке дисциплине која се ослања на биолошке процесе. Такође, она подразумева стварање и развој база података, алгоритама, информатичких и статистичких техника за решавање формалних и практичних проблема анализирајући биолошке структуре. Препознавање образаца, анализа података, машинско учење и визуелизација биолошких података само су неки од многобројних биоинформатичких приступа решавања проблема.

Протеини су важна класа биолошких молекула присутних у свим организмима и учествују у готово свим процесима у организму и користе се сваког дана. Управо због овога, јако је битно истраживати како њихову грађу тако и функцију. Структурна, транспортна и каталитичка функција протеина само су неке од бројних [7]. Мотивација за рад овог пројекта управо је последица великог значаја ове класе једињења, као и потребе да се она истражи.

Концепт машинског учења и скривених Марковљевих модела [1] све више је заступљен

у решавању разних проблема попут бионформатичких задатака, препознавања рукописа или говора [3]. Због њихове велике примене, подстрек за рад овог пројекта постао је још већи. За његову реализацију били су потребни математички концепти коришћених алгоритама, након чега је уследила примена на конкретном примеру.

Циљ пројекта је било препознавање топологије протеина [8]. Пре уласка у математичке структуре коришћене при имплементацији, осврнућемо се на теоријске темеље топологије протеина. Успешно имплементиране функције и њихов опис приказани су у Глави 3, док Глава 4 садржи сажетак евалуације и верификацију рада програма.

Глава 2

Теоријски темељи

Ова глава се осврће на теоријске основе биолошких структура над којим је вршена примена, као и на основне математичке моделе који су коришћени у самом пројекту и његовој имплементацији.

2.1 Увод у топологију трансмембранских протеина

Мада је до сад истражен велики број протеина, знање о њима сваким даном све више и више расте. Слободно се може рећи да су протеини једна од најкомплексинијих једињења.

2.1.1 Структура протеина

Основна грађа свих протеина су *аминокиселине* међусобно повезане у *полипептид* [6]. Сваки протеин има јединствен склоп аминокиселина. Грађа протеина може бити *примарна*, што представља управо низ повезаних аминокиселина, *секундарна* која је прилично стабилна и тежи одређеним структурним обрасцима, *терцијална* која објашњава потпуни тродимензиони склоп полипептида и *кватернарна* структура протеина који у себи садрже више полипептидних подјединица.

Аминокиселина има укупно 20 и могу бити из следећег скупа: {аланин, валин, леуцин, изолеуцин, глицин, пролин, фенилаланин, тирозин, триптофан, лизин, аргинин, хистидин, аспарагинска киселина, глутаминска киселина, аспарагин, глутамин, серин, треонин, метионин, цистеин}

2.1.2 Трансмембрански протеини

Према положају у мембрани протеини се деле на:

- *периферне* протеине који се налазе са обе стране мембране и ван су двослоја липида мембране;
- *интегралне* протеине који урањају у двослој липида и са њим су интегрисани. Без утицаја одређених хемијских супстанци се не могу уклонити из мембране. Имају хидрофобне и хидрофилне делове те могу бити једном делом у мембрани или пролазити кроз њу. Они који пролазе кроз њу називају се **трансмембрански протеини**. Њихова **топологија** представља спецификацију хеликса и њихову оријентацију у мембрани односно ћелијском и ванћелијском простору.

2.2 Увод у скривене Марковљеве моделе

2.2.1 Марковљеви ланци

Најбољи увод у скривене Марковљеве моделе свакако би представљао опис **Марковљевих ланаца**. Оба модела имају могућност учења и задовољавају **Марковљево својство**.

Дефиниција 1. Нека је S пребројив скуп *стања* и $\{X_n\}_{n\geq 0}$ низ дискретних променљивих које могу узети вредности $\{x_n\}_{n\geq 0}$ за $\forall n \geq 0$. $x_n \in S$. Овај низ задовољава *Марковљево својство* ако важи

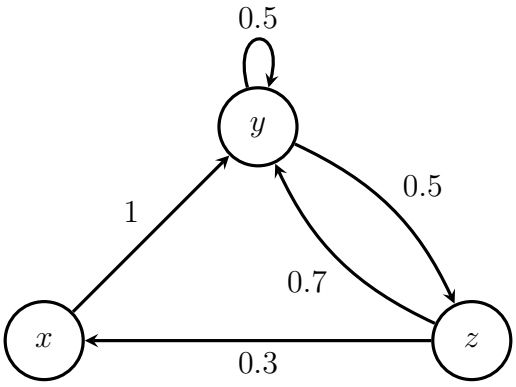
$$\forall n \geq 1, \mathbb{P}(X_n = x_n | X_{n-1} = x_{n-1}, ..., X_0 = x_0) = \mathbb{P}(X_n = x_n | X_{n-1} = x_{n-1})$$

и тада се назива *Марковљев ланац*.
Другим речима, свако следеће стање система је одређено једино тренутном стањем у коме се он налази. Расподела вероватноће следећег стања на основу тренутног се не мења током времена, па ову расподелу можемо сматрати **временски хомогеном**.

Дефиниција 2. Марковљев ланац $\{X_n\}_{n\geq 0}$ је *временски хомоген ако важи*

$$\forall n \geq 1, \forall x, y \in S, \mathbb{P}(X_n = y | X_{n-1} = x) = \mathbb{P}(X_1 = y | X_0 = x)$$

Под претпостаком временске хомогности и коначности скупа S, да бисмо потпуно описали Марковљев ланац довољно је дефинисати следеће:



Слика 2.1: Пример временски хомогеног Марковљевог ланца са сетом стања $S = \{x, y, z\}$.

- **Вектор вероватноће почетних стања,** $\vec{pi}$, дефинисан са $\vec{pi}_x \stackrel{\text{def}}{=} \mathbb{P}(X_0 = x)$;
- **Матрицу вероватноће преласка,** $\mathbf{T}$, дефинисану са $\mathbf{T}_{xy} \stackrel{\text{def}}{=} \mathbb{P}(X_1 = y | X_0 = x)$.

За конкретан низ стања $\{x_t\}_{t=0}^{T}$ вероватноћа да систем прође кроз низ стања $\vec{y}$ износи

$$\mathbb{P}(\{x_t\}_{t=0}^{T}) = \mathbb{P}(X_0 = x_0) \prod_{t=1}^{T} \mathbb{P}(X_t = x_t | X_{t-1} = x_{t-1}) = \vec{pi}_{x_0} \prod_{t=0}^{T-1} \mathbf{T}_{x_t x_{t+1}}$$

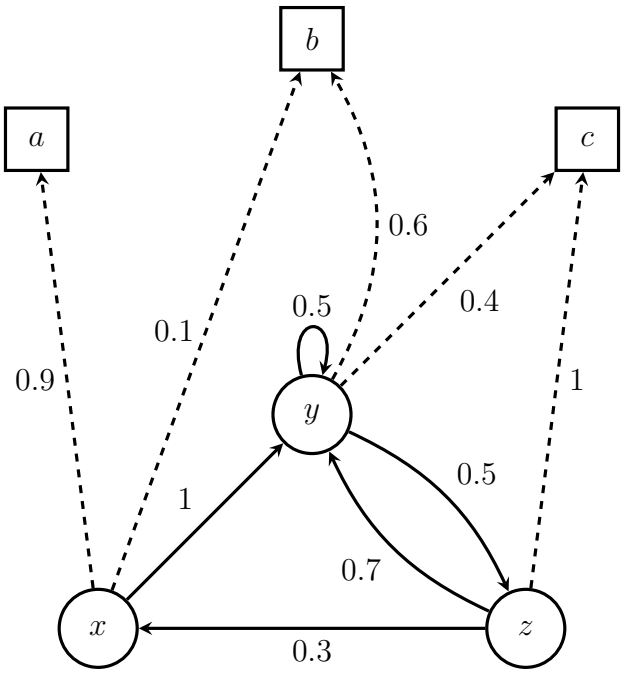
2.2.2 Скривени Марковљеви модели

Скривени Марковљеви модели се могу посматрати као нека врста надоградње Марковљевих ланаца.

Дефиниција 3. *Скривени Марковљев модел* (СММ) је Марковљев ланац у којем низ стања није приметан. [2]

Ово значи да и уз познате параметре Марковљевих ланаца (вектора вероватноће почетних стања и матрице вероватноће преласка у случају временске хомогености) је немогуће директно одредити низ стања $\{X_n\}_{n \geq 0}$ кроз који систем пролази. У овом случају, треба посматрати низ излаза које свако стање производи, $\{Y_n\}_{n \geq 0}$. Стање може произвести **излаз** из коначног низа датих излаза, O. Може се приметити да излаз који је произведен зависи једино од стања у којем се систем налазио у датом тренутку.

$$\forall n \geq 0, \mathbb{P}(Y_n = y_n | X_n = x_n, ..., X_0 = x_0, Y_{n-1} = y_{n-1}, ..., Y_0 = y_0) = \mathbb{P}(Y_n = y_n | X_n = x_n)$$



Слика 2.2: Пример надограђеног Марковљевог ланца са 2.1 и сетом излаза $O = \{a, b, c\}$.

Под претпоставком да је низ могућих излаза коначан и да се расподела њихове вероватноће да односу тренутног стања током времена не мења

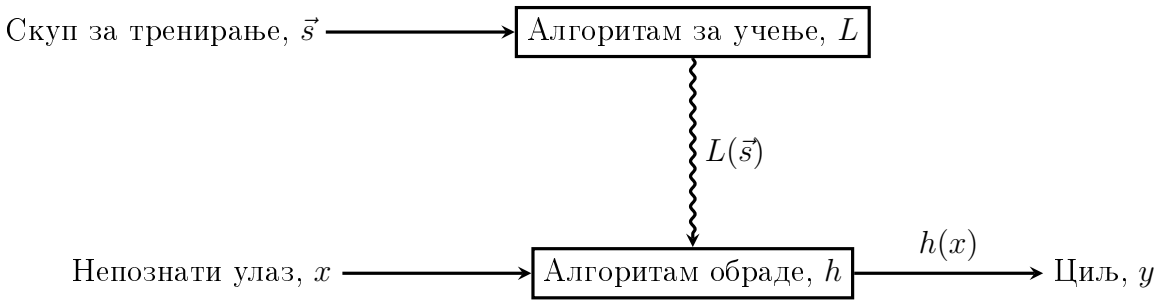
$$\forall n \geq 0, \mathbb{P}(Y_n = y_n | X_n = x_n) = \mathbb{P}(Y_0 = y_n | X_0 = x_n)$$

онда је једини додатни параметар који је потребан да се СММ у потпуности опише **матрица вероватноће излаза**, $\mathbf{O}$, дефинисана са $\mathbf{O}_{xy} \stackrel{\text{def}}{=} \mathbb{P}(Y_0 = y | X_0 = x)$.

2.2.3 Машинско учење

Машинско учење је област вештачке интелигенције чији је циљ адаптирање на нове ситуације коришћењем имплементираних алгоритама. Машинско учење има фазу „учења" која представља извођење и побољшавање коришћених параметара, уз претходно учење из базе искустава.

Врло погодан у решавању многих проблема машинског учења у којима посматрамо уређене низове је управо СММ. У циљу обухватања свих параметара СММ-а биће коришћена нотација $\mathbf{\Theta} = (\vec{\pi}, \mathbf{T}, \mathbf{O})$. Три су основна проблема којима се се бави СММ и који су описани у овом раду:



Слика 2.3: Графички приказ концепта машинског учења.

- *Вероватноћа постојања посматраног низа излаза.* За дати низ излаза, $\{y_t\}_{t=0}^{T}$, дефинише се вероватноћа да је он производ датог СММ-а

$$\mathbb{P}(Y_0 = y_0, ...Y_T = y_T|\mathbf{\Theta})$$

  Овај проблем се решава **forward алгоритмом**.

- *Највероватнији низ стања који би произвео дати низ излаза.* Уз дати низ излаза, $\{y_t\}_{t=0}^{T}$, одређује се највероватнији низ стања, $\{\hat{x}_t\}_{t=0}^{T}$, који би га произвео уз дат СММ $\mathbf{\Theta}$.

$$\{\hat{x}_t\}_{t=0}^{T} \stackrel{\text{def}}{=} \underset{\{x_t\}_{t=0}^{T}}{\arg\max} \mathbb{P}(\{x_t\}_{t=0}^{T}|\{y_t\}_{t=0}^{T}, \mathbf{\Theta})$$

  Овај проблем се решава **Viterbi алгоритмом**.

- *Прилагођавање параметара модела.* Уз дати низ излаза $\{y_t\}_{t=0}^{T}$ и СММ $\mathbf{\Theta}$, конструише се нови СММ $\mathbf{\Theta'}$, за који је већа веоватноћа да би произвео дати низ излаза.

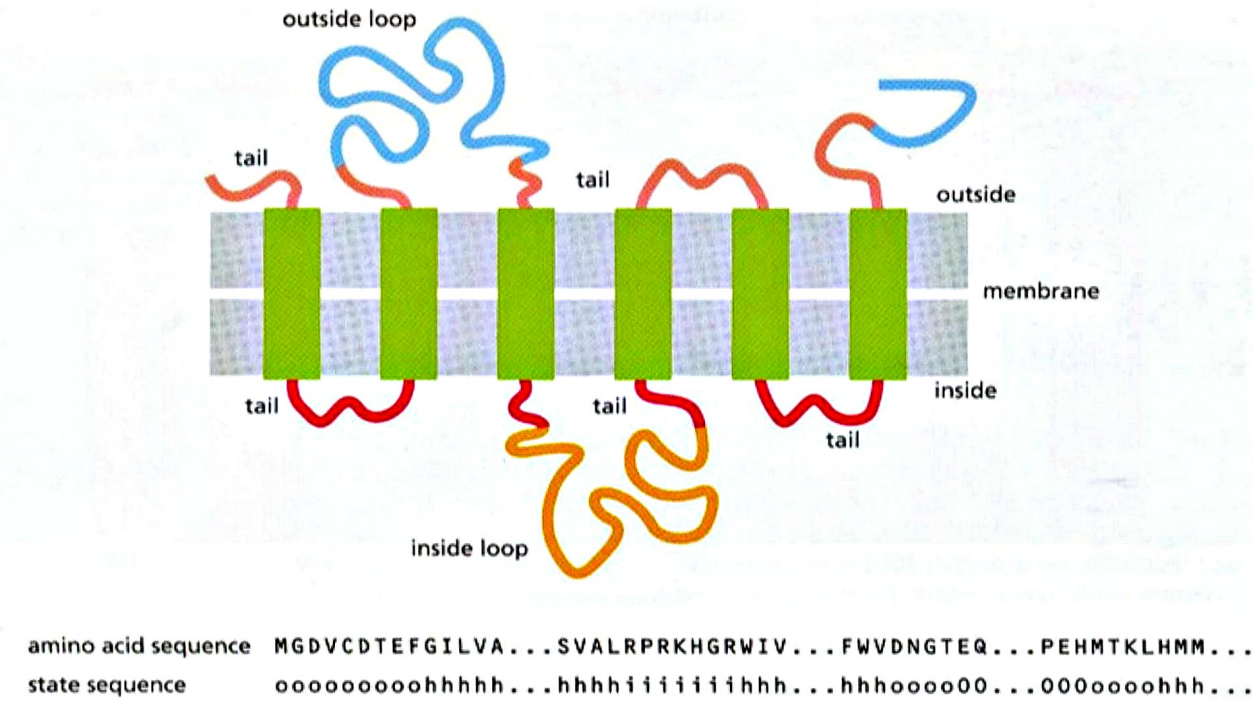
$$\mathbb{P}(\{y_t\}_{t=0}^{T}|\mathbf{\Theta'}) \geq \mathbb{P}(\{y_t\}_{t=0}^{T}|\mathbf{\Theta})$$

  Овај проблем се решава **Baum-Welch алгоритмом**.

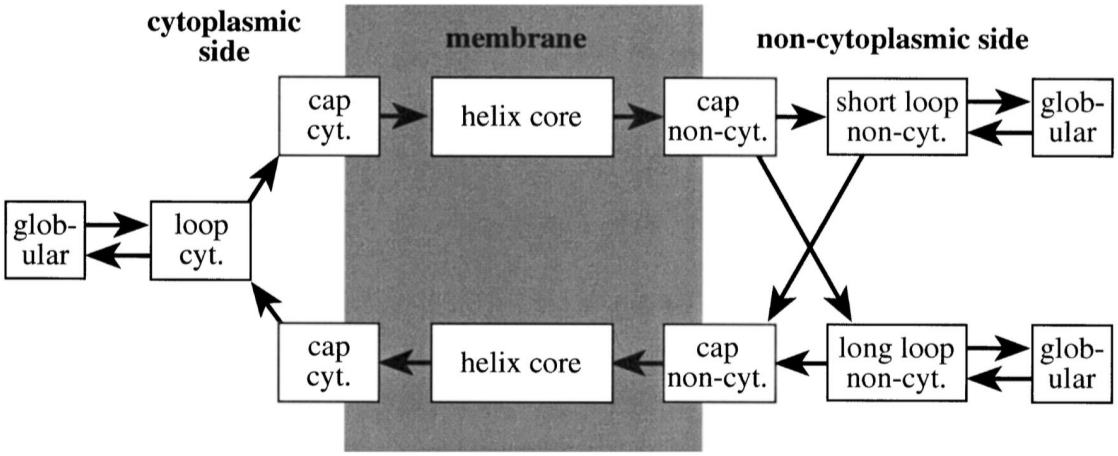
2.3 Скривени Марковљеви модели и протеини

Главни циљ овог пројекта био је да методом машинског учења и уз скривене Марковљеве моделе предвиди положаје аминокиселина протеина у односу на ћелијску мембрану.

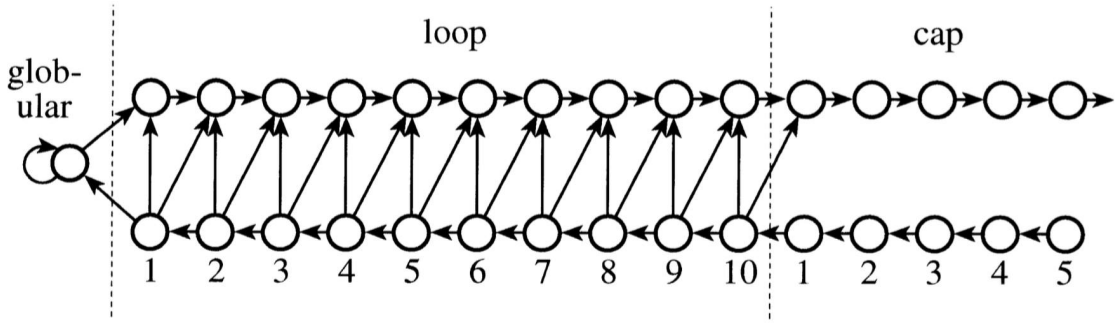
Говорећи у контексту СММ-ова, можемо рећи да је скуп излаза управо скуп аминокиселина. Стања на вишем нивоу можемо поделити на она која се налазе у ћелијском простору *(inside - i)*, ван њега *(ouside - o)* или у ћелијској мембрани *(membrane - m / helix - h)*. Свака од ових група има одређене групе стања која су приказана на предстојећим сликама.



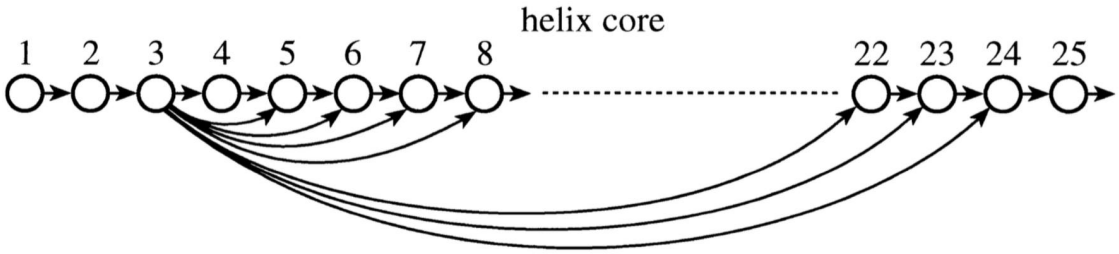
Слика 2.4: Графички приказ положаја пептида у односу на ћелијску мембрану.



Слика 2.5: Графички приказ стања аминокиселина на вишем нивоу.



Слика 2.6: Графички приказ стања аминокиселина у ванмембранским стањима.



Слика 2.7: Графички приказ стања аминокиселина у хеликсу.

Посматрано на нижем нивоу, свака група стања има одређени и тачно дефинисан број стања [5]. Тако рецимо, уз овако дефинисан CMM може се приметити да helix мора садржати макар 5 стања, док сар увек има константан број стања, и то 5.

Уз овакву конструкцију модела, можемо рећи да је број стања 133, а број излаза 20. Свако од стања има свој положај у односу на ћелијску мембрану.

Глава 3

Имплементација

У оквиру пројекта су успешно имплементиране функције потребне за ефикасно одређивање параметара $\mathbf{\Theta}$-а, као и оне које уз помоћ постојећих параметара одређују потребне перформансе. У овој глави су описане имплементиране функције поткрепљене математичком позадином и пропраћене одговарајућим псеудокодовима.

3.1 Forward алгоритам

Forward алгоритам користи одређивању вероватноће да је дати низ излаза произведен од стране датог CMM-а. Другим речима, forward алгоритам има за задатак да израчуна вредност L која је дефинисана на следећи начин:

$$L = \mathbb{P}(\vec{y}|\Theta) = \sum_{\vec{x} \in S^T} \mathbb{P}(\vec{y}|\vec{x}, \Theta)\mathbb{P}(\vec{x}|\mathbf{\Theta}).$$

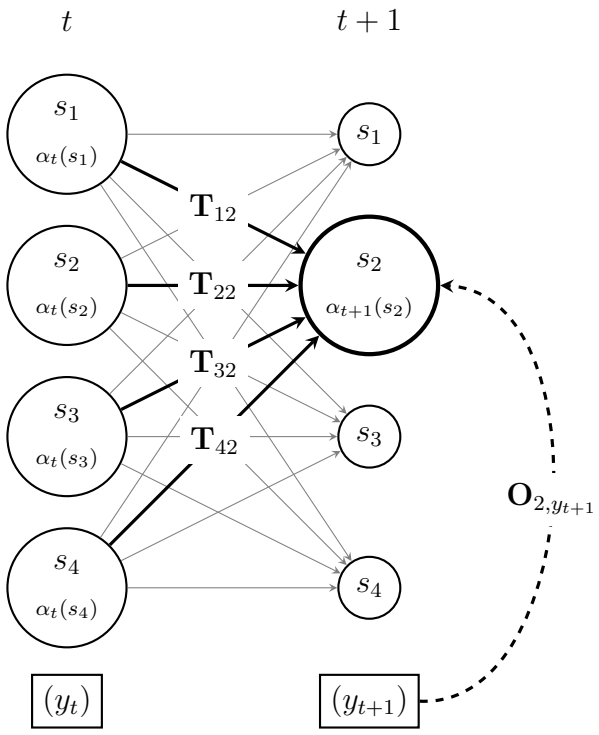
Може се приметити да је начин израчунавања L на овај начин велике сложености, што доводи до приступа динамичким програмирањем.

Дефинисаћемо матрицу α као матрицу вероватноће доласка у одређено стање након одређеног броја корака. Другим речима, нотацију $\alpha_t(x)$ ћемо користити као вероватноћу да је након t корака систем завршио у стању x.

$$\alpha_t(x) \stackrel{\text{def}}{=} \mathbb{P}(\{y_t\}_{i=1}^t, x_t = x|\mathbf{\Theta}).$$

Након овако дефинисане матрице, лако је израчунати вредност L.

$$L = \sum_{x \in S} P(\vec{y}, x_T = x|\mathbf{\Theta}) = \sum_{x \in S} \alpha_T(x).$$



Слика 3.1: Илустрација познатих или претходно израчунатих параметара од којих вредност $\alpha_{t+1}(s_2)$ зависи.

Базни случај, тј. вероватноћа да је у првом кораку систем био у стању x и притом произвео излаз y_1, се лако рачуна као:

$$\alpha_1(x) = \mathbb{P}(y_1, x_1 = x|\mathbf{\Theta}) = \pi_x \mathbf{O}_{x,y_1}.$$

Након израчунатих свих вредности $\alpha_t(x)$ за сва стања x, управо како претходна слика илуструје, можемо извести $\alpha_{t+1}(x)$ на следећи начин:

$$\begin{aligned}
\alpha_{t+1}(x) &= \mathbb{P}(\{y_i\}_{i=1}^{t+1}, x_{t+1} = x|\mathbf{\Theta}) \\
&= \mathbb{P}(y_{t+1}|x_{t+1} = x, \{y_i\}_{i=1}^{t}, \mathbf{\Theta})\mathbb{P}(\{y_i\}_{i=1}^{t}, x_{t+1} = x|\mathbf{\Theta}) \\
&= \mathbb{P}(y_{t+1}|x_{t+1} = x, \{y_i\}_{i=1}^{t}, \mathbf{\Theta}) \sum_{x' \in S} \mathbb{P}(\{y_i\}_{i=1}^{t}, x_{t+1} = x, x_t = x'|\mathbf{\Theta}) \\
&= \mathbf{O}_{x,y_{t+1}} \sum_{x' \in S} \mathbb{P}(\{y_i\}_{i=1}^{t}, x_t = x'|\mathbf{\Theta})\mathbb{P}(x_{t+1} = x|x_t = x', \mathbf{\Theta}) \\
&= \mathbf{O}_{x,y_{t+1}} \sum_{x' \in S} \alpha_t(x')\mathbf{T}_{x'x}.
\end{aligned}$$

Forward алгоритам има проблем поткорачења. Уочавамо да се вредности параметара,

који се користе, рачунају као суме, а не као производи, па није могуће решити овај проблем логаритмовањем. Нормализација $\alpha_t(x)$ за свако t решава проблем поткорачења **коефицијентом скалирања** c_t.

$$c_t = \frac{1}{\sum_{x' \in S} \alpha_t(x)}.$$

Вредност $\alpha_t(x)$ која се заправо чува у наредним корацима је

$$\hat{\alpha}_t(x) = c_t \alpha_t(x).$$

Користећи коефицијент скалирања, ову тражену вредност лако можемо представити као

$$L = \mathbb{P}(\vec{y}|\boldsymbol{\Theta}) = \frac{\sum_{x \in S} \hat{\alpha_T}(x)}{c_1 c_2 \cdots c_T} = \prod_{t=1}^{T} \frac{1}{c_t}$$

што се након логаритмовања може представити и као

$$\log L = -\sum_{t=1}^{T} \log c_t.$$

Временска сложеност алгоритма је $O(Tn^2)$, док је меморијска $O(Tn)$.

Псеудокод алгоритма *forward* приложен је у наставку.

FORWARD($\vec{y}$)

for all $x \in S$ // Базни случај
    $\alpha_1(x) \leftarrow \pi_x \mathbf{O}_{x,y_1}$
$c_1 \leftarrow \frac{1}{\sum_{x' \in S} \alpha_1(x)}$ // Рачунање коефицијента скалирања
for all $x \in S$
    $\alpha_1(x) \leftarrow c_1 \alpha_1(x)$ // Нормализација
for $t \leftarrow 2$ **to** T
    for all $x \in S$
        $\alpha_t(x) \leftarrow \mathbf{O}_{x,y_t} \sum_{x' \in S} \alpha_{t-1}(x') \mathbf{T}_{x'x}$ // Вредност $\alpha_t(x)$
    $c_t \leftarrow \frac{1}{\sum_{x' \in S} \alpha_t(x)}$ // Рачунање коефицијента скалирања
    for all $x \in S$
        $\alpha_t(x) \leftarrow c_t \alpha_t(x)$ // Нормализација
$L \leftarrow -\sum_{t=1}^{T} \log c_t$ // Вероватноћа вектора y
return (α, c, L)

3.2 Backward алгоритам

Backward вероватноће, $\beta_t(x)$, дефинишемо као вероватноће да кренувши од $(t+1)$. корака систем произведе све излазе до краја, уз то да се на кораку t налази у стању x.

$$\beta_t(x) \stackrel{\text{def}}{=} \mathbb{P}(\{y_i\}_{i=t+1}^{T} | x_t = x, \mathbf{\Theta}).$$

Backward вероватноће се ефикасно могу израчунати **backward алгоритмом**, који се на неки начин може замислити као „инверз" forward алгоритма. Он је подрутина Baum-Welch алгоритма.

Базни случајеви су $\beta_T(x) = 1$, што представља вероватноћу произвођења празне секвенце низа, без обзира на то којим стањем се почиње. На сличан начин на који су рекурентно изведене вредности за $\alpha_t(x)$, тако се и у овом случају могу извести $\beta_t(x)$, ако су претходно израчунате вредности $\beta_{t+1}(x)$ за свако x.

$$\begin{aligned}
\beta_t(x) &= \mathbb{P}(\{y_i\}_{i=t+1}^{T}, x_t = x | \mathbf{\Theta}) \\
&= \sum_{x' \in S} \mathbb{P}(\{y_i\}_{i=t+1}^{T}, x_{t+1} = x' | x_t = x, \mathbf{\Theta}) \\
&= \sum_{x' \in S} \mathbb{P}(x_{t+1} = x' | x_t = x, \mathbf{\Theta}) \mathbb{P}(\{y_i\}_{i=t+1}^{T} | x_{t+1} = x', x_t = x, \mathbf{\Theta}) \\
&= \sum_{x' \in S} \mathbf{T}_{xx'} \mathbb{P}(\{y_i\}_{i=t+1}^{T} | x_{t+1} = x', \mathbf{\Theta}) \\
&= \sum_{x' \in S} \mathbf{T}_{xx'} \mathbb{P}(y_i | x_{t+1=x'}, \mathbf{\Theta}) \mathbb{P}(\{y_i\}_{i=t+2}^{T} | x_{t+1} = x', \mathbf{\Theta}) \\
&= \sum_{x' \in S} \mathbf{T}_{xx'} \mathbf{O}_{x', y_{t+1}} \beta_{t+1}(x').
\end{aligned}$$

Слично као и код forward алгоритма јавља се проблем поткорачења који ће се и овде решити управо коефицијентом скалирања, па је аналогно вредност $\beta_t(x)$ која се заправо чува у наредним корацима

$$\hat{\beta}_t(x) = c_{t+1} \beta_t(x).$$

Специјално, $\beta_T(x)$ вредност није скалирана.

Као и код forward алгоритма, временска сложеност је $O(Tn^2)$, док је меморијска $O(Tn)$.

Псеудокод алгоритма *backward* приложен је у наставку.

BACKWARD$(\vec{y}, c)$

1 **for all** $x \in S$ // Базни случај
2 $\beta_T(x) \leftarrow 1$
3 **for** $t \leftarrow T-1$ **downto** 1
4 **for all** $x \in S$
5 $\beta_t(x) \leftarrow \sum_{x' \in S} \mathbf{T}_{xx'} \mathbf{O}_{x',y_{t+1}} \beta_{t+1}(x')$ // Вредност $\beta_t(x)$
6 $\beta_t(x) \leftarrow c_{t+1}\beta_t(x)$ // Нормализација
7 **return** (β)

3.3 Viterbi алгоритам

Задатак Viterbi алгоритма је да пронађе највероватнији низ стања који је произвео дати низ излаза, уз познате параметре СММ-а $\mathbf{\Theta} = (\vec{\pi}, \mathbf{T}, \mathbf{O})$ [4].

Алгоритам користи приступ динамичког програмирања. Он има готово идентичну замисао имплементације као forward алгоритам, односно користи се концепт рачунања појединих вредности одједном да се исте не би рачунале више пута. Основна грађа овог алгоритма је рачунање колика би била вероватноћа да се након t обрађених елемената (пређених корака) у низу излаза систем нађе у стању x, или, како ћемо означавати ову вредност $V_t(x)$. Базни случај, $V_1(x)$, ћемо лако израчунати као

$$V_1(x) = \vec{\pi}_x \mathbf{O}_{x,y_1}$$

који управо представља вероватноћу кретања стањем x и његовим прозвођењем првог излаза. Након израчунавања свих вредности $V_t(x)$ за неко t, $V_{t+1}(x)$ се лако може израчунати као:

$$V_{t+1}(x) = \max_{x' \in S} V_t(x') \mathbf{T}_{x'x} \mathbf{O}_{x,y_{t+1}}$$

који представља завршавање у стању x' у претходном кораку, прелазак из овог у стање x и најзад његово произвођење $(t+1)$- вог излаза. Такође је сврсисходно чување стања на кораку t које је са максималном вероватноћом довело до $(t+1)$-вог стања. Вредност поља ове матрице се рачуна на следећи начин:

$$x^*{}_{t+1}(x) = \arg\max_{x' \in S} V_t(x') \mathbf{T}_{x'x} \mathbf{O}_{x,y_{t+1}}.$$

Оптималан низ стања, односно низ стања који је са највећом вероватноћом произвео задати низ излаза, се може одредити методом бектрекинга. Последње стање $vect_T$ се лако може одредити као највероватније стање у којем се систем налазио у последњем кораку, односно као:

$$vect_T = \arg\max_{x \in S} V_T(x)$$

док се стање на кораку t може одредити као стање које је довело до стања које је највероватније до којег се дошло на кораку $t+1$, односно релацијом:

$$vect_t = x^*_{t+1}(vect_{t+1}).$$

Главни проблем у самој имплементацији овог алгоритма је био поткорачење, односно немогућност чувања бројева много мањих од 1 са одређеном тачношћу. С обзиром да су већина рачунања управо множење оваквих бројева, проблем се врло лако може решити логаритмовањем, тако да се израчунавање своди на сабирање логаритмованих вредности.

Временска сложеност алгоритма је $O(Tn^2)$, док је меморијска $O(Tn)$. У наставку је приказан псеудокод овог алгоритма.

```
VITERBI(y⃗)
 1  for all x ∈ S                                          // Базни случај
 2      V_1(x) ← log π_x + log O_{x,y_1}
 3  for t ← 2 to T                          // Прављење матрице V и низа x*
 4      for all x ∈ S
 5          V_t(x) ← V_{t−1}(1) + log T_{1,x} + log O_{x,y_t}
 6          x*_t(x) ← 1
 7          for all x′ ∈ S \ {1}                          // Тражење максимума
 8              p ← V_{t−1}(x′) + log T_{x′x} + log O_{x,y_t}
 9              if p > V_t(x)
10                  V_t(x) ← p
11                  x*_t(x) ← x′
12  maxx ← V_T(1)
13  vect_T ← 1
14  for all x ∈ S              // Тражење највероватнијег стања завршетка
15      if V_T(x) > maxx
16          maxx ← V_T(x)
17          vect_T ← x
18  if maxx < 0                // Провера случаја када дати низ y није могућ
19      for t ← 1 to T
20          vect_t ← −1
21  else for t ← T downto 2
22          vect_{t−1} ← x*_t(vect_t)
23  return (vect)
```

3.3.1 k-best Viterbi

Као што је већ објашњено, задатак Viterbi алгоритма је да на основу датог низа излаза y одреди највероватнији низ стања који је произвео овај низ. Међутим, у пракси се често догађа да, иако је највероватнији, овај низ није заправо тражени низ стања. С тим у вези, надоградња Viterbi алгоритма која се зове *k-best Viterbi* настоји решавању овог проблема, односно повећању вероватноће налажења изворног низа.

За разлику од Viterbi алгоритма, у k-best Viterbi алгоритму се не налази једно најбоље решење које представља највероватнији низ стања, већ k таквих. Вредност параметра k није фиксна и у самој имплементацији је битно мењати је зарад веће ефикасности рада Viterbi алгоритма. Након k одређених низова потребно је одредити низ стања на који ових k низова указују као највероватнији за произвођене датог низа излаза. На месту t

биће стање x ако се у k одређених низова ово стање највише пута понавља на месту t.

Приступ решавању проблема налажења k најбољих решења је приступ динамичким програмирањем. Вредност $V_t^l(x)$ представља l-ту по реду вероватноћу завршавања у стању x након t корака, где l узима вредност из скупа $1, 2, ..., k$. Базни случај је

$$V_1^l(x) = \pi_x \mathbf{O}_{x,y_1}.$$

Дефиниција 4. Нека је $\Psi_{t+1}^r(x) = V_t^l(x')\mathbf{T}_{x'x}\mathbf{O}_{x,y_{t+1}}, x' \in S, l \in \{1, 2, ..., k\}$, при чему r узима вредности из скупа $\{1, 2, ..., kn\}$. Тада дефинишемо пермутацију бројева $\Psi_{t+1}^s(x)$ на следећи начин:

$$\Psi_{t+1}^{(1)}(x) \geq \Psi_{t+1}^{(2)}(x) \geq ... \geq \Psi_{t+1}^{(kn)}(x),$$

односно, кажемо да је $\Psi_{t+1}^{(r)}(x)$ r-та по реду вредност у скупу вредности

$$\{\Psi_{t+1}^1(x), \Psi_{t+1}^2(x), ..., \Psi_{t+1}^{kn}(x)\}.$$

Тада за свако $t \in \{1, 2, ..., T-1\}$ и $l \in \{1, 2, ..., k\}$ важи:

$$V_{t+1}^l(x) = \Psi_{t+1}^{(l)}(x).$$

Дефиниција 5. Слично, нека је $\Omega_{t+1}^r(x) = \arg\max_{x' \in S} V_t^l(x')\mathbf{T}_{x'x}\mathbf{O}_{x,y_{t+1}}, x' \in S, l \in \{1, 2, ..., k\}$, при чему r узима вредности из скупа $\{1, 2, ..., kn\}$. Тада дефинишемо пермутацију бројева $\Omega_{t+1}^s(x)$ на следећи начин:

$$\Omega_{t+1}^{(1)}(x), \Omega_{t+1}^{(2)}(x), ..., \Omega_{t+1}^{(kn)}(x),$$

при чему важи

$$\Psi_{t+1}^{(1)}(x) \geq \Psi_{t+1}^{(2)}(x) \geq ... \geq \Psi_{t+1}^{(kn)}(x).$$

Тада за свако $t \in \{1, 2, ..., T-1\}$ и $l \in \{1, 2, ..., k\}$ важи:

$$x^{*l}{}_{t+1}(x) = \Omega_{t+1}^{(l)}(x).$$

Дефиниција 6. Нека је $\Delta_T^r = \underset{x \in S}{\arg}\Psi_T^r(x), l \in \{1, 2, ..., k\}$, при чему r узима вредности из скупа $\{1, 2, ..., kn\}$. Тада дефинишемо пермутацију бројева Δ_T^r на следећи начин:

$$\Delta_T^{(1)}, \Delta_T^{(2)}, ..., \Delta_T^{(kn)},$$

при чему важи

$$\Psi_T^{(1)}(x) \geq \Psi_T^{(2)}(x) \geq ... \geq \Psi_T^{(kn)}(x).$$

k-ти по реду оптималан низ стања се може одредити методом бектрекинга. Последње стање $vect_T^l$ се лако може одредити као:

$$vect_T^l = \Delta_T^{(l)},$$

док се стање на кораку t може одредити релацијом:

$$vect_t^l = x^{*l}{}_{t+1}(vect_{t+1}^l).$$

3.4 Baum-Welch алгоритам

Baum-Welch алгоритам се користи у такозваном „тренингу", односно служи за одређивање параметара $\mathbf{\Theta}$-а на основу базе знања [3]. Имплементација алгоритма се састоји у надовезаним итерацијама. Након сваке итерације од тренутног модела $\mathbf{\Theta}$ добијамо нови модел $\mathbf{\Theta}'$. На овај начин, вредности параметара конвергирају онима који најбоље описују дату базу знања. При великом броју итерација или након квадратног одступања параметара $\mathbf{\Theta}$-а и $\mathbf{\Theta}'$-а које не прелази процењену вредност, можемо рећи да су вредности параметара прилижно једнаке жељеним.

Свака итерација се састоји из два корака. У првом се рачуна тзв. „очекивање" *(Expectation)*, односно рачунање помоћних параметара уз помоћ којих ће се у другом кораку „максимизовати" *(Maximisation)* вредности параметара система. Ова два корака се управо зову *E step*, односно *M step*.

3.4.1 E step

Овај корак се састоји у рачунању два помоћна параметра, γ и ξ.

1. $\gamma_x(t)$ представља вероватноћу да се систем у тренутку t нашао у стању x.

$$\gamma_x(t) \stackrel{\text{def}}{=} \mathbb{P}(X_t = x|\vec{y}, \mathbf{\Theta}).$$

Користећи Бајесову теорему долазимо до следећег:

$$\begin{aligned}
\mathbb{P}(X_t = x|\vec{y}, \mathbf{\Theta}) &= \frac{\mathbb{P}(\vec{y}|X_t = x, \mathbf{\Theta})\mathbb{P}(X_t = x|\mathbf{\Theta})}{\mathbb{P}(\vec{y}|\mathbf{\Theta})} \\
&= \frac{\mathbb{P}(\{y_i\}_{i=1}^t|X_t = x, \mathbf{\Theta})\mathbb{P}(X_t = x|\mathbf{\Theta})\mathbb{P}(\{y_i\}_{i=t+1}^T|X_t = x, \mathbf{\Theta})}{\mathbb{P}(\vec{y}|\mathbf{\Theta})} \\
&= \frac{\mathbb{P}(\{y_i\}_{i=1}^t, X_t = x|\mathbf{\Theta})\mathbb{P}(\{y_i\}_{i=t+1}^T|X_t = x, \mathbf{\Theta})}{\mathbb{P}(\vec{y}|\mathbf{\Theta})} \\
&= \frac{\alpha_t(x)\beta_t(x)}{\mathbb{P}(\vec{y}|\mathbf{\Theta})}.
\end{aligned}$$

Изразивши $\mathbb{P}(\vec{y}|\mathbf{\Theta})$ преко коефицијента скалирања добијамо следеће:

$$\begin{aligned}
\mathbb{P}(X_t = x|\vec{y}, \mathbf{\Theta}) &= \frac{\alpha_t(x)\beta_t(x)}{\mathbb{P}(\vec{y}|\mathbf{\Theta})} \\
&= \frac{\alpha_t(x)\beta_t(x)}{\prod_{s=1}^T \frac{1}{c_s}} \\
&= \left(\alpha_t(x)\prod_{s=1}^t c_s\right)\left(\beta_t(x)\prod_{s=t+1}^T c_s\right).
\end{aligned}$$

Одавде лако можемо израчунати вредност $\gamma_x(t)$:

$$\gamma_x(t) = \hat{\alpha}_t(x)\hat{\beta}_t(x).$$

2. $\xi_{ij}(t)$ представља вероватноћу преласка из стања i у стање j у тренутку t.

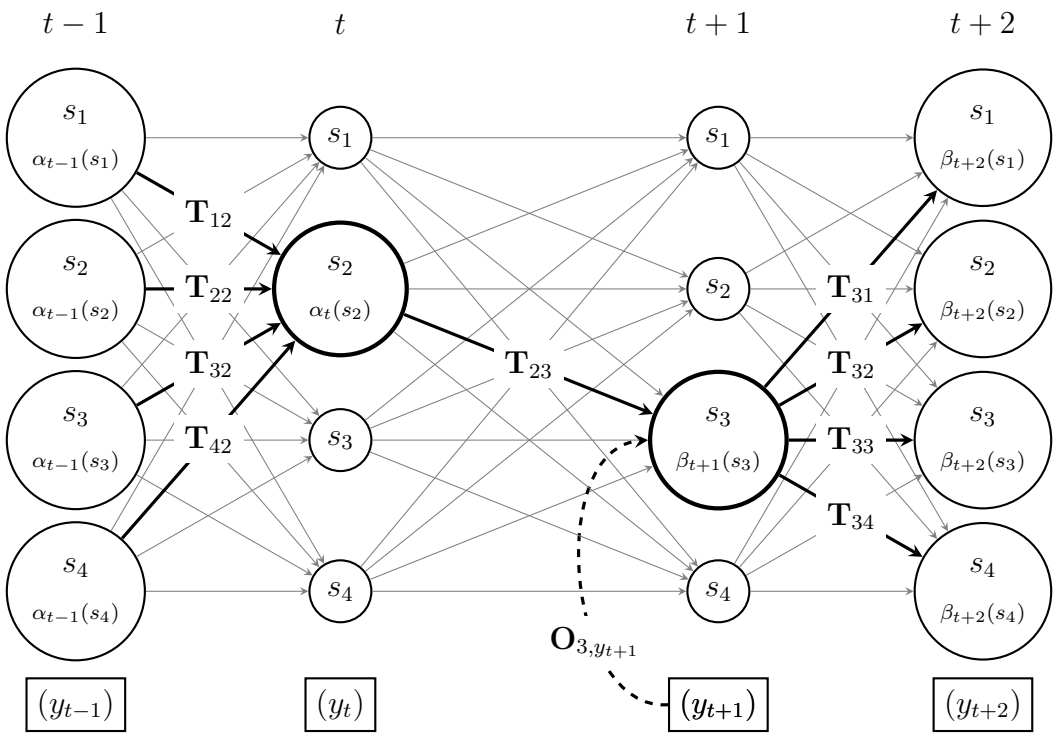
$$\xi_{ij}(t) \stackrel{\text{def}}{=} \mathbb{P}(X_t = i, X_{t+1} = j|\vec{y}, \mathbf{\Theta}).$$

Користећи претпоставке модела и Бајесову теорему $\mathbb{P}(X_t = i, X_{t+1} = j|\vec{y}, \mathbf{\Theta})$ можемо представити преко већ познатих параметара:

$$\mathbb{P}(X_t = i, X_{t+1} = j|\vec{y}, \mathbf{\Theta}) = \frac{\alpha_t(i)\mathbf{T}_{ij}\mathbf{O}_{j,y_{t+1}}\beta_{t+1}(j)}{\prod_{s=1}^T \frac{1}{c_s}}.$$

Дакле, вредност $\xi_{ij}(t)$ је:

$$\xi_{ij}(t) = c_{t+1}\hat{\alpha}_t(i)\mathbf{T}_{ij}\mathbf{O}_{j,y_{t+1}}\hat{\beta}_{t+1}(j).$$



Слика 3.2: Илустрација која приказује на основу чега се израчунава $\mathbb{P}(X_t = i, X_{t+1} = j|\vec{y}, \mathbf{\Theta})$, са четири стања СММ-а, уз $i = s_2$ и $j = s_3$.

3.4.2 M step

Овај корак се састоји у модификовању тренутних параметара $\mathbf{\Theta}$-а, што се изводи на следећи начин:

1. Вероватноћа почетног стања, π'_x, представља вероватноћу бивања у стању x у тренутку 1:
$$\pi'_x = \gamma_x(1)$$

2. Вероватноћа преласка, $\mathbf{T}'_{ij}$, се може израчунати као вероватноћа да након што је систем био у стању i прелази у стање j:
$$\mathbf{T}'_{ij} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$$

3. Вероватноћа произведеног излаза, $\mathbf{O}'_{xy'}$, се рачуна као вероватноћа да стање x произведе излаз y':
$$\mathbf{O}'_{xy'} = \frac{\sum_{t=1}^{T} \mathbb{I}(y_t = y')\gamma_x(t)}{\sum_{t=1}^{T} \gamma_x(t)}$$
$\mathbb{I}$ је индикаторска функције, тј. узима вредност 1 ако важи $y_t = y'$, а 0 у супротном.

Овако модификовани параметри чине нови модел $\mathbf{\Theta}' = (\vec{\pi}', \mathbf{T}', \mathbf{O}')$.

3.4.3 Имплементација Baum-Welch алгоритма

Неизоставан део имплементације Baum-Welch алгоритма је процена броја итерација и квадратног одступања који задовољавају конвергенцију параметара $\mathbf{\Theta}$-а стварним вредностима истих. Као уобичајена пракса за одређивање вредности квадратног одступања односно толеранције (*tolerance*) се узима вредност 10^{-6}, док је за максималан број итерација та вредност реда величине 10^6. Наравно, битно је напоменути да ови параметри варирају и да је поред саме имплементације алгоритма врло битна и процена њихових вредности зарад ефикасности одређивања параметара $\mathbf{\Theta}$-а.

При имплементацији овог алгоритма због, како меморијске, тако и временске сложености, није потребно E step и M step извршавати засебно један од другог, али ни независно од саме модификације параметара $\mathbf{\Theta}$-а. Када би имплементација била утемељена на математичким принцима објашњеним у претходном поглављу, али не и оптимизована на начин који представља симултано рачунање вредности помоћних параметара γ и ξ, у

контексту сваке итерације меморијска сложености би била $O(n^2T)$. Имплементација на начин описан у предстојећем псеудокоду смањује меморијску сложеност сваке итерације на $O(nT)$, док је временска сложеност у том случају $O(n^2T)$.

BaumWelch($\vec{y}, iterations, tolerance$)

```
1  oldL ← 0
2  for iter ← 1 to iterations
3      (α, c, L) ← Forward(y)
4      β ← Backward(y, c)
5      for all x ∈ S
6          π_x = α_1(x)β_1(x)                          // Модификовање π
7          DD ← Σ_{t=1}^{T-1} α_t(x)β_t(x)             // DD ← Σ_{t=1}^{T-1} γ_t
8          for all x' ∈ S
9              NN ← Σ_{t=1}^{T-1} c_{t+1}α_t(x)O_{x',y_{t+1}}β_{t+1}(x')  // NN ← Σ_{t=1}^{T-1} ξ_t
10             T_{xx'} ← T_{xx'}NN/DD                  // Модификовање T
11         DD ← DD + α_T(x)β_T(x)                      // DD ← Σ_{t=1}^{T} γ_t
12         OO_{x,y'} ← Σ_{t=1}^{T} I(y_t = y')α_t(x)β_t(x)   // Модификовање O
13         for all y' ∈ O
14             OO_{x,y'} ← OO_{x,y'}/DD
15         O ← OO
16     if |oldL − L| > tolerance break // Провера испуњења одступања
17 odlL ← L
```

3.4.4 Тренинг са више низова у бази знања

Претходни опис Baum-Welch алгоритма се односио на случај када се у бази знања налази не више од једног низа y. Међутим, то је редак случај, тако да је неопходно дискутовати рад алгоритма у случају више одвојених низова. Нека их је k. Увешћемо ознаке $\gamma_x^l(t)$ и $\xi_{ij}^l(x)$ за l-ти низ. Сада једноставно можемо изразити $\mathbf{T}'_{ij}$ и $\mathbf{O}'_{xy'}$ на следећи начин:

$$\mathbf{T}'_{ij} = \frac{\sum_{l=1}^{k}\sum_{t=1}^{T-1}\xi_{ij}^l(t)}{\sum_{l=1}^{k}\sum_{t=1}^{T-1}\gamma_i^l(t)},$$

$$\mathbf{O}'_{xy'} = \frac{\sum_{l=1}^{k}\sum_{t=1}^{T}\mathbb{I}(y_t^l = y')\gamma_x^l(t)}{\sum_{l=1}^{k}\sum_{t=1}^{T}\gamma_x^l(t)}.$$

Такође, неопходно је израчунати и вектор π, и то на следећи начин:

$$\pi'_x = \frac{1}{k}\sum_{l=1}^{k}\gamma_x^l(1).$$

3.4.5 Restricted Baum-Welch

Поред описа машинског учења и Марковљевих модела, циљ овог рада је и приказ примене истих на конкретном примеру. Од многобројних примена изабрано је препознавање елемената трансмембраских протеина. Поред могућих стања у којима се свака од аминокиселина може наћи, неизбежно је напоменути да се може десити да у одређеном тренутку одређена аминокиселина не сме бити у неком од могућих стања. Оваква стања ћемо звати *забрањена стања*. С обзиром на ово, неопходно је модификовати Baum-Welch алгоритам зарад ефикаснијег одређивања вредности параметара. Restricted Baum-Welch се састоји у модификовању матрица α и β, односно forward и backward алгоритама.

Скуп $tags_t$ дефинишемо као скуп недозвољених стања у тренутку t. До елемената ових скупова дошло се биолошким истраживањима.

За модификовање матрице α потребно је да вредности $\alpha_t(x)$ буду једнаке нули онда када је у тренутку t забрањено стање x. Подсетимо се да $\alpha_t(x)$ представља вероватноћу да је након t корака систем завршио у стању x. Међутим ако је x забрањено стање у тренутку t, ова вероватноћа је једнака нули. Модификовањем матрице α односно forward алгоритма на овај начин управо постижемо жељени резултат. Почевши од базног случаја, вредности $\alpha_1(x)$ биће једнаке:

$$\alpha_1(x) = \mathbb{I}(x \notin tags_1)\pi_x \mathbf{O}_{x,y_1},$$

при чему је $\mathbb{I}$ индикатор функције, тј. узима вредност 1 ако стање x није забрањено у тренутку t, а 0 у супротном. Слично, можемо одредити вредности за наредне кораке:

$$\alpha_{t+1}(x) = \mathbb{I}(x \notin tags_{t+1})\mathbf{O}_{x,y_{t+1}} \sum_{x' \in S} \alpha_t(x')\mathbf{T}_{x'x}.$$

Псеудокод овако модификованог *forward* алгоритма приложен је у наставку.

FORWARD$(\vec{y}, tags)$

for all $x \in S$ // Базни случај
    if $x \notin tags_1$
        $\alpha_1(x) \leftarrow \pi_x \mathbf{O}_{x,y_1}$
    else $\alpha_1(x) \leftarrow 0$
$c_1 \leftarrow \frac{1}{\sum_{x' \in S} \alpha_1(x)}$ // Рачунање коефицијента скалирања
for all $x \in S$
    $\alpha_1(x) \leftarrow c_1 \alpha_1(x)$ // Нормализација
for $t \leftarrow 2$ **to** T
    for all $x \in S$
        if $x \notin tags_t$
            $\alpha_t(x) \leftarrow \mathbf{O}_{x,y_{t+1}} \sum_{x' \in S} \alpha_{t-1}(x') \mathbf{T}_{x'x}$ // Вредност $\alpha_t(x)$
        else $\alpha_t(x) \leftarrow 0$
    $c_t \leftarrow \frac{1}{\sum_{x' \in S} \alpha_t(x)}$ // Рачунање коефицијента скалирања
    for all $x \in S$
        $\alpha_t(x) \leftarrow c_t \alpha_t(x)$ // Нормализација
$L \leftarrow -\sum_{t=1}^{T} \log c_t$ // Вероватноћа вектора y
return (α, c, L)

Аналогно бива модификована и матрица β. С обзиром да $\beta_t(x)$ представља вероватноћу произвођења низа y до самог краја у услов да се систем у тренуку t нашао у стању x, довољно је дефинисати да је вредност $\beta_t(x)$ једнака нули ако је стање x забрањено у стању t. Базни случај би у том случају изгледао:

$$\beta_T(x) = \mathbb{I}(x \notin tags_T),$$

док би за све остале кораке вредност $\beta_t(x)$ била једнака:

$$\beta_t(x) = \mathbb{I}(x \notin tags_t) \sum_{x' \in S} T_{xx'} \mathbf{O}_{x',y_{t+1}} \beta_{t+1}(x').$$

Као и у случају модификовања forward алгоритма, $\mathbb{I}$ је индикаторска функције, тј. узима вредност 1 ако стање x није забрањено у тренутку t, а 0 у супротном.

Псеудокод овако модификованог *backward* алгоритма приложен је у наставку.

BACKWARD$(\vec{y}, c, tags)$

1 **for all** $x \in S$ // Базни случај
2 **if** $x \notin tags_T$
3 $\beta_T(x) \leftarrow 1$
4 **else** $\beta_T(x) \leftarrow 0$
5 **for** $t \leftarrow T-1$ **downto** 1
6 **for all** $x \in S$
7 **if** $x \notin tags_t$
8 $\beta_t(x) \leftarrow \sum_{x' \in S} \mathbf{T}_{xx'} \mathbf{O}_{x', y_{t+1}} \beta_{t+1}(x')$ // Вредност $\beta_t(x)$
9 **else** $\beta_t(x) \leftarrow 0$
10 $\beta_t(x) \leftarrow c_{t+1} \beta_t(x)$ // Нормализација
11 **return** (β)

Глава 4

Евалуација

Ова глава садржи преглед метода коришћених за верификацију правилног рада имплементираних функција. Такође, представљена је и обрада базе знање зарад одређивања ефикасноси рада алгоритама и вероватноће доласка до одређених грешака.

4.1 Тестирање алгоритама

Неизоставан део сваког пројекта свакако је тестирање функција. Након извршеног тестирања закључак је да сваки од алгоритама одређује потребне параметре исправно. У наредном одељку ће бити приказани примери на којима су имплементирани алгоритми тестирани.

4.1.1 Forward и backward алгоритам

Као почетни параметри $\mathbf{\Theta}$-а узети су:

$$n = 2, m = 2$$

$$\pi = \begin{pmatrix} 0.5 & 0.5 \end{pmatrix}$$

$$\mathbf{T} = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$$

$$\mathbf{O} = \begin{pmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{pmatrix}$$

За низ излаза y узет је низ:

$$\vec{y} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Очекиване вредности нормализованих матрица α и β су:

$$\hat{\alpha} = \begin{pmatrix} 0.8182 & 0.8834 & 0.1907 & 0.7308 & 0.8673 \\ 0.1818 & 0.1166 & 0.8093 & 0.2692 & 0.1327 \end{pmatrix}$$

$$\hat{\beta} = \begin{pmatrix} 0.5923 & 0.3763 & 0.6533 & 0.6273 & 0.5000 \\ 0.4077 & 0.6237 & 0.3467 & 0.3727 & 0.5000 \end{pmatrix}$$

4.1.2 Viterbi алгоритам

Као почетни параметри $\mathbf{\Theta}$-а узети су:

$$n = 3, m = 4$$

$$\pi = \begin{pmatrix} 0.3 & 0.3 & 0.4 \end{pmatrix}$$

$$\mathbf{T} = \begin{pmatrix} 0.2 & 0.4 & 0.4 \\ 0.1 & 0.6 & 0.3 \\ 0.8 & 0.1 & 0.1 \end{pmatrix}$$

$$\mathbf{O} = \begin{pmatrix} 0.7 & 0.1 & 0.1 & 0.1 \\ 0.3 & 0.2 & 0.4 & 0.1 \\ 0.4 & 0.2 & 0.2 & 0.2 \end{pmatrix}$$

За низ излаза y узет је низ:

$$\vec{y} = \begin{pmatrix} 2 & 2 & 3 & 0 & 0 & 3 & 1 & 3 \end{pmatrix}$$

За највероватнији низ стања очекивање уз дате параметре је:

$$\vec{vect} = \begin{pmatrix} 1 & 1 & 2 & 0 & 2 & 0 & 2 & 0 \end{pmatrix}$$

4.1.3 Baum-Welch алгоритам

Као почетни параметри $\mathbf{\Theta}$-а узети су:

$$n = 2, m = 3$$

$$\pi = \begin{pmatrix} 0.5 & 0.5 \end{pmatrix}$$

$$\mathbf{T} = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

$$\mathbf{O} = \begin{pmatrix} 0.4 & 0.1 & 0.5 \\ 0.1 & 0.5 & 0.4 \end{pmatrix}$$

За низ излаза y узет је низ:

$$\vec{y} = \begin{pmatrix} 2 & 0 & 0 & 2 & 1 & 2 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 0 & 0 & 1 \end{pmatrix}$$

Након конвергенције, очекивани параметри су:

$$\pi' = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

$$\mathbf{T}' = \begin{pmatrix} 0.6909 & 0.3091 \\ 0.0934 & 0.9066 \end{pmatrix}$$

$$\mathbf{O}' = \begin{pmatrix} 0.5807 & 0.0010 & 0.4183 \\ 0.0000 & 0.7621 & 0.2379 \end{pmatrix}$$

4.2 База знања

Биолошким истраживањима многобројних протеина у разним организмима предвиђени су положаји аминокиселина у полипептидним ланцима посматраних протеина. Имајући увид у ове податке извршена је процена доброг рада имплементираних алгоритама.

Скуп података се састојао од 160 трансмембранских протеина. Информације о сваком протеину представљене су помоћу два низа. Први је представљао низ амикосилеина које се налазе у његовом саставу. Други је представљао положаје у којима се свака од њих налази, односно њихово постојање у ћелији (*i* - *inside*), међућелијском протору (*o* - *outside*) или ћелијској мембрани (*M* - *membrane*).

4.3 Перформансе мерења

За процену колико је направљен модел ефикасан потребно је над базом знања извршити истраживања и то уз мерење тачно одређених параметара. Мерени параметри говоре о учесталости одређених грешака.

Највероватнији низ стања који производи дати низ излаза може у себи садржати узастопна ванмембранска или мембранска стања. Посматрајмо континуални низ мембранских стања. Овај низ ћемо звати *хеликс*. Посматрани хеликс низа стања до којег се дошло применом CMM-а на конкретан низ излаза кажемо да је *коректан* ако се поклапа са стварним низом стања, који се налази у бази знања, на бар 5 позиција. У контексту једног хеликса, може доћи до одређених одступања од жељеног низа. Хеликс који није

коректан, а поклапа се са хеликсом низа стања базе знање на бар једној позицији се назива *shifted*. Ово је прва од могућих грешака које модел CMM и његови параметри и методе могу направити. Такође, ако на месту на којој хеликс не постоји можемо оучити хеликс у предвиђеном низу, можемо рећи да смо уочили један *over predicted* хеликс. Слично, у секвенцама у којима нема хеликса, а у низу стања из базе знања постоји, кажемо да је пронађен *under predicted* хеликс. Перформансе које се такође одређују као могуће грешке су *falsely split* (пример када је требало на одређеном месту препознати један, али је препознато више од једног хеликса) и *falsely merged* (пример налажења једног хеликса, мада у низу за тренинг се примећује да на тим позицијама постоји више таквих).

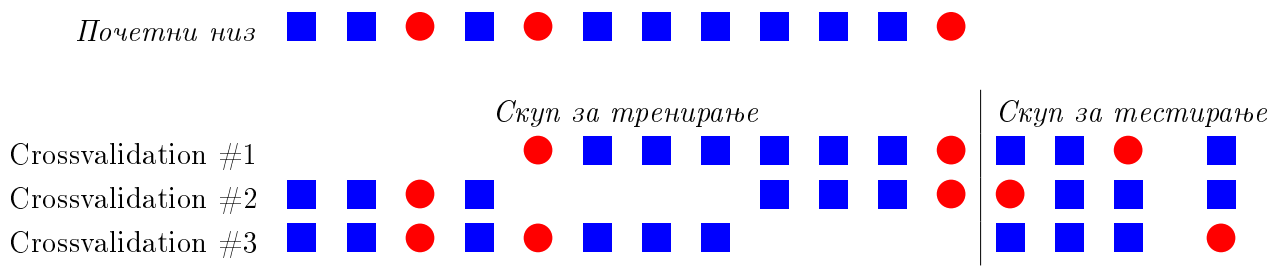
Посматрајући саме протеине, број грешака и број коректних стања хеликса се одређује сумирањем свих горепоменутих грешака односно коректности у односу на све хеликсе. Перформансе које се у овом случају посматране су следеће: *correct* (представља број протеина чији су сви хеликси коректни), *overpredicted* (представља број протеина чији су сви хеликси коректни, али имају бар једну overpredicted претпоставку за неки хеликс), да су *underpredicted* (представља број коректних протеина са бар једним underpredicted хеликсом), *over&under predicted* (број протеина који у себи садрже обе врсте хеликса), *invertedly* (број протеина који су коректни, али су супротно оријентисани (односно делови који се налазе у ћелији иначе моделовани су да буду ван ње и обрнуто); приметимо да због конструкције самог CMM-а је ово могуће, а да се положаји мембранских стања не промене).

4.4 Експериментална поставка

Уз помоћ базе знања извршена је процена ефикасности имплементираних алгоритама.

Концепт алгоритма *crossvalidation* се састоји у модификовању параметара $\boldsymbol{\Theta}$-а користећи одређени скуп низова аминокиселина, а затим одређивању највероватнијих низова стања који су произвели низове излаза из другог скупа аминокиселина, уз упоређивање истих са низовима из базе знања.

Као што је већ напоменуто, скуп података се састојао од 160 протеина. Овај сет података је подељен у 10 група од по 16 протеина. Алгоритам је понављан 10 пута, с тим да је за тестирање бирана увек друга група протеина, док је остатак користио тренирању.



Слика 4.1: Илустрација алгоритма crossvalidation на датом примеру.

4.5 Резултати

Табеларни приказ резултата мерених перформанси за засебне хеликсе:

Врста мерења	Број измерених	Вероватноћа догађаја
Број хеликса	696	
Shifted	0	
Over-predicted	21	3.02%
Under-predicted	28	4.02%
Falsely merged	1	0.14%
Falsely split	0	

Табеларни приказ резултата мерених перформанси за протеине:

Врста мерења	Број измерених	Вероватноћа догађаја
Број протеина	160	
Correct-predicted	117	73.12%
Invertedly predicted	5	3.12%
Over-predicted	18	11.25%
Under-predicted	23	14.37%
Over&under-predicted	3	1.87%

У алгоритму k-best Viterbi вредност параметра k је мењања, а вредности приказане у табелама су продукт рада агоритма за $k = 10$.

Алгоритам је у потпуности препознао све хеликсе коректно у 117 од 160 протеина, што је компатабилно резултатима представљеним у раду [5], где су такође примењене и додатне оптимизације поврх алгоритама описаних у раду. Самим тим, пројекат се може сматрати успешно имплементираним.

Глава 5

Закључак

Сусревши се први пут са, до тада мени страним, појмом скривених Марковљевих модела и упознавши се са начином размишљања које је заступљено у појму машинског учења са којим раније нисам имала додирних тачака, заинтересовала сам се за примену истих и дубљу анализу рада алгоритама којима се служе. Разумевање топологије протеина сматрам јако битном ствари, па отуда и мотивација за одабир управо овог примера за рад са скривним Марковљевим моделима. Циљ пројекта је првенствено била имплементација алгоритама коришћених у препознавању топологије трансмембранских протеина, али као врло битан део рада сматрам анализу архитектуре модела протеина кроз скривене Марковљеве моделе и разумевање математичког поткрепљења свих алгоритама.

Пројекат сматрам успешним с обзиром да су све потребне, а у овом раду описане, функције исправно имплементиране. Користећи познату базу знања извршен је тренинг и одређени су параметри неопходни за даља предвиђања топологије трансмембранских протеина.

Машинско учење и скривени Марковљеви модели као такви пружају у неку руку једноставно разумевање математичког модела који има широку примену при имплементацији кроз одређене алгоритме. С тим у вези, сматрам да је за сваког програмера корисно да зна основне нацрте ових концепата.

На самом крају бих желела да изразим захвалност:

- **Петру Величковићу** - ментору овог рада, студенту докторских студија универзитета у Кембрицу, за безгранично стрпљење и разумевање, свеопшту присутност и преданост успеху овог рада, несебично одвајање времена и неизмерну подршку у сваком смислу, непрестано мотивисање у вези са пројектом, али и унапређивањем свих аспеката знања информатике уопште;

- **Свим професорима** - из школе, али и ван ње, који су својим дугогодишњим радом помогли у мом школовању и стицању знања из свих области на било који начин;

- **Свима осталима** - који нису наведени, а допринели су стварању овог рада и унапређивању мог информатичког знања уопште.

Литература

[1] BAUM, L. E., & PETRIE, T. (1966) Statistical inference for probabilistic functions of finite state Markov chains. The annals of mathematical statistics, 1554-1563.

[2] BAUM, L. E., PETRIE, T., SOULES, G., & WEISS, N. (1970) A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. The annals of mathematical statistics, 164-171.

[3] RABINER, L. R. (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE, 77(2), 257-286.

[4] Andrew J Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. Information Theory, IEEE Transactions on, 13(2):260– 269, 1967.

[5] A. Krogh, B. Larsson, G. von Heijne, and E. L. L. Sonnhammer. Predicting transmembrane protein topology with a hidden Markov model: Application to complete genomes. Journal of Molecular Biology, 305(3):567-580, January 2001.

[6] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter. Molecular Biology of the cell, fifth edition, 125-192, 2008.

[7] David L. Nelson, Michael M Cox. Principles pd biochemistry, fifth edition, 153-182.

[8] Gunnar von Heijne. Membrane-protein topology, 2006.

[9] Machine learning
https://en.wikipedia.org/wiki/Machine_learning#Theory
Последњи приступ сајту: 27.05.2016.

[10] Hidden Markov model
https://en.wikipedia.org/wiki/Hidden_Markov_model
Последњи приступ сајту: 27.05.2016.

[11] Bioinformatics
https://en.wikipedia.org/wiki/Bioinformatics
Последњи приступ сајту: 27.05.2016.

[12] Amino acid
https://en.wikipedia.org/wiki/Amino_acid
Последњи приступ сајту: 27.05.2016.