Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

Zarovnávanie sekvencií s použitím metód klasifikácie

Diplomová práca

2014 **Bc. Michal Hozza**

Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

Zarovnávanie sekvencií s použitím metód klasifikácie

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra Informatiky
Školiteľ:	Mgr. Tomáš Vinař, PhD.
Konzultant:	Mgr. Michal Nánási

Bratislava, 2014 **Bc. Michal Hozza**

Poďakovanie...

Bc. Michal Hozza

Abstrakt

Zarovnávanie dvoch DNA sekvencií je jedným zo základných bioinformatických problémov. Obvykle takéto zarovnanie hľadáme pomocou jednoduchých párových skrytých Markovovských modelov (pHMM). V tejto práci sa zaoberáme možnosťami použitia prídavnej informácie o funkcii vstupných sekvencií na zlepšenie kvality takýchto zarovnaní. Informácie sme zakomponovali pomocou klasifikátorov, ktoré rozhodujú či dané pozície majú byť zarovnané k sebe alebo nie. Ako klasifikátor sme použili Random forest.

Ukázalo sa, že klasifikátor sa dokáže naučiť, ktoré okná majú byť zarovnané k sebe a ktoré nie.

Vyvinuli sme 2 modely pre zarovnanie sekvencií s anotáciami za pomoci klasifikátora, ktoré sú založené na párových skrytých Markovovských modeloch.

Model s klasifikátorom ako emisiou, kde sme nahradili emisné tabuľky stavov výstupom z klasifikátora.

Model s klasifikátorovou páskou, kde navyše modelujeme aj výstup z klasifikátora.

Kľúčové slová: zarovnávanie sekvencií, strojové učenie, Random forest, anotácie

Abstract

English abstract

Key words: ...

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Najnovšie technológie sekvenovania DNA produkujú stále väčšie množstvo sekvencií rôznych organizmov. Spolu s tým stúpa aj potreba rozumieť týmto dátam. Dôležitým krokom k ich porozumeniu je *zarovnávanie sekvencií*. Zarovnávanie dvoch DNA sekvencií je teda jedným zo základných bioinformatických problémov. Správne zarovnanie identifikuje časti sekvencie, ktoré vznikli z toho istého predka (zarovnané bázy), ako aj inzercie a delécie v priebehu evolúcie (medzery v zarovnaní). Je nápomocné pri zisťovaní ich štruktúry a následne funkcie jednotlivých častí.

Existujú rôzne algoritmy na zarovnávanie sekvencií. Väčšina z nich je založená na pravdepodobnostnom modeli, pričom sa snažia nájsť zarovnanie s čo najväčšou pravdepodobnosťou. Algoritmy sú zvyčajne založené na dynamickom programovaní a pracujú v kvadratickom čase v závislosti od dĺžok sekvencií. Niekedy sa na urýchlenie použijú rôzne heuristické algoritmy, ktoré nie vždy nájdu najpravdepodobnejšie zarovnanie, ale pracujú oveľa rýchlejšie.

My sme sa v práci zaoberali algoritmom, ktorý hľadá zarovnanie pomocou jednoduchých párových skrytých Markovovských modelov (pHMM) [DEKM98], kde kvalita výsledného zarovnania je ovplyvnená len pravdepodobnostným modelom.

Základný model berie do úvahy len jednotlivé *bázy* a pravdepodobnosti *substitúcie (mutácie)*, *inzercie* a *delécie*. Náš model navyše uvažuje aj prídavné informácie (takzvané anotácie) získané z externých programov (napr. anotácie o génoch z vyhľadávača génov).

Keďže množstvo dodatočnej informácie môže byť veľmi veľké – napríklad pre 3 binárne anotácie by sme mali $2^3 \times 2^3 = 64$ krát väčší počet parametrov – je ťažké skonštruovať vhodnú skórovaciu maticu pre zarovnávací algoritmus. Namiesto nej sme teda použili klasifikátory[1], ktoré sme trénovali na sekvenciách so známym zarovna-

[1]program, ktorý na základe vstupnej informácie a vopred natrénovaných parametrov klasifikuje dáta

ním a potom použili na zarovnanie nových sekvencií. Naše klasifikátory vracajú čísla z intervalu $\langle 0, 1 \rangle$, ktoré určujú, či dané dve bázy majú byť zarovnané spolu.

Ako klasifikátor sme použili *náhodný les* [Bre01], pretože aktuálne patrí medzi najlepšie klasifikátory.

ToDo: napisat este strucny obsah prace

do niektorej triedy z danej množiny tried

1 Zarovnávanie sekvencií

V tejto kapitole si stručne popíšeme čo je to globálne a lokálne zarovnanie a ukážeme základné algoritmy na hľadanie globálneho a lokálneho zarovnania. Tieto algoritmy budeme neskôr modifikované používať pri našom riešení.

1.1 Podobnosť sekvencií, sekvenčná homológia a zarovnanie

V prírode vznikajú evolúciou nové sekvencie modifikáciou už existujúcich. Preto môžme často spozorovať podobnosť medzi neznámou sekvenciou a sekvenciou o ktorej už niečo vieme. Ak zistíme podobnosti medzi sekvenciami, môžeme preniesť informácie o štruktúre a/alebo funkcii na novú sekvenciu.

Podobné sekvencie, ktoré sa vyvinuli mutáciami zo sekvencie v spoločnom predkovi sa nazývajú *homologické* a pod pojmom *hľadanie homológov* rozumieme hľadanie takých podobností, ktoré s veľkou pravdepodobnosťou vznikli práve spoločnou evolučnou históriou.

Počas evolúcie dvoch homologických sekvencií nastane veľa *inzercií*, *delécií* a *substitúcií*, preto predtým ako môžeme začať porovnávať sekvencie, ich musíme zarovnať tak, aby homologické časti sekvencií boli na rovnakom mieste v zarovnaní. [DEKM98, BV11]

1.2 Párové zarovnávanie

Párové zarovnávanie je základná úloha zarovnávania sekvencií, kde sa k sebe zarovnávajú dve sekvencie. V tejto práci sa budeme zaoberať len párovým zarovnávaním.

Kľúčové problémy sú:

Sekvencia 1:

gagacccgcctaggtgaatatttagcagc
gattaaataccacgtaTATAAGGTGGACC
GTTCCTCGAGAGGTTCTTCCGGCAATGAC
GGCCAGAGCAAAAGCCACGTgtaggactg
catacgcctctacgcctccactgacgcga
tgatgtggcgtggatctgtttgctcttgg
tataggtcacggagacggctggtactgat
cccttcgggagtaaaaatataatgaccat
ggcccaggcttcaggagggagttgtgcgg

Sekvencia 2:

tgtacagcactgcaacgagcatctggggg
ttggttattccgatggcgctggacagcta
gcggacagtagttctcaggccttagtaga
aaggtgggaaccccctATGAGGTCGACCG
TTTCAGCGTGACTATAGACGTCATTGAAG
CAATATACAGGAACACCACCTacttagga
agggagttcggtgcagtaaagcattctta
cctcagggcacggtagagaacactacaac
cagaatagcaacgtgatgcggcgactctc

Lokálne zarovnanie:

```
TATAAGGTGGACCGTT--------CCTCGAGAGGTTCTTCCGGCAATGGCCACGAGAGCAAAAGCCACGT
TATGAGGTCGACCGTTTCAGCGTGACTATAGACGTCATTGAAGCAATATACAGG------AACACCACCT
```

Obr. 1.1: Dve sekvencie a ich lokálne zarovnanie. Veľkými písmenami a červenou farbou sú vyznačené zarovnané časti. V zarovnaní sa nachádzajú zhody, nezhody a medzery v oboch sekvenciách

1. Aké typy zarovnávania by sme mali uvažovať
2. Skórovací systém, ktorý použijeme na ohodnotenie zarovnania a trénovanie
3. Algoritmus, ktorý použijeme na hľadanie optimálneho alebo dobrého zarovnania podľa skórovacieho systému
4. Štatistická významnosť zarovnania.

[DEKM98]

1.3 Typy zarovnaní

Základné typy zarovnaní sú *Globálne zarovnanie* a *Lokálne zarovnanie*.

Definícia 1.3.1 (Globálne zarovnanie)**.** Vstupom sú dve sekvencie $X = x_1x_2 \dots x_n$ a $Y = y_1y_2 \dots y_m$ Výstupom je zarovnanie celých sekvencií X a Y.

Definícia 1.3.2 (Lokálne zarovnanie)**.** Vstupom sú dve sekvencie $X = x_1x_2 \dots x_n$ a $Y = y_1y_2 \dots y_m$ Výstupom je zarovnanie nejakých podreťazcov $x_i \dots x_j$ a $y_k \dots y_l$ sekvencií.

V praxi sa väčšinou snažíme nájsť zarovnania s najvyšším skóre. [BV11]

1.4 Skórovacie systémy

Takmer všetky metódy zarovnania hľadajú zarovnanie dvoch reťazcov na základe nejakej *skórovacej schémy*[1]. Skórovacie schémy môžu byť veľmi jednoduché, napr. $+1$ za *zhodu* a -1 za *nezhodu*. Hoci ak chceme mať schému, kde biologicky najkorektnejšie zarovnanie má najvyššie skóre, musíme vziať do úvahy, že biologické sekvencie majú evolučnú históriu, 3D štruktúru a mnohé ďalšie vlastnosti obmedzujúce ich evolučné procesy. Preto skórovací systém vyžaduje starostlivé premyslenie a môže byť veľmi zložitý. [DEKM98]

[1] metóda, ktorou priradíme zarovnaniu skóre - zvyčajne čím väčšie skóre, tým realistickejšie zarovnanie by to malo byť

1.4.1 Skórovacie matice

Skoro vždy však chceme rôzne zhody a nezhody skórovať rôzne - nie len všetky zhody $+1$ a nezhody -1. Skóre môže záviseť od toho aké bázy sú v danom stĺpci zarovnania. Na to sa používa *skórovacia matica* (obr. 1.2), kde máme definované skóre pre každú dvojicu. Skórovacie matice sa využívajú najmä pri zarovnávaní proteínov, kde niektoré dvojice majú podobné chemické vlastnosti. [DEKM98, BV11]

Iným typom skórovacej schémy je napríklad *párový skrytý markvovský model* (viac v 1.6)

	A	C	G	T	-
A	6	-1	-2	-1	-3
C	-1	5	-3	-2	-4
G	-2	-3	5	-2	-2
T	-1	-2	-2	6	-1
-	-3	-4	-2	-1	∞

Obr. 1.2: Ukážka skórovacej matice. Všimnime si, že môžu byť rôzne skóre aj za jednotlivé zhody, nezhody alebo medzery.

1.5 Algoritmy na hľadanie zarovnaní

Pre danú skórovaciu schému potrebujeme algoritmus, ktorý nájde optimálne zarovnanie dvoch sekvencií. Budeme uvažovať zarovnávanie s medzerami. Medzery používame na znázornenie delécie v danej sekvencii, alebo inzercie v druhej sekvencii. Na označenie medzier používame pomlčku `'-'` a medzeru do sekvencie medzi 2 bázy pridáme tak, že medzi 2 bázy napíšeme jednu alebo viac pomlčiek - počet pomlčiek zodpovedá počtu medzier a teda aj počtu báz, ktoré má na danom mieste jedna sekvencia navyše oproti druhej. Do sekvencie môžeme pridať ľubovoľne veľa medzier, aby sme dosiahli lepšie skóre. Pre 2 sekvencie dĺžky n existuje

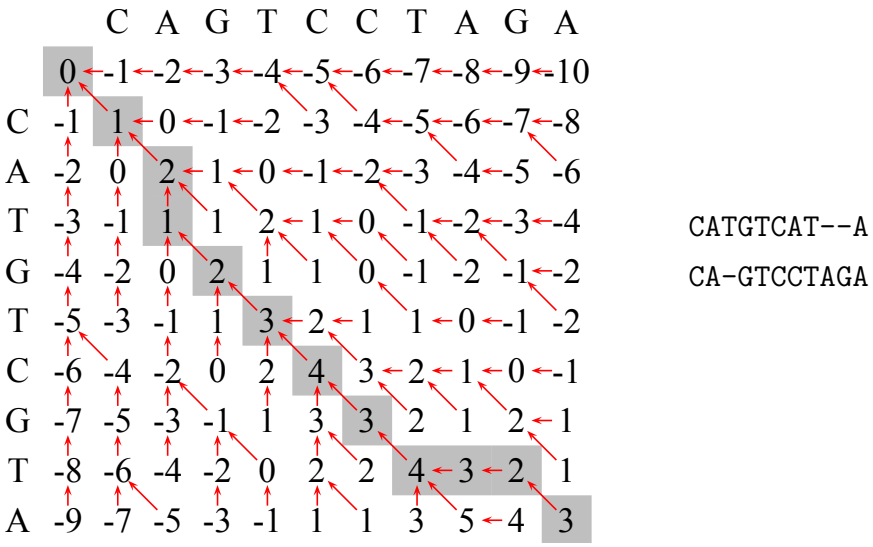
$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \simeq \frac{2^{2n}}{\sqrt{\pi n}}$$

možných globálnych zarovnaní. [DEKM98] Čiže nie je možné v rozumnom čase nimi prejsť.

Algoritmy na hľadanie zarovnaní využívajú *dynamické programovanie*. Často sa používajú aj rôzne heuristiky na urýchlenie výpočtu. My sa budeme zaoberať len algoritmami využívajúcimi dynamické programovanie. Pre rôzne typy zarovnaní a skórovacie schémy máme rôzne algoritmy zarovnávania. [DEKM98, BV11]

1.5.1 Algoritmus pre globálne zarovnanie: Needelman-Wunch

Máme dané 2 sekvencie $X = x_1x_2 \dots x_n$ a $Y = y_1y_2 \dots y_m$, budeme zarovnávať všetky znaky sekvencie X a všetky znaky sekvencie Y. Definujeme si jednoduchú skórovaciu tabuľku kde $s(x, y)$ bude udávať skóre pre danú dvojicu báz (napr. $+1$ za zhodu, -1 za nezhodu) a za nezhodu budeme dávať penaltu $-d$.



Obr. 1.3: Tabuľka dynamického programovania pre globálne zarovnanie (vľavo) a výsledné zarovnanie (vpravo). Skóre je +1 za zhodu a -1 za nezhodu alebo medzeru.

Algoritmus postupne vypĺňa 2-rozmernú maticu A. Riadky zodpovedajú bázam sekvencie X a stĺpce bázam Y. Na políčku $A[i, j]$ bude skóre najlepšieho zarovnania prvých i báz sekvencie X a prvých j báz Y.

Keď zarovnávame sekvenciu s prázdnou sekvenciou, tak skóre bude $-n$, kde n je dĺžka

sekvencie. Bude tam n pomlčiek, každá nám dá skóre -1. Takto vyplníme riadky a stĺpce $A[i,0]$ a $A[0,j]$.

Ak chceme vyplniť políčko $A[i,j]$, musíme si uvedomiť ako môže vyzerať posledný stĺpec zarovnania $x_1x_2\ldots x_i$ a $y_1y_2\ldots y_j$. Máme iba 3 možnosti ako môže vyzerať posledný stĺpec najlepšieho zarovnania. Buď obsahuje x_i alebo y_j alebo oboje. V prípade, že posledný stĺpec obsahuje oboje, cena tohto stĺpca je $s(x_i,y_j)$. Ak by sme posledný stĺpec zmazali, dostali by sme zarovnanie $x_1x_2\ldots x_{i-1}$ a $y_1y_2\ldots y_{j-1}$, pričom musí ísť o najlepšie zarovnanie. To už máme vypočítané v políčku $A[i-1,j-1]$, čiže výsledné skóre bude $A[i-1,j-1]+s(x_i,y_j)$.

V prípade, že posledný stĺpec obsahuje len x_i zarovnané s pomlčkou, skóre stĺpca bude -1 a po zmazaní dostávame zarovnanie $x_1x_2\ldots x_{i-1}$ a $y_1y_2\ldots y_j$, výsledné skóre bude teda $A[i-1,j]-1$. V prípade, že posledný stĺpec obsahuje len y_i, tak skóre vypočítame analogicky.

Najlepšie skóre bude maximálne skóre pre všetky 3 prípady. Dostávame teda nasledujúci vzťah pre výpočet $A[i,j]$:

$$A[i,j]=\max\begin{cases}A[i-1,j-1]+s(x_i,y_j)\\A[i-1,j]-d\\A[i,j-1]-d\end{cases}$$

Maticu vieme vypĺňať po riadkoch, pričom každé políčok vieme vypočítať z troch políčok, ktoré už sú vypočítané. Políčko $A[0,0]=0$ a krajné políčka potom vieme vypočítať ako $A[i,0]=A[i-1,j]-d$ a $A[0,j]=A[i,j-1]-d$

Ak nás zaujíma aj zarovnanie – nie len jeho skóre – vieme si pre každé políčko zapamätať ktorá z troch možností dosiahla maximálnu hodnotu (červené šípky na Obr. 1.3). Na základe tejto informácie potom vieme zrekonštruovať zarovnanie tak, že postupne z posledného políčka ($A[n,m]$) budeme prechádzať na políčko, z ktorého sme vypočítali aktuálnu hodnotu.

Časová zložitosť je $O(nm)$, pretože vypĺňame nm políčok, každé v konštantnom čase. Zjavne aj pamäťová zložitosť je $O(nm)$.

Pamäťová zložitosť sa dá zredukovať na $O(n+m)$ za cenu zhruba dvojnásobného času výpočtu [Hir75].

1.5.2 Algoritmus pre lokálne zarovnanie: Smith-Waterman

		C	A	G	T	C	C	T	A	G	A
	0	0	0	0	0	0	0	0	0	0	0
A	0	0	1	0	0	0	0	0	1	0	1
T	0	0	0	0	1	0	0	1	0	0	0
G	0	0	0	1	0	0	0	0	0	1	0
T	0	0	0	0	2	1	0	1	0	0	0
C	0	1	0	0	1	3	2	1	0	0	0
T	0	0	0	0	1	2	2	3	2	1	0
A	0	0	1	0	0	1	1	2	4	3	2
T	0	0	0	0	1	0	0	2	3	3	2

```
GT-CTA
GTCCTA
```

Obr. 1.4: Tabuľka dynamického programovania pre lokálne zarovnanie (vľavo) a výsledné zarovnanie (vpravo). Skóre je +1 za zhodu a -1 za nezhodu alebo medzeru.

Algoritmus pre lokálne zarovnania sa líši len v niekoľkých malých detailoch. Opäť vypĺňame maticu A, s tým, že v $A[i, j]$ bude najvyššie skóre lokálneho zarovnania medzi sekvenciami $x_1x_2 \ldots x_i$ a $y_1y_2 \ldots y_j$, ktoré buď obsahuje bázy x_i aj y_j, alebo je prázdne. Teda na ľubovoľnom mieste uvažujeme aj prázdne zarovnanie so skóre 0 (v matici nebudú záporné čísla). Vzťah pre výpočet $A[i, j]$ vyzerá takto:

$$A[i,j] = \max \begin{cases} 0 \\ A[i-1, j-1] + s(x_i, y_j) \\ A[i-1, j] - d \\ A[i, j-1] - d \end{cases}$$

V tomto prípade sú všetky krajné políčka nulové.

Časová aj pamäťová zložitosť sú, rovnako ako pri globálnom zarovnaní $O(nm)$.

1.5.3 Afínne skórovanie medzier

V jednoduchom skórovaní sme dávali za pomlčku vždy rovnaké skóre (-1). Pri evolúcii sa však môže stať, že sa naraz zmaže niekoľko susedných báz. Pri *afínnom skórovaní*

medzier teda zavedieme dva typy skóre. Skóre za *začatie medzery* a skóre za *rozšírenie medzery.*

Algoritmus globálneho zarovnania vieme upraviť nasledovne: Namiesto matice A teraz budeme mať 3 matice M, I_x, I_y zodpovedajúce trom situáciám (Obr. 1.5).

ACTx$_i$	ACTTAx$_i$	ACTx$_i$--
AGTy$_j$	AGTy$_j$--	AGTATy$_j$
(a) Mutácia(M)	(b) Inzercia v X (I_x)	(c) Inzercia v Y (I_y)

Obr. 1.5: Tri situácie pri afínnoom skórovaní medzier

Nech $M[i,j]$ je najlepšie skóre prvých i báz zo sekvencie X a prvých j báz zo sekvencie Y, pričom x_i je zarovnané k y_j, $I_x[i,j]$ je najlepšie skóre ak x_i je zarovnané k medzere a $I_y[i,j]$ je najlepšie skóre ak y_j je zarovnané k medzere.

Označme si d penaltu za začatie medzery a e penaltu za rozšírenie medzery. Vzťahy pre výpočet políčok sú nasledovné:

$$M[i,j] = \max \begin{cases} M[i-1,j-1] + s(x_i, y_j) \\ I_x[i-1,j-1] + s(x_i, y_j) \\ I_y[i-1,j-1] + s(x_i, y_j) \end{cases}$$

$$A[i,j] = \max \begin{cases} M[i-1,j] - d \\ I_x[i-1,j] - e \end{cases}$$

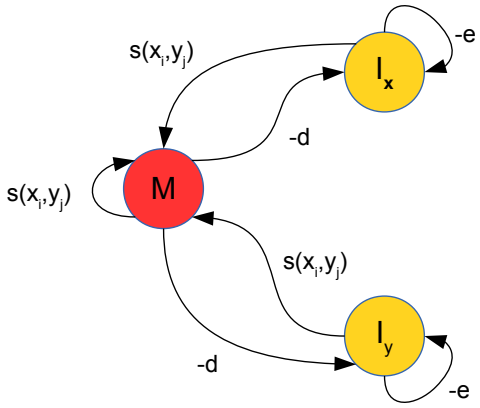
$$A[i,j] = \max \begin{cases} M[i,j-1] - d \\ I_y[i,j-1] - e \end{cases}$$

V týchto rovniciach predpokladáme, že delécia nie je nasledovaná inzerciou. Toto platí v optimálnej sekvencii, ak $-d - s$ je menšie ako najmenšie skóre nezhody.

Tieto vzťahy vieme popísať stavovým diagramom na obrázku 1.6:

Časová zložitosť je $O(nm)$, pretože vypĺňame $3nm$ políčok, každé v konštantnom čase. Pamäťová zložitosť je opäť $O(nm)$.

[DEKM98]



Obr. 1.6: Stavový diagram pre zarovnanie sekvencií - obsahuje *Match* (M), *InsertX* (I_x) a *InsertY* (I_y) stav a prechody medzi nimi spolu s ich cenou. Napríklad prechod z M do I_x znamená vloženie medzery do Y-ovej sekvencie a to penalizujeme $-d$

1.6 Zarovnávanie pomocou skrytých Markovovských modelov

1.6.1 Skryté markovovské modely (HMM)

Skrytý markovovský model (Hidden Markov Model, HMM) je pravdepodobnostný model, ktorý generuje náhodnú sekvenciu spolu s jej anotáciou (stavmi). HMM si môžme predstaviť ako konečný automat. Skladá sa z niekoľkých stavov, prechodov medzi nimi a emisií. Na rozdiel od bežných konečných automatov, HMM emitujú symboly v stave, nie počas prechodu. HMM sa skladá z 3 distribúcií

- distribúcia začiatočných stavov (HMM začne v stave i)
- distribúcia prechodov (HMM prejde zo stavu i do stavu j)
- distribúcia emisií (HMM v stave i vygeneruje symbol x)

Generovanie sekvencie teda vyzerá nasledovne: Na začiatku je HMM v niektorom stave (každý stav i má nejakú pravdepodobnosť π_i, že bude začiatočný). Potom v každom kroku HMM emituje symbol x s pravdepodobnosťou $e_{i,x}$ a prejde do stavu j s pravdepodobnosťou $a_{i,j}$. Po n krokoch takto vygenerujeme sekvenciu dĺžky n, pričom každý symbol je oanotovaný stavom, ktorý ho vygeneroval.

V takomto modeli vieme počítať pravdepodobnosť, že model vygeneruje sekvenciu x dĺžky n s anotáciou s ako súčin pravdepodobností prechodov a emisií. Výpočet vyzerá nasledovne:

$$P[X = x|S = s] = \pi_{s_1} e_{s_1,x1} a_{s_1,s_2} e_{s_2,x_2} a_{s_2,s_3} e_{s_3,x_3} \dots a_{s_{n-1},s_n} e_{s_n,x_n}$$

. [BV11, DEKM98]

1.6.2 Zarovnávanie pomocou párových skrytých markovovských modelov (pHMM)

Párové skryté Markovovské modely (pHMM) sa od tých obyčajných líšia v tom, že namiesto jednej sekvencie generujú dvojicu sekvencií. Na pHMM sa dajú použiť podobné algoritmy ako na HMM, ale treba ich upraviť tak, aby pracovali v dvoch rozmeroch.

V časti 1.5.3 sme si ukázali jednoduchý algoritmus na globálne zarovnávanie s afínnym skórovaním medzier. K tomuto algoritmu sme si uviedli aj jednoduchý stavový automat (Obr. 1.6). Tento automat vieme previesť na pHMM.

Na to aby sme automat previedli na pHMM, musíme urobiť niekoľko zmien – musíme nastaviť emisné a prechodové pravdepodobnosti, tak aby sčítavali do jedna. Pre jednoduchosť pridáme aj prechody medzi stavmi X a Y. Ak ich pravdepodobnosti nastavíme na nula, máme model ekvivalentný predchádzajúcemu. [DEKM98]

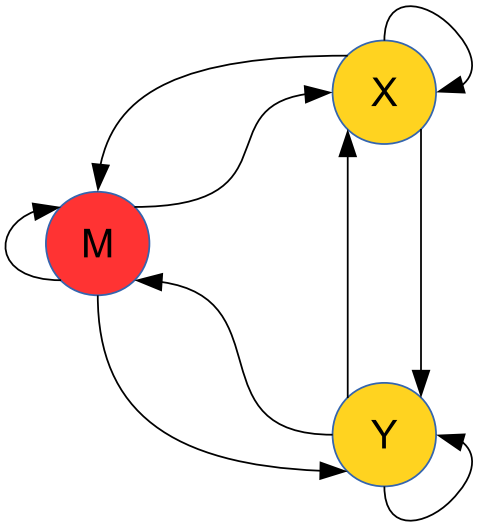
1.6.3 Viterbiho algoritmus na pHMM

Hľadáme najpravdepodobnejšiu postupnosť stavov A ak máme dané sekvencie X a Y, teda $\arg\max_A \Pr(A, X, Y)$. Úlohu budeme riešiť dynamickým programovaním.

Podproblém $V[i, j, u]$ je pravdepodobnosť najpravdepodobnejšej postupnosti stavov končiacej v $X[i]$ a $Y[j]$ v stave u.

Inicializácia:

$$V[0, 0, M] = 1 \wedge V[i, 0, *] = 0, V[0, j, *] = 0 \quad \forall i, j \tag{1.1}$$



Obr. 1.7: Párový HMM pre zarovnávanie sekvencií

Rekurentné vzťahy:

$$V[i,j,M] = e_{M,(x_i,y_j)} \max \begin{cases} a_{M,M} V[i-1,j-1,M] \\ a_{I_x,M} V[i-1,j-1,I_x] \\ a_{I_y,M} V[i-1,j-1,I_y] \end{cases} \tag{1.2}$$

$$V[i,j,I_x] = e_{I_x,x_i} \max \begin{cases} a_{M,I_x} V[i-1,j,M] \\ a_{I_x,I_x} V[i-1,j,I_x] \\ a_{I_y,I_x} V[i-1,j,I_y] \end{cases} \tag{1.3}$$

$$V[i,j,I_y] = e_{I_y,y_j} \max \begin{cases} a_{M,I_y} V[i,j-1,M] \\ a_{I_y,I_y} V[i,j-1,I_y] \\ a_{I_x,I_y} V[i,j-1,I_x] \end{cases} \tag{1.4}$$

Algoritmus funguje takto: Nech n a m sú dĺžky sekvencií X a Y. Na začiatku nainicializuje hodnoty pre $V[0,*,*]$ a $V[*,0,*]$ podľa 1.1. Potom postupne pre všetky $i = 1 \ldots n$, $j = 1 \ldots m$ a $u \in \{M, I_x, I_y\}$ dynamicky vypočíta $V[i,j,u]$ podľa 1.2, 1.3 a 1.4.

Maximálne $V[n,m,u] \forall u \in \{M, I_x, I_y\}$ je pravdepodobnosť najpravdepodobnejšej postupnosti stavov. Aby sme stavy vedeli zrekonštruovať, pamätáme si pre každé $V[i,j,u]$ stav w, ktorý viedol k maximálnej hodnote. Táto postupnosť stavov nám udáva najpravdepodobnejšie zarovnanie.

Časová zložitosť tohto algoritmu je $O(nm)$.

Poznámka: pre dlhé sekvencie budú čísla $V[i, j, u]$ veľmi malé a môže dôjsť k podtečeniu. V praxi teda používame zlogaritmované hodnoty a namiesto násobenia súčet. [DEKM98]

1.6.4 Nastavenie parametrov pHMM

Ak máme oanotované trénovacie sekvencie, môžme z nich parametre odvodiť frekvenčnou analýzou. Emisie získame tak, že vyfiltrujeme symboly s príslušným stavom a spočítame frekvencie pre každý stav zvlášť, pričom v Match stave počítame frekvencie dvojíc symbolov. Tranzície získame tak, že pre každý stav spočítame frekvencie nasledujúcich stavov. Tento postup sa volá *metóda maximálnej vierohodnosti (v angličtine maximum likelihood estimation)*. [DEKM98, Wik14b]

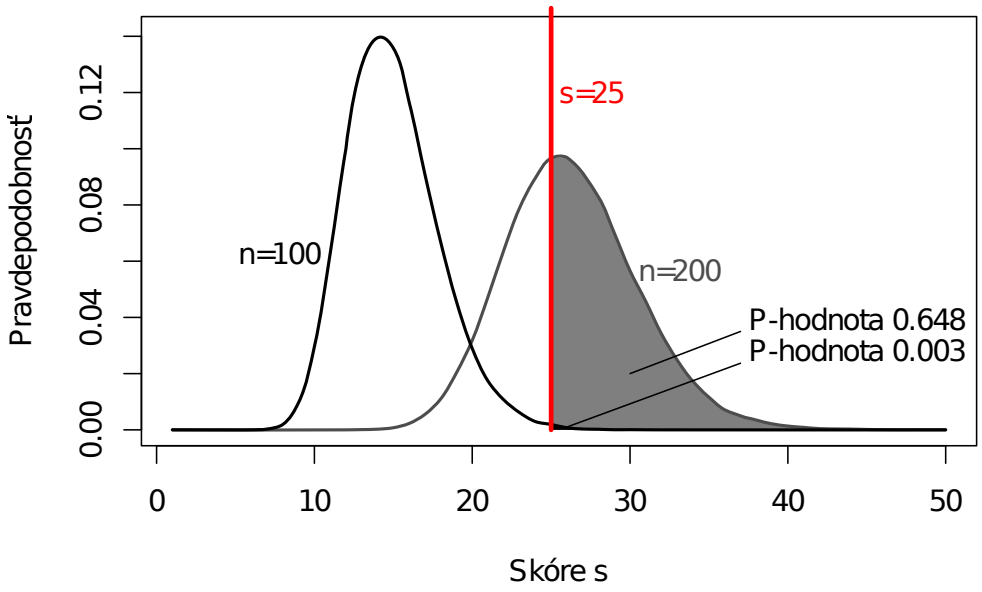
Parametre modelu môžme teda ľahko natrénovať z existujúcich párových zarovnaní.

1.7 Štatistická významnosť zarovnania

Smith-Watermanov algoritmus nájde najlepšie lokálne zarovnanie pre ľubovoľné dve sekvencie. Treba však rozhodnúť, či je zarovnanie dostatočne vierohodné nato, aby predstavovalo skutočnú podobnosť sekvencií a nie len najlepšie zarovnanie dvoch nesúvisiacich sekvencií. Ako vodítko pri rozhodnutí sa používajú identifikátory *štatistickej významnosti* zarovnania: *P-hodnota (P-value)* alebo *E-hodnota (E-value)*.

P-hodnota zarovnania je pravdepodobnosť, že medzi náhodne generovanými sekvenciami tej istej dĺžky by sme našli zarovnanie s rovnakým skóre alebo vyšším. Keďže P-hodnota závisí od dĺžok sekvencií a skóre, musíme ju počítať pri každom zarovnaní. Je však časovo náročné robiť to generovaním veľkého množstva zarovnaní, preto sa používajú matematicky odvodené vzorce na odhad tejto hodnoty. ([KA90], [MB06]).

E-hodnota vyjadruje strednú hodnotu počtu zarovnaní so skóre aspoň takým ako má naše zarovnanie medzi náhodne generovanými sekvenciami. E-hodnota teda môže byť aj väčšia ako jedna. Ak je E-hodnota väčšia ako jedna, tak čisto náhodou by sme očakávali aspoň jedno také silné zarovnanie, a teda zarovnania s takouto (a nižšou) E-hodnotou nebudeme považovať za štatisticky významné.



Obr. 1.8: P-hodnota lokálneho zarovnania so skóre $s = 25$ medzi 2 sekvenciami dĺžky $n = 100$ alebo $n = 200$ (skórovanie $+1$ zhoda, -1 nezhoda alebo medzera). Rozdelenie bolo získané zarovnávaním 100000 párov náhodných sekvencií. Pri $n = 100$ je P-hodnota približne 0.003 a zodpovedá malej čiernej ploche pod krivkou napravo od zvislej čiary pre $s = 25$. Pri dlhších sekvenciách zodpovedá P-hodnota veľkej sivej ploche napravo od zvislej čiary. Pri takto dlhých sekvenciách očakávame skóre 25 alebo väčšie vo viac ako 60% prípadov čisto náhodou. Nejde teda o štatisticky významné zarovnanie.

V štatistike sa v rôznych testoch štandardne používajú prahy na P-hodnotu 0.05 alebo 0.01. Pri zarovnávaní sekvencií však často používame ešte nižší prah, teda uvažujeme len zarovnania s P-hodnotou menšou ako napr. 10^{-5}. Pri malých hodnotách sú P-hodnota a E-hodnota približne rovnaké, teda taký istý prah môžme použiť aj na E-hodnotu.

2 Súvisiaca práca

V tejto kapitole si uvedieme stručný prehľad modelov, ktoré zahŕňajú doplnkové informácie do zarovnania pomocou metód klasifikácie a stručne uvedieme v čom sa bude náš model líšiť.

V princípe môžme rozlišovať dva typy modelov - *generatívny model* a *diskriminačný model*.

Konvenčné techniky odhadu pre zarovnávania sa zakladajú na generatívnom modeli. Generatívny model (napr. HMM) sa snaží modelovať proces, ktorý generuje dáta ako pravdepodobnosť $P(X, Y, Z)$, kde $X = x_1x_2 \dots x_n$, $Y = y_1y_2 \dots y_m$ a Z je zarovnanie. Ak poznáme $P(X, Y, Z)$ (alebo jej dobrý odhad),

$$\arg\max_z P(X = x, Y = y, Z = z)$$

predikuje zarovnanie z z dvoch sekvencií x a y. Aby sme zľahčili odhad pravdepodobnosti$P(X, Y, Z)$, rozložíme ju pomocou nezávislých predpokladov na procese, ktorý generuje x a y. To síce vedie k efektívnym a jednoduchým problémom odhadu, ale obmedzuje to interakcie v rámci sekvencií, ktoré by sme mohli modelovať. [YJEP07]

Výskum v oblasti strojového učenia dokázal, že diskriminačné učenie (SVM, RandomForest) zvyčajne produkuje oveľa presnejšie pravidlá ako generatívne učenie (HMM, naive Bayes classifier). [YJEP07] Môže to byť vysvetlené tým, že $P(Z|X, Y)$, je už vhodné na vyhodnotenie optimálnej predikcie

$$\arg\max_z P(Z = z|X = x, Y = y).$$

[YJEP07]
Diskriminačné učenie aplikované na problém zarovnania bude priamo odhadovať $P(Z|X, Y)$ alebo prislúchajúcu diskriminačnú funkciu, a preto sa zamerá na podstatnú časť problému odhadu. [YJEP07]

Aktuálne existuje len niekoľko prístupov k diskriminačnému učeniu modelov zarovnávaní. Jeden z možných prístupov je riešiť *problém inverzného zarovnania* pomocou strojového učenia. [YJEP07]

Definícia 2.0.1 (Inverzné zarovnanie)**.** Máme dané sekvencie a k nim zarovnanie. Inverzné zarovnanie nám vráti váhový model, s ktorým daný algoritmus na zarovnávanie vráti požadované zarovnanie k daným sekvenciám.

Problém inverzného zarovnania bol prvý krát formulovaný v [GS96]. Na tomto probléme je postavený aj model v [YJEP07], kde sa na trénovanie Support Vector Machine (SVM) dá pozerať ako na riešenie tohto problému. V článku sa zaoberajú použitím *Structural SVM* algoritmu na zarovnávanie proteínových sekvencií. Diskriminačné učenie umožňuje zahrnutie množstva dodatočnej informácie – státisíce parametrov. Navyše SVM umožňuje trénovanie pomocou rôznych účelových funkcií (loss functions). SVM algoritmus má lepšiu úspešnosť ako generatívna metóda SSALN, ktorá je veľmi presným generatívnym modelom zarovnaní zahŕňajúcim informáciu o štruktúre.

Podobný prístup je aj v CONTRAlign [DGB06], kde sa používajú Conditional Random Fields (CRF). Tento prístup tiež ťaží z benefitov diskriminačného učenia, avšak na rozdiel od [YJEP07] neumožňuje použitie účelových funkcií.

3 Klasifikácia na základe lokálnej informácie

Cieľom našej práce bolo zahrnúť dodatočnú informáciu do zarovnávania sekvencií, ktorá je poskytnutá formou anotácií k príslušným bázam. Keďže náš zarovnávač je založený na pHMM (sekcia 1.6.2), potrebujeme nejako určiť emisné pravdepodobnosti v jednotlivých stavoch. Keďže chceme okrem jednotlivých báz zahrnúť aj iné informácie, rozhodli sme sa na tento účel využiť klasifikátory. Klasifikátory určujú, ktoré pozície sa majú zarovnať k sebe a túto informáciu zakomponovávame do výsledného zarovnávača pomocou modelov, ktoré si popíšeme v kapitole 4.

V tejto kapitole sa budeme venovať výberu klasifikátora a jeho vstupných atribútov. Najskôr si stručne popíšeme algoritmus klasifikátora, potom typy vstupných atribútov, ktoré sme navrhli a nakoniec sa budeme venovať vlastnostiam klasifikátora s danými typmi atribútov a výberu finálneho klasifikátora pre použitie v našich modeloch.

3.1 Náhodné lesy

Ako klasifikátor sme si vybrali *náhodný les (angl. Random forest)*, pretože patrí v súčasnosti medzi najlepšie klasifikátory. V tejto sekcii si popíšeme základný algoritmus a v čom spočívajú jeho výhody a jeho najdôležitejšie vlastnosti.

Dôležité vlastnosti náhodných lesov

Klasifikátor náhodný les má niekoľko dôležitých vlastností, kvôli ktorým sme si ho vybrali:

- má vysokú presnosť (podľa [BC] je najpresnejší spomedzi všetkých vtedajších klasifikačných algoritmov)

- je efektívny aj na veľkých dátach
- dokáže obsiahnuť tisícky vstupných premenných
- natrénovaný náhoný les môžme uložiť a neskôr použiť na ďalšie dáta
- dáva odhad, ktoré premenné sú dôležité na klasifikáciu.

Navyše má ešte niekoľko ďalších užitočných vlastností, ktoré však pre nás neboli rozhodujúce:

- dokáže sa vysporiadať aj s väčším množstvom chýbajúcich dát
- produkuje interný nevychýlený odhad chyby aj počas trénovania lesa
- dokáže počítať vzťahy medzi atribútmi a klasifikáciou
- počítanie vzdialenosti môže byť použité aj na neoanotované[1] dáta, čo sa dá použiť na klastrovanie a detekciu outlierov.
- Ponúka experimentálnu metódu na detekciu interakcie medzi premennými.

3.1.1 Klasifikačné stromy

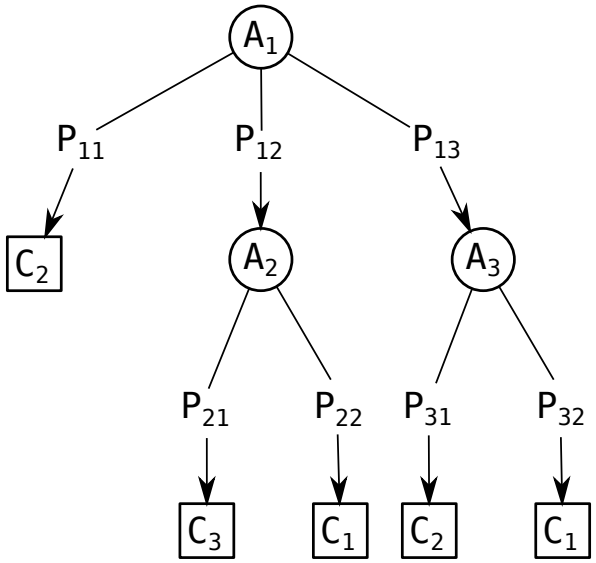
Keďže náhodný les je vlastne les *klasifikačných stromov* [2], je nevyhnutné, aby sme si najskôr povedali niečo o nich.

Klasifikačný strom je strom vybudovaný na základe trénovacej množiny, ktorý na základe vstupných údajov (vstupného vektora) predpovedá hodnotu výstupnej premennej. Klasifikačný strom sa skladá z *uzlov* a *listov*. V uzle sa nachádza rozdeľovacie kritérium, podľa ktorého sa vieme rozhodnúť do ktorej vetvy máme pokračovať. V listoch sa nachádzajú triedy, do ktorých sa vstupný vektor klasifikuje.

Klasifikácia vstupného vektora vyzerá nasledovne: prechádzame strom zhora nadol a postupne sa zaraďujeme do vetiev podľa rozdeľovacích kritérií. Podľa toho, v ktorom liste skončíme, sa určí výstupná hodnota.

[1] bez informácie o triede

[2] niekedy označované ako rozhodovacie stromy



Obr. 3.1: Klasifikačný strom. A sú atribúty, P sú konkrétne hodnoty atribútov a C sú triedy, do ktorých klasifikujeme dáta.

Trénovanie

Na trénovanie rozdovacích stromov sa používajú rôzne algoritmy. My si spomenieme najjednoduchší z nich – algoritmus ID3 [Wik14c]. Budeme rozlišovať dva typy listov: uzavretý a neuzavretý list. Uzavreté listy majú už priradenú triedu. Neuzavreté listy ešte nemajú priradenú triedu. Ku každému neuzavretému list prislúcha nejaká sada príkladov.

Algoritmus začína so stromom, ktorý má práve jeden vrchol – neuzavretý list. Kým existuje nejaký neuzavretý list l opakuje:

- Ak sú v jednom liste všetky príklady jednej triedy, označí list touto triedou a získame uzavretý list.
- Inak vyberie najinformatívnejší atribút A_l a vrcholu l pridá deti, rozdelené podľa hodnôt atribútu A_l.

Najinformatívnejší atribút vyberáme pomocou miery *zisk informácie (angl. information gain)*. Zisk informácie je založený na rozdiele entrópie pred a po rozdelení podľa daného atribútu. Nech C je množina tried a p_c je pravdepodobnosť, že náhodný prvok

z S patrí do C, potom entropiu množiny S počítame ako:

$$E(S) = -\sum_{c \in C} p_c \log p_c$$

Zisk informácie potom vypočítame takto:

$$IG(S, A) = E(S) - \sum_{v \in A} \frac{\left|S_{\{A=v\}}\right|}{|S|} E\left(S_{\{A=v\}}\right)$$

3.1.2 Náhodný les

Klasifikácia

Náhodný les sa skladá z mnohých klasifikačných stromov, ktoré tvoria komisiu. Pri klasifikácii nejakého vstupného vektora sa predloží vstupný vektor každému stromu. Každý strom následne vráti klasifikáciu a hovoríme, že stromy *hlasujú*. Náhodný les následne zvolí klasifikáciu podľa väčšiny z hlasov všetkých stromov v lese.

Trénovanie

Pri trénovaní náhodných lesov sa využívajú dve hlavné techniky. Prvou z nich je takzvaný *bagging*, alebo tiež *bootstrapping*. Máme N trénovacích vektorov. Trénovaciu množinu pripravíme pre každý strom zvlášť tak, že z pôvodných N vektorov vyberieme náhodne s opakovaním N vektorov, ktoré budú použité na natrénovanie daného rozhodovacieho stromu.

Druhá technika spočiva v zmene trénovacieho algoritmu pre stromy. Nech trénovacie vektory majú M atribútov. Namiesto všetkých M atribútov sa v každom vrchole vyberie náhodne len $m << M$ z nich. Najlepší atribút na rozdelenie vrcholu sa potom vyberá z týchto m atribútov, pričom m je konštantné počas celého algoritmu.

Každý strom je potom plne natrénovaný, bez orezávania.

Parameter m je jediným parametrom, na ktorý je náhodný les citlivý. Ako bolo ukázané v [Bre01], chyba náhodného lesa závisí na dvoch veciach: korelácii medzi dvoma storomami v lese (zvýšenie korelácie vedie k zväčšeniu chyby) a sily[3] každého stromu v lese (zvýšenie sily jednotlivých stromov vedie k zníženiu chyby). Ak znížime m znížime oboje – koreláciu aj silu stromov. Optimálny rozsah m je zvyčajne celkom veľký a

[3]silný klasifikátor je taký, ktorý má malú chybu

v praxi sa zvykne často používať $m = \sqrt{M}$ alebo sa zvykne m zvoliť, tak aby sa minimalizovala *out of bag*[4] chyba.

3.2 Použitie náhodných lesov na klasifikáciu zarovnaní

Keďže v našich modeloch máme dva typy stavov, použili sme dva typy klasifikátorov: *Match klasifikátor* pre Match stav a *Indel klasifikátor* pre Inzert stavy.

Klasifikátory dostanú na vstupe dáta asociované s pozíciami v daných sekvenciách a rozdeľujú ich do dvoch tried – nula a jedna. V Match klasifikátore trieda jedna znamená, že dané dve pozície majú byť zarovnané k sebe a nula znamená, že nie. V Indel klasifikátore trieda jedna označuje pozície, ktoré majú byť zarovnané k medzere a nula tie, ktoré nemajú. Ako výstupnú hodnotu pre naše modely berieme pravdepodobnosť, že dané dáta patria do triedy jedna.

Tieto pravdepodobnosti sú akousi mierou istoty daného klasifikátora, a keďže tieto dva klasifikátory sú nezávislé, súčet ich výstupov nemusí byť jedna. Sú totiž tri možnosti, čo sa môže stať:

- dve bázy sú zarovnané k sebe
- báza je zarovnaná s nejakou inou bázou (túto možnosť klasifikátory nepokrývajú)
- báza je zarovnaná s medzerou

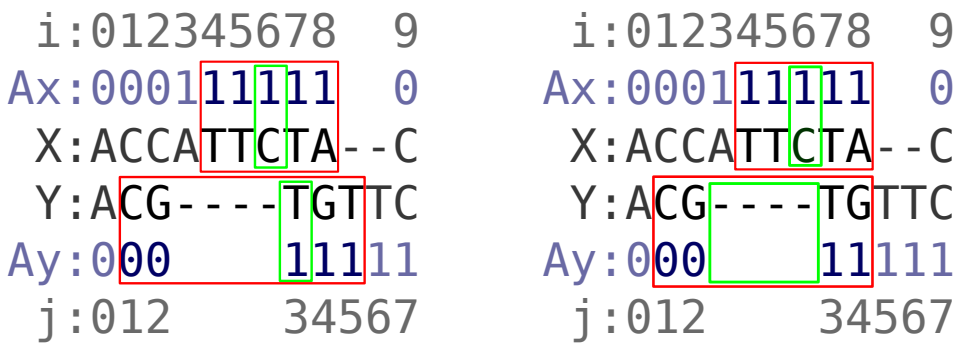
3.2.1 Výber atribútov

Klasifikátor dokáže pracovať len s poľom číselných atribútov, preto bolo treba najskôr dáta dať do tejto podoby. Navyše bolo treba rozhodnúť, aké dáta klasifikátoru podstrčíme. Potrebovali sme, aby klasifikátor videl bázy a ich anotácie na daných pozíciách. Navyše sme sa rozhodli pridať ako pomôcku aj okolie daných báz spolu s anotáciami. Toto okolie budeme volať *okno*. Vyskúšali sme štyri rôzne formy okna a zisťovali sme, ktorá forma je pre klasifikátor najvhodnejšia. Na to sme mali dve miery: úspešnosť klasifikátora a dôležitosť atribútov. Pri úspešnosti sme sledovali rozdelenie výstupu klasifikátora pre pozitívne a negatívne atribúty. Dôležitosť atribútov sme mali hlavne na kontrolu a vizualizáciu, ktoré dáta považuje klasifikátor za dôležité.

[4]dáta, ktoré sa nenachádzajú v jednotlivých trénovacích množinách pre dané stromy, viac v [BC]

Definícia okna

Okno veľkosti w pozostáva z $2w$ blokov veľkosti $k = (1 + \#\text{anotácií})$. Majme teda dve sekvencie, $X = x_1x_2 \dots x_n$ a $Y = y_1y_2 \dots y_n$ a pozície i a j. Pri Match klasifikátore okno veľkosti w obsahuje $x_{i-w/2} \dots x_i \dots x_{i+(1+w)/2}$, $y_{j-w/2} \dots y_j \dots y_{j+(1+w)/2}$ a všetky anotácie príslušných báz. (Obr. 3.2a) Pri Indel klasifikátore používame tiež dve pozície – prvá je pozícia bázy, na ktorú sa pýtame a druhá je pozícia medzery medzi dvoma bázami. Predpokladajme teraz, že X je sekvencia s bázou. Okno Indel klasifikátora veľkosti w obsahuje $x_{i-w/2} \dots x_i \dots x_{i+(1+w)/2}$, $y_{j-w/2} \dots y_j \dots y_{j+(1+w)/2-1}$ a všetky anotácie príslušných báz. (Obr. 3.2b)



(a) Match klasifikátor (b) Indel klasifikátor

Obr. 3.2: Okno klasifikátora pre pozície $i = 6$ a $j = 3$

Typ dát A - okno bez úpravy

Ako prvý typ dát sme zobrali okno tak, ako sme ho definovali v predošlej sekcii. Dáta obsahujú priamo všetky bázy a anotácie tak, ako sú v okne.

Typ dát B - zhody v stĺpcoch okna

Druhý typ dát obsahuje aktuálnu bázu spolu s jej anotáciami a navyše pole veľkosti $k * w$, ktoré má na i-tom mieste jedna ak $okno_X[i] = okno_Y[i]$, ináč nula (w je veľkosť okna, k je veľkosť bloku, $okno_X$ je časť okna zodpovedajúca sekvencii X a $okno_Y$ sekvencii Y). V Indel klasifikátore je jedna malá zmena: pozícia 0 aj 1 v x-ovej sekvencii sú porovnávané s pozíciou 1 v y-ovej a pozícia 2 v x-ovej sa porovnáva s pozíciou 2

v y-ovej. Pozíciu 1 v y-ovej sekvencii sme zopakovali preto, že sme experimentom zistili, že pre klasifikátor je dôležitá.

Typ dát C - matica zhôd v okne

Tretí typ dát je podobný ako typ B. Rozdiel je v tom, že teraz pole obsahuje nielen zhody po dvojiciach ale celú maticu zhôd. Teda opäť máme aktuálne bázy s anotáciami a pole má veľkosť $k * w^2$. Každý riadok sa skladá z jednotlivých blokov a v tabuľke v x-tom riadku, y-tom stĺpci a i-tom mieste v bloku je jedna práve vtedy, keď $okno_X[x+i] = okno_Y[y+i]$.

Typ dát D - kombinácia A a B

Posledný typ dát je kombináciou typov dát A a B. Dáta opäť obsahujú všetky bázy a anotácie tak ako v type A a navyše sme pridali pole zhôd z dát typu B. Táto informácia je síce redundantná a klasifikátor by si ju mal vedieť odvodiť aj sám, no experimenty ukázali, že to pomôže.

3.2.2 Trénovanie a testovanie klasifikátorov

Pre oba typy klasifikátorov sme sa snažili nájsť vhodné pozitívne a negatívne trénovacie príklady. Z oboch sme do trénovacej množiny zahrnuli rovnako, aby bola trénovacia množina vyvážená. Ak by sme nemali vyváženú trénovaciu množinu, klasifikátory by zvýhodnili triedu, ktorej príkladov by bolo viacej. Napríklad, keby sme pri Indel klasifikátore zobrali všetky pozície s medzerou ako pozitívne a všetky zarovnané pozície ako negatívne príklady, klasifikátor, ktorý by dával hodnotu nula, by mal pomerne vysokú úspešnosť, pretože zarovnaných pozícií je rádovo viac ako pozícií s medzerou. Preto by natrénovaný klasifikátor mal tendenciu dávať nižšie hodnoty, aby znížil trénovaciu chybu a tomu sa chceme vyhnúť.

Výber pozitívnych príkladov bol v oboch prípadoch intuitívny. Pri výbere negatívnych príkladov sme sa museli zamyslieť nad vhodným protipólom k pozitívnym dátam.

Výber pozitívnych a negatívnych príkladov pre Match klasifikátor

Match klasifikátor sme chceli natrénovať tak, aby pre pozície, ktoré majú byť pri sebe, dával hodnoty blízke jednej a pre pozície ktoré k sebe byť zarovnané nemajú dával hodnoty blízke nule. Ako pozitívne príklady sme teda vybrali pozície z trénovacích sekvencií, ktoré boli zarovnané k sebe.

Ako negatívne príklady sme vyskúšali dva prístupy: *náhodné dáta* a *dáta s posunom.*

Náhodné dáta: Náhodné dáta sme vyberali ako náhodné pozície, ktoré neboli zarovnané k sebe. Toto sa však ukázalo ako nedostatočné riešenie. Preto sme sa rozhodli pristúpiť k inému spôsobu výberu negatívnych vzoriek.

Dáta s posunom: Dáta s posunom sme vyberali posunutím jednej z dvoch pozícií v zarovnanom okne. Počas zarovnávania sa totiž väčšinou lokálne rozhodujeme, či zarovnať dané pozície k sebe, alebo vložiť medzeru. Ak by sme vložili medzeru, nastal by posun v jednej zo sekvencií. Rozhodli sme sa teda týmto inšpirovať a vyrábať negatívne príklady posunom zarovnaných pozícií. Pre každú zarovnanú pozíciu si najskôr rovnomerne náhodne vyberieme sekvenciu, ktorú budeme posúvať a potom z exponenciálneho rozdelenia náhodne vyberieme veľkosť posunu. Presný vzťah pre výpočet posunu Δ je

$$\Delta = (2D - 1) \cdot (1 + \lfloor S \rfloor) \qquad D \sim Alt(0.5),\ S \sim Exp(0.75),$$

kde prvý člen nám určuje smer posunu (čo je to isté ako: ktorá sekvencia sa bude posúvať) a druhý člen určuje veľkosť posunu (chceme celočíselnú hodnotu ≥ 1).

Výber pozitívnych a negatívnych príkladov pre Indel klasifikátor

Indel klasifikátor sme chceli natrénovať tak, aby pre miesta, ktoré majú byť zarovnané s medzerou, dával čo najvyššie hodnoty a pre miesta, ktoré nemajú byť zarovnané k medzere, dával čo najnižšie hodnoty. Ako pozitívne príklady sme sa teda rozhodli vyberať pozície, ktoré boli v trénovacích sekvenciách zarovnané k medzere. Ako negatívne príklady sme vybrali pozície, ktoré boli zarovnané k sebe, teda tie isté, čo sme pri trénovaní Match klasifikátora považovali za pozitívne. Akurát v tomto prípade mala jedna

zo sekvencií okno skrátené o jedna, teda akoby bola medzera pred bázou, s ktorou je aktuálne uvažovaná pozícia zarovnaná.

Parametre trénovacej a testovacej množiny

Dáta pre naše experimenty pochádzajú z nášho simulátora. Trénovacia sada sa skladá z 20 zarovnaní sekvencií, pričom každé zarovnanie má dĺžku 10000. Celá trénovacia sada pre Match klasifikátor má 329428 príkladov. Pre Indel klasifikátor má sada 34150 príkladov. Testovacia sada sa skladá z jedného zarovnania s dĺžkou 10000. Pre Match klasifikátor obsahuje 16498 príkladov. Pre Indel klasifikátor obsahuje 1674 príkladov. Vo všetkých sadách je polovica príkladov pozitívnych a polovica negatívnych.

3.3 Výsledky experimentov

Použili sme klasifikátory tak, ako sme popísali v sekcii 3.2 a výsledky sa nachádzajú v tabuľke 3.1.

3.3.1 Dôležitosť atribútov

Atribúty sme označili takto: stredná pozícia je 0, pozície naľavo sú -1, -2 a pozície napravo 1 a 2, pričom v Indel klasifikátore v sekvencii s medzerou pozícia 0 chýba. Bázy v sekvencii X označujeme x_i a v Y y_i. Anotácie v sekvencii X označujeme a_i a v Y b_i. Zhody medzi bázami i, j budeme označovať $m_{i,j}$, pričom označenie zhôd v type dát B a D zjednodušíme na m_i. Podobne zhody medzi anotáciami označíme $l_{i,j}$ (resp. l_i).

V tabuľke 3.1 si môžeme všimnúť, že klasifikátory sa zamerali najmä na bázy a okrem prípadu C anotácie skoro nebrali do úvahy. Toto správanie zodpovedá tomu, že v praxi bázy majú podstatne väčší význam pri zarovnávaní sekvencií.

V prípade dát typu A a B sa Match klasifikátory nezamerali na stredné bázy ako by sme to čakali, ale všetky bázy brali približne s rovnakou váhou. Indel klasifikátory na tom boli podobne, hoci v prípade dát typu B je vidieť výraznejšiu preferenciu atribútu zhody báz x_0, y_1 (l_0), avšak atribúty l_1 a l_2 klasifikátoru neprišli dôležité.

Pri type dát C si môžeme všimnúť, že najdôležitejšie atribúty sú na uhlopriečke, čo zodpovedá typu dát B, ibaže v tomto prípade sa nám vyskytli na niektorých pozíciách

Typ dát	chyba		atribúty	
	trénovacia	testovacia	najvýznamnejšie	najbezvýznamnejšie
A	93,07%	83,57%	x_0, y_{-1}, y_1, x_*, y_*	$\{a, b\}_*$
B	84,05%	84.31%	m_{-1}, m_0, m_*	l_*
C	80,44%	79,79%	$m_{-2,-2}$, $m_{2,2}, l_{-1,-1}, l_{1,1}, m_{0,0}$	ostatné
D	93,65%	84,32%	$m_0, m_{-1}, m_1, m_2, m{-}2$	l_*, $\{a, b\}_*$

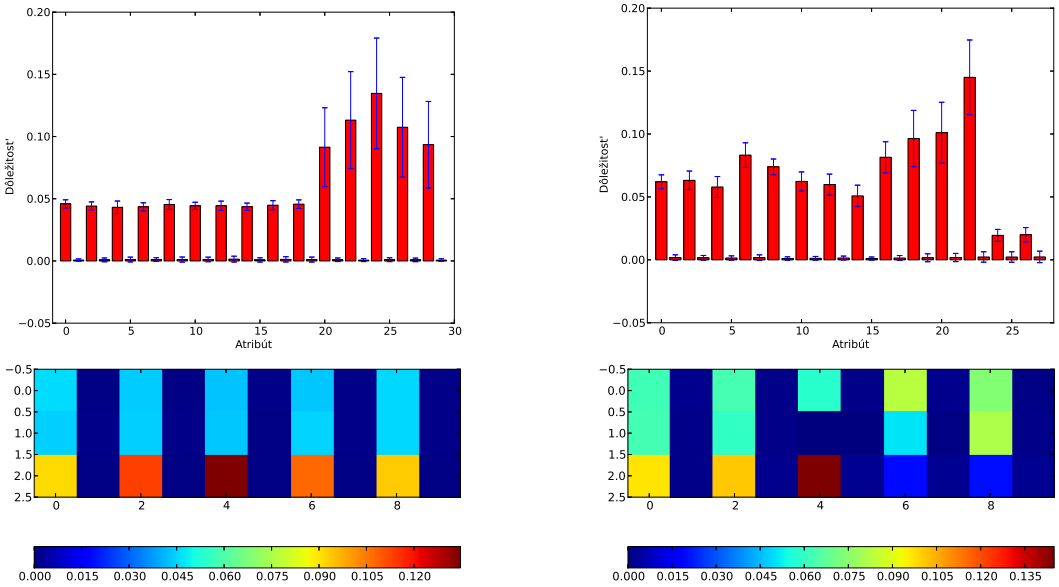
(a) Match klasifikátor

Typ dát	chyba		atribúty	
	trénovacia	testovacia	najvýznamnejšie	najbezvýznamnejšie
A	88,51%	74,13%	x_0, x_1, x_{-1}, x_*, y_*	$\{a, b\}_*$
B	77,17%	75,75%	m_0, m_{-1}, m_{-2}	l_*, m_1, m_2
C	78,03%	75,03%	$l_{2,2}$, $m_{-2,-2}$, $m_{0,1}$, $l_{-1,-1}$, x_0, a_0	ostatné
D	88,78%	76,46%	$m_0, m_{-1}, m_{-2}, x_1, \{x, y\}_*$	$\{l, a, b\}_*$

(b) Indel klasifikátor

Tabuľka 3.1: Porovnanie vlastností klasifikátorov pri rôznych typoch dát.

anotácie namiesto báz. Avšak ak sa nám tam vyskytla anotácia, bázu už klasifikátor nepovažoval za dôležitú a aj naopak. Indel klasifikátor navyše považoval za dôležitejšiu aj bázu na aktuálnej pozícii a jej anotáciu. Pri type dát C, na rozdiel od ostatných, klasifikátor viac berie do úvahy aj anotácie.



(a) Match klasifikátor (b) Indel klasifikátor

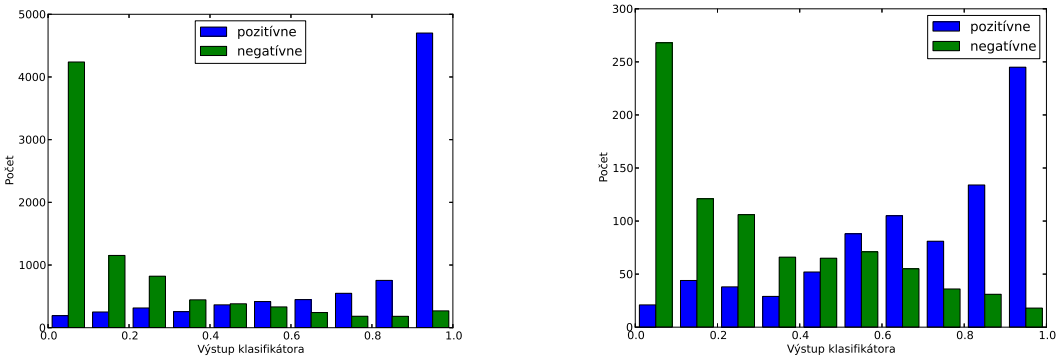
Obr. 3.3: **Dôležitosť atribútov pre typ dát D** - hodnoty sú normalizované, aby súčet bol jedna, modrý pásik označuje štandardnú odchýlku cez jednotlivé stromy v náhodnom lese. Pod grafom je tepelná mapa pre lepšiu vizualizáciu.

Pri dátach typu D sme dostali v prípade Match klasifikátora očakávanú distribúciu dôležitosti atribútov (obr. 3.3). Dáta typu B boli uprednostnené voči typu A, avšak aj bázy z typu A boli dôležité. Pri Indel klasifikátore sme dostali graf, ktorý je zložením grafov z dát typu A a B. Zaujímavosťou je, že pri tomto type dát má výraznejšiu úlohu práve zhoda na stredovej pozícii, teda pozície už nie sú také rovnocenné ako to bolo v type B.

3.3.2 Úspešnosť klasifikátora

V tabuľke 3.1 vidíme, že hoci typ C obsahuje nadmnožinu atribútov B, jeho úspešnosť je nižšia a tento typ nemá zmysel používať. Ďalej si môžeme všimnúť, že kombinácia

typov A a B využila klady oboch a mierne ich vylepšila.

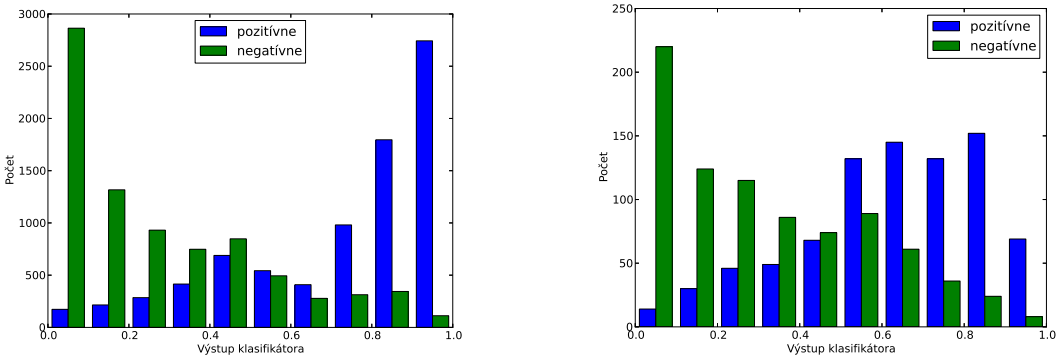


(a) Match klasifikátor

(b) Indel klasifikátor

Obr. 3.4: Distribúcia výstupu z klasifikátora pri type dát D – modré sú pozitívne príklady a zelené su negatívne. Na x-ovej osi je výstup klasifikátora a na y-je počet inštancií, pre ktoré výstup z klasifikátora padol do daného chlievika

Úspešnosť klasifikátorov s rôznym typom dát sme skúmali aj podrobnejšie, pozerali sme sa aj na distribúciu výstupov klasifikátora. Silný klasifikátor má totiž distribúciu takú, že väčšina pozitívnych príkladov má vysoké výstupné hodnoty a väčšina negatívnych klasifikátorov má nízke výstupné hodnoty. Najlepšiu distribúciu výstupov sa nám podarilo dosiahnuť práve pri type D (obr. 3.4). Na obrázku 3.5 sú pre porovnanie najhoršie distribúcie z ostatných typov dát.



(a) Typ C: Match klasifikátor

(b) Typ A: Indel klasifikátor

Obr. 3.5: Horšie distribúcie výstupov z klasifikátora – modré sú pozitívne príklady a zelené sú negatívne. Na x-ovej osi je výstup klasifikátora a na y-je počet inštancií, pre ktoré výstup z klasifikátora padol do daného chlievika

3.4 Zhrnutie

V tejto kapitole sme si predstavili klasifikátory, ktoré použijeme v našich modeloch. Uviedli sme dva typy klasifikátorov – pre Match stav a Inzert stav. Venovali sme sa výberu atribútov pre klasifikátory a ich trénovaniu. Predstavili sme si štyri množiny atribútov pre naše klasifikátory, pre ktoré sme porovnali vlastnosti klasifikátora.

Nakoniec sme sa rozhodli použiť ako množinu atribútov typ D, pretože s týmito atribútmi mali klasifikátory najlepšie vlastnosti a aj najvyššiu úspešnosť.

4 Modely

Hlavnou úlohou našej práce bolo preskúmať využitie dodatočnej informácie pri zarovnávaní sekvencií. Dodatočnú informáciu máme vo forme anotácií k sekvenciám. Anotácie sme sa rozhodli zakomponovať do zarovnania pomocou klasifikátorov, ktoré zosumarizujú informáciu obsiahnutú v anotáciách do jendého čísla. Prácu sme rozdelili na dve časti. Prvú časť tvorí kapitola 3, kde sme riešili problém ako základe lokálnej informácie rozhodúť, či dané pozície majú byť zarovnané k sebe alebo k medzere. Navrhli sme dva klasifikátory – pre Match a Inzert stav, pričom prvý rozhoduje, či dané dve pozície majú byť zarovnané k sebe alebo nie. Druhý rozhoduje, či daná pozícia má byť zarovnaná k medzere.

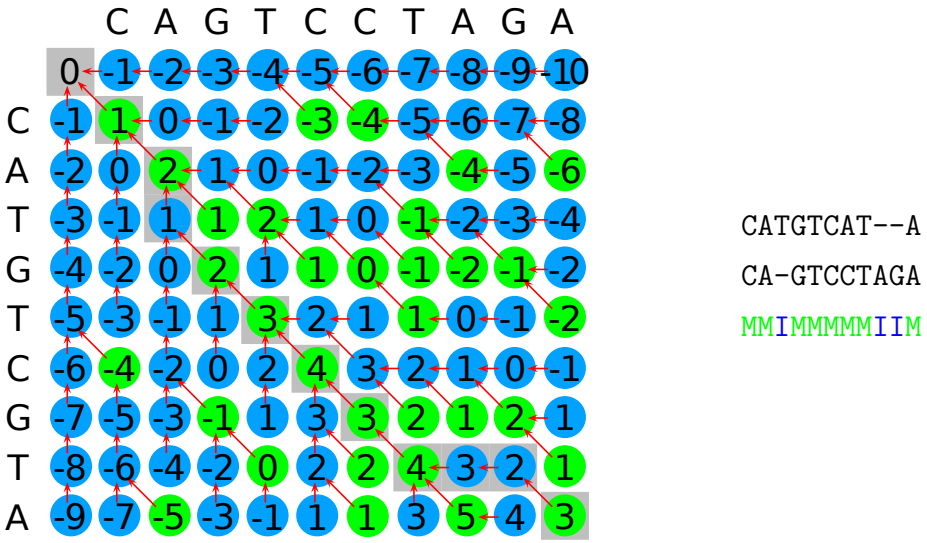
V tejto kapitole sa venujeme druhej časti, ktorú tvorí zakomponovanie výsledkov z klasifikátorov do modelov pre zarovnávanie sekvencií. V kapitole 1.6 sme si zadefinovali pHMM pre zarovnávanie sekvencií (obr. 1.7). Najskôr si popíšeme architektúry našich dvoch modelov, ktoré sú založené na pHMM, pričom pHMM sme modifikovali tak, aby zahŕňal výstupy z klasifikátorov. Potom sa budeme venovať trénovaniu jednotlivých modelov a nakoniec si modely porovnáme a zhodnotíme ich úspešnosť.

4.1 Integrácia párových HMM a výsledkov z klasifikátorov

Pri zarovnávaní sekvencií si klasifikátor z ktorého zoberieme výstup vyberieme podľa toho, v ako stave sa práve nachádzame. Pri Viterbiho algoritme konštruujeme dvojrozmernú tabuľky dynamického programovania, v ktorých sú na políčku (i, j) optimálne hodnoty pre zarovnania končiace na i-tej pozícii v sekvencii X a na j-tej pozícii v sekvencii Y. Okrem toho si pamätáme aj od kiaľ sme do daného políčka prišli. Do týchto tabuliek si môžme pridať aj informáciu o výstupe klasifikátora. To ktorý klasifikátor

použijeme závisí od smeru, ktorým sme sa pohli. Ak sme sa pohli diagonálne, použijeme Match klasifikátor, ináč Indel klasifikátor.

Ak teda pridáme k zarovnaniu pásku s výstupom klasifikátora, môžme si ju predstaviť ako cestu v tabuľke, ktorá zodpovedá ceste zarovnania (obr. 4.1). Takúto pásku využijeme v jednom z našich modelov, ktoré definujeme v nasledujúcej sekcii.

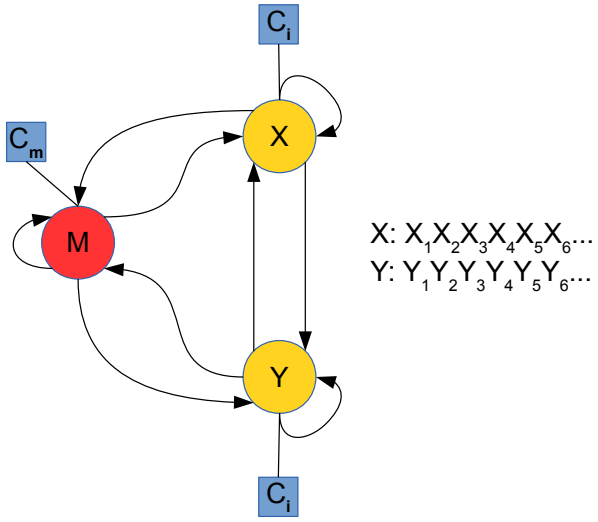


Obr. 4.1: Použité klasifikátory v klasifikátorovej páske pri zarovnávaní sekvencií. Pre jednoduchosť je použitá tabuľka z algoritmu Needelman-Wunch (kapitola 1.5.1) na globálne zarovnanie sekvencií. Viterbiho algoritmus používa tri tabuľky a cesta zarovnania vedie cez všetky tri. Zelenou farbou sú označené políčka, kde sa použije Match klasifikátor, modrou sú políčka, kde sa použije Indel klasifikátor. Šedou je vyznačené zarovnanie spolu s klasifikátorovou páskou pre zarovnanie. Vpravo je vypísané to isté zarovnanie spolu s klasifikátormi z klasifikáotrovej páske.

4.1.1 Definície modelov

Prvým modelom je *model s klasifikátorom ako emisiou (ďalej len model A)*. Tento model vyzerá rovnako ako základný model, aj pravdepodobnosti prechodov zostanú rovnaké, ale emisnú pravdepodobnosť nahradíme výstupom z klasifikátora.

V základnom modeli pre zarovnanie (kapitola 1.6) je pravdepodobnosť emisie v match stave $P(X = x \wedge Y = y)$, kde X, Y sú náhodné premenné pre bázy v sekvenciách



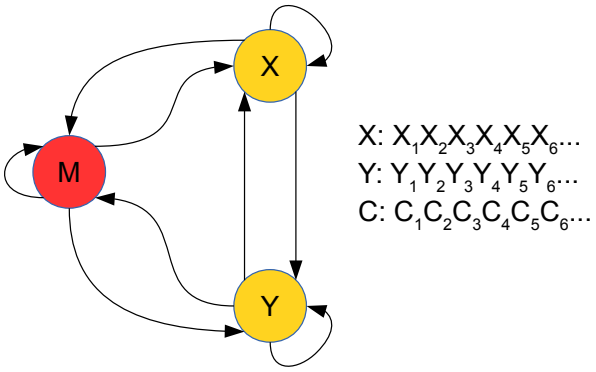
Obr. 4.2: Model s klasifikátorom ako emisiou

X a Y a x, y sú konkrétne bázy. Pre Inzert stavy analogicky. Narozdiel od toho, výstup klasifikátora je definovaný ako $P(C = 1|W = w)$, kde $C \in \{0, 1\}$ je náhodná premenná pre výstupnú triedu, pričom nula znamená, že dáta nemajú byť zarovnané k sebe a 1 znamená, že majú byť, W je náhodná premenná pre okno klasifikátora, zahŕňajúce všetky atribúty a w je konkrétna hodnota okna. Pre porovnanie, ak by sme mali Match klasifikátor, a ako atribúty len príslušné bázy bolo by to $P(C = 1|X = x \wedge Y = y)$. Vidíme teda, že tieto dve pravdepodobnosti sú definované ináč a zatiaľ čo

$$\sum_{x \in X, y \in Y} P(X = x \wedge Y = y) = 1,$$

pre $\sum_{x \in X, y \in Y} P(C = 1|W = w)$ takáto rovnosť neplatí. Viterbiho algoritmus však bude fungovať aj s hodnotami $P(C = 1|W = w)$ namiesto emisií a bude hľadať zarovnanie, ktoré bude maximalizovať súčin s týmito hodnotami. O takomto modeli už však nemôžme hovoriť ako o pravdeopdobnostnom, ani ako o pHMM. Tento model je len inšpirovaný pHMM.

Keďže predošlý model nie je korektný pravdepodobnostný model, navrhli sme alternatívny model, *model s klasifikátorovou páskou (ďalej len model B)*, ktorý navyše modeluje aj výstup z klasifikátora a teda je korektný pravdepodobnostný model. Nemodelujeme len dvojicu sekvencií, ale aj sekvenciu výstupov klasifikátora vo forme pásky, ktorú sme si definovali na začiatku kapitoly 4. Pásku s výstupom z klasifikátora považujeme za akúsi pomôcku pre náš zarovnávač. Klasifikátor môže vrátiť ľubovoľnú



Obr. 4.3: Model s klasifikátorovou páskou

hodnotu z intervalu $\langle 0, 1\rangle$. Keďže výstup z klasifikátora je spojitý, museli sme sa s týmto problémom vysporiadať. Preto sme navrhli dve varianty tohto modelu – diskrétnu a spojitú.

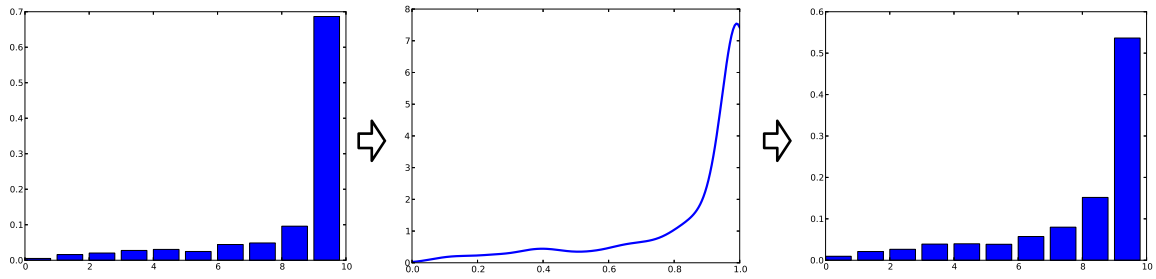
V diskrétnom modeli sme hodnoty klasifikátora rovnomerne rozdelili na b hodnôt a výstup sme zaokrúhlili na najbližšiu z nich. V našich modeloch sme zvolili $b = 10$.

Alternatíva k predošlej metóde je použiť spojitý HMM. Hlavný rozdiel medzi spojitými a diskrétnymi HMM je, že spojité HMM nepočítajú s pravdepodobnosťou, ale s hustotou. Hodnota hustoty v danom bode na rozdiel od pravdepodobnosti môže byť aj väčšia ako jedna. Podľa [HHL89], jedným zo štandardných spôsobov reprezentácie hustoty v spojitých HMM je zmes gausiánov, takže využijeme interpoláciu takouto distribúciou a to budeme brať ako hustotu výstupu klasifikátora (obr. 4.4b).

4.1.2 Diskrétizácia skôre klasifikátorov

Pre diskrétnu verziu modelu B sme sa zaoberali spôsobom diskretizácie. Diskretizovať hodnoty môžme dvoma spôsobmi. Buď jednoducho spočítame histogram pre trénovacie dáta, alebo interpolujeme vstupnú vzorku pomocou spojitej distribúcie, napr. pomocou zmesi gausiánov (tak ako v spojitej verzii) a potom toto rozdelenie diskretizujeme (obr. 4.4). Druhá spomenutá metóda má výhodu v tom, že vyhladí šum a doplní chýbajúce dáta, ale keďže je pouzitá aproximácia, zavádza určitú nepresnosť.

Ako vidíme z tabuľky 4.1, spojitá verzia modelu sa nám neosvedčila. Má totiž nedostatok, že distribučná funkcia nedosahuje maximum pri výstupe klasifikátora rovnom jedna, ale kúsok predtým, čo znamená istú penalizáciu v prípade, že si je klasifikátor



(a) Histogram (b) Zmes gausiánov (c) Diskretizovaný gausián

Obr. 4.4: Dve možnosti diskretizácie výstupu klasifikátora: buď použijeme histogram 4.4a, alebo dáta najskôr aproximujeme pomocou zmesi gausiánov 4.4b a následne zdiskretizujeme ako na obr. 4.4c.

	Úspešnosť
Histogram	82,25%
Nasekaný gausián	82,98%
Zmes gausiánov	65,59%

Tabuľka 4.1: Porovnanie úspešností modelu B pri rôznom spracovaní pásky.

príliš istý. To, či použijeme vyhladenie pomocou gausiánu až tak nezaváži, ale rozhodli sme sa ho predsa len využiť, pretože úspešnosť bola trochu vyššia takmer na všetkých testovaných sekvenciách.

4.2 Trénovanie modelov

Množiny pre trénovanie klasifikátora ostali také, ako sme si popísali v kapitole 3.2.2. Množina biologických dát pre trénovanie klasifikátora sa skladala z 30 sekvencií s dĺžkami od 1000 do 1500.

V prípade simulovaných dát 1 a 2 boli trénovacie množiny pre naše modely jedno zarovnanie dĺžky 10000. Testovacie množiny boli 6 zarovnaní dĺžky 1000. Zarovnania boli vygenerované našim simulátorom, ktorý je popísaný v kapitole 5.3.1. ToDo: fixnut V tabuľke 4.2 uvádzame pravdepodobnosť mutácie pre jednotlivé dáta. V prípade bio-

anotácia X	anotácia Y	sim1	sim2
1	1	0,20	0,01
1	0	0,35	0,30
0	1		
0	0	0,40	0,99

Tabuľka 4.2: Pravdepodobnosť mutácie v simulovaných dátach 1 a 2

logických dát má trénovacia množina pre modely 41 sekvencií, pričom každá má dĺžku od 1000 do 2500. Testovacia množina sa skladá zo 6 sekvencií s dĺžkami od 1000 do 1500.

Naše modely trénujeme pomocou metódy maximálnej vierohodnosti, tak ako sme si to popísali v kapitole 1.6.4. V HMM trénujeme prechodové a emisné pravdepodobnosti. Prechodové pravdepodobnosti sme trénovali vo všetkých modeloch rovnako a emisné pravdepoodbnosti rôzne.

V modeli A sme trénovali iba prechodové pravdepodobnosti, emisie sme mali priamo z natrénovaného klasifikátora.

V modeli B sme trénovali aj prechodové aj emisné pravdepodobnosti. Označme si bázy v sekvenciách X, Y a výstup klasifikátora C. Veľkými písmenami X, Y, C budeme označovať náhodné premenné pre bázy a výstup klasifikátora, malými písmenami budeme označovať konkrétne hodnoty. V match stave generujeme trojice: (x, y, c) a v Inzert stavoch generujeme dvojice (x, c) alebo (y, c). Popíšeme si len trénovanie emisií v Match stave, Inzert stavy trénujeme analogicky. V diskrétnej verzii modelu si potrebujeme, vyrobiť tabuľku $e[x, y, c] = P(X = x, Y = y, C = c) \quad \forall x, y, c$. V spojitej verzii si potrebujeme zapamätať hustotu C pre každú možnú dvojicu (x, y). Teda budeme mať tabuľku $e[x, y] = f_{x,y}(t) \quad \forall x, y$, kde $f_{x,y}(t)$ je hustota náhodnej premennej C za predpokladu, že $X = x$ a $Y = y$. Keďže distribúciu C chceme počítať pre každú možnú dvojicu báz zvlášť, musíme si ukázať, že pravdepodobnosť $P(X = x, Y = y, C = c)$ vieme rozložiť pomocou $P(C|X \cap Y)$. To vyplýva priamo z definície podmienenej pravdepodobnosti:

$$P\left(C = c \wedge X = x \wedge Y = y\right) = P\left(C = c | X = x \wedge Y = y\right) P\left(X = x \wedge Y = y\right)$$

4.3 Výsledky experimentov

V tejto sekcii sme porovnáme naše výsledky s výsledkami referenčného modelu a zarovnávača *muscle* [Edg04] na našich troch dátových množinách popísaných v sekcii 4.2.

Potom uvedieme výsledky nikoľkých experimentov, v ktorých sme porovnávali naše modely s rôznymi parametrami a v rôznych podmienkach.

Keďže k biologickým dátam nepoznáme správne zarovnanie, rozhodli sme sa zaviesť okrem percentuálnej zhody ďalšiu mieru. Túto mieru budeme označovať *tranzitivita* a budeme ju počítať z troch párových zarovnaní, ktoré dostaneme z troch sekvencií zarovnaním všetkých ich kombinácií. Označme si tri sekvencie ako A, B, C. Vyrobíme si tri zarovnania: AB, BC a AC. Tranzitivitu spočítame ako percentuálnu zhodu medzi zložením prvých dvoch zarovnaní ($AB \circ BC$) a tretieho zarovnania AC. V ideálnom prípade by sme dostali 100% a mali by sme multizarovnanie troch sekvencií. V praxi sú však hodnoty nižšie. Táto miera určuje konzistentnosť zarovnávača, čo je jedna z dôležitých vlastností kvalitného zarovnávača. V prípade biologických dát považujeme túto mieru za dôležitejšiu, pretože v tomto prípade náš zarovnávač môže nájsť biologicky správnejšie zarovnanie ako bolo to pôvodné. Avšak my sme si na naše testy vybrali multizarovnania 7 sekvencií, ktoré sú považované za veľmi kvalitné. Preto budeme požadovať aj vysoké percento zhody ako sekundárne kritérium.

Dáta	Model A		Model B		Ref. Model		Muscle	
	Zhoda	Tranz.	Zhoda	Tranz.	Zhoda	Tranz.	Zhoda	Tranz.
Sim	79,75%	44,97%	84,35%	56,5%	**85,78%**	**61,03%**	82,72%	58,76%
Bio	**91,40%**	96,63%	91,24%	**96,89%**	91,34%	96,45%	91,28%	95,98%

Tabuľka 4.3: Porovnanie našich modelov s referenčným modelom a zarovnávačom muscle.

V tabuľke 4.3 sme porovnávali naše modely so základným modelom a zarovnávačom muscle. Na simulovaných dátach, ktoré boli generované náhodne, naše modely nedokázali prekonať referenčné modely. Avšak na biologických dátach, v ktorých je viacej pravidelností, dokázali dodatočné inforácie pomôcť dosiahnuť lepšiu tranzitivitu a v prípade modelu A aj lepšiu úspešnošť.

4.3.1 Vplyv dodatočnej informácie na zarovnanie

V tejto sekcii sme zisťovali, aký vplyv má dodatočná informácia na zarovnanie. Vyskúšali sme zakázať anotácie pri veľkosti okna jedna a 9. Pri veľkosti okna rovnej jedna nemáme okrem anotácie žiadnu ďalšiu informáciu. Pri veľkosti okna 9 máme aj okolie sekvencie.

Veľkosť okna	Anotácia	Model A	Model B
1	nie	74,42%	18,35%
1	áno	76,95%	57,86%
9	nie	78,43%	82,67%
9	áno	79,75%	84,35%

Tabuľka 4.4: Vplyv dodatočnej informácie na zarovnanie.

V tabuľke 4.4 si môžeme všimnúť, že zatiaľ čo pri modeli A je úspešnosť veľmi podobná, hoci aj anotácie aj väčšie okno mierne pomohli. Naopak pri modeli B sú rozdiely výrazne väčšie, vidíme, že aj samotná anotácia výrazne pomohla a väčšie okno je pre tento model nutnosťou.

4.3.2 Použitie jedného klasifikátora pre Match aj Inzert stav

V tejto sekcii sme vyskúšali nahradenie Indel klasifikátora Match klasifikátorom, ktorý vracia opačné hodnoty. Teda trénujeme len jeden klasifikátor.

	Model A	Model B
1 klasifikátor	74,53%	82,72%
2 klasifikátory	79,75%	84,35%

Tabuľka 4.5: Porovnanie použitia jedného alebo dvoch typov klasifikátorov.

Z tabuľky 4.5 vidno, že použitie dvoch typov klasifikátorov má svoje opodstatnenie, hoci naše modeli dokážu fungovať aj z jedným.

4.4 Zhrnutie

5 Implementácia

V tejto kapitole sa budeme zaoberať implementáciou nášho riešenia. Popíšeme si dôvody pre výber jazyka a knižníc a niektoré časti nášho zarovnávača slúžiace na predspracovanie dát, klasifikáciu a zarovnávanie. Potom si popíšeme dôležité pomocné programy – simulátor a program na trénovanie modelov – a nakoniec si popíšeme použitie, konfiguráciu a možnosti rozšírenia programu.

5.1 Výber jazyka a použité knižnice

Ako jazyk pre písanie nášho zarovnávača sme si vybrali Python (vo verzii 2.7). Dôvodov sme mali niekoľko: V prvom rade zarovnávač sme nepísali od začiatku, ale ako základ sme zobrali zarovnávač od Michala Nánásiho (viď sekcia 5.1.1), ktorý už bol napísaný v jazyku Python 2.7. Ďalšími dôvodmi sú pohodlnosť písania v jazyku Python a dostupnosť veľkého množstva užitočných knižníc. Knižnice pre Python sa dajú písať aj v jazyku C alebo C++, čo umožňuje skĺbiť rýchlosť C a jednoduchosť jazyka Python.

5.1.1 Realigner

Realigner je program na zarovnávanie a prezarovnávanie[1], ktorý napísal Michal Nánási pre potreby [NVB13]. Umožňuje prezarovnanie sekvencií pomocou párových HMM a Viterbiho algoritmu. Prezarovnanie dokáže robiť aj v rámci lokálneho okolia okolo pôvodného zarovnania. Program je rozšíriteľný o nové modely, čo sme využili pre naše potreby.

[1] modifikáciu existujúceho zarovnania

5.1.2 Pythonové knižnice

V našom programe sme sa okrem štandardnej knižnice jazyka Python rozhodli využiť niekoľko knižníc. Najdôležitejšími boli knižnice *numpy*, *scipy*, *scikit-learn*, *pyplot*, *track* a *pandas*.

Numpy je matematická knižnica, ktorá obsahuje množstvo užitočných matematických funkcií. Pre nás z nej boli dôležité hlavne štatistické funkcie, napr. generovanie náhodných čísel z rôznych rozdelení a počítanie štandardnej odchýlky. Okrem toho je táto knižnica základom aj pre väčšinu ostatných použitých knižníc.

Scipy obsahuje funkcie pre pokročilejšie vedecké výpočty. My sme z nej využili hlavne funkciu na odhad rozdelenia pomocou gausiánov.

Scikit-learn je knižnica zaoberajúca sa hlavne strojovým učením. Odtiaľ sme využili náhodné lesy.

Pyplot je knižnica na kreslenie grafov a využili sme ju na vygenerovanie všetkých našich vizualizácií.

Track je knižnica na prácu s `'.bed'` súbormi, v ktorých sme mali uložené naše anotácie.

Pandas je knižnica na analýzu a úpravu veľkých dát. Poslúžila nám najmä pri príprave anotácií pre biologické zarovnania.

5.2 Triedy na zarovnávanie s klasifikátorom

V tejto sekcii si popíšeme najdôležitejšie triedy, ktoré sme doplnili do programu Realigner. Ide o triedy na presdpracovanie dát, klasifikáciu a triedy pre stavy pHMM.

5.2.1 Predspracovanie dát

ToDo: obrazok classdiagramu

Na predspracovanie dát nám slúžili triedy *DataPreparer*, *ComparingDataPreparer*, *FullComparingDataPreparer*, *CombinedDataPreparer*, *IndelDataPreparer*, *ComparingIndelDataPreparer*, *FullComparingIndelDataPreparer* a *CombinedIndelDataPreparer* (obr **??**).

Trieda DataPreparer je rodičovskou triedou pre ostatné triedy, takže si popíšeme

najmä ju. Trieda IndelDataPreparer je rodičom Indel verzií tried. Tieto triedy majú dve hlavné úlohy: Prvá je predpríprava dát na klasifikáciu a druhá je výroba trénovacej množiny zo sekvencií.

Pri predpríprave dát je úlohou dať dáta do formy vhodnej pre klasifikátor. To znamená získať okno a anotácie a ďalšie atribúty tak ako sme ich popísali v kapitole 3.2.1 a previesť ich do číselnej podoby. Naše štyri typy tried, ktoré zodpovedajú štyrom typom dát, sa líšia iba v tejto úlohe. Najdôležitejšia práca sa vykoná v metóde `_prepare_sequence`, ktorá pripraví dáta pre jednu sekvenciu, a potom sa zlepia dáta z volania tejto funkcie pre obe sekvencie. Indel verzie tried obsahujú príslušné úpravy aby v sekvencii s medzerou použili kratšie okno.

Pri predspracovaní trénovacej množiny treba zo zarovnaní vytvoriť pozitívne a negatívne príklady. Táto metóda je implementovaná len v DataPreparer a IndelDataPreparer triede, ostatné ju zdedia. Pri výbere pozitívnych príkladov v triede DataPreparer označujeme pozície v sekvencii, ktoré sú zarovnané. To má na starosti metóda `prepare_positive_data`. Metóda `prepare_negative_data` vyberá negatívne príklady použitím zarovnaných pozícií z pozitívnych príkladov a náhodne ich poposúva podľa vzorca spomenutého v sekcii 3.2.2. Okrem toho sme implementovali aj náhodný výber negatívnych dát v metóde `prepare_negative_data_random`, kde negatívne príklady vyberáme náhodne z nezarovnaných pozícií s tým, že vyberieme rovnako veľa negatívnych ako pozitívnych príkladov. V Indel klasifikátore sa oba typy príkladov vyberajú v metóde `prepare_training_data`, pričom jedna sekvencia sa zvolí za medzerovú a za pozitívne príklady sa vezme všetko, čo je zarovnané k medzere v medzerovej sekvencii. Za negatívne sa vezme rovnako veľa náhodných zarovnaných pozícií.

5.2.2 Abstrakcia klasifikátora

V pre väčšiu modularitu sme sa rozhodli naprogamovať vrstvu medzi klasifikátorom a zvyškom programu. Trieda sprostredkúvajúca túto vrstvu sa volá *PairClassifier*. PairClassifier poskytuje niekoľko metód navyše oproti tým, ktoré poskytuje klasifikátor. Jej hlavnými výhodami sú:

- možnosť rýchlej výmeny klasifikátora bez úpravy zvyšku programu
- automatické predspracovanie pri klasifikácii a trénovaní pomocou príslušnej triedy

DataPreparer

- automatické trénovanie – ak klasifikátor nie je natrénovaný, tak sa automaticky natrénuje z prednastavenej trénovacej množiny
- hromadná klasifikácia s predspracovaním
- otočenie výstupu klasifikátora

5.2.3 HMM stavy s klasifikátorom

V zarovnávači bol implementovaný Viterbiho algoritmus, ktorý pracoval s párovými HMM. Tento algoritmus pracoval s triedou *GeneralizedPairState*, ktorá poskytovala metódu `emission` vracajúcu emisiu báz na daných pozíciách. Od tejto triedy dedia naše triedy *ClassifierState*, *ClassifierIndelState*, ktoré tvoria náš model A a preťažili sme metódu `emission` tak, aby vracala hodnoty z klasifikátora. Museli sme však urobiť niekoľko zmien. Viterbiho algoritmus si pýta emisie po jednej. Avšak volanie C-čkovej funkcie z knižnice scikit-learn z Pythonu je pomalé, takže pre dlhé sekvencie by zarovnanie trvalo veľmi dlho. Preto sme si najskôr naraz, počas jedného volania funkcie predpočítali emisie pre všetky pozície a tie sme potom zapamätali a vracali v momente, keď si ich Viterbiho algoritmus vypýtal. Na to sme využili možnosť hromadnej klasifikácie v našej triede PairClassifier (sekcie 5.2.2). Tieto dve triedy sa líšia iba v použitej triede na predspracovanie dát.

Od tried ClassifierState a ClassifierIndelState potom dedia triedy *ClassifierAnnotationState* a *ClassifierAnnotationIndelState*, ktoré tvoria diskrétnu verziu nášho modelu B. Triedy mali v súbore uložené tabuľky emisií a podľa aktuálnych pozícií si zistili bázy a aktuálny výstup klasifikátora, ktoré slúžili ako indexy do tabuľky. Tieto triedy navyše obsahujú metódy na trénovanie emisií.

Od ClassifierAnnotationState a ClassifierAnnotationIndelState dedia triedy *ContignuousClassifierAnnotationState* a *ContignuousClassifierAnnotationIndelState*, ktoré stvárňujú spojitú verziu modelu B. Tieto triedy majú v tabuľke emisií pre každú dvojicu báz uloženú serializovanú verziu objektu *gaussian_kde* (viac detailov je v sekcii 5.3.2), ktorý obsahuje distribúciu výstupu klasifikátora. Konkrétnu hodnotu potom získa volaním príslušného objektu k danej dvojici báz (resp. báze pri IndelState) s hodnotou klasifikátora. Tieto triedy takisto obsahujú aj metódy na trénovanie emisií.

Od triedy *GeneralizedPairState* navyše dedia aj naše triedy pre referenčný model – *SimpleMatchState* a *SimpleIndelState*, ktoré triedu GeneralizedPairState rozširujú o metódy na trénovanie emisných pravdepodobností.

ToDo: obrazok classdiagramu

5.3 Pomocné programy

V tejto sekcii sa budeme venovať dvom hlavným pomocným programom. Prvým z nich je náš simulátor, ktorý nám umožňuje generovať zarovnania a druhým je program na trénovanie zarovnaní. Okrem toho sme implementovali množstvo programov na tvorbu vizualizácií pre túto prácu, alebo konverziu dát z rôznych zdrojov do formátov podporovaných našim zarovnávačom, o ktorých si však hovoriť nebudeme.

5.3.1 Simulátor

Simulátor slúži na generovanie zarovnaní pre naše experimenty. Náhodne vygeneruje tri[2] sekvencie, ktoré vznikli zo spoločného predka a vyrobí korektné spoločné zarovnanie všetkých troch sekvencií. Okrem toho vyrobí aj dodatočné informácie, ktoré majú pomôcť pri zarovnávaní.

Algoritmus

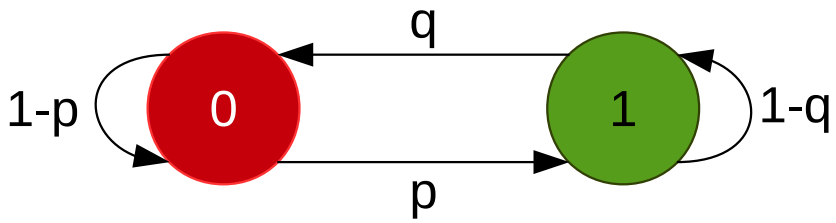
Ako prvú simulátor vygeneruje informáciu o tom, ktoré časti sekvencie prislúchajú génom a ktoré nie. Informácia má podobu binárneho vektora. Najskôr sa informácia vygeneruje pre *základnú (master) sekvenciu* a z nej sa potom skopíruje do troch dcérskych sekvencií, pričom s určitou pravdepodobnosťou sa niektoré gény zmažú (jednotky sa prepíšu na nulu). Táto informácia bude tvoriť pásku s anotáciou o géne.

Potom simulátor vygeneruje základnú sekvenciu a z nej následne pomocou mutácie a delécie (inzercia = delécia v zvyšných sekvenciách) odvodí tri ďalšie sekvencie. Mutácia má rôzne pravdepodobnosti v prípade, že ide o gén v jednej či druhej sekvencii, alebo oboch. Simulátor rovno generuje zarovnanie tým, že keď robí deléciu, nahradzuje bázy pomlčkami.

[2] aby sme vedeli počítať tranzitivitu

Generovanie informácie o génoch

ToDo: awk: Gény bývajú súvislé úseky, takže ich treba generovať tak, že niekedy začneme gén, potom generujeme jednotky, potom skončíme gén a generujeme nuly. Potom môžme opäť začať gén atď.



Obr. 5.1: Markovova reťaz použitá na generovanie informácie o génoch. p je pravdepodobnosť, že začneme generovať gén, q je pravdepodobnosť, že prestaneme generovať gén.

Generovanie robíme pomocou *Markovovskej reťaze (Markov Chain)* obr. 5.1. Generujeme podľa aktuálneho stavu a v každom kroku sa náhodne rozhodneme, či sa prepneme do iného stavu, alebo ostaneme v tom istom. Rozhodnutie robíme pomocou *falošnej mince (biased coin)*, kde hlava padne s danou pravdepodobnosťou, ktorú nastavíme ako parameter. V našom prípade je to p pre stav nula a q pre stav jedna.

Simulácia mutácie a delécie

Ak už máme vygenerovanú základnú sekvenciu, potom vyrobíme zmutovanú sekvenciu tak, že s určitou pravdepodobnosťou nahradíme bázu zo základnej sekvencie inou bázou. Pravdepodobnosť závisí aj od toho, či je na danej pozícii gén v oboch sekvenciách, v jednej, alebo v žiadnej. Na rozhodovanie máme pre každú možnosť jednu falošnú mincu a podľa toho, ktorá z možností nastala, hodíme si príslušnou mincou.

Deléciu simulujeme opäť pomocou podobnej Markovovskej reťaze, pretože počas evolúcie majú tendenciu vypadávať súvislé úseky. Stav nula bude, že nemažeme a stav jedna, že mažeme, teda nahradzujeme danú bázu znakom `'-'`.

Využitie

Simulátor je prvá vec, ktorú sme implementovali a slúžil na väčšinu našich experimentov. Simulované dáta majú totiž oproti biologickým niekoľko výhod. Prvou je,

že poznáme správnu odpoveď, a teda môžme objektívne zhodnotiť úspešnosť našich modelov. Druhou je možnosť zvoliť si parametre, ktoré menia závislosť mutácií na anotáciách a testovať správanie modelov, ak zvýšime túto závislosť. Ďalšou výhodou je, že si môžme vygenerovať ľubovoľne veľa ľubovoľne dlhých sekvencií.

5.3.2 Trénovanie modelov

Trénovanie modelov má na starosti trieda *ModelTraining* a je urobené pomocou metódy maximálnej vierohodnosti (kapitola 1.6.4). Trénovanie prechodových pravdepodobností je pre všetky modely spoločné. Trénovanie emisných pravdepodobností je v každom stave spravené zvlášť, pretože stavy emitujú rôzne symboly a jednotlivé stavy majú rôzne spôsoby trénovania. Trieda ModelTraining najskôr načíta model z ktorého zistí stavy. Potom načíta trénovacie sekvencie a oanotuje ich stavmi. Následne vypočíta prechodové pravdepodobnosti. Spočíta všetky výskyty nasledujúcich stavov vo všetkých sekvenciách pre každý stav zvlášť. To vydelí celkovým počtom výskytov daného stavu. Potom oanotované sekvencie predloží jednotlivým stavom, aby vypočítali emisné pravdepodobnosti.

Máme štyri základné typy stavov, ktoré sme si popísali v časti 5.2.3 a pre každý máme triedu pre Match a Inzert verziu:

- SimpleMatchState, SimpleIndelState
- ClassifierState, ClassifierIndelState
- ClassifierAnnotationState, ClassifierAnnotationIndelState
- ContignuousClassifierAnnotationState, ContignuousClassifierAnnotationIndelState

V SimpleMatchState, SimpleIndelState algoritmus priamo spočíta výskyty príslušných dvojíc báz (resp. báz) a vydelí celkovým počtom výskytu stavu. V ClassifierState, ClassifierIndelState sa emisie netrénujú, takže tabuľka bude prázdna. V ClassifierAnnotationState, ClassifierAnnotationIndelState sa pre každý stav a pre každú dvojicu báz (resp. bázu) spočítajú hodnoty z klasifikátora a tie sa následne aproximujú pomocou gausiánov. Na toto využijeme triedu `gaussian_kde` z modulu `scipy.stats`. Táto trieda robí odhad distribučnej funkcie z množiny hodnôt. Šírku pásma (parameter,

ktorý určuje hladkosť výslednej funkcie) sme nechali na predvolenej hodnote – „scott". Tá nastavuje šírku pásma podľa vzorca:

$$n^{-1/(d+4)},$$

kde n je počet vstupov a d je počet dimenzií (v našom prípade $d = 1$). Viac detailov je v [com13] alebo [Wik14a]. Následne túto funkciu diskretizujeme na k hodnôt. To spravíme pomocou určitého integrálu, pomocou metódy `integrate_box(a, b)`, ktorá spočíta určitý integrál od a po b. Keďže gaussian_kde je definovaný na intervale $\langle -\infty, \infty \rangle$ a

$$\int_{-\infty}^{\infty} gaussian\_kde(x)\mathrm{d}x = 1,$$

orezaním definičného oboru na $\langle 0, 1 \rangle$ sa môže stať, že

$$\int_{0}^{1} gaussian\_kde(x)\mathrm{d}x < 1.$$

V tom prípade výslednú hodnotu ešte musíme normalizovať vydelením hodnotou

$$\mu = \int_{0}^{1} gaussian\_kde(x)\mathrm{d}x.$$

Ako sme si ukázali v 4.2 výslednú pravdepodobnosť dopočítame vynásobením diskretizovanej pravdepodobnosti daného výstupu klasifikátora v prípade daných báz a pravdepodobnosti daných báz. V ContignuousClassifierAnnotationState, ContignuousClassifierAnnotationIndelState postupujeme rovnako, ako v predchádzajúcom prípade, ale distribúciu už nediskretizujeme. Namiesto toho si ju uložíme do súboru ako g spolu s číslom $\lambda = p/\mu$, kde p je pravdepodobnosť daných báz a μ je normalizačná konštanta – tá istá ako v predchádzajúcom prípade. Hustotu emisie potom počas zarovnávania vypočítame tak, že zoberieme parametre pre danú dvojicu báz (resp. bázu) – g a λ – a pre výstup klasifikátora c bude hustota $f(c) = \lambda g(c)$.

5.4 Použitie

Hlavný program sa spúšťa príkazom

```
PYHONPATH=. python bin/Realigner.py vstup výstup
```

s následujúcimi parametrami

- `-mathtype`: float/LogNum – označuje, či sa má počítať s logaritmovanými hodnotami alebo s reálnymi
- `-algorithm`: viterbi – v našich modeloch je podporovaný len viterbi
- `-beam_width`: číslo – šírka okolo lúča vstupného zarovnania, ktorá sa má prehľadávať vo viterbiho algoritme
- `-model`: ClassificationHMM.js/OracleHMM.js/SimpleHMM2.js/vlastný – model, ktorý sa má použiť (naše modely sa nachádzajú v data/models)
- `-sequence_regexp`: sekvenciaX sekvenciaY regulárne výrazy na vyfiltrovanie dvoch sekvencií zo vstupného súboru zarovnania
- `-annotation_model`: súbor.js – súbor s konfiguráciou anotácií pre jednotlivé sekvencie

Pozičné parametre

- `1`: vstup – súbor so vstupným zarovnaním
- `2`: výstup – súbor kam sa má uložiť výstupné zarovnanie

Napríklad:

```
PYHONPATH=. python bin/Realigner.py\
     --mathType LogNum --algorithm viterbi --beam_width 30\
     --model data/models/ClassificationHMM.js\
     --sequence_regexp ^sequence1$ ^sequence2$\
     --annotation_model annotations.js\
     input_alignment.fa output_alignment.fa
```

Predtým, ako prvýkrát spustíme zarovnávač, je nutné model najskôr natrénovať. To spravíme spustením programu

`python model_training.py  model trénovacie_dáta,`

kde prvý parameter je model, ktorý chceme trénovať a druhý je adresár s trénovacími sekvenciami. Ak ešte nemáme natrénovaný klasifikátor, tak sa natrénuje automaticky z dát ktoré sa nachádzajú v priečinku `data/sequences/train_sequences`. Napríklad

```
python model_training.py\
    data/models/OracleHMM.js data/sequences/model_training/bio,
```

Klasifikátor sa potom uloží do `data/clf` odkiaľ sa bude automaticky načítavať aj pri trénovaní modelov aj pri zarovnávaní. Pokiaľ treba klasifikátor natrénovať odznovu, je nutné ho z `data/clf` vymazať.

5.4.1 Konfigurácia a Formáty súborov

Konfigurácia našich modelov sa nachádza v dvoch súboroch:

- `classifier_alignment/constants.py` – tu sa dá zmeniť veľkosť okna a či sú povolené anotácie
- `classifier_alignment/config.py` – tu sa dá zmeniť klasifikátor, datapreparer a či sa má použiť ten istý klasifikátor aj na Match aj na Inzert stav alebo nie. V tomto súbore sú preddefinované klasifikátory a datapreparer-y takže stačí zmeniť index označujúci, ktorý z nich sa má použiť.

Súbory zarovnania sú vo formáte FASTA[3], súbory anotácií sú vo formáte BED[4]. Súbory modelov sú vo formáte JSON[5], nasledujúceho tvaru:

```
{
  "model": {
    "states": [
      {
        "name": "Match",
        "durations": [[1, 1], 1.0]],
        "startprob": 0.33,
        "endprob": 1.0,
        "emission": [],
        "onechar": "M",
        "__name__": "SimpleMatchState"
      },
      {
```

[3] `http://en.wikipedia.org/wiki/FASTA_format`
[4] `http://genome.ucsc.edu/FAQ/FAQformat.html#format1`
[5] JavaScript object notation

```
14          "name": "InsertX",
15          "durations": [[[1, 0], 1.0]],
16          "startprob": 0.33,
17          "endprob": 1.0,
18          "emission": [],
19          "onechar": "X",
20          "__name__": "SimpleIndelState"
21        },
22        {
23          "name": "InsertY",
24          "durations": [[[0, 1], 1.0]],
25          "startprob": 0.33,
26          "endprob": 1.0,
27          "emission": [],
28          "onechar": "Y",
29          "__name__": "SimpleIndelState"
30        }
31      ],
32      "__name__": "GeneralizedPairHMM",
33      "transitions": []
34    }
35  }
```

Emisné a prechodové tabuľky sa vyplnia pri trénovaní modelu. Zaujímavé sú pre nás len položky `__name__` v jednotlivých stavoch. Tam treba vložiť meno príslušnej triedy pre stav.

Súbor pre konfiguráciu anotácií je JSON nasledujúceho tvaru:

```
1  {
2    "__name__": "Annotations",
3    "annotations": ["gene"],
4    "sequences": [
5      {
6        "name": "sequence1", "annotations": [
7          {
8            "id": "gene",
9            "file": "data/sequences/simulated/
                 simulated_alignment_sequence1_gene.bed"
```

```
10            "offset": 47
11          }
12        ]
13      },
14      {
15        "name": "sequence2", "annotations": [
16          {
17            "id": "gene",
18            "file": "data/sequences/simulated/
                 simulated_alignment_sequence2_gene.bed"
19            "offset": 47
20          }
21        ]
22      },
23      {
24        "name": "sequence3", "annotations": [
25          {
26            "id": "gene",
27            "file": "data/sequences/simulated/
                 simulated_alignment_sequence3_gene.bed"
28            "offset": 47
29          }
30        ]
31      }
32    ]
33  }
```

Do `annotations` treba dať zoznam anotácií ktoré sa majú použiť a do `sequences` treba dať zoznam sekvencií – meno musí byť rovnaké ako vo FASTA súbore. Pre každú sekvenciu treba vyplniť v `annotations` pre každú anotáciu (id musí byť rovnaké ako v zozname anotácií) zdrojový súbor, odkiaľ sa má daná anotácia načítať. Navyše je možné špecifikovať `offset` v prípade, že sekvencia, ktorú chceme zarovnať, nezačína v genóme na pozícii 0 a anotácie sú číslované podľa pozícií v genóme.

5.5 Rozšírenie programu

Program je možné ľahko rozšíriť rôznymi spôsobmi. Je možné pridať:

- novú triedu pre stav
- nový DataPreparer
- nový klasifikátor

Nová trieda pre stav musí dediť od GeneralizedPairState, alebo niektorého z jej potomkov. Najdôležitejšiou metódou je emission, ktorá dostane dve pozície, dve sekvencie a dve hodnoty, ktoré označujú koľko symbolov z ktorej sekvencie treba vygenerovať, jej výstupom je pravdepodobnosť, že sa vygenerujú dané znaky z daných sekvnecií od daných pozícií. Ak chceme túto triedu použiť v modeli, musíme ju ešte zaregistrovať do triedy *HMMLoader* pomocou metódy `addFunction`.

Nový DataPreparer musí dediť od triedy DataPreparer, alebo niektorej z jej potomkov. Najdôležitejšou je metóda `prepare_data`, ktorá dostane obe sekvencie, anotácie a aj dve pozície v sekvenciách. Výstupom musí byť pole číselných atribútov. Preťažením metódy `prepare_training_data` sa dá zmeniť generovanie trénovacej množiny.

Nový klasifikátor musí implementovať nasledujúce metódy:

- `fit(X, y)` – dostane pole trénovacích vektorov a pole cieľových hodnôt, z ktorých natrénuje svoje parametre
- `predict_proba(X)` – dostane pole vstupných vektorov a výstupom je dvojrozmerné pole – pre každý vstupný vektor a každú triedu pravdepodobnosť, že vektor patrí do danej triedy

Navyše musí byť serializovateľný pomocou modulu *pickle*.

Záver

Literatúra

[BC] Leo Breiman and Adele Cutler. Random forests. [Online; accessed 14-Jan-2013].

[BPSS11] Alvis Brazma, Helen Parkinson, Thomas Schlitt, and Mohammadreza Shojatalab. EBI Research - Functional Genomics - Introduction To Biology. 2011. [Online; accessed 26-Oct-2012].

[Bre01] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[BV11] Broňa Brejová and Tomáš Vinař. *Metódy v bioinformatike [Methods in Bioinformatics]*. Knižničné a edičné centrum FMFI UK, 2011. Lecture notes.

[com13] SciPy community. *SciPy Reference Guid*, 2013. `http://docs.scipy.org/doc/scipy/scipy-ref.pdf`.

[DEKM98] R. Durbin, S.R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[DGB06] Chuong Do, Samuel Gross, and Serafim Batzoglou. Contralign: Discriminative training for protein sequence alignment. In Alberto Apostolico, Concettina Guerra, Sorin Istrail, Pavel Pevzner, and Michael Waterman, editors, *Research in Computational Molecular Biology*, volume 3909 of *Lecture Notes in Computer Science*, pages 160–174. Springer Berlin / Heidelberg, 2006. 10.1007/11732990_15.

[Edg04] Robert C Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, 32(5):1792–1797, 2004.

[GS96] D. Gusfield and P. Stelling. [28] parametric and inverse-parametric sequence alignment with xparal. *Methods in enzymology*, 266:481–494, 1996.

[HHL89] XD Huang, Hsiao-Wuen Hon, and Kai-Fu Lee. Multiple codebook semi-continuous hidden markov models for speaker-independent continuous speech recognition. 1989.

[Hir75] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, June 1975.

[KA90] S Karlin and S F Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87(6):2264–2268, 1990.

[MB06] Alexander Yu. Mitrophanov and Mark Borodovsky. Statistical significance in biological sequence analysis. *Briefings in Bioinformatics*, pages 2–24, 2006.

[NJ02] Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 2:841–848, 2002.

[NVB13] Michal Nánási, Tomáš Vinař, and Broňa Brejová. Probabilistic approaches to alignment with tandem repeats. In *Algorithms in Bioinformatics*, pages 287–299. Springer, 2013.

[NW70] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970.

[Sri] Sargur N. Srihari. Machine Learning: Generative and Discriminative Models.
`http://www.cedar.buffalo.edu/~srihari/CSE574/Discriminative-Generative.pdf`. [Online; accessed 14-Jan-2013].

[Sut07] Ivan Sutóris. Tvorba klasifikačných stromov pri procese data miningu, 2007.

[SW81] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[Wik14a] Wikipedia. Kernel density estimation. `http://en.wikipedia.org/wiki/Kernel_density_estimation`, 2014. [Online; accessed 1-May-2014].

[Wik14b] Wikipedia. Maximum likelihood. `http://en.wikipedia.org/wiki/Maximum_likelihood`, 2014. [Online; accessed 17-Apr-2014].

[Wik14c] Wikipedia. Maximum likelihood. `http://en.wikipedia.org/wiki/ID3_algorithm`, 2014. [Online; accessed 26-Apr-2014].

[Wik14d] Wikipedia. Random forest. `http://en.wikipedia.org/wiki/Random_forest`, 2014. [Online; accessed 26-Apr-2014].

[YJEP07] Chun-Nam Yu, Thorsten Joachims, Ron Elber, and Jaroslaw Pillardy. Support vector training of protein alignment models. In Terry Speed and Haiyan Huang, editors, *Research in Computational Molecular Biology*, volume 4453 of *Lecture Notes in Computer Science*, pages 253–267. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-71681-5_18.