Algoritmos eficientes para análise de campos aleatórios condicionais semi-markovianos e sua aplicação em sequências genômicas

Ígor Bonadio

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Ciência da Computação
Orientador: Prof. Dr. Alan Mitchell Durham

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, abril de 2018

Algoritmos eficientes para análise de campos aleatórios condicionais semi-markovianos e sua aplicação em sequências genômicas

Esta versão da dissertação/tese contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 06/08/2018. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Alan Mitchell Durham (orientador) - IME-USP
- Prof. Dr. Ronaldo Fumio Hashimoto - IME-USP
- Prof. Dr. Georgios Joannis Pappas Junior - UnB
- Prof. Dr. André Yoshiaki Kashiwabara - UTFPR
- Prof. Dr. David Corrêa Martins Júnior - UFABC

Agradecimentos

Agradeço a minha esposa, Jessica Eto, e a toda minha família o apoio que recebi durante o desenvolvimento desse trabalho. Agradeço ao Prof. Dr. Alan Mitchell Durham a orientação e a oportunidade de me aprofundar em Bioinformática. Agradeço aos meus amigos Renato Cordeiro Ferreira e Mauro Medeiros as diversas discussões que tivemos e a parceria que formamos. Agradeço a todos os membros do nosso grupo de pesquisa: Aline Rodrigheri Ioste, Almir José Ferreira, Bruno Tenório, Igor Fratel, Pedro Nachtigall e Rodrigo Bossini. Agradeço também André Barbosa, Cinthia Marie Tanaka, Felipe Toledo Farias, Henrique Przibisczki Oliveira e Rafael Ballet, membros do meu time no Elo7, a compreensão e ajuda durante a reta final deste trabalho.

Resumo

Bonadio, I. **Algoritmos eficientes para análise de campos aleatórios condicionais semi-markovianos e sua aplicação em sequências genômicas**.

Campos Aleatórios Condicionais são modelos probabilísticos discriminativos que tem sido utilizados com sucesso em diversas áreas como processamento de linguagem natural, reconhecimento de fala e bioinformática. Entretanto, implementar algoritmos eficientes para esse tipo de modelo não é uma tarefa fácil. Nesse trabalho apresentamos um arcabouço que ajuda no desenvolvimento e experimentação de Campos Aleatórios Condicionais Semi Markovianos (semi-CRFs). Desenvolvemos algoritmos eficientes que foram implementados em C++ propondo uma interface de programação flexível e intuitiva que habilita o usuário a definir, treinar e avaliar modelos. Nossa implementação foi construída como uma extensão do arcabouço ToPS que, inclusive, pode utilizar qualquer modelo já definido no ToPS como uma função de característica especializada. Por fim utilizamos nossa implementação de semi-CRF para construir um preditor de promotores que apresentou performance superior aos preditores existentes.

Palavras-chave: campos aleatórios condicionais, predição de genes, predição de promotores.

Abstract

Bonadio, I. **Efficient algorithms for semi-markov conditional random fields and their application for the analysis of genomic sequences**.

Conditional Random Fields are discriminative probabilistic models that have been successfully used in several areas like natural language processing, speech recognition and bioinformatics. However, implementing efficient algorithms for this kind of model is not an easy task. In this thesis we show a framework that helps the development and experimentation of Semi-Markov Conditional Random Fields (semi-CRFs). It has an efficient implementation in C++ and an intuitive API that allow users to define, train and evaluate models. It was built as an extension of ToPS framework and can use ToPS' probabilistic models as specialized feature functions. We also use our implementation of semi-CRFs to build a high performance promoter predictor.

Keywords: conditional random fields, gene predition, promoter prediction.

Sumário

Lista de Abreviaturas

CRF	Campo Aleatório Condicional (*Conditional Random Field*)
LCCRF	Campo Aleatório Condicional de Cadeias Lineares (*Linear Chain Conditional Random Field*)
semi-CRF	Campo Aleatório Condicional Semi-Markoviano (*Semi-Markov Conditional Random Field*)
UTR	Região não traduzida (*Untranslated Region*)
TSS	Sítio de início de transcrição (*Transcription Start Site*)
ToPS	*Toolkit for Probabilistic Models of Sequences*
GHMM	Modelo de markov oculto generalizado (*Generalized Hidden Markov Model*)
HMM	Modelo de markov oculto (*Hidden Markov Model*)

Lista de Figuras

Capítulo 1

Introdução

Determinar a melhor rotulação para uma dada sequência de observações é uma tarefa comum em diversas áreas: Em bioinformática, a tarefa conhecida como predição de genes (Kashiwabara *et al.*, 2013) consiste em, a partir de uma sequência de DNA, determinar a localização dos genes e suas estruturas; Em processamento de linguagem natual, determinar se uma palavra em um texto é uma entidade (como um nome de uma pessoa, empresa ou lugar) é uma tarefa conhecida como Reconhecimento de Entidades (NER, do inglês *Named-entity Recognition*) (Habibi *et al.*, 2017); Em reconhecimento de fala, uma tarefa comum é segmentar um sinal de fala e associá-lo a um texto (Chan *et al.*, 2016); E em visão computacional uma de suas tarefas consiste em determinar a localização de objetos em uma imagem ou video (Long *et al.*, 2015).

Tradicionalmente, um modelo bastante conhecido e utilizado ao se abordar esse tipo de problema é o modelo oculto de Markov (HMM, do inglês *Hidden Markov Model*) (Jose *et al.* , 2016; Kupiec, 1992; Rabiner, 1989), bem como suas variações conhecidas como modelo oculto semi-Markoviano (HSMM, do inglês *hidden semi-Markov model*) (Yu e Kobayashi , 2003) e modelo oculto de Markov generalizado (GHMM, do inglês *Generalized Hidden Markov Model*) (Stanke *et al.*, 2006). Esse tipo de modelo possui algumas limitações como, por exemplo, a necessidade de se modelar a distribuição das observações embora o objetivo real seja determinar os rótulos (Sutton, 2008). Além disso, se algum tipo de rótulo puder ser associado a muitos tipos de observações diferentes, então esse rótulo será penalizado e sua chance de pertencer a rotulação ótima será baixa (Sutton e Mccallum, 2002).

Visando contornar os problemas descritos acima, foram introduzidos em 2001, os campos aleatórios condicionais (CRF, do inglês *Conditional Random Fields*) (Lafferty *et al.*, 2001) e deste então vêm sendo utilizados em diversas áreas com bastante sucesso (Bernal *et al.*, 2012; DeCaprio *et al.*, 2007; Kudo *et al.*, 2004; Morales-Cordovilla *et al.*, 2018; Sha e Pereira, 2003; Vinson *et al.*, 2007; Yang e Cardie, 2012; Yao *et al.*, 2014). A principal característica desses modelos é que representamos diretamente a probabilidade condicional $p(\text{rótulos}|\text{observações})$ e consequentemente não há a necessidade de se modelar explicitamente a distribuição das observações. Além disso, a adição de conhecimento do domínio é feita a partir de funções de características que podem ser definidas de forma arbitrária, mas que frequentemente são funções indicadoras do tipo $1_{\{cond\}}$, cujo valor é 1 se a condição *cond* for verdadeira ou 0 caso

contrário. Em Reconhecimento de Entidades, por exemplo, podemos adicionar uma função $1_{\{\text{palavra começa com letra maiúsica}\}}1_{\{\text{rótulo=entidade}\}}$ para indicar que normalmente os nomes de entidades começam com letra maiúscula.

De modo geral, os algoritmos de CRF executam em tempo exponencial. Entretanto algumas variantes como CRF de cadeias lineares (LCCRF, do inglês *Linear Chain Conditional Random Field*) e CRF Semi-Markoviano (Semi-CRF, do inglês *Semi-Markov Conditional Random Field*) possuem algoritmos polinomiais. Em bioinformática, mais especificamente em predição de genes, área que abordamos neste trabalho, LCCRFs e Semi-CRFs apresentaram bons resultados iniciais (Culotta e Mccallum, 2001; DeCaprio *et al.* , 2007; Gross e Brent, 2006). Mas, mesmo sendo polinomiais, os algoritmos desses CRFs mostraram-se proibitivos para a análise de sequências de organismos mais complexos como *H. sapiens*.

Neste trabalho focamos em melhorar os algoritmos de LCCRF e semi-CRF. Para isso desenvolvemos uma nova implementação de CRFs como uma extensão do arcabouço ToPS (*Toolkit for Probabilistic Models of Sequences*) (Kashiwabara *et al.*, 2013). ToPS é um arcabouço que tem como objetivo facilitar a integração e composição de modelos probabilísticos, que antes de nossa adição, já contava com 8 modelos diferentes. Nossa implementação de CRF se aproveita dos múltiplos núcleos disponíveis nos computadores atuais para diminuir o tempo de execução dos algoritmos, bem como identifica automaticamente pontos de otimização, como o grau de conectividade entre rótulos e o tamanho máximo do contexto necessário para se computar a probabilidade em uma dada posição da sequência.

Outro problema é que a modelagem de problemas utilizando CRFs não é trivial. A utilização das tradicionais funções de características para definir a relação entre rótulos e/ou observações pode ser difícil principalmente se o número de rótulos e observações for grande. Propomos aqui, também, uma linguagem gráfica para descrição de CRFs que facilitará não apenas o entendimento do modelo mas também servirá como base para nossa implementação de CRFs.

Por fim, utilizamos nossa implementação de CRF para explorar a área de predição de genes. Atualmente os programas de predição de gene focam em identificar as regiões codificantes de proteínas e desconsideram as regiões não traduzidas (UTRs, do inglês *Untranslated Regions*) (Burge, 1997; Stanke e Waack, 2003; Stanke *et al.*, 2006). Em particular, identificar corretamente a região UTR localizada antes da região codificante, chamada de 5'UTR, é importante pois auxilia no estudo e caracterização de promotores (Abeel *et al.*, 2008a,b), principalmente da região conhecida como *core*. Para tanto é necessário localizar o sítio de início de transcrição (TSS, do inglês *Transcription Start Site*), que é uma tarefa difícil. Diversas técnicas foram revisadas por Bajic *et al.* (2004) que concluíram que o número de falsos positivos tornava proibitivo o uso dessas abordagens.

Finalmente, desenvolvemos um preditor de TSS chamado TSSFinder basedo em nossa implementação de CRF. Nosso preditor busca, a partir de um códon de iniciação previamente anotado, por um TSS e, se houver, um TATA-box. Nosso preditor apresenta ótimos resultados e supera os preditores existentes.

1.1 Organização do Texto

No capítulo 2 comparamos dois tipos de modelos probabilísticos: os geradores e os disciminativos. Após isso apresentamos os modelos HMM, GHMM, CRF gerais, LCCRF e semi-CRF, que são os modelos mais utilizados em rotulação de sequências.

No capítulo 3 definimos alguns conceitos biológicos e as principais abordagens utilizadas na predição de genes e de promotores. Revisamos também algumas ferramentas que utilizam CRF para realizar predição de genes genes.

No capítulo 4 descrevemos a nossa linguagem gráfica de especificação de CRFs utilizando máquinas de estados e como ela foi usada para desenvolvermos uma implementação eficiente de LCCRF e semi-CRF. Apresentamos também o padrão secretário, desenvolvido durante esse trabalho, e como ele foi aplicado em nossa modelagem orientada a objetos. Por fim comparamos nossa implementação com outras já existentes.

No capítulo 5 descrevemos nosso preditor de TSS chamado TSSFinder. Apresentamos o processo de validação utilizado bem como os resultados e uma compração com os outras ferramentas disponíveis.

Finalizamos este trabalho com o capítulo 6, apresentando nossas conclusões e considerações futuras.

Capítulo 2

Modelos Probabilísticos para Rotulação de Sequências

Tradicionalmente modelos geradores tais como HMM e GHMM são bastante utilizados para rotulação de sequências (Burge, 1997; Kashiwabara *et al.*, 2013; Stanke e Waack, 2003; Stanke *et al.*, 2006). Esses modelos representam a distribuição de probabilidade conjunta $p(y, x)$, sendo x a sequência de dados que observamos e y a sequência de rótulos atribuídos à cada elemento de x que desejamos predizer. Como os conjuntos das observações e rótulos são independentes, podemos, utilizando a regra de Bayes, calcular a probabilidade *a posteriori* e realizar a classificação:

$$p(y|x) = \frac{p(x \cap y)}{p(x)} = \frac{p(x|y)p(y)}{p(x)} = \frac{p(x, y)}{p(x)} \tag{2.1}$$

Estimar a verossimilhança, $p(x|y)$, pode ser obtida através de uma contagem dos símbolos do conjunto de treinamento. Uma abordagem similar pode ser feita para estimar a probabilidade *a priori*, $p(y)$, ou até mesmo utilizar o conhecimento sobre o problema em questão para estimá-la. Devido a isso, esta classe de modelos precisa enumerar todas as possíveis sequências de observações (Sutton e Mccallum, 2002). Este problema, em muitos domínios, é considerado intratável a não ser que algumas suposições de independência entre observações sejam assumidas.

Além disso, note que modelos geradores definem indiretamente a probabilidade $p(y|x)$, necessária para a classificação. Uma alternativa são os modelos discriminativos, que modelam diretamente essa probabilidade condicional.

Campos Aleatórios Condicionais (CRF, do inglês *Conditional Random Fields*), introduzidos por Lafferty *et al.* (2001), são exemplos de modelos discriminativos. Neste capítulo compararemos a abordagem discriminativa com a geradora, apresentaremos os modelos HMM e GHMM e apresentaremos, como alternativa discriminativa, a definição de CRFs gerais e de duas variantes cujas estruturas são limitadas: CRFs de cadeias lineares e semi-Marcovianos.

2.1 Modelos Geradores e Modelos Discriminativos

Para entendermos a diferença entre modelos geradores e modelos dicriminarivos, apresentaremos um problema simples que consiste em classificar um dado objeto a uma classe. Uma forma de abordar este problema é utilizando classificadores probabilísticos.

Duda *et al.* (2000) apresentam um exemplo que consiste em classificar peixes entre as categorias salmão e robalo. Uma possível forma de classificação é simplesmente a categoria com maior probabilidade *a priori* $P(c)$, onde $c \in \{salmo, robalo\}$. Esta probabilidade indica um conhecimento prévio de que, por exemplo, existem mais salmões do que robalos no mar (ou seja $P(\text{salmão}) > P(\text{robalo})$) e, portanto classificar todos os peixes como salmão irá acarretar uma classificação com menor taxa de erro do que classificar todos como robalo. Esta abordagem poderia até ser suficiente se estamos desejando classificar um único peixe que tem uma abundancia muito maior do que as outras, já que, se aplicarmos o mesmo procedimento em vários peixes, todos serão classificados como pertencentes a mesma classe.

Entretanto, é possível extrair algumas características dos peixes e utilizá-las para diferenciá-los. Essas características podem ser o tamanho e a intensidade do brilho das escamas, por exemplo. A partir disso, podemos calcular $P(c|x, y)$, onde x representa o tamanho, y o brilho e $c \in \{$ salmão, robalo $\}$ e decidir por *salmão* se $P(\text{salmão}|x, y) > P(\text{robalo}|x, y)$, caso contrário decidir por *robalo*. Esta regra de decisão é conhecida como regra decisão bayesiana.

O problema, então, passa a ser como estimar essas distribuições de probabilidade. Existem basicamente duas abordagens: geradora, que estima a probabilidade $P(c|x, y)$ indiretamente; e discriminativa, que estima a probabilidade $P(c|x, y)$ diretamente.

Com o objetivo de ressaltar as diferenças entre ambas as abordagens, apresentaremos a seguir dois modelos probabilísticos: Naïve Bayes, que é um modelo gerador simples; e Regressão Losgística, que pode ser vista como uma versão discriminativa das Naïve Bayes (Sutton, 2008).

2.1.1 Naïve Bayes

Naïve Bayes é um modelo gerador simples que assume que as características que definem um objeto são independentes.

Seja $C = \{c_1, c_2, ..., c_n\}$ um conjunto de n classes e $x = \{x_1, x_2, x_m\}$ um conjunto de m características que descreve um objeto. O objetivo de uma Naïve Bayes é, dado um conjunto de características x, encontrar encontrar a classe mais provável, ou seja, encontrar k^* tal que

$$k^* = \arg\max_k P(c_k|x) \tag{2.2}$$

Sabemos, pela regra de Bayes, que

$$P(c_k|x) = \frac{P(x|c_k)P(c_k)}{P(x)} \tag{2.3}$$

e

$$\begin{aligned} P(x|c_k)P(c_k) &= P(c_k)P(x_1, x_2, ..., x_m|c_k) \\ &= P(c_k)P(x_1|c_k)P(x_2, ..., x_m|c_k, x_1) \\ &= P(c_k)P(x_1|c_k)P(x_2|c_k, x_1)...P(x_m|c_k, x_1, x_2, ..., x_{m-1}) \end{aligned} \tag{2.4}$$

Sendo uma Naïve Bayes, assumiremos que as características de um objeto são independentes. Então

$$\begin{aligned} P(x_1, x_2, ..., x_m|c_k) &= P(x_1|c_k) \\ P(x_2, ..., x_m|c_k, x_1) &= P(x_2|c_k) \\ &... \\ P(x_m|c_k, x_1, x_2, ..., x_{m-1}) &= P(x_m|c_k) \end{aligned} \tag{2.5}$$

E por fim, temos que

$$P(c_k|x) = P(c_k)\prod_{i=1}^{m}\frac{P(x_i|c_k)}{P(x)} \tag{2.6}$$

2.1.2 Regressão Logística

Regressão logística é um modelo discriminativo bastante relacionado com Naïve Bayes por sua simplicidade (Sutton, 2008).

É definida a partir da intuição de se escrever uma função linear que se comporte como uma distribuição de probabilidade, ou seja, que tenha valores entre 0 e 1. Um função que se encaixa nesta descrição é a função logística:

$$\log\frac{p(x)}{1 - P(x)} = \beta_0 + x\beta \tag{2.7}$$

que nos leva a

$$P(x) = \frac{\exp\{\beta_0 + x\beta\}}{1 + \exp\{\beta_0 + x\beta\}} \tag{2.8}$$

Esta idéia pode ser aplicada ao problema de classificação. Seja $C = \{c_1, c_2, ..., c_n\}$ um conjunto de n classes e $x = \{x_1, x_2, x_m\}$ um conjunto de m características que descreve um objeto. O objetivo é, novamente, encontrar um k^* tal que

$$k^* = \arg\max_k P(c_k|x) \tag{2.9}$$

Neste caso, quando $n \geq 2$, temos n parametros $\beta_0^{(0)}$, $\beta^{(0)}$, $\beta_0^{(1)}$, $\beta^{(1)}$, ..., $\beta_0^{(n)}$, $\beta^{(n)}$ e a

probabilidade condicional é descrita como

$$P(c_k|x) = \frac{\exp\{\beta_0^{(c_k)} + x\beta^{(c_k)}\}}{\sum_{c \in C} \exp\{\beta_0^{(c)} + x\beta^{(c)}\}} \tag{2.10}$$

Note que, para o caso de somente 2 classe, a equação 2.10 é transformada na equação 2.8 para $\beta_0 = \beta_0^{(1)} - \beta_0^{(0)}$ e $\beta = \beta^{(1)} - \beta^{(0)}$.

2.2 Cadeias de Markov

CRFs são comumente comparados com HMMs e GHMMs, que são modelos geradores. Nessa seção apresentaremos esses dois modelos.

Se um processo Markoviano possui estados discretos, este é chamado de Cadeia de Markov, e pode ser definido para tempos discretos ou contínuos, e para conjuntos de estados finitos ou infinitos (Kulp *et al.*, 1996). Durante essa seção abordaremos somente as Cadeias de Markov de tempo discreto para um conjunto de estados finito, já que o objetivo deste trabalho é a segmentação de sequências de símbolos pertencentes à um alfabeto discreto.

De modo geral, para qualquer modelo probabilístico podemos escrever a probabilidade de uma sequência x de comprimento L como:

$$\begin{aligned} P(x) &= P(x_L, x_{L-1}, ...x_1) \\ &= P(x_L|x_{L-1}, ...x_1)P(x_{L-1}|x_{L-2}, ...x_1)...P(x_1) \end{aligned} \tag{2.11}$$

Se esse modelo é uma cadeia de Markov, podemos reescrever a equação (2.11) como

$$P(x) = P(x_L|x_{L-1})P(x_{L-1}|x_{L-2})...P(x_2|x_1)P(x_1) \tag{2.12}$$

Note que comportamento probabilístico do presente depende somente de seu passado imediato.

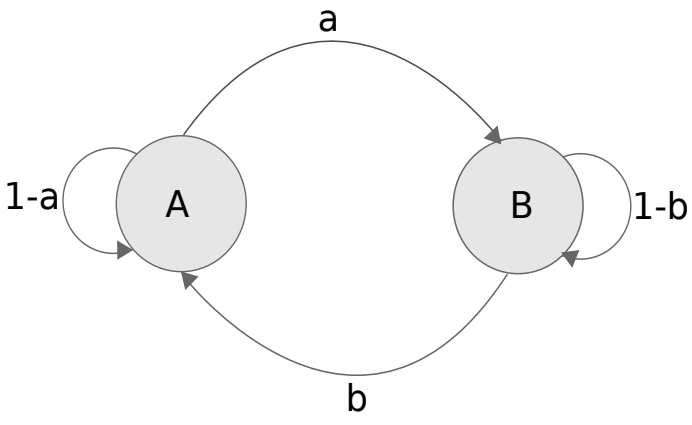

Figura 2.1: *Grafo direcionado representando uma Cadeia de Markov*

Uma Cadeia de Markov pode ser representada graficamente por um grafo direcionado cujos vértices representam os *estados* que são associados à um símbolo, sendo que cada um desse estados pode ser conectado à um outro qualquer. Como exemplo temos a figura 2.1, que representa um Cadeia de Markov simples que possui apenas dois estados A e B.

Cada aresta desse grafo tem uma probabilidade de transição associada, que também pode ser descrita pela equação abaixo:

$$a_{ij} = P(x_t = i | x_{t-1} = j) \tag{2.13}$$

onde i e j são dois estados da cadeia de Markov. Ou seja, podemos reescrever a equação (2.12) como:

$$P(x) = P(x_1) \prod_{i=2}^{L} a_{x_{i-1} x_i} \tag{2.14}$$

2.2.1 Modelo Oculto de Markov

Modelo Oculto de Markov é um tipo de modelo probabilístico gerador bastante popular em segmentação de sequências. Seguindo a notação de Rabiner (1989), podemos definir um HMM como uma quíntupla (S, V, A, B, D), onde:

* $S = \{s_1, s_2, ..., s_N\}$, o conjunto de estados ocultos
* $V = \{v_1, v_2, ..., v_M\}$, o conjunto de símbolos visíveis
* $A = \{a_{ij}\}$, o conjunto que define as probabilidades de transição de um estado s_i para um estado s_j para todo $1 \leq i, j \leq N$. A equação (2.15) define essas probabilidades, sendo $y = (y_1, y_2, ..., y_L)$ a sequência de rótulos cujo comprimento é L e t o instante atual.

$$a_{ij} = P(y_t = s_j | y_{t-1} = s_i), 1 \leq i, j \leq N \tag{2.15}$$

* $B = \{b_j(k)\}$, o conjunto que define as probabilidades de emissão do símbolo $v_k \in V$ pelo estado $s_j \in S$. A equação (2.16) define essas probabilidades, sendo $y = (y_1, y_2, ..., y_L)$ a sequência de rótulos, $x = (x_1, x_2, ..., x_L)$ a sequência de símbolos observados, ambas de comprimento L, e t o instante atual.

$$b_j(k) = P(x_t = v_k | x_t = s_j), 1 \leq j \leq N, 1 \leq k \leq M \tag{2.16}$$

* Um conjunto de probabilidades iniciais $D = \{d_i\}$. A equação (2.17) define essas probabilidade, sendo y_1 o primeiro rótulo.

$$d_i = P(y_1 = s_i), 1 \leq i, j \leq N \tag{2.17}$$

Note que as equações (2.18) e (2.19) devem ser satisfeitas.

$$\sum_{j=1}^{N} a_{ij} = 1, 1 \leq i \leq N \tag{2.18}$$

$$\sum_{k=1}^{M} b_j(k) = 1, 1 \leq j \leq N \tag{2.19}$$

Alem disso podemos definir a probabilidade conjunta de uma sequência de rótulos y e de uma sequência de símbolos visíveis x ambas de tamanho L como:

$$P(y, x) = d_{y_1} P(x_1|y_1) \prod_{t=2}^{T} P(y_t|y_{t-1}) P(x_t|y_t) \tag{2.20}$$

Uma representação possível deste modelo é a apresentada na figura 2.2, que modela o problema do cassino desonesto. Este problema consiste de um cassino que alterna, de acordo com as probabilidades definidas pelas arestas na figura 2.2, a utilização dois tipos de dados: Um honesto, onde todas as faces têm a mesma probabilidade de ocorrer em um lançamento, e um desonesto, cuja face número 1 tem maior probabilidade de acontecer do que as outras.

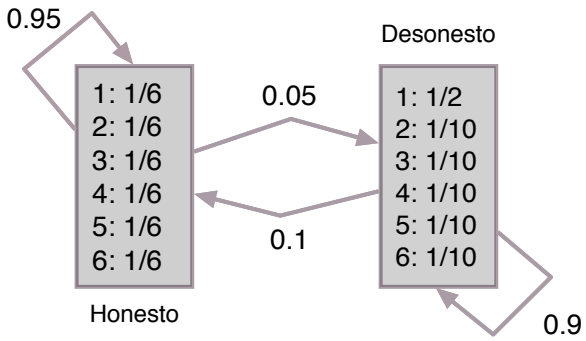

Figura 2.2: *Exemplo de HMM para o problema do Cassino Desonesto (?)*

2.2.2 Modelo Oculto Generalizado de Markov

Os HMMs nos oferecem a possibilidade de associarmos uma distribuição de probabilidade a cada estado fazendo com que este possa emitir observações. Porém, a duração de um segmento modelado por um HMM é sempre geométrica. O Modelo Oculto Generalizado de Markov (GHMM, em inglês *Generalized Hidden Markov Model*) tem como proposta permitir a modelagem explícita da duração, bem como definir um modelo probabilístico qualquer para as possíveis obervações (Kulp *et al.*, 1996). Isso faz com que o GHMM seja um modelo semi-markoviano, pois a propriedade markoviana vale entre segmentos, mas não entre cada um dos símbolos da sequência de rótulos.

A definição formal de um GHMM é dada pela quíntupla (X, Y, a, D, B), onde

* X é o conjunto de estados/rótulos do GHMM;
* Y é o conjunto de símbolos observáveis;
* a é a função de probabilidade de transição entre os estados, sendo que $\sum_{i \in Y} a_{i,j} = 1$, para todo $j \in Y$;
* $D = \{d_j(l)\}$ para todo $j \in Y$ é um conjunto de distribuições de probabilidade de duração, sendo que $\sum_{l \in \mathbb{N}^*} d_j(l) = 1$.
* $B = \{b_j(x)\}$ para todo $j \in Y$ é um conjunto de distribuições de probabilidade de emissão de símbolos observáveis, sendo que $\sum_{x \in X^l} b_j(x) = 1$, onde $l \in \mathbb{N}^*$ e X^l é o conjunto de todas as palavras de comprimento l sobre o alfabeto X.

Como podemos notar o GHMM nos proporciona uma maior flexibilidade, entretanto dificulta a inferência estatística quando compararmos com HMM, que é um modelo mais simples.

2.3 Campos Aleatórios Markovianos

As definições de campos aleatórios condicionais (CRF do ingles *Conditional Random Fields*) estão diretamente relacionadas as campos aleatórios markovianos (Hammersley e Clifford , 1971), que abordaremos nesta seção.

2.3.1 Notação

Seja $G = (V, E)$ um grafo, tal que

- $V = \{v_1, v_2, ..., v_n\}$ é um conjunto de vértices
- Cada vértice $v_i \in V$ é também uma variável aleatória
- E é o conjunto de arestas
- $(v_i, v_j) \in E$ se e somente se

$$P(v_i|\{v_1, v_2, ..., v_n\} \setminus \{v_i\}) \neq P(v_i|\{v_1, v_2, ..., v_n\} \setminus \{v_i, v_j\}) \tag{2.21}$$

e

$$P(v_j|\{v_1, v_2, ..., v_n\} \setminus \{v_j\}) \neq P(v_j|\{v_1, v_2, ..., v_n\} \setminus \{v_j, v_i\}) \tag{2.22}$$

ou seja, existe uma aresta em G se e somente se as variáveis conectadas são dependentes.

Definimos também:

- Um conjunto de vizinhos de um vértice v_i no grafo G como $N(v_i)$ (figura 2.3) tal que $v_j \in N(v_i)$ se e somente se $(v_i, v_j) \in E$.
- $C \subseteq V$ é um clique, ou seja, um subgrafo completo, (figura 2.3) se e somente se $C \subseteq \{v, N(v)\}$, $\forall v \in C$. Em outras palavras, C é um clique se e somente se ele é composto por apenas um único vértice ou se todo vértice de C é também vizinho de todos os outros vértices de C.
- O conjunto de todos os cliques de um grafo G é denotado $\mathcal{C}(G)$

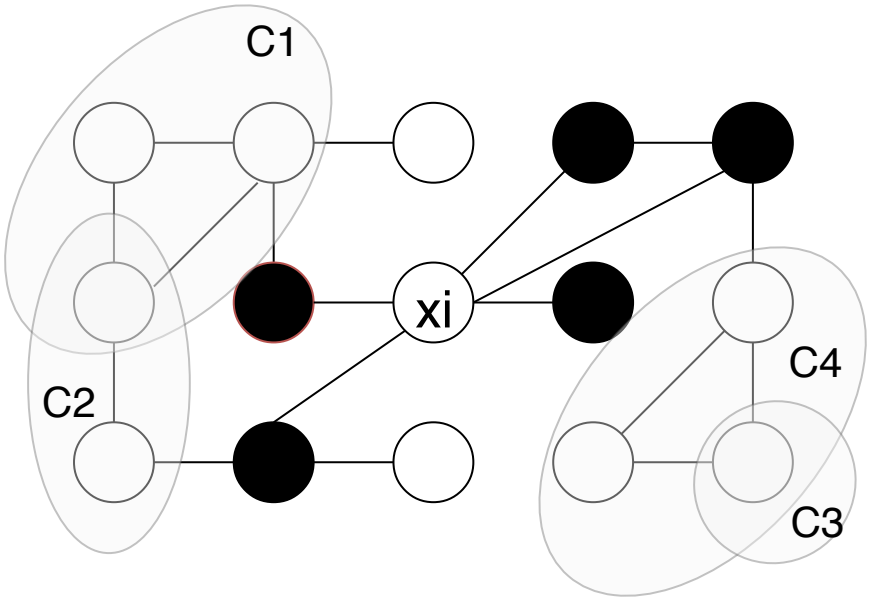

Figura 2.3: *A figura representa um grafo G, sendo que os vértices representam variáveis aleatórias e as arestas indicam a não independência entre duas variáveis aleatórias. Em preto destacamos os vértices vizinhos de x_i, ou seja $N(x_i)$. Em cinza destacamos 4 cliques de G, C_1, C_2 e C_3.*

Este grafo G define um Campo Aleatório Markoviano.

2.3.2 Teoremas

Seja $y = \{y_1, y_2, ..., y_n\}$ um evento do espaço amostral de V, podemos apresentar os principais teoremas obtidos por Hammersley e Clifford (1971):

TEOREMA 1: Propriedades Markovianas locais e globais são equivalentes.
Ou seja, uma variável aleatória apenas depende dos seus vizinhos:

$$P(y_i|y \setminus \{y_i\}) = P(y_i|N(y_i)) \tag{2.23}$$

e, para A, B e S, 3 subconjuntos disjuntos de y, tal que A e B estão separados por S

em G:

$$P(A|B,S) = P(A|S) \tag{2.24}$$

TEOREMA 2: P é Markoviana se e somente se esta distribuição de probabilidade puder ser escrita na forma de

$$P(y) = \frac{1}{Z} \prod_{C \in \mathcal{C}(G)} \phi_c(y_C) \tag{2.25}$$

onde y_C é um evento do clique C de G, ϕ_C é um função real arbitrária sobre os eventos do clique C e Z é um fator de normalização para que P seja uma distribuição de probabilidade válida, ou seja

$$Z = \sum_{y} \prod_{C \in \mathcal{C}(G)} \phi_c(y_C) \tag{2.26}$$

Note que isso implica no fato de que $P(y)$ é fatorável de acordo com os cliques do grafo G.

2.4 Campos Aleatórios Condicionais Gerais

Campos Aleatórios Condicionais (CRF, do inglês *Conditional Random Fields*) são modelos discriminativos que foram construídos utilizando as definições de campos aleatórios markovianos que vimos anteriormente. A diferença entre Campos Aleatórios e CRFs está no fato de que as variáveis aleatórias definidas pelo Campo Aleatório, que a partir de agora chamaremos de rótulos, estão condicionadas à um conjunto de observações (Lafferty *et al.*, 2001), como está apresentado na figura 2.4.

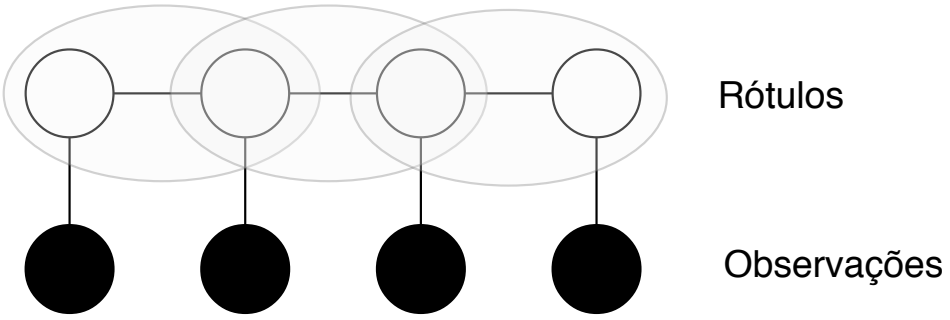

Figura 2.4: *Grafo G que define um Campo Aleatório. Em cinza destacamos os cliques de tamanho 2, e em sua interseção temos os cliques de tamanho 1.*

Tanto as observações quanto os rótulos podem ter qualquer conformação espacial, entretanto neste trabalho abordaremos o caso em que ambas são sequências de símbolos.

Seja $G = (V, E)$ um Campo Aleatório sobre as variáveis aleatórias $V = \{v_1, v_2, ..., v_n\}$. Seja também y um evento do espaço amostral de V e x uma sequência fixada de símbolos observados. Então $p(y|x)$ é um CRF se $p(y|x)$ é fatorada de acordo com G e pode ser escrita

na forma de:

$$p(y|x) = \frac{1}{Z(x)} \prod_{C \in \mathcal{C}(G)} \phi_c(y_C, x) \tag{2.27}$$

onde $Z(x)$ é um fator de normalização para que p seja uma distribuição de probabilidade válida, ou seja

$$Z(x) = \sum_{y} \prod_{C \in \mathcal{C}(G)} \phi_c(y_c, x) \tag{2.28}$$

e ϕ_C é um função real sobre os vértices do clique C e a sequência x que tem a forma de:

$$\phi_c(y_C, x) = \exp\{\sum_{k=1}^{K_C} \lambda_k f_k(y_C, x)\} \tag{2.29}$$

Cada clique C possui K_C funções reais arbitrárias f_k sobre os vértices do clique C e a sequência de observações x. Essas funções são comumente chamadas de *funções de características* e são parametrizadas por λ_k.

Por fim, podemos apresentar a forma na qual normalmente são apresentados os CRFs:

$$p(y|x) = \frac{1}{Z(x)} \exp\{\sum_{C \in \mathcal{C}(G)} \sum_{k=1}^{K_C} \lambda_k f_k(y_C, x)\} \tag{2.30}$$

e o fator de normalização $Z(x)$:

$$Z(x) = \sum_{y} \exp\{\sum_{C \in \mathcal{C}(G)} \sum_{k=1}^{K_C} \lambda_k f_k(y_C, x)\} \tag{2.31}$$

Note que estimar $p(y|x)$ é intratável no caso geral pois depende diretamente da enumeração de todos os cliques de um dado grafo. Sendo assim algumas simplificações são necessárias para que sua utilização seja possível. Neste trabalho apresentaremos duas variações de CRF que limitam os tipos de arestas possíveis em G tornando a avaliação de $p(y|x)$ possível: CRFs de Cadeias Lineares (Sutton e Mccallum, 2002) e CRFs Semi-Markovianos (Sarawagi e Cohen, 2005).

2.5 Campos Aleatórios Condicionais de Cadeias Lineares

Um a simplificação que podemos fazer com relação a estrutura de um CRF é permitir apenas cliques de tamanho 2 e 1, formando assim uma cadeia linear.

Seja G um grafo que define um Campo Aleatório sobre um conjunto de rótulos V, tal que G é uma cadeia. Seus cliques são compostos por um único vértice ou por dois vértices consecutivos nessa cadeia. Podemos definir um Campo Aleatório Condicional de Cadeias Lineares (LCCRF, do inglês *Linear-Chain Conditional Random Field*) pela distribuição de

probabilidade de uma sequência de eventos y do espaço amostral de Y dada uma sequência de observações fixa x, ambas de tamanho N, tal que:

$$p(y|x) = \frac{1}{Z(x)} \exp\{\sum_{t=1}^{N}(\sum_{e=1}^{E_t} \eta_{t,e} g_{t,e}(y_t, y_{t-1}, x) + \sum_{r=1}^{R_t} \mu_{t,r} h_{t,r}(y_t, x))\} \tag{2.32}$$

O primeiro somatório interno representa a fatoração dos cliques que contem dois vértices consecutivos. Sendo assim, temos, para cada clique t, E_t funções de características g_e ponderadas por η_e. O segundo somatório, que representa a fatoração dos cliques contendo apenas um único vértice. Neste caso, cada clique t possui R_t funções de características h_r ponderadas por μ_p.

Para simplificar as próximas definições, uniformizaremos as funções de características g e h para que tenham 3 argumentos:

$$\begin{aligned}
g_{t,k}(y_t, y_{t-1}, x) &= f_{t,k}(y_t, y_{t-1}, x) && ,1 \leq k \leq E_t \\
h_{t,k}(y_t, x) &= f_{t,k+E_t}(y_t, y_{t-1}, x) && ,1 \leq k \leq P_t \\
\eta_{t,k} &= \lambda_{t,k} && ,1 \leq k \leq E_t \\
\mu_{t,k} &= \lambda_{t,k+E_t} && ,1 \leq k \leq P_t
\end{aligned}$$

A partir dessa uniformização temos:

$$p(y|x) = \frac{1}{Z(x)} \exp\{\sum_{t=1}^{N}\sum_{k=1}^{K_t} \lambda_{t,k} f_{t,k}(y_t, y_{t-1}, x)\} \tag{2.33}$$

onde $K_t = E_t + P_t$ e o fator de normalização $Z(x)$ é:

$$Z(x) = \sum_{y} \exp\{\sum_{t=1}^{N}\sum_{k=1}^{K_t} \lambda_{t,k} f_{t,k}(y_t, y_{t-1}, x)\} \tag{2.34}$$

Outra simplificação importante na notação é uniformizar a fatoração dos cliques. Podemos notar que podemos agrupar todas as funções de características de modo que:

$$p(y|x) = \frac{1}{Z(x)} \exp\{\sum_{t=1}^{N}\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x)\} \tag{2.35}$$

onde $K = \sum_t K_t$ e que para cada posição p, f_k é igual a $f_{t,k}$ se t é igual a p. De forma análoga podemos definir os parâmetros λ_k.

2.5.1 Algoritmos de Inferência

Nesta seção apresentaremos alguns algoritmos seguindo a notação que Sutton e Mccallum (2002) utilizaram em seu trabalho. Cabe destacar que todos os algoritmos apresentados utili-

zam a técnica de programação dinâmica para que possam ser calculados em tempo eficiente, e são bastante relacionados com suas variantes para HMMs e GHMMs.

Antes de apresentar os algoritmos, vamos definir novamente um LCCRF, já que essa definição é importante para o entendimento dos mesmos. Seja um LCCRF definido como:

$$p(y|x) = \frac{1}{Z(x)} \prod_{t=1}^{N} \Psi_t(y_t, y_{t-1}, x) \tag{2.36}$$

onde

$$Z(x) = \sum_{y} \prod_{t=1}^{N} \Psi_t(y_t, y_{t-1}, x) \tag{2.37}$$

e onde cada fator é definido como a combinação de todas as funções de característica de uma dada posição:

$$\Psi_t(y_t, y_{t-1}, x) = \exp\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x)\} \tag{2.38}$$

Definimos também um conjunto L que contêm os possíveis rótulos que uma variável aleatória de uma sequência y pode assumir.

Uma das tarefas mais comuns de inferência em LCCRFs é a de se encontrar qual sequência de estados ocultos y melhor descreve a sequência de estados observados x de tamanho N. Essa tarefa pode ser executada a partir do algoritmo de *Viterbi*, que encontra $y^* = \arg\max_y p(y|x)$, também conhecido como caminho de *Viterbi*, e pode ser calculado a partir da seguinte recursão:

$$\delta_t(j) = \max_{i \in L} \Psi_t(j, i, x)\delta_{t-1}(i) \tag{2.39}$$

para todo $t \in \{1, 2, ..., |x|\}$ e $j \in L$.

O caminho mais provável pode ser obtido facilmente a partir de δ. Para isso é necessário armazenar ponteiros de onde cada valor $\delta_t(j)$ foi calculado.

Outros dois algoritmos bastante comuns em HMM são o *Forward* e o *Backward*. O algoritmo *Forward* consiste no cálculo de $\alpha_t(j) = p(x_{1...t}, y_t = j)$, onde $x_{1...t}$ é a sequência dos t primeiros símbolos de x e pode ser obtido eficientemente a partir da seguinte recursão:

$$\alpha_t(j) = \sum_{i \in L} \Psi_t(j, i, x)\alpha_{t-1}(i) \tag{2.40}$$

onde $t \in \{1, 2, ..., |x|\}$, $j \in L$ e

$$Z(x) = \sum_{y} \alpha_N(y). \tag{2.41}$$

Já o algoritmo *Backward* é definido como $\beta_t(i) = p(x_{t+1...T}, y_t = i)$, onde $x_{t+1...T}$ é

a sequência dos últimos t símbolos de x e pode ser eficientemente calculado pela seguinte recursão:

$$\beta_t(i) = \sum_{j \in L} \Psi_{t+1}(j, i, x)\beta_{t+1}(j) \tag{2.42}$$

onde $t \in \{1, 2, ..., |x|\}$, $j \in L$ e, de forma semelhante,

$$Z(x) = \sum_{y} \beta_0(y) \tag{2.43}$$

Combinando os resultados obtidos pela computação dos algoritmos *Forward* e *Backward* podemos calcular a distribuição marginal, que é utilizada durante a etapa de treinamento de LCCRFs, como definido na seguinte equação:

$$p(y_{t-1}, y_t|x) = \alpha_{t-1}(y_{t-1})\Psi_t(y_t, y_{t-1}, x)\beta_t(y_t) \tag{2.44}$$

Outra forma de inferência em campos aleatórios condicionais é a conhecida *posterior decoding* descrita em 2.45.

$$\hat{\pi_t} = \arg\max_k p(y_t = k|x) \tag{2.45}$$

onde

$$p(y_t = k|x) = \frac{\alpha_t(k)\beta_t(k)}{P(x)} \tag{2.46}$$

Observe que os algoritmos descritos são similares aos utilizados em HMMs (Durbin *et al.*, 1998). A diferença está na avaliação das funções de características que envolve um somatório em K de todas as possíveis funções de característica. Isso faz com que os algoritmos de LCCRF tenham sua complexidade aumentada em um fator K em relação aos algoritmos de HMM.

2.5.2 Treinamento

A etapa de treinamento de um LCCRF consiste em encontrar os parâmetros $\theta = \{\lambda_{t,k}\}$. Seja, então, um conjunto de treinamento $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^N$, onde $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, ...x_T^{(i)}\}$ é a sequência de estados observados de entrada e $y^{(i)} = \{y_1^{(i)}, y_2^{(i)}, ...y_T^{(i)}\}$ é a sequência de símbolos que rotulam x. Assim, podemos obter os parâmetros θ ótimos a partir da maximização do log da probabilidade condicional definida por $l(\theta)$.

$$l(\theta) = \sum_{i=1}^{N} \log p(y^{(i)}|x^{(i)}) \tag{2.47}$$

Substituindo 4.3 em 2.47 temos

$$l(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, x^{(i)}) - \sum_{i=1}^{N} \log Z(x^{(i)}) \tag{2.48}$$

Podemos melhorar $l(\theta)$ adicionando uma penalidade baseada na norma Euclidiana de θ e no parâmetro de regularização $1/2\sigma^2$, onde σ^2 é somente um parâmetro que indica a intensidade dessa penalidade. Essa adição é conhecida como regularização e é necessária para que evitemos o *overfitting*, ou seja, para que o nosso modelo tenha a capacidade de generalização e não fique preso somente aos exemplos contidos no conjunto de treinamento (Sutton e Mccallum, 2002). Sendo assim temos o log da probabilidade condicional regularizada

$$l(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, x^{(i)}) - \sum_{i=1}^{N} \log Z(x^{(i)}) - \sum_{t=1}^{T} \sum_{k=1}^{K_t} \frac{\lambda_k^2}{2\sigma^2} \tag{2.49}$$

e suas derivadas parciais definidas como

$$\frac{\partial l(\theta)}{\partial \lambda_q} = \sum_{i=1}^{N} \sum_{t=1}^{T} f_q(y_t^{(i)}, y_{t-1}^{(i)}, x^{(i)}) - \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{y,y'} f_q(y, y', x^{(i)}) p(y, y'|x^{(i)}) - \sum_{t=1}^{T} \sum_{k=1}^{K_t} \frac{\lambda_k}{\sigma^2} \tag{2.50}$$

Uma característica interessante é que $l(\theta)$ é estritamente concava, ou seja possui um único ótimo global (Lafferty *et al.*, 2001).

De modo geral $l(\theta)$ não pode ser maximizada em sua forma fechada, sendo necessário um método numérico como o BFGS (Broyden-Fletcher-Goldfarb-Shanno) que é uma aproximação do método de Newton. Vale notar que o treinamento é uma etapa bastante custosa computacionalmente, isso porque o cálculo de $Z(x)$ e da distribuição marginal $p(y_t, y_{t-1}|x)$ são obtidos através dos algoritmos *forward backward* que têm no caso geral complexidade $O(TKM^2)$, sendo K o número máximo de funções de características em um clique e M o número de estados não observáveis possíveis. Esse algoritmo é computado para cada instância do conjunto de treinamento e para cada cálculo do gradiente num total de complexidade $O(TKM^2NG)$, sendo G o número de gradientes calculados.

2.6 Campos Aleatórios Condicionais Semi-Markovianos

Os LCCRFs apresentados no capítulo anterior possuem a limitação de não conseguirem modelar a duração de um estado (ou o comprimento de um segmento). Para resolver esse problema, Sarawagi e Cohen (2005) introduziram os Campos Aleatórios Condicionais Semi-Markovianos (semi-CRF, do inglês *Semi-Markov Conditional Random Fields*).

Assim como em LCCRFs, temos um conjunto de rótulos $V = \{v_1, v_2, ..., v_N\}$. Definimos

então um conjunto de segmentos de V chamado $V_S = \{v_{s1}, v_{s2}, ..., v_{sP}\}$, onde cada segmento $v_{si} = (\ell_i, b_i, e_i)$ de forma que ℓ_i, b_i e e_i indicam, respectivamente, o rótulo, o início e o fim desse segmento. Além disso, deve-se satisfazer:

$$\begin{aligned} \ell_i \neq & \ell_{i+1} \\ b_i \leq & e_i \\ e_i + 1 = & b_{i+1} \\ b_1 = & 1 \\ e_P = & T \end{aligned} \tag{2.51}$$

Observe então que uma sequência de tamanho T é representada por uma sequência de P segmentos.

Seja, então, G um grafo que define uma Campo Aleatório sobre o conjunto de segmentos V_S, tal que G é um cadeia. Podemos definir um semi-CRF pela distribuição de probabilidade de um evento s do espaço amostral de V_S dada uma sequência de observações x de tamanho N, tal que:

$$p(s|x) = \frac{1}{Z(x)} \exp\{\sum_{t=1}^{P}\sum_{k=1}^{K_t}(\lambda_{t,k}^T f_{t,k}^T(\ell_{t-1}, s_t, x) + \lambda_{t,k}^D f_{t,k}^D(\ell_{t-1}, s_t, x) + \lambda_{t,k}^O f_{t,k}^O(\ell_{t-1}, s_t, x))\} \tag{2.52}$$

onde $Z(x)$ é uma função de normalização para que $P(s|x)$ seja uma distribuição de probabilidade válida:

$$Z(x) = \sum_s \exp\{\sum_{t=1}^{P}\sum_{k=1}^{K_t}(\lambda_{t,k}^T f_{t,k}^T(\ell_{t-1}, s_t, x) + \lambda_{t,k}^D f_{t,k}^D(\ell_{t-1}, s_t, x) + \lambda_{t,k}^O f_{t,k}^O(\ell_{t-1}, s_t, x))\} \tag{2.53}$$

e que para cada t temos K_t funções de característica $f_{t,k}^T$, $f_{t,k}^D$ e $f_{t,k}^O$ que modelam transições, durações e observações, e que seus respectivos pesos são $\lambda_{t,e}^T$, $\lambda_{t,e}^D$ e $\lambda_{t,e}^O$.

Assim como fizemos com LCCRFs, podemos uniformizar as funções de características, então:

$$p(s|x) = \frac{1}{Z(x)} \exp\{\sum_{t=1}^{P}\sum_{k=1}^{K}(\lambda_k^T f_k^T(\ell_{t-1}, s_t, x) + \lambda_k^D f_k^D(\ell_{t-1}, s_t, x) + \lambda_k^O f_k^O(\ell_{t-1}, s_t, x))\} \tag{2.54}$$

e

$$Z(x) = \sum_{s} \exp\{\sum_{t=1}^{P}\sum_{k=1}^{K}(\lambda_k^T f_k^T(\ell_{t-1}, s_t, x) + \lambda_k^D f_k^D(\ell_{t-1}, s_t, x) + \lambda_k^O f_k^O(\ell_{t-1}, s_t, x))\} \tag{2.55}$$

onde $K = \sum_t K_t$ e que para cada posição p, f_k^X é igual a $f_{t,k}^X$, $X \in \{T, D, O\}$, se t é igual a p. De forma análoga podemos definir os parâmetros λ_k.

Por fim, podemos notar que, sendo s uma sequência de segmentos de um y, onde y é um evento do espaço amostral de V, $P(s|x) = P(y|x)$.

2.6.1 Algoritmos de Inferência

Os algoritmos de inferência de semi-CRFs são bem parecidos com os de LCCRFs. A diferença está na análise das durações dos segmentos de cada rótulo.

Por conveniência, seja

$$\Psi(\ell_{t-1}, s_t, x) = \exp\{\sum_{e=1}^{K}(\lambda_k^T f_k^T(\ell_{t-1}, s_t, x) + \lambda_k^D f_k^D(\ell_{t-1}, s_t, x) + \lambda_k^O f_k^O(\ell_{t-1}, s_t, x)\}) \tag{2.56}$$

Então, a partir de 2.56 e 2.54 temos

$$p(s|x) = \frac{1}{Z(x)} \prod_{t=1}^{P} \Psi(\ell_{t-1}, s_t, x) \tag{2.57}$$

e

$$Z(x) = \sum_{s} \prod_{t=1}^{P} \Psi(\ell_{t-1}, s_t, x) \tag{2.58}$$

Definimos também um conjunto L que contêm os possíveis rótulos que uma variável aleatória de uma sequência y pode assumir.

Podemos encontrar a sequência de rótulos y que melhor descreve a sequência de símbolos observados x, $y^* = \arg\max_y p(y|x)$, utilizando o algoritmo de *Viterbi*:

$$\delta_t(j) = \max_{i \in L, d \in D} \Psi(i, (j, t-d+1, t), x)\delta_{t-d}(i) \tag{2.59}$$

Podemos calcular também o valor de $Z(x)$ utilizando o algoritmo *Forward*:

$$\alpha_t(j) = \sum_{d \in D}\sum_{i \in L} \Psi(i, (j, t-d+1, t), x)\alpha_{t-d}(i) \tag{2.60}$$

onde

$$Z(x) = \sum_{i \in L} \alpha_T(i). \tag{2.61}$$

E de forma análoga, temos o algoritmo *Backward*:

$$\beta_t(i) = \sum_{d \in D} \sum_{j \in L} \Psi_{t+d-1}(i, (j, t, t+d-1), x) \beta_{t+d}(j) \tag{2.62}$$

onde

$$Z(x) = \sum_{i \in L} \beta_0(i) \tag{2.63}$$

Assim como fizemos um paralelo entre os algoritmos de LCCRFs e HMMs, os algoritmos de semi-CRFs são similares aos utilizados em GHMMs. Novamente a diferença está na avaliação das K funções de características para cada clique do campo aleatório. Isso faz com que os algoritmos de semi-CRFs tenham sua complexidade aumentada em um fator K em relação aos algoritmos de GHMM.

2.6.2 Treinamento

O algoritmo de treinamento que apresentaremos para semi-CRFs é similar ao já apresentado para LCCRFs. O objetivo é encontrar um conjunto de paretros $\theta = \{\lambda^X_{t,k}\}$, onde $X \in \{T, D, O\}$. Seja, então, um conjunto de treinamento $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^N$, onde $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, ...x_T^{(i)}\}$ é a sequência de estados observados de entrada e $y^{(i)} = \{y_1^{(i)}, y_2^{(i)}, ...y_T^{(i)}\}$ é a sequência de símbolos que rotulam x. Temos então um conjunto S que contêm os segmentos $s^{(i)}$ de $x^{(i)}$ por $y^{(i)}$.

Assim, podemos obter os parâmetros θ ótimos a partir da maximização do log da probabilidade condicional definida por $l(\theta)$.

$$l(\theta) = \sum_{i=1}^{N} \log p(s^{(i)} | x^{(i)}) \tag{2.64}$$

Substituindo 2.54 em 2.64 temos

$$l(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{P} \sum_{k=1}^{K} (\lambda_k^T f_k^T(\ell_{t-1}, s_t, x) + \lambda_k^D f_k^D(\ell_{t-1}, s_t, x) + \lambda_k^O f_k^O(\ell_{t-1}, s_t, x)) - \sum_{i=1}^{N} \log Z(x^{(i)}) \tag{2.65}$$

que pode ser simplificado para

$$l(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{P} \sum_{k=1}^{K} \sum_{U \in \{T,D,O\}} \lambda_k^U f_k^U(\ell_{t-1}, s_t, x) - \sum_{i=1}^{N} \log Z(x^{(i)}) \tag{2.66}$$

Assim como em LCCRF, para evitarmos o *overfitting*, podemos melhorar $l(\theta)$ adicionando uma penalidade baseada na norma Euclidiana de θ e no parâmetro de regularização $1/2\sigma^2$, onde σ^2 é somente um parâmetro que indica a intensidade dessa penalidade. Sendo assim temos o log da probabilidade condicional regularizada

$$l(\theta) = \sum_{i=1}^{N}\sum_{t=1}^{P}\sum_{k=1}^{K}\sum_{U\in\{T,D,O\}} \lambda_k^U f_k^U(\ell_{t-1}^{(i)}, s_t^{(i)}, x^{(i)}) - \sum_{i=1}^{N}\log Z(x^{(i)}) - \sum_{t=1}^{P}\sum_{k=1}^{K_t}\frac{\lambda_k^2}{2\sigma^2} \quad (2.67)$$

e suas derivadas parciais definidas como

$$\frac{\partial l(\theta)}{\partial \lambda_k^U} = \sum_{i=1}^{N}\sum_{t=1}^{P} f_k^U(\ell_{t-1}^{(i)}, s_t^{(i)}, x^{(i)}) - \sum_{s}\sum_{t=1}^{P} f_k^U(\ell_{t-1}, s_t, x)p(s'|x^{(i)}) - \sum_{t=1}^{P}\sum_{k=1}^{K_t}\frac{\lambda_k}{\sigma^2} \quad (2.68)$$

que podem ser eficientemente calculadas utilizando a seguinte equação:

$$\frac{\partial l(\theta)}{\partial \lambda_k^U} = \sum_{i=1}^{N}\sum_{t=1}^{P} f_k^U(\ell_{t-1}^{(i)}, s_t^{(i)}, x^{(i)}) - \frac{1}{Z(x^{(i)})}\sum_{y\in L}\eta^k(|x^{(i)}|, y) - \sum_{t=1}^{P}\sum_{k=1}^{K_t}\frac{\lambda_k}{\sigma^2} \quad (2.69)$$

onde $\eta^k(i, y)$ pode ser calculado utilizando a seguinte recursão:

$$\eta^k(i, y) = \sum_{d=1}^{D}\sum_{y'\in L}(\eta^k(i-d, y') + \alpha_{i-d}(y')f_k(y, y', x, i-d, i))\Psi(\ell, (y', i-d, i), x) \quad (2.70)$$

A função $l(\theta)$ é estritamente concava, ou seja, possui um único ótimo global (Lafferty *et al.* , 2001). E como de modo geral $l(\theta)$ não pode ser maximizada em sua forma fechada, o uso de algoritmos de otimização é indicado, como, por exemplo, o LBFGS.

2.7 Comparação

A primeira característica que diferencia modelos geradores dos discriminitativos é que os geradores modelam indiretamente o problema de classificação, enquanto que os modelos discriminativos modelam diretamente a probabilidade $p(c|x)$. Como consequência, o treinamento de modelos geradores é mais simples, já que um simples contagem é suficiente para estimar a verossimilhança. Já o treinamento de modelos discriminativos consiste em encontrar parâmetros que maximizem $p(c|x)$ o que envolve técnicas mais avançadas como otimização de funções não lineares (Wallach, 2002).

Entretanto, modelos geradores necessitam de um relaxamento dos relacionamentos entre as variáveis aleatórias pois caso contrário existirão muitos parâmetros a serem estimados, o

que demandará um conjunto de treinamento muito grande. Já os modelos discriminativos não sofrem desta característica (Sutton, 2008).

Outra vantagem dos modelos discriminativos é que estes normalmente têm desempenho superior ao dos modelos geradores. Ng e Jordan (2002) provaram que modelos discriminativos tem menor erro assintótico. Além disso, compararam o desempenho de naïve Bayes e regressão logística em problemas reais e observaram, como esperado, que naïve Bayes tinha erro médio menor quando o conjunto de treinamento era pequeno, entretanto, ao aumentar o conjunto de treinamento a regressão logística passava a ter um erro médio menor.

As vantagens em se utilizar HMM ou GHMM é que estes são modelos mais simples e é possível interprestar seus parâmetros de maneira mais direta. Por exemplo, na modelagem de um cassino desonesto utilizando HMM, como descrita na figura 2.2, deixa explícita a dinâmica de trocas entre os dados bem como que os lançamentos de dado desonestos resultam na face 1 na metade das vezes. Já ao se utilizar LCCRF ou semi-CRF a interpretação dos parâmetros passa a ser menos óbvia. Essa caracteristica é recorrente ao se utilizar outros modelos discriminativos.

Uma vantagem de se utilizar LCCRFs ou semi-CRFs é que é mais simples extrair novas características das sequências observadas. Para isso precisa-se apenas adicionar novas funções de características. Essas funções de características podem ser de qualquer tipo, permitindo explorarmos uma maior diversidade de modelagens.

Se comparado com outros modelos discriminativos para rotulação de sequências, como redes neurais artificiais, LCCRFs e semi-CRF têm como vantagem a possibilidade de se especificar a arquitetura geral das possíveis rotulações. Isso permite que um especialista possa transferir conhecimento com relação a estrutura das sequências que estão sendo analisadas. Inclusive é comum a utilização de CRF como uma última camada de uma rede neural, funcionando como um ajuste fino na segmentação de imagens (Chen *et al.*, 2018; Kamnitsas *et al.*, 2017; Roy e Todorovic, 2017).

Capítulo 3

Predição de Genes e de Início de Sítio de Transcrição

Uma fita de DNA é composta por uma série de nucleotídeos conectados através de ligações fosfodiester, sendo eles adenina, citosina, guanina e timina normalmente referenciados como A, C, G e T, respectivamente. Essa molécula é encontrada em organismos vivos na forma de fita dupla, pois os pares de nucleotídeos A-T e G-C formam ligações de hidrogênio, tornando a molécula bastante estável (Alberts *et al.*, 2014). Sendo assim, uma molécula de DNA fita dupla possui duas sequências de nucleotídeos complementares.

Segundo o dogma central da biologia molecular, o DNA de um organismo serve como molde para a produção de uma molécula de RNA complementar a uma das fitas de DNA que, por sua vez, serve como molde para a síntese de proteína (Lodish *et al.*, 2008). Estes processos são conhecidos por transcrição e tradução, respectivamente (figura 3.1).

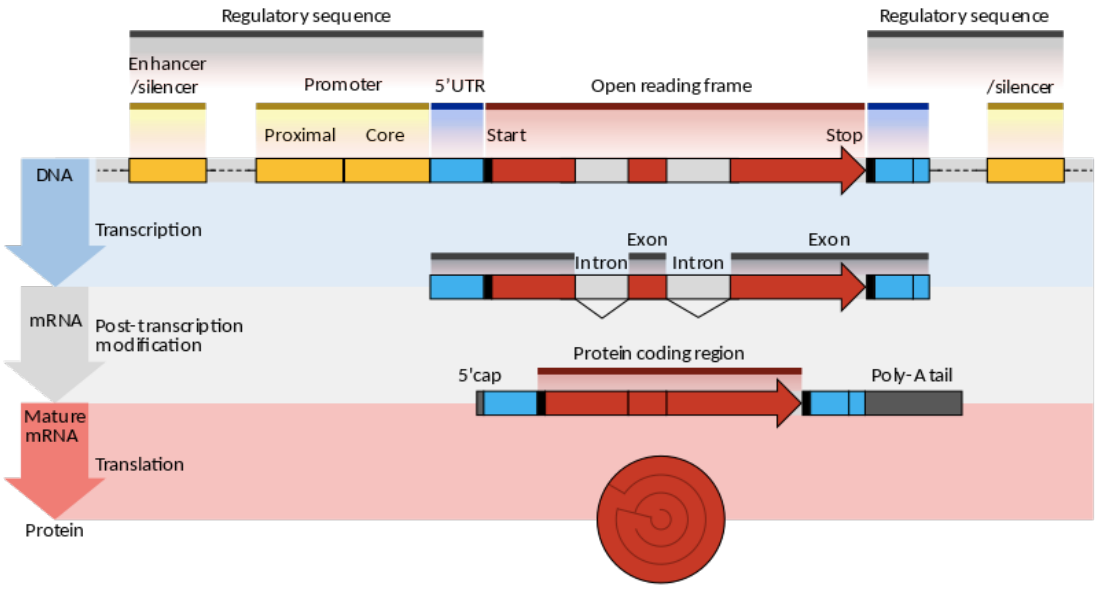

Figura 3.1: *Dogma central da biologia molecular: A partir de uma molécula de DNA, transcreve-se um RNA que é traduzido para proteína. (Shafee T, Lowe R, 2017)*

Essas regiões do DNA que servem como molde para a síntese de RNA são chamadas genes. Embora em alguns casos o RNA seja o produto final, neste trabalho estamos interessados

apenas nos genes que são transcritos em RNAs que são traduzidos para proteínas. Estes RNAs são chamados de RNAs mensageiros ou simplesmente mRNA.

Os nucleotídeos dos mRNAs são lidos em trincas, chamadas códons, a partir do códon de iniciação (AUG) até um códon de parada (UAA, UAG ou UGA). Cada códon, com exceção os códons de parada, traduz para um aminoácido, que são sub unidades de um proteína. Essa relação entre códons e aminoácidos é conhecida como código genético. Esse código genético é degenerado, ou seja, um aminoácido corresponde a mais de um códon. Por exemplo, o aminoácido arginina é codificado por 6 códons: CGU, CGC, CGA, CGG, AGA e AGG.

Em organismos eucariotos, a organização dos genes é complexa (Alberts *et al.*, 2014). Isso porque a sequência que define um mRNA em um gene eucariótico é intercalada por sequências não codificadoras (chamadas de introns). Sendo assim o gene é composto por *exons*, que farão parte do mRNA maduro, e *introns*, que serão descartados. Após a transcrição de um gene codificador de proteína é gerado um RNA imaturo (pré-mRNA), que sofrerá algumas modificações pós-transcricionais, sendo transformado em um mRNA maduro, antes de ser transportado para fora do núcleo da célula. As principais modificações são:

- Adição de uma guanosina modificada na extremidade 5', conhecida como *5' CAP.*
- Adição de uma sequência de adenosinas na extremidade 3', conhecida como *cauda poliA*
- Remoção de regiões intrônicas não-codificadoras para proteínas presentes nos pré-mRNAs, processo conhecido como *splicing*

Em alguns casos, um mecanismo chamado de *splicing* é responsável por permitir que um único gene possa gerar diferentes mRNAs maduros e consequentemente diferentes proteínas, como podemos ver na figura 3.2.

Após essas modificações pós-transcricionais, o mRNA maduro é transportado para fora do núcleo onde será traduzido para proteína. Um detalhe importante é que a região que realmente codifica para proteína é normalmente menor do que o mRNA maduro. Essas regiões não traduzidas, que estão localizadas antes do códon de iniciação e depois do códon de terminação, são conhecidas como 5'UTR e 3'UTR do gene, respectivamente.

Embora todas as células de um organismo multicelular possuam o mesmo DNA, cada tipo de célula exerce uma função diferente e portanto produz conjuntos de proteínas diferentes. Essas células são capazes de alterar seus padrões de expressão gênica em resposta a sinais extracelulares, como, por exemplo, regular a quantidade disponível de um certa proteína (Alberts *et al.*, 2014). Uma região importante para o controle da expressão de genes codificantes de proteínas é conhecida como região promotora e está localizada antes do gene. Essa região é composta por:

- um promotor *core*, sequência de DNA com aproximadamente 100 nucleotídeos contendo as subsequências *TATA-box* (caracterizada pela repetição de sequências dos nucleotídeos T e A), elemento iniciador (Inr, do inglês *Initiator Element*) e o sítio de início de transcrição (TSS, do inglês *Transcription Start Site*),

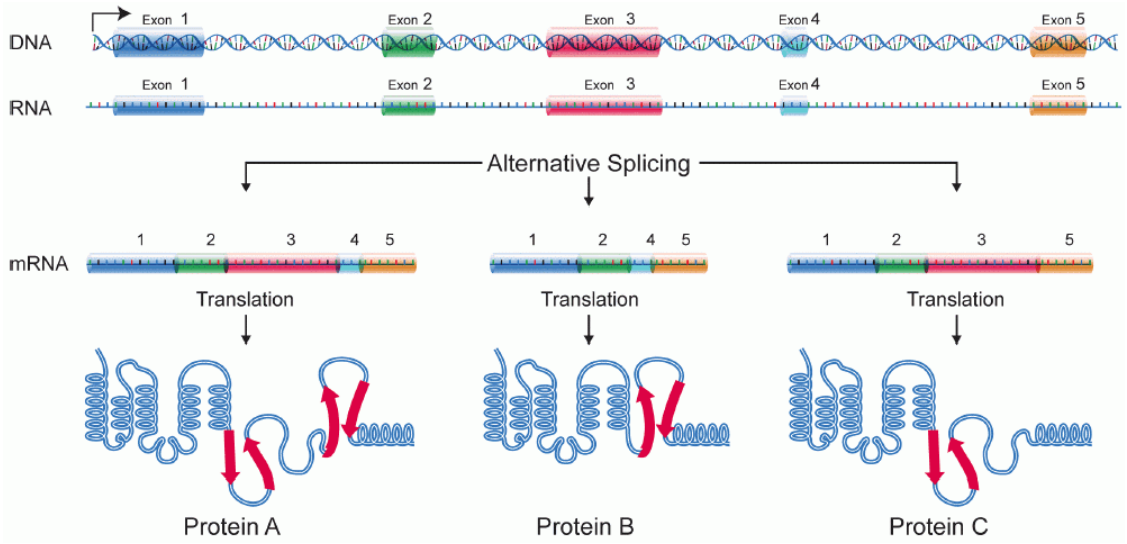

Figura 3.2: *Exemplo de splicing alternativo (NHGRI, 2018). Um mesmo gene pode gerar diferentes mRNAs, que por sua vez codificam proteínas diferentes. Na figura, o gene em questão é composto por 5 exons e pode gerar 3 mRNAs diferentes: A, transcrito dos exons 1, 2, 3, 4 e 5; B, transcrito dos exons 1, 2, 4 e 5; e C, transcrito dos exons 1, 2, 3 e 5. Por fim cada mRNA gerará uma proteína diferente.*

- e um conjunto de motivos localizados antes do promotor *core* contendo subsequências como *CAAT-box* (cuja sequência consenso é GGCCAATCT) e sítios de ligação de fatores de transcrição (TFBS, do inglês *Transcription Factor Binding Site*).

Determinar as regiões gênicas experimentalmente é um processo custoso e demorado. Além disso, as técnicas atuais dependem que o gene esteja sendo expresso no momento do experimento. Alguns genes são expressos somente em condições específicas e que são difíceis de serem reproduzidas (Mathe *et al.*, 2002). Para resolver esse problema diversos preditores de genes *in silico* foram propostos (Bernal *et al.*, 2012; Burge e Karlin, 1997; Gross *et al.*, 2007; Stanke e Waack, 2003; Stanke *et al.*, 2008). De modo geral, os preditores de gene são construídos para identificar genes que codificam para proteínas (Goel *et al.*, 2013). Mais ainda, eles apenas conseguem identificar a região codificante dos genes, ou seja, as 5'UTR e 3'UTR não são consideradas ou são identificadas com baixa precisão (Zhang , 2002). É comum, na área de predição de genes, a utilização do termo *exon*, mesmo que erroneamente, como sinônimo de CDS (ou sequência codificante, do inglês, *Coding Sequence*) pois os preditores conseguem apenas identificar a parte codificante dos *exons*.

Como já comentamos anteriormente, os preditores de genes de procariotos apresentam bons resultados (Angelova *et al.*, 2010). Entretanto, embora os preditores de genes de eucariotos consigam identificar cerca de 95% dos nucleotídeos das regiões codificantes, somente cerca de 50% do genes são identificados corretamente (Stanke e Waack, 2003). Esse baixo

desempenho está relacionado com a estrutura do gene de eucariotos, que é fragmentado em *exons*, normalmente pequenos, e *introns*, normalmente grandes. Qualquer erro ao identificar uma fronteira entre *exons* e *introns* produz uma predição errada.

Identificar a localização da região promotora, principalmente o TSS, pode auxiliar os preditores de genes a obter melhores resultados (Morton *et al.*, 2015), já que próximo a um TSS está localizado o sítio de iniciação de tradução. Portanto, neste trabalho focamos em identificar a região promotora de genes de eucariotos codificadores de proteínas.

3.1 Predição de genes

Segundo Mathe *et al.* (2002), os preditores de genes podem ser classificados em 2 categorias: Os que utilizam informações extrínsecas e os que utilizam apenas informações intrínsecas.

3.1.1 Predição Extrínseca

Predição Extrínseca é normalmente baseada na busca pela similaridade entre a sequência analisada e outras sequências armazenadas em algum banco de dados. Essa análise pode ser realizada utilizando algoritmos de alinhamento como Smith-Waterman (Smith e Waterman, 1981), FASTA (Pearson e Lipman, 1988) e BLAST (Altschul *et al.*, 1990). Entretanto, uma das maiores fraquezas deste tipo de sensor é o fato de que não se identificará nada além de sequências similares as sequências armazenadas no banco de dados.

A busca por similaridade em banco de dados de proteínas como o SwissProt (Bairoch *et al.*, 2004) pode auxiliar na localização de regiões exônicas e intrônicas. Estima-se que cerca de 50% dos genes podem ser encontrados devido a similaridade entre as proteínas que estes genes codificam e proteínas homólogas (Mathe *et al.*, 2002). Porém, a estrutura completa do gene ainda é difícil de ser determinada já que mesmo proteínas homólogas podem não compartilhar alguns de seus domínios.

Outro tipo de busca por similaridade é comparar a sequência em questão com sequências de transcritos. Esse tipo de abordagem melhora a identificação da estrutura gênica principalmente se os dados utilizados são provenientes do mesmo genoma que está sendo anotado (Mathe *et al.*, 2002).

3.1.2 Predição Intrínseca

Como destacamos na seção anterior, a maior desvantagem da predição extrínseca é que identifica-se apenas genes que já possuem alguma evidência (sequências de de organismos próximos, dados de expressão, banco de dados de proteínas, etc). Já a abordagem intrínseca permite a identificação de genes novos ainda não caracterizados. Esta é a abordagem mais utilizada pelos preditores de genes, como o Genscan (Burge, 1997) e o AUGUSTUS (Stanke e Waack, 2003) que modelam o problema utilizando um GHMM.

Essa abordagem analisa a sequência alvo e, a partir da distribuição dos nucleotídeos, infere as possíveis localizações dos genes. Na maioria dos preditores modernos esta inferência se dá em dois níveis: cada um dos componentes da arquitetura dos genes é representada por um *sensor* e os sinais desetes sensores são integrados por um modelo integrador.

Sensores

Preditores de genes *ab initio* em geral são compostos de dois tipos de sensores:

- **Sensores de sinais**

  Sensores de sinais são aqueles que são capazes de identificar padrões regiões de tamanho fixo, em geral associando a estas regiões valores de probabilidade. Isso pode ser feito através de várias técnicas incluindo:

  1. um algoritmo de similaridade (Kleffe *et al.*, 1996; Rogozin e Milanesi, 1997) que compara a sequência alvo com uma sequência consenso;
  2. PWMs (do inglês *position weight matrix*) que modelam a probabilidade de uma certa base aparecer em uma dada posição, utilizadas no preditores de Brunak *et al.* (1991); Hebsgaard *et al.* (1996); Tolstrup *et al.* (1997) seguem essa abordagem.
  3. WAMs (do inglês *weight array model*), introduzidas por Zhang e Marr (1993), que capturam possíveis dependências entre posições adjacentes de um sinal
  4. MDDs (do inglês *maximal dependence decomposition*) introduzidos por (Burge , 1997).

  Um exemplo de sensor é a caracterização de sítios de *splicing*. As fronteiras entre exons e introns são normalmente conservadas, como podemos ver na figura 3.3. A extremidade 5' de um intron normalmente inicia com GU enquanto que a extremidade 3' normalmente termina com AG. Essas regiões são comumente chamadas respectivamente de *donor* e *acceptor*.

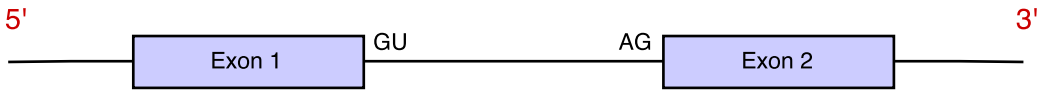

Figura 3.3: *A extremidade 5' de um intron normalmente inicia com GU (*donor site*), e a extremidade 3' termina com AG (*acceptor site*).*

- **Sensores de conteúdo**

  Sensores de conteúdo identificam regiões de tamanho variável, em geral atribuindo a elas um valor de probabilidade. A abordagem intrínseca baseia-se no fato de que as regiões codificantes são compostas por codons que, a partir das regras do código

genético, podem ser traduzidos para um aminoácido. Outras características são importantes, como: a composição de nucleotídeos das regiões exônicas são ricas em G e C se comparadas com introns; e a variação de hexâmeros (sequência de 6 nucleotídeos) têm capacidade de discriminar regiões codificantes das não codificantes (Fickett e Tung , 1992).

Um dos modelos mais utilizados na caracterização de regiões codificantes são as cadeias de Markov 3-periódicas introduzidas no preditor GeneMark (Borodovsky e Mcininch , 1993). Esse tipo de modelo define 3 cadeias de Markov (Hsu, 2013), uma para cada posição dentro do códon. Quanto maior a ordem dessas cadeias de markov, maior a capacidade de se caracterizar o relacionamento entre os nucleotídeos. Entretanto isso implica em um maior número de parâmetros a serem estimados e consequentemente é necessário um maior conjunto de treinamento para estimá-los. Na prática, preditores como GeneMark e Genscan (Burge e Karlin, 1997), utilizam cadeias de Markov de ordem 5 a fim de caracterizar hexâmeros.

Modelos Integradores

Para garantir uma predição que maximize a probabilidade do gene predito é necessário integrar as distribuições de probabilidades geradas pelos sensores em um modelo integrador. Econtramos dois modelos integradores utilizados para predição de genes na literatura: GHHMs e CRFs.

- **Preditores Baseados em GHMM** GHMMs utilizam diretamente os modelos de sensores descritos acima como modelos de emissão dos estados. A isso acrescentam-se distribuições para modelar a duração de cada emissão. Sinais tem sua duração fixa, já estados associados a sensores de conteúdo podem ter sua duração modelada com auto transições ou com uma distrubuição defindа por outro modelo. COMPLETAR, CORRIGIR BLABLA

- **Preditores Baseados em CRF**

  Dois preditores de genes foram propostos com integração de sensores utilizando CRFs: CONRAD e CRAIG.

  CONRAD (DeCaprio *et al.*, 2007) é um preditor de genes baseado em semi-CRF que obteve bons resultados na identificação de genes de fungos. Inicialmente, codificaram um semi-CRF que reproduzisse o mesmo comportamento do GHMM implementado pelo preditor Twinscan. Após isso, treinaram o semi-CRF para que os parâmetros fossem estimados de maneira discriminativa, ou seja, otimizando a função log da probabilidade condicional regularizada apresentada na equação (2.67). Para melhorar ainda mais a acurácia do preditor, novas funções de características foram incluídas. Um conjunto de funções de características modelam a existência de *gaps* em alinhamentos de sequências de espécies próximas. Um outro conjunto de funções de características modela, para cada espécie, se em uma dada posição da sequência analisada existe um

alinhamento com uma das sequências dessa espécie. E por fim um conjunto de funções de características que indicam a existência de um alinhamento com ESTs. Testes iniciais foram realizados o fungo *Cryptococcus neoformans*.O modelo obtido teve performance superior ao Twinscan em 15.9%. Porém os autores limitaram sua abordagem a fungos. Como em outras espécies os segmentos correspondentes a éxons e íntrons podem ter um tamanho grande, a implementação tradicional de Semi-CRFs iria envolver um overhead excessivo tanto paro o treinamento como para a predição. Em CONRAD o comprimento de cada segmento (correspondente a exons e introns) é limitado a um intervalo pequeno, o que resultaria em predições de baixa qualidade. Isso reforça a necessidade de uma implementação eficiente de semi-CRF que suporte durações de tamanho arbitrário.

Outro exemplo de preditor de genes baseado em CRF é o CRAIG (Bernal *et al.*, 2007), que conseguiu bons resultados, ao predizer sinais pertencentes aos genes de *H. sapiens*, como sítio de iniciação de tradução (14% melhor do que o Genizilla), sítios de terminação de tradução (6% melhor do que o AUGUSTUS), *donor sites* (4% melhor do que o GENSCAN++) e *acceptor sites* (7% melhor do que o GENSCAN++). Para que o modelo pudesse ser aplicado em um organismo complexo como *H. sapiens* as durações de rótulos de sinais foram modeladas com tamanho fixo enquanto que todo os outros rótulos foram modelados com tamanho geométrico.

Como CRAIG tem bom desempenho para identificar sinais de tamanho fixo, Bernal *et al.* (2012) o mesmo grupo desenvolveu uma extensão do preditor chamada de eCRAIG. Esse preditor combina anotações provenientes de diversas fontes como ENSEMBL, Pairagon+mRNA_EST, NSCAN, Aceview, Exogean, ExonHunter, MARS, Twinscan e do próprio CRAIG. Nos testes realizados em *A. thaliana*, eCRAIG melhorou o *F-score* em 5.8% absolutos da identificação de genes completos, se comparado com os preditores GeneMarkHMM, GenScan, GlimmerA, GlimmerM, TwinScan.

3.2 Predição de Promotores

O objetivo dos preditores de promotores varia, podendo, em alguns caso focar em determinar a posição da região promotora de genes codificadores de proteína, ou focar em determinar a posição do TSS (Abeel *et al.*, 2009). Entretanto, como apresentado por Frith e colaboradores (Frith *et al.*, 2008), normalmente um gene não possui apenas um único TSS, mas sim vários, formando uma região chamada de Região de Início de Transcrição (TSR, do inglês *Transcription Start Region*). Algumas TSR são mais compactas, ou seja, o sinal de início de transcrição é mais forte em um pequeno segmento, enquanto que algumas são mais dispersa, ou seja, o sinal de início de transcrição está espalhado por uma região maior. Essa característica faz com que a avaliação de preditores de TSS seja baseada em uma janela de distância entre o TSS predito e o anotado (Abeel *et al.*, 2009).

Os primeiros métodos para determinar a posição do TSS consistiam em escolher uma

distância fixa entre o sítio de iniciação de tradução e o TSS (Yamamoto *et al.*, 2011). Porém, devido ao tamanho variado das regiões ’5 UTR, esse método tende a ter baixa precisão.

A partir dos anos 2000 novos preditores de TSS baseados em critérios mais sofisticados foram propostos, mas, mesmo considerando certo resultados dentro uma janela com distância máxima de 500 nucleotídeos entre o TSS predito e o anotado, menos de 35% dos TSS eram identificados (Abeel *et al.*, 2009; Narlikar e Ovcharenko, 2009).

Nos últimos 3 anos, 2 preditores foram desenvolvidos e elevaram a qualidade das predições de TSS: TIPR (Morton *et al.*, 2015) e TSSPlant (Shahmuradov *et al.*, 2017).

O preditor TIPR analisa o conteúdo da sequência de DNA procuando por padrões que caracterizam TFBSs. Para cada padrão de TFBS conhecido, é construído um classificador utilizando o modelo de regressão logistica, que quando aplicado, atribui a probabilidade de um dado nucleotídeo pertencer, ou não, à uma TSR. Morton *et al.* (2015) afirmam que essa característica faz com que TIPR possa ser utilizado para localizar não somente o TSS, mas também determinar o tipo de TSR, se é mais compacta ou dispersa.

O TSSPlant é um preditor de TSSs de plantas, que também consegue identificar se o promotor possui uma região *TATA-Box*. Para isso foram construídos dois classificadores baseados em redes neurais artificiais, um para identificar promotores que possuem TATA-Boxes e outro para identificar promotores que não possuem TATA-Boxes. Cada classificador utiliza um conjunto de PWMs que caracterizam sinais importantes de promotores, como INR (do inglês *initiator element*), DPE (do ingles *downstream promoter element*) and YP (do ingles *youth promoter*). Para treinar um modelo geral para plantas Shahmuradov *et al.* (2017) utilizaram sequência de *A. thaliana* e *O. sativa*. Mas, assim como o TIPR, TSSPlant não oferece uma ferramenta para treinar novos modelos.

Capítulo 4

Implementação de CRFs

Nossa implementação de CRFs foi feita como uma extensão do arcabouço probabilístico ToPS (Kashiwabara *et al.*, 2013), desenvolvido pelo nosso grupo de pesquisa. ToPS é um sistema orientado a objetos implementado em C++ que tem como objetivo forncer um ambiente para implementação de novos modelos. Ao inciarmos nosso trabalho o ToPS implementava 8 modelos probabilisticos: (i) processo de distribuição idêntica e independente; (ii) cadeia de Markov de alcance variável; (iii) cadeia de Markov não-homogênea; (iv) modelo de Markov oculto; (v) modelo de Markov oculto de perfil; (vi) modelo de Markov oculto pareado; (vii) modelo de Markov oculto generalizado; e (viii) ponderação de sequencias baseada em similaridade. A utilização do arcabouço como base da implementacão nos possibilitou gerar um sitema que está habilitado a utilizar os modelos probabilísticos já existentes como funções de característica, o que irá simplificar, no futuro, o desenvolvimento de preditores de genes baseados em CRFs..

Antes de apresentarmos nossa implementação de LCCRF e semi-CRF, apresentaremos uma nova forma de representá-los utilizando uma máquina de estados. Após isso, essa nova representação será mapeada diretamente na forma como modelamos nossa implementação. Por fim comparamos nossa extensão do ToPS com duas ferramentas já existentes, CRF++ (Kudo, 2003) e CRFSuite (Okazaki, 2007).

4.1 Representação Gráfica

CRFs não possuem uma representação gráfica muito intuitiva. Tradicionalmente eles são representados na forma de Grafos de Fatores. Como alternativa, desenvolvemos uma nova forma de representação utilizando máquinas de estados, que, embora não seja tão geral quanto os Grafo de Fatores, facilita a especificação e entendimento de LCCRF e semi-CRFs.

4.1.1 Grafo de Fatores

Tradicionalmente alguns modelos geradores são representados como grafos direcionados. Como exemplo, temos a representação na figura 4.1 de um HMM para o problema da moeda desonesta, que consiste em um jogo onde se alterna entre moedas honestas e desonestas.

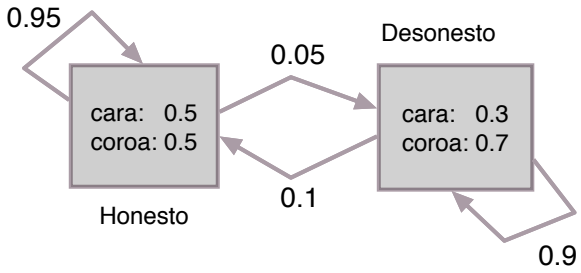

Figura 4.1: *Exemplo de HMM para o problema da moeda desonesta. Nesta representação os vértices representam os dois tipos de moedas e as arestas representam as probabilidades de alternar ou manter a utilização de uma certa moeda.*

Podemos também representar uma sequência de lançamentos como a figura 4.2. Nesta figura apresentamos um grafo direcionado que representa o lançamento de uma moeda honesta seguido de 2 lançamentos de uma moeda desonesta e novamente mais um lançamento de uma moeda honesta. Cada estado tem arestas que representam as possíveis emissões.

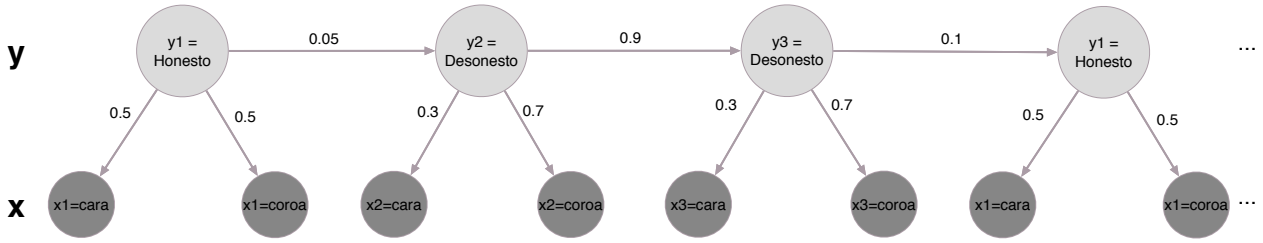

Figura 4.2: *Representação alternativa a apresentada na figura 4.1*

Já para a representação de modelos discriminativos, a utilização de grafos de fatores é mais comum (Sutton e Mccallum, 2002).

Podemos utilizar grafos de fatores para modelar uma família de distribuição de probabilidade. Para isso, representamos esta distribuição de probabilidade como o produto de funções locais Ψ em que cada uma depende de um pequeno conjunto de variáveis aleatórias.

Seja V um conjunto de variáveis aleatórias, $A_i \subset V$ uma coleção de subconjuntos de V, e $\Psi_{A_i} : V^{n_i} \to \mathbb{R}^+$ um conjunto de funções, onde n_i é o número de elementos de A_i e v_{A_i} os elementos de A_i.

Sendo assim, para uma sequência v temos que

$$P(v) = \frac{1}{Z} \prod_i \Psi_{A_i}(v_{A_i}) \tag{4.1}$$

onde Z é um fator de normalização definido por

$$Z = \sum_{v} \prod_{i} \Psi_{A_i}(v_{A_i}) \tag{4.2}$$

Podemos representar um conjunto de fatores como na figura 4.3, onde temos um grafo bipartido $G = (V, F, E)$ em que cada variável aleatória é representada como um vértice $v_s \in V$ que está conectado à um vértice Ψ_{A_i}, que é chamado de fator somente se v_s é um argumento de Ψ_{A_i}.

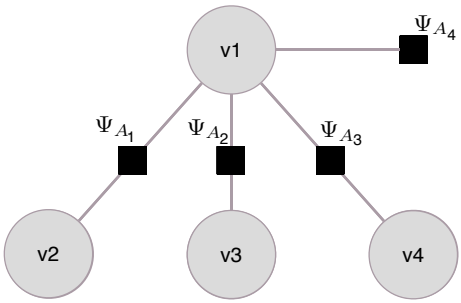

Figura 4.3: *Grafo de fatores. Os circulos representam variáveis aleatórias e os quadrados representam funções que fatoram a probabilidade definida pelo grafo. Os argumentos dessas funções são determinados pelas arestas que ligam funções e variáveis aleatórias*

Voltando ao exemplo do lançamento de moedas desonestas, podemos representar o lançamento descrito na figura 4.2 na forma de um grafo de fatores como o da figura 4.4, onde os fatores relativos às observações são:

- $f1(y_i, x_i) = 0.5$ se $x_i = cara$ e $y_i = honesto$
- $f2(y_i, x_i) = 0.5$ se $x_i = coroa$ e $y_i = honesto$
- $f4(y_i, x_i) = 0.3$ se $x_i = cara$ e $y_i = desonesto$
- $f5(y_i, x_i) = 0.7$ se $x_i = coroa$ e $y_i = desonesto$

e os fatores relativos às transições são:

- $f3(y_i, y_{i-1}) = 0.05$ se $y_i = desonesto$ e $y_{i-1} = honesto$
- $f6(y_i, y_{i-1}) = 0.9$ se $y_i = desonesto$ e $y_{i-1} = desonesto$
- $f7(y_i, y_{i-1}) = 0.1$ se $y_i = honesto$ e $y_{i-1} = desonesto$
- $f8(y_i, y_{i-1}) = 0.95$ se $y_i = honesto$ e $y_{i-1} = honesto$

O modelo representando na figura 4.4 apenas não é um CRF e sim um HMM pois não define uma distribuição de probabilidade como a descrita no capítulo 2. Para tanto seus fatores deveriam ser:

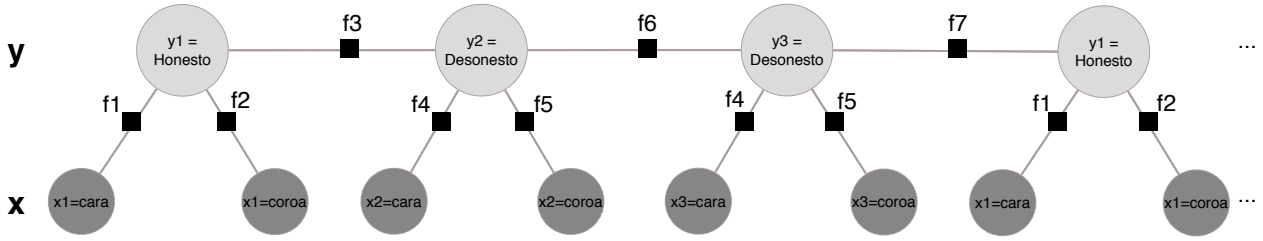

Figura 4.4: *Representação alternativa a apresentada na figura 4.2*

- $f1(y_i, x_i) = \log 0.5$ se $x_i = cara$ e $y_i = honesto$
- $f2(y_i, x_i) = \log 0.5$ se $x_i = coroa$ e $y_i = honesto$
- $f4(y_i, x_i) = \log 0.3$ se $x_i = cara$ e $y_i = desonesto$
- $f5(y_i, x_i) = \log 0.7$ se $x_i = coroa$ e $y_i = desonesto$
- $f3(y_i, y_{i-1}) = \log 0.05$ se $y_i = desonesto$ e $y_{i-1} = honesto$
- $f6(y_i, y_{i-1}) = \log 0.9$ se $y_i = desonesto$ e $y_{i-1} = desonesto$
- $f7(y_i, y_{i-1}) = \log 0.1$ se $y_i = honesto$ e $y_{i-1} = desonesto$
- $f8(y_i, y_{i-1}) = \log 0.95$ se $y_i = honesto$ e $y_{i-1} = honesto$

e a distribuição:

$$P(y|x) = \frac{1}{Z(x)} \exp\{\sum_{t=1}^{N} \sum_{k=1}^{8} f_k(y_t, y_{t-1}, x)\} \tag{4.3}$$

Observe que alguns LCCRFs se assemelham a HMMs, porém utilizando um grafo de fatores em sua representação. Seja $1_{\{a=b\}}$ uma função cujo valor é 1 se a é igual à b e 0 caso contrário temos que um LCCRF é semelhante a um HMM se suas funções de características são do tipo $f_{ij}(y, y', x) = 1_{\{y=i\}}1_{\{y'=j\}}$ para cada transição entre estados não-observáveis (i, j) e $f_{io}(y, y', x) = 1_{\{y=i\}}1_{\{x=o\}}$ para cada par estado não-observado e seu símbolo emitido (i, o). E seus respectivos pesos são $\lambda_{ij} = \log p(y' = i|y = j)$ e $\lambda_{oi} = \log p(x = o|y = i)$. Assim, adicionar novas características ao modelo é bastante simples. Por exemplo, se desejamos que a observação atual dependa do estado atual e anterior podemos adicionar a função de característica $1_{\{y_t=j\}}1_{\{y_{t-1}=i\}}1_{\{x_t=o\}}$ como observado na Figura 4.5.

4.1.2 Máquina de estados

O problema de se utilizar grafo de fatores para representar CRFs é que elas definem a probabilidade $p(y|x)$ para dadas sequências x e y específicas. Muitas vezes queremos

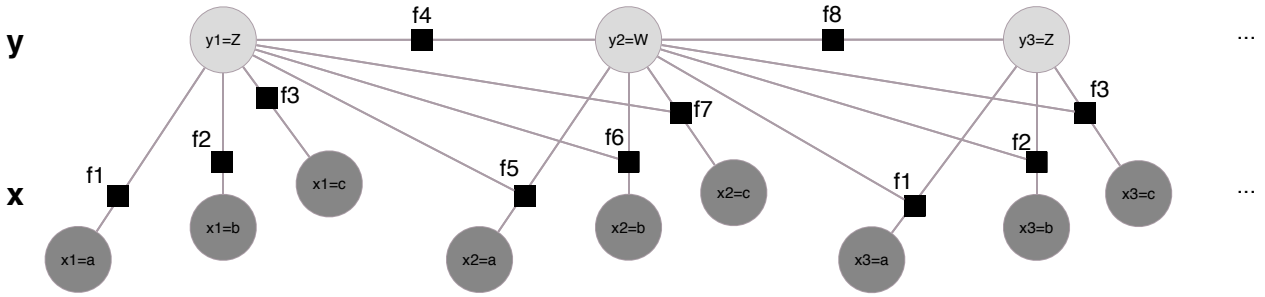

Figura 4.5: *LCCRF em que a observação atual depende do estado atual e anterior. Novamente os fatores $f4$ e $f8$ possuem funções de características do tipo $1_{\{y_t=i\}}1_{\{y_{t-1}=j\}}$, e o restante seguem o formato $1_{\{y_t=i\}}1_{\{y_{t-1}=j\}}1_{\{x_t=o\}}$.*

delinear um modelo de forma mais geral, como uma arquitetura que define as possíveis respostas/rotulações. Em predição de genes essa é uma abordagem bem comum (Burge, 1997; Korf *et al.*, 2001; Stanke e Waack, 2003). Primeiramente descreve-se a arquitetura do gene, separando cada parte bem caracterizada em um estado. Esses estados são interligados de maneira lógica (a sempre formar genes válidos) e para cada um define-se um modelo probabilístico capaz de discriminar o sinal que ele representa.

Para facilitar o entendimento, utilizaremos novamente como exemplo o problema das moedas desonestas representado na figura 4.1. Formalmente um semi-CRF para esse problema pode ser definido como fizemos na seção 2.6:

$$P(s|x) = \frac{1}{Z(x)} \exp\{\sum_{t=1}^{P}\sum_{k=1}^{K_t}(\lambda_{t,k}^{T} f_{t,k}^{T}(\ell_{t-1}, s_t, x)+\lambda_{t,k}^{D} f_{t,k}^{D}(\ell_{t-1}, s_t, x)+\lambda_{t,k}^{O} f_{t,k}^{O}(\ell_{t-1}, s_t, x))\} \tag{4.4}$$

onde cada estado $s_t = (\ell_t, b_t, e_t)$ é uma tripla contendo o rótulo (ℓ_t), a posição de início (b_t) e a posição de fim (e_t) do segmento que ele representa tal que

$$\begin{aligned} \ell_i &\neq \ell_{i+1} \\ b_i &\leq e_i \\ e_i + 1 &= b_{i+1} \\ b_1 &= 1 \\ e_P &= T \end{aligned} \tag{4.5}$$

e $f_{t,k}^{T}$, $f_{t,k}^{D}$, $f_{t,k}^{O}$, são, respectivamente as funções de características de transição, duração e observação, e $\lambda_{t,k}^{T}$, $\lambda_{t,k}^{D}$, $\lambda_{t,k}^{O}$, são os pesos associados às funções de característica de transição, duração e observação.

Considere uma sequência de observações $x = \{$ cara, cara, coroa, cara, cara, cara $\}$ e uma sequência de rotulações $y = \{$ desonesta, desonesta, honesta, honesta, desonesta, desonesta

}. A representação de $p(y|x)$ utilizando um grafo não-direcionado é dada na figura 4.6. Note que não é possível determinar a relação entre os rótulos (é possível não existir dois rótulos iguais consecutivos?) nem mesmo entre rótulos e observações (a moeda desonesta sempre observa uma cara?) ou quais os tipos de funções de características existem.

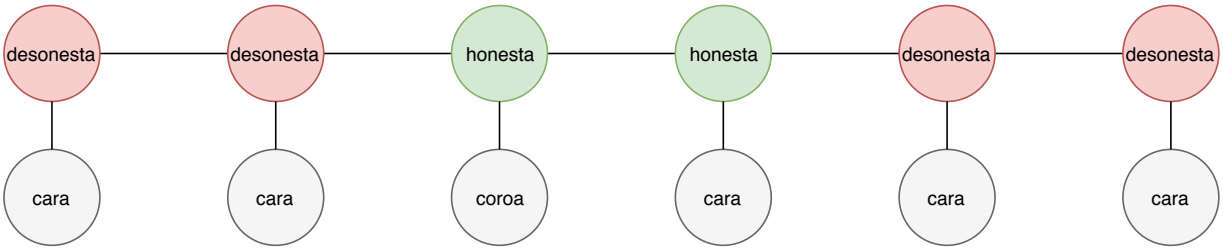

Figura 4.6: *Grafo não direcionado representando a probabilidade p(y|X) para x = { cara, cara, coroa, cara, cara, cara } e y = { desonesta, desonesta, honesta, honesta, desonesta, desonesta }*

Dada essa dificuldade, propomos uma representação alternativa capaz de descrever a arquitetura lógica do problema. Um semi-CRF pode ser definido como uma máquina de estados, tal que cada vértice é composto por:

- um identificador que corresponde ao rótulo associado ao estado da máquina de estados;
- uma função de característica que representa uma duração, que pode ser de tamanho fixo, ou definida por um modelo probabilístico;
- um conjunto de funções de características que podem ser uma função indicadora (é avaliada como 1 se um conjunto de símbolos é observado ou 0 caso contrário), um modelo probabilístico ou qualquer outra função;
- um conjunto de arestas que determinam as possíveis transições entre estados. Essas arestas definem indiretamente as funções de características indicadoras que habilitam as transições entre estados.

Voltando ao nosso exemplo das moedas desonestas, podemos representá-lo como na figura 4.7, que inclusive é muito parecida com a definição do mesmo problema utilizando HMM na figura 4.1.

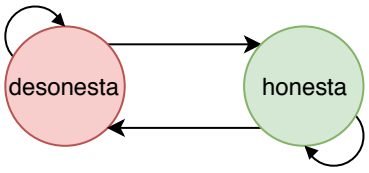

Figura 4.7: *Máquina de estados representando o problema da moeda desonesta.*

As funções de característica de duração associadas aos estados dessa máquina de estados são:

- $f^D_{t,1}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{e_t - b_t = 1\}} 1_{\{\ell_{t-1} = honesto\}} 1_{\{\ell_t = honesto\}} \log(0.95)$
- $f^D_{t,2}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{e_t - b_t = 1\}} 1_{\{\ell_{t-1} = honesto\}} 1_{\{\ell_t = desonesto\}} \log(0.05)$

- $f^D_{t,3}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{e_t - b_t = 1\}} 1_{\{\ell_{t-1} = desonesto\}} 1_{\{\ell_t = honesto\}} \log(0.1)$
- $f^D_{t,4}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{e_t - b_t = 1\}} 1_{\{\ell_{t-1} = desonesto\}} 1_{\{\ell_t = desonesto\}} \log(0.9)$

as funções de característica de observações são:

- $f^O_{t,1}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{\ell_t = honesto\}} 1_{\{x_t = cara\}}$
- $f^O_{t,2}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{\ell_t = honesto\}} 1_{\{x_t = coroa\}}$
- $f^O_{t,3}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{\ell_t = desonesto\}} 1_{\{x_t = cara\}}$
- $f^O_{t,4}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{\ell_t = desonesto\}} 1_{\{x_t = coroa\}}$

e os parâmetros associados a essas funções de característica são:

- $\lambda^D_{t,k} = \log(1)$ para todo $k \in \{1, 2, 3, 4\}$
- $\lambda^O_{t,k} = \log(0.5)$ para todo $k \in \{1, 2\}$
- $\lambda^O_{t,3} = \log(0.3)$
- $\lambda^O_{t,4} = \log(0.7)$

Nossa representação dispensa a definição das funções de característica de transições, pois a própria máquina de estados já representa isso. Ou seja, para todas as k arestas saindo de um estado i e chegando em um estado j temos uma função de característica do tipo $f^T_{t,k}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{\ell_{t-1} = i\}} 1_{\{\ell_t = j\}}$. Já os parâmetros $\lambda^T_{t,k}$ devem ser especificados, e como nessa modelagem da moeda desonesta a transição está embutida na duração dos estados $\lambda^T_{t,k} = 0$ para todo k.

Observe que as funções de características de duração forçam segmentos de tamanho 1 já que $e_t - b_t$ deve ser igual a 1, caso contrário é avaliada como zero. Essa condição viola a definição de semi-CRF, mas quando a duração é geométrica podemos relaxar essa condição para que a computação seja mais eficiente, como veremos mais adiante.

A vantagem de nossa representação gráfica utilizando uma máquina de estados é que é fácil visualizar que pode existir uma alternância qualquer entre as moedas honesta e desonesta. A importância disso aumenta quando o modelo em questão tem dezenas de estados esparsamente conectados como um preditor de gene. Outra vantagem é que essa abordagem se assemelha a definição de um GHMM como descrevemos no início dessa seção.

Vale destacar que nesse exemplo utilizamos apenas funções indicadoras. Poderíamos ter utilizado funções mais complexas, como modelos probabilísticos, para modelar as durações e observações, por exemplo:

- Caso a duração do estado *honesto* fosse modelada por outro tipo de distribuição, poderíamos remover a auto-transição da máquina de estados referente a ele e especificar um função de característica do tipo:
  $f^D_{t,5}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{\ell_{t-1} \neq honesto\}} 1_{\{\ell_t = honesto\}} d(e_t - b_t)$
  onde d é uma função qualquer.

- Caso exista uma distribuição de probabilidade definida por $p(x|\theta)$ para o estado *honesto*, onde θ são os parâmetros dessa distribuição e $x_{i:j}$ é um segmento de x contendo os elementos da posição i até j, poderíamos definir uma função de característica de observação do tipo:

$$f^{O}_{t,5}(\ell_{t-1}, (\ell_t, b_t, e_t), x) = 1_{\{\ell_t = honesto\}} \log p(x_{b_t:e_t}|\theta)$$

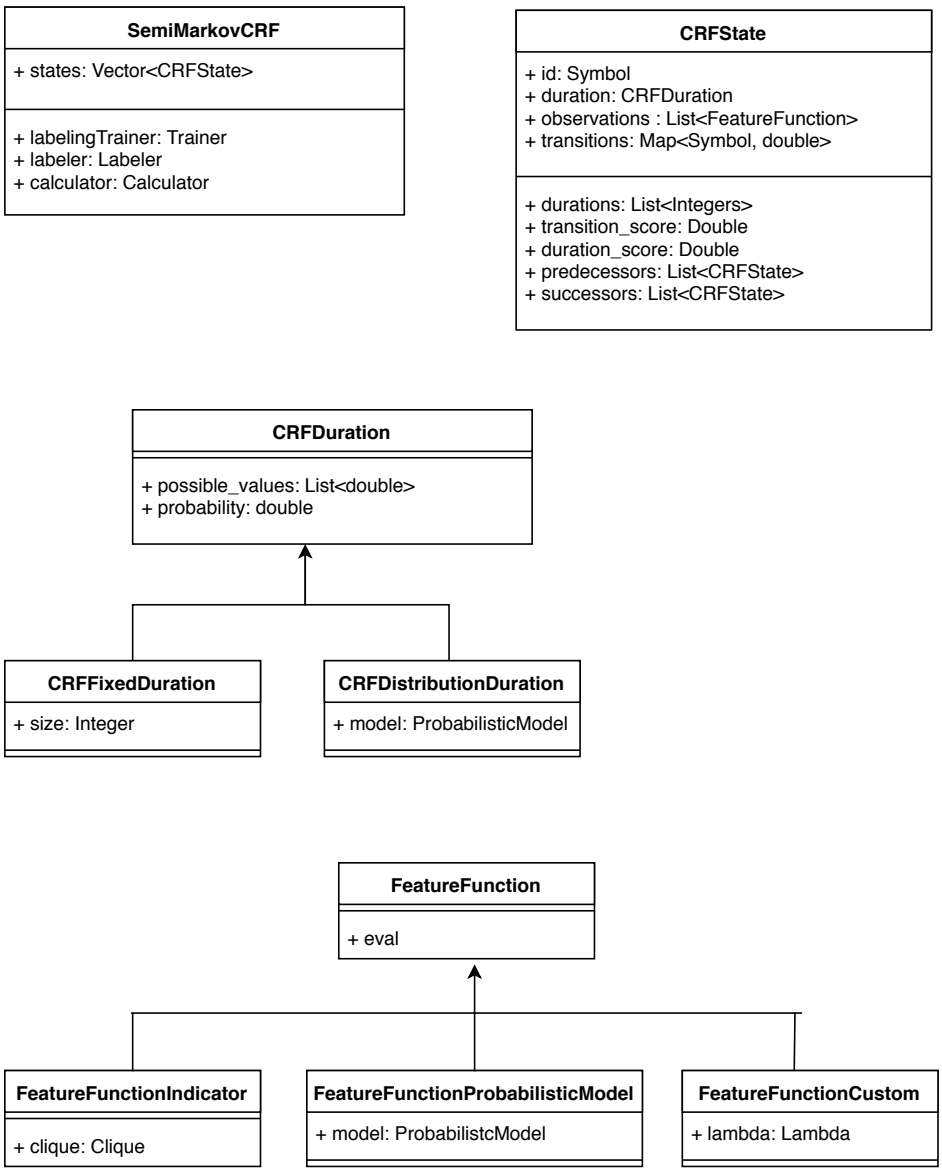

Figura 4.8: *Diagrama das principais classes envolvidas na definição de um semi-CRF utilizando o arcabouço ToPS.*

4.2 Modelagem

Desenvolver algoritmos de semi-CRF capazes de analisar grandes volumes de dados é um desafio devido a suas complexidades computacionais. CRF++ (Kudo, 2003) e CRFSuite (Okazaki, 2007), por exemplo, apenas permitem a modelagem de problemas utilizando

LCCRF com funções de características indicadoras. Implementações de semi-CRFs normalmente são realizadas para resolver problemas específicos (Bernal *et al.*, 2012; Vinson *et al.*, 2007), de tal forma que o programador pode explorar certas características do seu problema para desenvolver uma implementação eficiente. Entretanto, esse tipo de abordagem dificulta a extensão do modelo e não permite utilizá-lo para resolver outros tipos de problemas.

Nosso grupo de pesquisa desenvolver o arcabouço ToPS para resolver um problema parecido (Kashiwabara *et al.*, 2013). Não existia uma ferramenta eficiente e flexível o suficiente que pudesse ser usada para experimentar arquiteturas de GHMM. Neste trabalho utilizaremos o ToPS como base para nossa implementação de semi-CRF.

Nossa modelagem de semi-CRFs orientada a objeto é baseada na representação gráfica apresentada na seção anterior. Na figura 4.8 podemos ver que um *SemiMarkovCRF* é composto por uma lista de estados. Cada estado representa um rótulo (*id*) e é conectado a outros estados (*transitions*) formando, assim, uma máquina de estados. Os estados possuem uma duração (*duration*) e um conjunto de funções de características (*observations*).

4.2.1 Durações

Os algoritmos de *viterbi*, *forward* e *backward* descritos na seção 2.6.1 dependem de um conjunto contendo as possíveis durações dos estados. Como esses três algoritmos são bastante similares, utilizaremos apenas o algoritmo de *viterbi* para demonstrar como exploramos a duração dos estados em nossa implementação para reduzir suas complexidades.

O algoritmo de *viterbi* consiste em preencher, para uma sequência de observações x, $t \in \{1, 2, ..., |x|\}$ e para todo estado $j \in |L|$, uma tabela δ seguindo a seguinte recursão:

$$\delta(t, j) = \max_{i \in L, d \in \{1,2,...,t-1\}} \Psi(i, (j, t-d, t), x)\delta(t-d, i) \tag{4.6}$$

onde

$$\Psi(\ell_{t-1}, s_t, x) = \exp\{\sum_{k=1}^{K}(\lambda_k^T f_k^T(\ell_{t-1}, s_t, x) + \lambda_k^D f_k^D(\ell_{t-1}, s_t, x) + \lambda_k^O f_k^O(\ell_{t-1}, s_t, x)\}) \tag{4.7}$$

Se assumirmos que $d = \{1, 2, ..., t-1\}$ são todas as possíveis durações, calcular todos os valores $\delta(t, j)$ tem complexidade $O(|x|^2|L|^2K)$. Entretanto, assumir que o conjunto de durações é o mesmo para todo estado é uma modelagem imprecisa. Durante a especificação de um semi-CRF pode-se determinar que um estado tem duração periódica (múltiplo de 3, por exemplo, para representar códons), limitada, fixa ou até mesmo geométrica. Portanto decidimos criar uma hierarquia de classes, cuja classe raiz é *CRFDuration*, que representa as possíveis durações:

- *CRFFixedDuration* é responsável por modelar sinais de tamanho fixo (estados sem auto-transição) ou de duração geométrica (estados com auto-transição). Para esse tipo de estado $D = \{d\}$, onde d é o tamanho do sinal.

- *CRFDistributionDuration* é responsável por representar uma distribuição como uma duração. Essa distribuição é um *ProbabilisticModel*, classe base de todo modelo probabilístico disponível no ToPS. Para esse tipo de estado, D pode ser qualquer conjunto de durações.

Portanto, para cada estado, ao calcular o valor máximo de acordo com a equação (4.6), chamamos o método *possible_values* de sua duração. Se essa duração for do tipo *CRFFixedDuration* diminuímos em $|x|$ o número de operações dessa iteração.

Embora essa estratégia melhore a performance dos algoritmos, não diminui sua complexidade, a não ser que todas as durações sejam do tipo *CRFFixedDuration* (o que seria equivalente a definir um LCCRF). Para isso podemos adicionar um limite superior tal que toda duração seja menor do que uma constante M. Esse simples limite faz com que a complexidade caia para $O(|x||L|^2K)$

4.2.2 Funções de Característica

Outro ponto de otimização que exploramos na implementação dos algoritmos de *viterbi*, *forward* e *backward* é a computação das funções de características (equação (4.7)). O cálculo de Ψ contribui linearmente em K (número de funções de características) para a complexidade desses algoritmos. Isso porque estamos assumindo que calcular cada função de característica leva um tempo constante, o que muitas vezes não é verdade.

Nossa solução consiste em armazenar os valores das funções de características de observação e duração em *caches* de maneira que facilite a computação da equação (4.7) para qualquer subsequência das observações. Para tanto, as função de características são modeladas como subclasses de *FeatureFunction*s, que fornece uma interface uniforme para que cada tipo de função de característica possa ser implementada explorando suas particularidades.

- *FeatureFunctionIndicator* define funções indicadoras que são implementadas como condicionais; Esse é o tipo de função de característica mais comum ao se modelar um CRF. Por exemplo, em processamento de linguagem natural podemos adicionar uma função $1_{\{\text{palavra começa com letra maiúscia}\}}$ para indicar que uma palavra pode ser um nome.

- *FeatureFunctionProbabilisticModel* utiliza qualquer modelo do ToPS com função de característica. Esse tipo de função de característica é importante quando temos algum conhecimento prévio sobre o problema que estamos modelando. Por exemplo, sabemos que cadeias de markov interpoladas 3-periódicas são boas para modelar a região codificante do gene (Kashiwabara *et al.*, 2013; Stanke *et al.*, 2008), e, portanto, podemos utilizá-las como funções de características de um preditor de gene baseado em CRF. Integrar CRFs com outros modelos probabilísticos foi a principal motivação ao implementá-los como uma extensão do arcabouço ToPS.

- *FeatureFunctionCustom* define funções de características como funções arbitrárias e são impementadas como fechamentos (ou, em inglês, *closures*). Esse tipo de função pode

ser utilizado, por exemplo, para calcular o *score* do alinhamento entre a sequência de observações e um conjunto de sequências em um banco de dados. Embora esse tipo de função seja mais geral, deve-se garantir que essas funções sejam fatoráveis para que possamos aplicar as otimizações descritas nessa seção.

Todas as funções de característica definidas acima são fatoráveis, característica que nos permitir aplicar o seguinte algoritmo para calcular o *score* das sequências de observações.

Seja sub_score o valor das funções de características de observação e duração dada uma subsequência de x iniciando na posição a e terminando na posição b:

$$sub\_score(x, a, b, i) = \exp\{\sum_{k=1}^{K}(\lambda_k^D f_k^D(\_, (i, a, b), x) + \lambda_k^O f_k^O(\_, (i, a, b), x))\} \quad (4.8)$$

podemos pré-computar valores para todos os prefixos dessa dada sequência:

$$\begin{aligned} &prefix\_score(x, 0, i) = 1 \\ &prefix\_score(x, a, i) = prefix\_score(x, a-1, i) sub\_score(x, a-1, a, i) \end{aligned} \quad (4.9)$$

e por fim obter em $O(1)$ os valores dessas funções para qualquer uma das suas subsequências:

$$score(x, a, b, i) = prefix\_score(x, b, i) - prefix\_score(x, a, i) \quad (4.10)$$

Essa estratégia é importante, pois após uma pré-computação de complexidade $O(|x|K)$ podemos calcular o algoritmo de *viterbi* e, de forma semelhante, os algoritmos *forward* e *backward*:

$$\delta_t(j) = \max_{i \in L, d \in D} score(x, t-d+1, t, j) trans(i, j) \delta_{t-d}(i) \quad (4.11)$$

em tempo $O(|x||L|^2)$, onde $trans(i, j)$ é igual ao parâmetro λ_k^T associado a uma das k funções de característica de transição tal que $f_k^T(\ell_{t-1}, s_t, x) = 1_{\{\ell_{t-1}=i\}} 1_{\{\ell_t=j\}}$.

4.2.3 Conectividade

Se analisarmos a complexidade dos algoritmos de *viterbi*, *forward* e *backward*, o número de estados, M, aumenta o tempo de execução desses algoritmos em um fator $O(M^2)$. Em casos reais, como predição de genes, o número de estados pode passar de 60 (Kashiwabara *et al.*, 2013). E ao se representar esses tipos de problemas utilizando máquinas de estados, podemos perceber que a conectividade dos estados é baixa. Sendo assim ao mantermos uma lista de predecessores em *SemiMarkovCRF* podemos limitar a busca por valores possíveis, por

exemplo no *viterbi*:

$$\delta_t(j) = \max_{i \in Predecessores_j, d \in D} score(x, t-d+1, t, j) trans(i, j) \delta_{t-d}(i) \tag{4.12}$$

fazendo com que a complexidade total caia para $O(|x||L|S)$, onde S é o grau máximo dos vértices da máquina de estados e utilizando uma quantidade de memória proporcional a $O(|x||L|)$.

4.3 Treinamento

Como apresentamos na seção anterior, nossa implementação dos algoritmos de predição (*viterbi*, *foward*, *backward*) é eficiente e executa em tempo similar a um GHMM. Entretanto, estimar os parâmetros de um CRF é bastante custoso, como visto nas equações 2.5.2 e 2.6.2. Isso porque, a cada iteração temos que calcular a derivada da função log da probabilidade condicional regularizada que envolve o cálculo dos algoritmos *forward* e *backward*.

Um alternativa ao método tradicional de treinamento de CRFs é utilizar diferenciação automática (Baydin *et al.*, 2015) para calcular as derivadas parciais de cada parâmetro. Essa técnica é bastante popular no desenvolvimento de Redes Neurais Artificiais (ANN, do inglês *Artificial Neural Networks*) (Lecun *et al.*, 2015) e disponível em diversas bibliotecas para definição de ANN profundas como TensorFlow (Abadi *et al.*, 2016) , PyTorch (PyTorch Community, 2016) e MXNet (Chen *et al.*, 2015). A vantagem é que para cada iteração é necessário calcular apenas o algoritmo *forward* (utilizado no cálculo da função log da probabilidade condicional regularizada).

Nossa implementação utiliza a biblioteca PyTorch em conjunto com o arcabouço ToPS. PyTorch é uma biblioteca escrita em Python, mas as operações de tensores são implementadas em C. Embora fosse possível acessar as funcionalidades do PyTorch pela API C, escolhemos utilizar a API Python por ser mais simples e mais bem documentada. Portanto, foi necessário desenvolver uma API Python para os modelos do ToPS para que PyTorch e ToPS pudessem ser utilizados juntos. Devido a arquitetura orientada a objetos do ToPS, para que os modelos pudessem ser acessados por programas Python, foi necessário apenas exportar a interface de modelos probabilísticos mais geral chamada *ProbabilisticModel*.

Outra vantagem dessa abordagem é que agora podemos combinar modelos probalísticos (implementados no ToPS) com redes neurais (implementadas com PyTorch) para extrair features de sequências de símbolos.

4.4 Padrão Secretário

Após a publicação do arcabouço ToPS (Kashiwabara *et al.*, 2013), iniciamos, em parceria com Renato Ferreira Cordeiro, uma refatoração do código para facilitar a adição de novos modelos probabilísticos. Foi durante esse processo que decidimos por transformar os objetos de modelos probabilísticos em objetos imutáveis que produzem objetos-*interface*

para seus comportamentos. *SemiMarkovCRF* produz três tipos diferentes desses objetos-*interface*: *Trainer*, responsável por treinar novos CRFs; *Labeler*, responsável por rotular sequências utilizando o algoritmo de *viterbi* ou *posterior decoding*; e *Calculator*, responsável por calcular os algoritmos *forward* e *backward* necessários durante o treinamento e para a execução do *posterior decoding*.

Posteriormente identificamos que esta estrutura se repetia em outros sistemas como, por exemplo, jogos e implementações de linguagens de programação. Esse padrão de projeto foi então publicado como *Padrão Secretário* (Ferreira *et al.*, 2016).

O objetivo do padrão secretário é diminuir o acoplamento entre clientes e classes que têm interfaces complexas de modo que seus métodos possam ser agrupados em diferentes subconjuntos. Cada método deve ser conceitualmente independente dos métodos de grupos diferentes, mas suas implementações podem compartilhar código com qualquer outro método. O padrão define objetos auxiliares, chamados *Secretários*, reponsáveis por representar um comportamento de uma classe principal, chamada *Chefe* (figura 4.9). Em nossa implementação de semi-CRF, a classe *SemiMarkovCRF* é a *Chefe* e as classes *Trainer*, *Labeler* e *Calculator* são os *Secretários*.

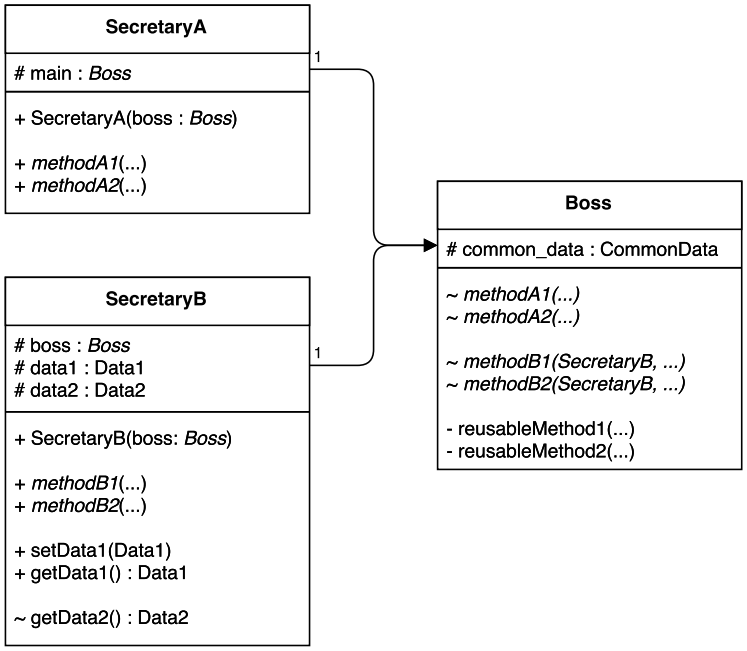

Figura 4.9: *Diagrama de classes do padrão secretário*

Os dados utilizados exclusivamente por um comportamento pode ser movido para o *Secretário* correspondente. Por exemplo, *Labeler*, responsável pela rotulação de sequências, calcula a matriz de *viterbi* (descrito na seção 4.2) para determinar a melhor rotulação de uma dada sequência de observações. Essa matriz, então, é armazenada no *Secretário Labeler*.

Já o código referente ao comportamento é mantido na classe *Chefe*. Continuando o exemplo, o código responsável por preencher a matriz de *viterbi* está implementado na classe *SemiMarkovCRF*.

Para acessar um comportamento, a aplicação cliente executa um método de um *Secre-*

tário, que por sua vez delega a computação para o *Chefe*, passando um ponteiro que dá acesso a si mesmo. Esse ponteiro pode ser utilizado para acessar os dados armazenados no *Secretário*. Toda chamada a um comportamento do *Chefe* deve ser indireta através de um *Secretário*, como descrito no diagrama de sequência 4.10.

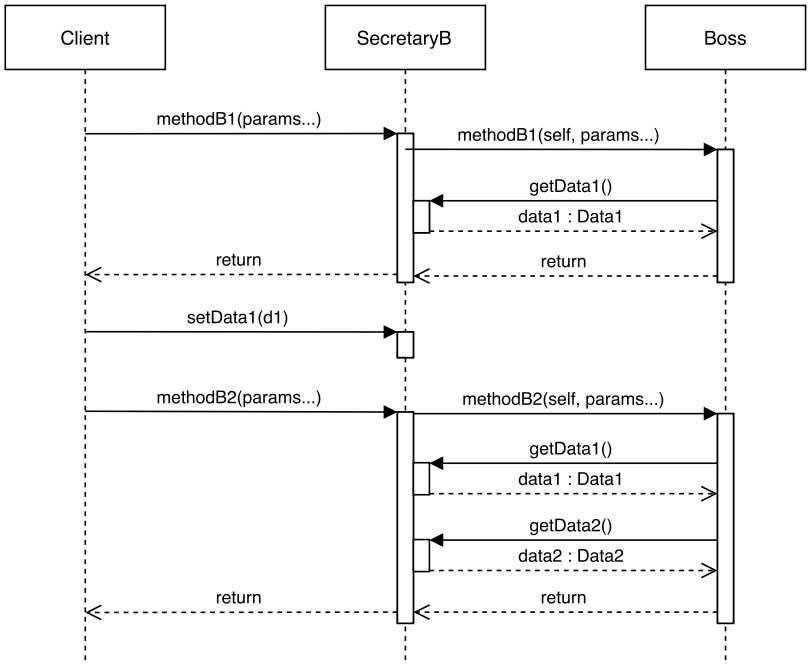

Figura 4.10: *Diagrama de sequência do padrão secretário*

O uso do padrão *secretário* permitiu a diminuição de recompilações de código e melhorou o reuso e duplicação de código, isso porque os algoritmos ficaram mais genéricos sendo necessário apenas variar o tipo de dado que o *secretário* armazena.

Outra vantagem ao se utilizar o padrão no ToPS é que, se o cliente utilizar mais de um comportamento ao mesmo tempo, o modelo permanece imutável. Isso permite que possamos paralelizar as computações sem se preocupar com inconsistências. Durante o cálculo dos gradientes descritos nas equações (2.50) e (2.68) chamamos dois secretários *Calculator* paralelamente, um para calcular o algoritmo *forward* e outro para calcular o algoritmo *backward*.

4.5 Comparação com Ferramentas Existentes

As ferramentas mais populares para a utilização de CRFs são CRFSuite (Okazaki, 2007) e CRF++ (Kudo, 2003). Ambas permitem apenas a definição de LCCRF que utilizem funções indicadoras como funções de característica. Nossa implementação é mais geral, implementa tanto LCCRF como Semi-CRFs e permite a definição de qualquer função como função de característica. Além disso, a integração de CRFs com outros modelos probabilísticos é facilitada devido a arquitetura orientada a objetos do ToPS.

4.5.1 CRF++

O arcabouço CRF++ é uma implementação de LCCRF de código aberto que fornece o algoritmo de *Viterbi* e treinamento utilizando LBFGS.

A modelagem de problemas é feita através da definição de *templates* de características. A partir destes *templates*, CRF++ extrai as funções de características necessárias a partir de um conjunto de treinamento e treina um novo modelo.

O conjunto de treinamento é definido como uma sequência de linhas, sendo que cada linha tem o mesmo número de colunas. Cada coluna, com exceção da última, contêm símbolos observáveis. Já a última coluna contêm rótulos atribuídos a posição em questão. Abaixo temos um exemplo desta estrutura:

```
He          PRP   B-NP
reckons     VBZ   B-VP
the         DT    B-NP << Posição Atual
current     JJ    I-NP
account     NN    I-NP
```

A definição desses *templates* de características é composta por *macros* do tipo $x[a, b]$, sendo a o deslocamento da linha e b o deslocamento da coluna em relação a posição atual. Por exemplo, para a sequência anterior, cuja posição atual é a terceira linha, temos:

```
x[0,0]            = the
x[0,1]            = DT
x[-1,0]           = reckons
x[-2,1]           = PRP
x[0,0],x[0,1]     = the,DT
```

Existem dois tipos de *templates* de características:

* *Unigram*: Esse tipo de *template* segue o formato "U01:%x[a,b]", sendo que U indica seu tipo e *x[a,b]* é a macro utilizada. Esse *template* define automaticamente um conjunto de funções de características de tamanho $(L \cdot N)$, onde L é o número de rótulos e N é o número de combinações possíveis geradas pelas macros deste template. São comumente utilizadas para adicionar dependências entre observações e o rótulo da posição atual.

* *Bigram*: Esse tipo de *template* é representado pelo formato "B01:%x[0,1]", sendo que B indica seu tipo e *x[a,b]* é a macro utilizada. Esse *template* define $(L \cdot L \cdot N)$, onde L é o número de rótulos e N é o número de combinações possíveis geradas pelas macros deste template. É responsável por gerar funções de características com dependência na transição do rótulo atual e anterior. Esse tipo de template tem um uso especial, sendo que pode-se defini-lo como "B", e assim gerar todas as $(N \cdot N)$ transições entre rótulos possíveis.

Essa abordagem reduz o número de funções de características que o usuário tem que definir, porém exige um maior conhecimento do funcionamento dos dois tipos de *templates* que o arcabouço disponibiliza.

Como exemplo, temos abaixo parte da definição do conjunto de treinamento do já mencionado problema do Cassino Desonesto, sendo que na primeira coluna temos as observações e na segunda coluna temos os rótulos.

```
6   Fair
5   Fair
2   Loaded
1   Loaded
1   Loaded
```

E a definição dos *templates* necessários para que o LCCRF seja similar à um HMM para o mesmo problema:

```
U00:%x[0,0]
B
```

Essa notação mostra-se bastante enxuta, já que, como podemos observar, utilizamos apenas dois *templates* para a definição de 16 funções de características. Entretanto, essa notação pode apenas representar funções de características indicadoras.

4.5.2 CRFSuite

CRFSuite é outra alternativa de código aberto que implementa o modelo LCCRF. Assim como CRF++, CRFSuite apenas implementa o algoritmo de *viterbi* e treinamento por LCFGS. Entretanto esta ferramenta exige que os dados sejam preprocessados antes do treinamento e rotulação. Esse preprocessamento mostrou-se bastante custoso (figura 4.11) o que pode ser um problema quando os dados a serem analisados estão disponíveis em grande volume.

CRFSuite segue a mesma abordagem do CRF++ e o usuário deve definir uma série de *templates* que extrairão funções de características automaticamente. Mas diferente do CRF++, a definição desses *templates* é realizada através de um *script* escrito na linguagem *Python*. Abaixo exibimos um *script* que extrai funções de características de uma série de lançamentos de dados de acordo com o problema do cassino desonesto:

```
1 #!/usr/bin/env python
2
3 separator = "\t"
4 fields = 'x y'
5
6 templates = (
7     (('x', 0), ),
```

```
8       )
9
10 import crfutils
11
12 def feature_extractor(X):
13     crfutils.apply_templates(X, templates)
14     if X:
15         X[0]['F'].append('__BOS__')      # BOS feature
16         X[-1]['F'].append('__EOS__')     # EOS feature
17
18 if __name__ == '__main__':
19     crfutils.main(feature_extractor, fields=fields, sep=separator)
```

O *script* acima gera automaticamente funções de características que definem as transições entre os rótulos, sendo que definimos explicitamente apenas as relações entre as observações x de deslocamento zero com o rótulo atual.

4.5.3 Comparação

O principal ponto que devemos observar é que nenhuma das ferramentas apresentadas suportam a definição de semi-CRFs, que são mais gerais. Em nossa extensão do ToPS optamos por implementar semi-CRFs que automaticamente utilizem estruturas de dados especializadas para que quando o usuário desejasse expressar um modelo mais simples, como um LCCRF, não houvesse perda de performance. Comparando o tempo de execução do algoritmo de *viterbi* definido nas figuras 4.11 e 4.11 podemos ver que nossa implementação, embora mais geral, não perde performance ao modelar LCCRFs. Não pudemos comparar o tempo de execução dos algoritmos *forward* e *backward* porque ambas ferramentas, CRF++ e CRFSuite, não oferecem interface para executarmos esses algoritmos.

Outro ponto importante é com relação ao tipo de funções de característica que podemos utilizar. CRF++ apenas permite funções indicadoras que podem ser geradas pelos seus *templates*. CRF-suite é um pouco mais geral pois permite que o usuário escreva um extrator de características externo, entretanto as características geradas ainda são binárias. Já nossa implementação de semi-CRF é ainda mais geral e permite a definição de qualquer função como uma função de característica. A única exigência é que essas funções sejam fatoráveis para que o algoritmo descrito na seção 4.2 funcione.

Um ponto em comum é que as três ferramentas podem ser utilizadas em programas escritos em Python, linguagem de programação bastante utilizada no desenvolvimento de algoritmos de aprendizado de máquina, reconhecimento de padrões e inteligência artificial.

Nossa implementação, por ser uma extensão do ToPS, também tem como vantagem o fato de ser facilitada a integração de semi-CRFs com outros modelos probabilísticos. Entretanto, até o momento, a definição de semi-CRFs é feita programaticamente via C++ ou Python.

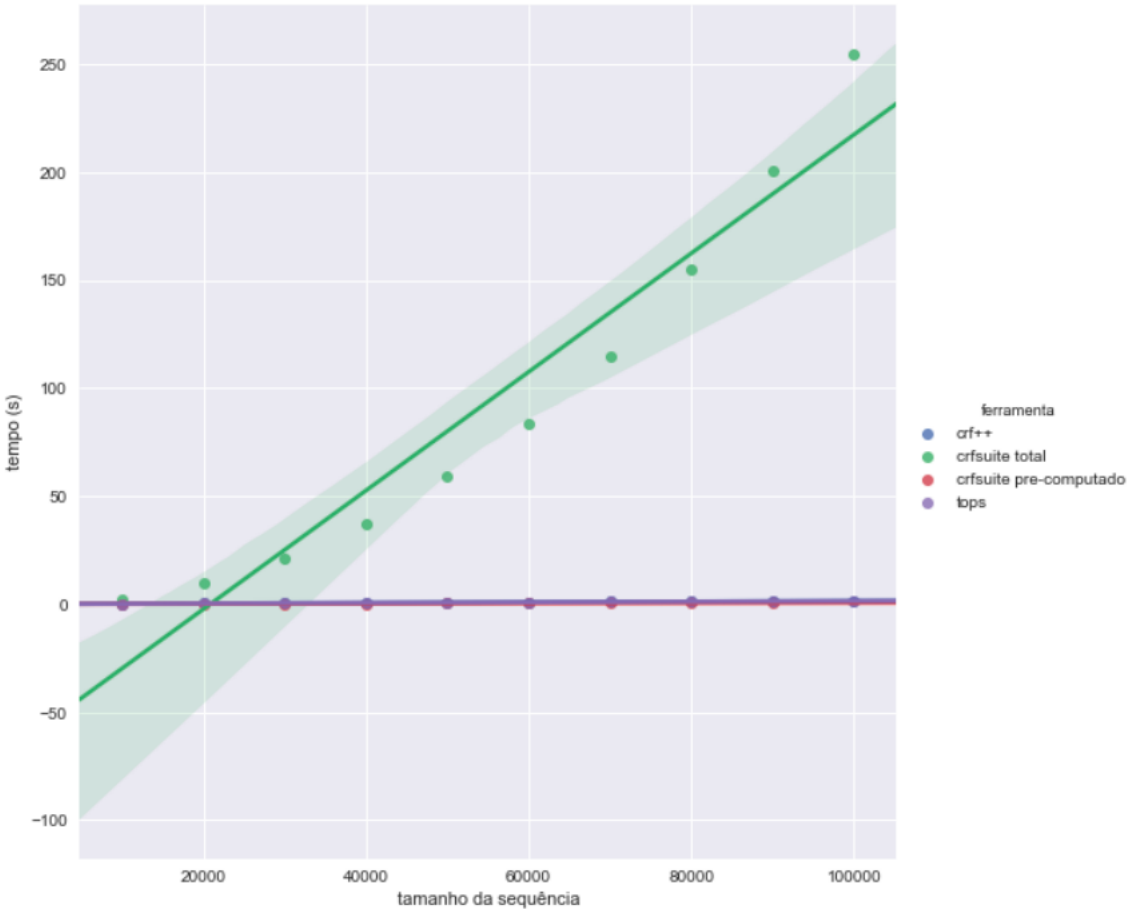

Figura 4.11: *Tempo de execução do algoritmo de viterbi nas 3 ferramentas analisadas (contando o tempo de pré-processamento do CRFSuite)*

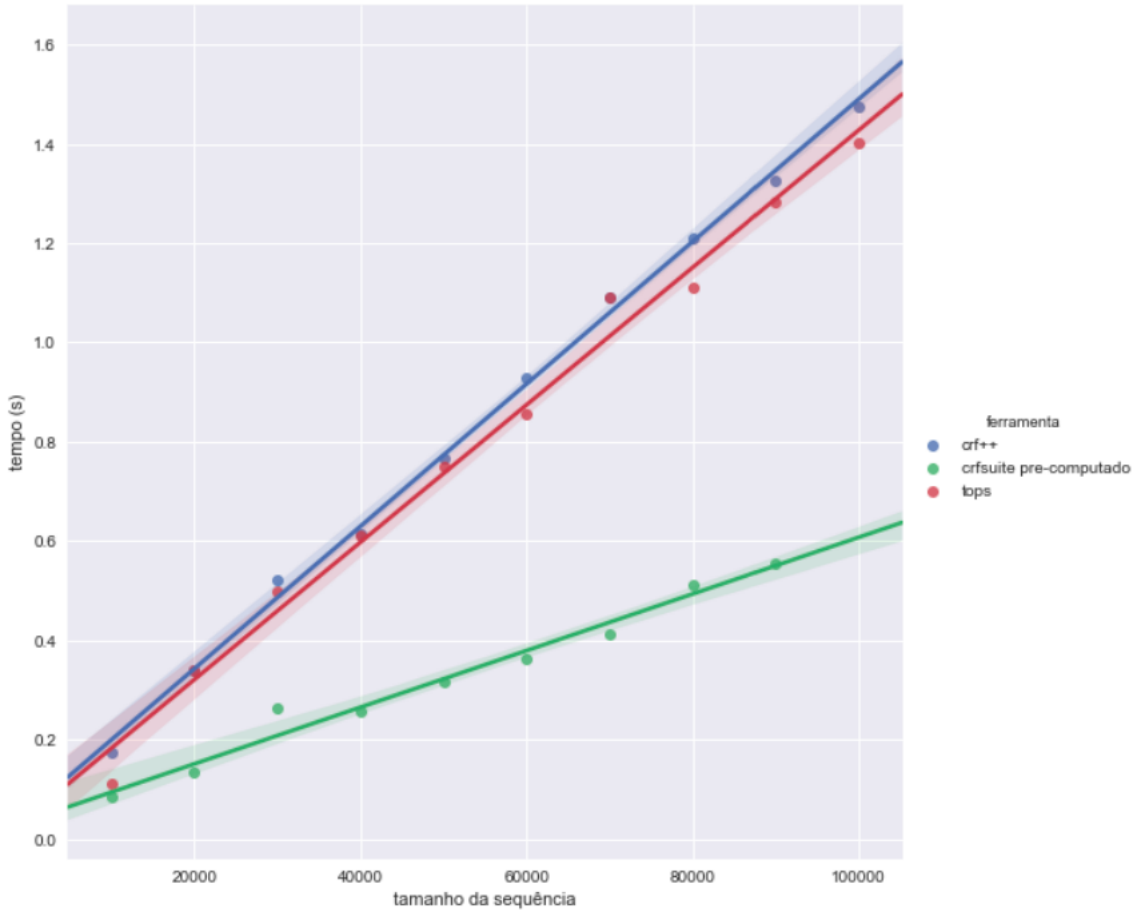

Figura 4.12: *Tempo de execução do algoritmo de viterbi nas 3 ferramentas analisadas (descontando o tempo de pré-processamento do CRFSuite)*

Capítulo 5

Preditor de Início de Sítio de Transcrição

A região promotora é composta por uma série de sequências de DNA de cerca de 5 a 10 nucleotídeos que funcionam como sítios de ligação de fatores de transcrição (Ladunga, 2010). Cada promotor é gene-específico e pode ter uma arquitetura complexa, o que dificulta sua identificação utilizando métodos *in silico* (Shahmuradov *et al.*, 2017). A região conhecida como *core* de um promotor está localizada a cerca de 40 a 60 nucleotídeos antes do sítio de início de transcrição (TSS, do inglês *Transcription Start Site*) que pode, ou não, conter uma região chamada de TATA-Box. Essa região é chamada de TATA-Box pois is rica em nucleotídeos Timina (T) e Adenosina (A) e está localizada a uma distância de cerca de 25 a 35 nucleotídeos do TSS (Yamamoto *et al.*, 2011). Os promotores que contem um TATA-box são chamados de *TATA+* e o que não contem são chamados de *TATA-*.

A principal referência para determinar a região correspondente ao promotor *core* é o TSS. O sinal do TSS pode ser determinado a partir de experimentos in vivo em molécula de pré-mRNA utilizando técnicas como OligoCap, CAGE, deepCAGE e PEAT (Dreos *et al.*, 2017; Narlikar e Ovcharenko, 2009). Entretanto esses experimentos tem um custo elevado o que torna muitas vezes seu uso proibitivo (Dreos *et al.*, 2017; Ladunga, 2010). Uma provável consequência disso é o número reduzido de organismos, apenas 14, cobertos pelo EPD(Dreos *et al.*, 2017), o principal repositório de sequência de promotores validados.

Uma alternativa é fazer análise *in silico*. Entretanto essa não é uma tarefa fácil pois o TSS é um sinal degenerado de apenas 2 nucleotídeos (Yamamoto *et al.*, 2011). Segundo Abeel *et al.* (2009), até o momento as ferramentas de predição de promotores *core in silico* podem ser divididas em 2 categorias: As que calculam uma pontuação para cada um dos nucleotídeos e escolhem o nucleotídeo da posição com a potuação mais alta como sendo o TSS; e as que procuram por motivos pré determinados.

Algumas técnicas foram propostas utilizando redes neurais (Bajic *et al.*, 2004), máquinas de vetores de suporte (Sonnenburg *et al.*, 2006) e mapas auto-organizáveis (Abeel *et al.* , 2008b). Entretanto nenhuma dessas ferramentas ultrapassa os 35% de acerto da posição do TSS aceitando uma janela de erro de 500 nucleotídeos (Abeel *et al.*, 2009). Nos últimos anos, 2 ferramentas foram propostas elevando a precisão da predição de TSS: TIPR (Morton *et al.* , 2015) e TSSPlant (Shahmuradov *et al.*, 2017).Nenhum destes modelos pode ser treinado pelo usuário. TIPR apenas fornece modelos para predição de *M. musculus* e *A. thaliana*, e

TSSPlant fornece apenas um modelo geral, aplicado a qualquer planta.

Utilizando a implementação de CRF apresentada neste trabalho, desenvolvemos, em conjunto com o aluno de doutorado Mauro Medeiros, um preditor de TSS que utiliza uma abordagem inédita baseada na segmentação/rotulação de sequências.

5.1 Modelo

TSSFinder é um caracterizador de sequências promotoras baseado em um LCCRF para o qual a sequência de observações x é uma sequência de nucleotídeos terminada pelo primeiro nucleotídeo de um sítio de iniciação (A) e os rótulos correspondem aos vários elementos e sinais da região promotora proximal.

Assim, a sequência de rótulos y, que queremos predizer, é definida pela máquina de estados apresentada na figura 5.1. Nosso preditor procura encontrar a sequência de rotulos que maximiza a probabilidade condicional $p(y|x)$.

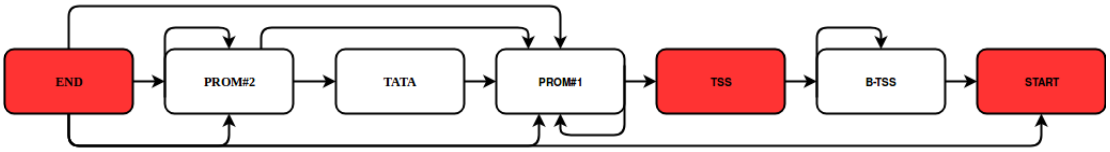

Figura 5.1: *Máquina de estados representando o CRF utilizado no MYOP-PROM. A partir do estado* Prom#1 *decide-se se o promotor é* TATA+ *ou* TATA-*. Os estados* TSS *e* TATA-box *foram subdivididos em 7 estados, que aqui omitimos por simplicidade.*

Definimos dois tipos de funções de características: *bigrams*, que representam as transições entre rótulos, e *unigrams*, que associam os rótulos às possíveis observações e registram as distâncias dos vários elementos em relação ao códon de inicação de tradução do gene.

Utilizando a notação que introduzimos nesse trabalho, podemos definir as funções de características *bigrams* a partir da figura 5.1. Para a caracterização dos rótulos utilizamos os *unigrams* da forma:

$$1_{\{y=Y\}}1_{\{x_{i-a}=N\}}1_{\{x_{i-b}=N'\}} \tag{5.1}$$

para todo $a \neq b$, $0 \leq a \leq 3$, $0 \leq b \leq 3$, $N, N' \in \{A, C, G, T\}$ e Y é um dos estados definidos pela máquina de estados da figura 5.1. Ou seja, para cada rótulo y em uma posição i observamos uma janela de tamanho 3 nucleotídeos. Nessa janela criamos uma função de característica diferente para cada um dos possíveis pares de nucleotídeos. Este tipo de *unigram* é amplamente utilizado em LCCRFS (Lafferty *et al.*, 2001; Sutton, 2008; Wallach, 2004).

Para melhorar a precisão do nosso modelo, adicionamos um conjunto de funções de características *unigram* que capturam a distribuição das distâncias entre um sinal e o sítio

de iniciação:

$$fd_D(y_t, y_{t-1}, x) = \begin{cases} 1 & \text{if } D - 50 \leq T - t \leq D \\ 0 & casocontrrio \end{cases} \tag{5.2}$$

para todo $50 \leq D \leq 600$ que é múltiplo de 50. Ou seja, as distâncias dos entre os elemento dos componentes da região promotora e o códon de inicação são dividida em bins de tamanho 50. Para cada um desse bins e cada rótulo uma função de característica nova é criada. O tamanho 50 foi obtigo por exerimentações sucessivas utiliando intervalos 50, 100, e 150. Intervalos menores não foram utilizados para reduzir o número total de features e, consequentemente, o tempo de treinamento.

5.1.1 Ferramentas

O resultado final deste experimento foi o desenvolvimento da ferramenta TSSFinder, um caracterizador das regiões de *core promoters* e genomas. Esta ferramenta é constituída de dois scripts em *Python* para implementar a ferramenta TSSFinder:

- **ttsfinder** - Recebe como parâmetro um modelo, um arquivo BED contendo anotação dos sítios de início, um arquivo FASTA contendo as sequências genômicas a serem analisadas e gera como saída 2 arquivos BED, um contendo a localização dos TSSs e outro contendo a localização dos TATA-boxes.
- **tssfinder-train** - Recebe como parâmetro 3 arquivos BED contendo anotação dos sítios de início, TATA-boxes e TSSs e um arquivo FASTA com as sequências genômicas. Como saída, o programa gerará um diretório contendo um modelo treinado.

5.2 Validação

Para medirmos a acurácia do preditor criado, utilizamo um processo de validação cruzada.

5.2.1 Datasets

Nosso preditor foi validado utilizando datasets de 6 organismos diferentes: *Arabidopsis thaliana*, *Drosophila melanogaster*, *Gallus gallus*, *Homo sapiens*, *Oryza sativa* and *Saccharomyces cerevisiae*. Para compará-lo aos preditores TIPR e TSSPlant, utilizamos apenas o dataset de *Arabidopsis thaliana*, único em comum organimos suportado pelos preditores TIPR e TSSPlant.

Selecionamos 19.620 promotores de *A. thaliana*, 14.750 de *D. melanogaster*, 5.595 de *G. gallus*, 14,605 de *H. sapiens* e 4.685 de *S. cerevisiae* do banco de dados EPD (Dreos *et al.* , 2017). Todos os promotores selecionados são de genes que que possuiam apenas um único sítio de iniciaçÃo anotado. A maioria desses promotores também tinha o TATA-box anotado. Já para o dataset de *O. sativa*, selecionamos 45.540 promotores que tinham TSSs validados a

partir de sequências de cDNA *full length* de *Oryza sativa ssp. japonica cv. Nipponbare* (Os-Nipponbare-Reference-IRGSP-1.0) obtidas do banco de dados RAP-DB (*Rice Annotation Project*) (Tanaka *et al.*, 2008).

5.2.2 Procedimento

Dado o tamanho dos nossos conjuntos de dados e o tempo requerido para trainamento dos modelos, utilizamos uma variação do processo de validação cruzada de fator 5 (*5-fold cross validation*). Cada dataset foi dividido aleatoriamente em 5 partes de tamanhos iguais. A partir disso efetuamos 5 experimentos, onde em cada um dos experimentos selecionamos 1 das partes para treinar e as outras 4 para rotulação utilizando o modelo treinado.

5.2.3 Resultados

Os resultados foram resumidos na tabela 5.1 e nos histogramas da figura 5.2. Como podemos notar, nosso método generaliza bem para diferentes organismos. Além disso, mostramos que, mesmo sem anotação de TATA-boxes nosso preditor foi capaz de obter bons resultados na predição de TSS de *O. sativa*.

Organismo	Erro máximo 50 (nt)	Erro máximo 100 (nt)	Erro máximo 150 (nt)	Erro máximo 200 (nt)	Erro máximo 250 (nt)
Arabidopsis thaliana	67.7 (± 0.6)	83.4 (± 0.4)	86.3 (± 1.2)	87.8 (± 1.6)	94.4 (± 0.7)
Drosophila melanogaster	48.4 (± 1.7)	60.3 (± 1.4)	67.7 (± 1.0)	72.0 (± 0.9)	76.1 (± 0.7)
Gallus gallus	52.9 (± 0.8)	74.2 (± 0.7)	84.7 (± 0.5)	90.8 (± 0.5)	94.5 (± 0.6)
Homo sapiens	32.2 (± 2.9)	47.7 (± 1.6)	57.0 (± 2.3)	64.6 (± 1.5)	70.0 (± 1.3)
Saccharomyces cerevisiae	75.7 (± 0.6)	94.9 (± 0.4)	99.9 (± 0.0)	99.9 (± 0.0)	99.9 (± 0.0)
Oryza sativa	47.1 (± 0.7)	68.4 (± 0.8)	79.1 (± 0.8)	87.2 (± 0.8)	93.3 (± 0.7)

Tabela 5.1: *Resultados do procedimento de validação do TSSFinder. Cada coluna apresenta a acurácia para uma dada janela de erro.*

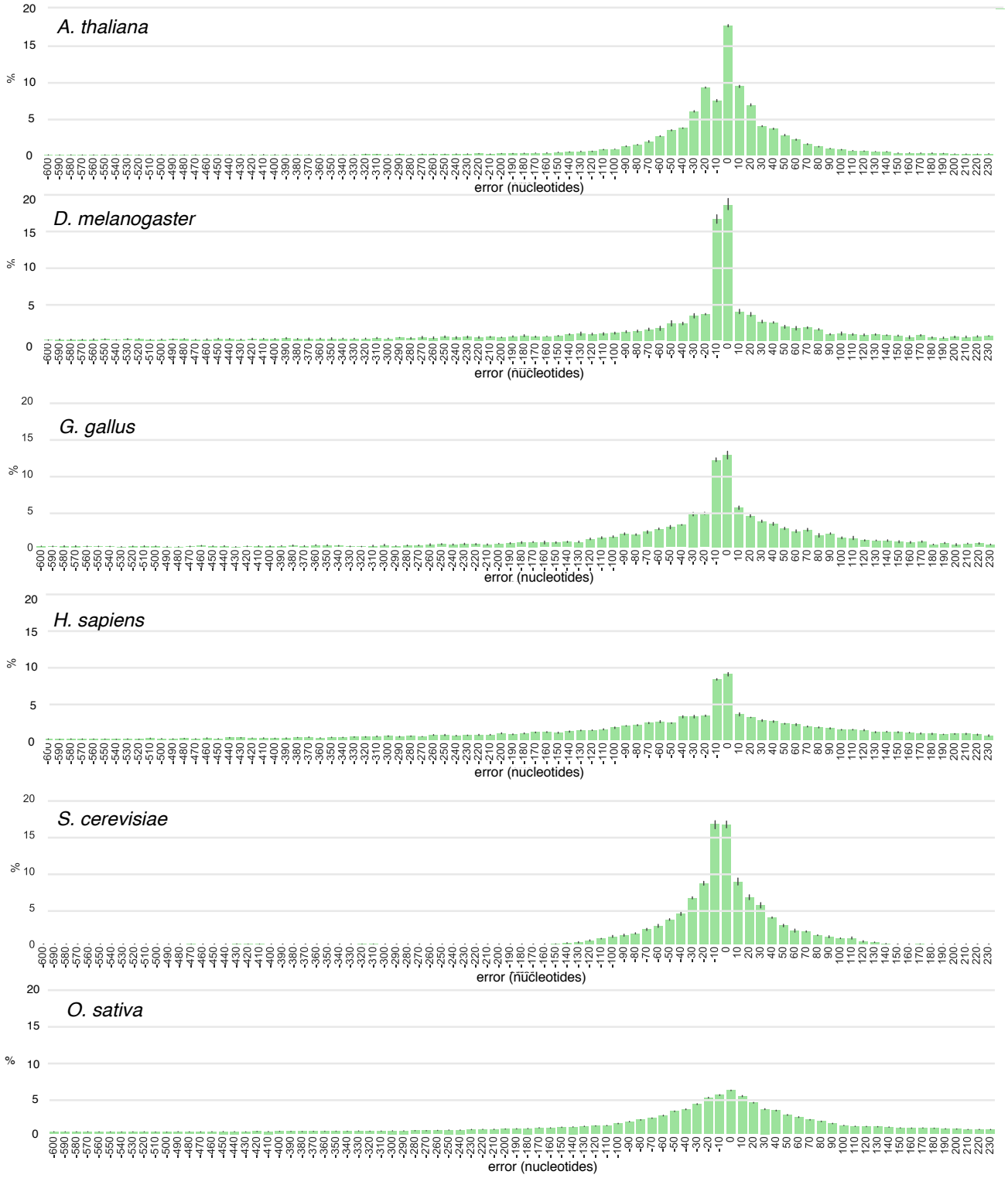

Figura 5.2: *Resultados do procedimento de validação do TSSFinder. Histograma de distância entre os TSS predito e o TSS real. Cada barra corresponde a um intervalo de 10 nucleotídeos.*

5.3 Comparação com outros preditores

Utilizamos o dataset de *A. thaliana* para comparar o TSSFinder com as ferramentas TIPR e TSSPlant. Isso porque, como dissemos anteriormente, os preditores TIPR e TSSPlant não podem ser treinados, tornando-se necessário aplicação em um organismo onde todos os preditores pudessem ser utilizados. Para isso removemos um dos folds de treinamento da fase anterior do conjunto de comparação e o utilizamos para treinamento do TSSFinder. As outras sequencias foram utilizadas para predição. Para TSSFinder e TSSPlant utilizamos, para cada gene, a região de 2000 nucleotídeos anterior ao códon de inicio de tradução. Para o TIPR, utilizamos, para cada gene, uma região de 10.001 nucleotídeos que incluia os 5000 nucleotídeos anteriores e 5000 nucleotídeos posteriores ao TSS, bem como o próprio TSS.

A tabela 5.2 e a figura 5.3 apresentam os resultados comparativos, comprovando que o TSSFinder obteve melhores resultados.

Preditor	Erro máximo 50 (nt)	Erro máximo 100 (nt)	Erro máximo 150 (nt)	Erro máximo 200 (nt)	Erro máximo 250 (nt)
TSSFinder	69.4	84.3	90.2	92.7	93.8
TIPR	42.7	52.3	56.1	58.07	59.3
TSSPlant	26.1	36.4	43.8	50.6	56.7

Tabela 5.2: *Resultados do procedimento de comparação entre os preditores TSSFinder, TIPR e TSSPlant. Cada coluna apresenta a acurácia para uma dada janela de erro.*

5.4 Curva de treinamento

Em nosso procedimento de validação utilizamos conjuntos grandes de treinamento, todos maiores do que 2900 sequências. Numa situação real não se pode esperar um conjunto tão grande de sequências préviamente validadas para treinarmos novos modelos. Desta maneira realizamos um experimento para avaliar qual o tamanho mínimom necessário para atingir-se uma boa precisão. Neste experimento utilizamos um dos *folds* utilizados na validacão de *A. thaliana* e fomos reduzindo sucessivamente o tamanho do conjunto de treinamento pela metade, medindo em seguida os resultaodos de precisão. O resultado pode ser observado na tabela 5.3.

Como podemos observar, mesmo com apenas 569 sequências de treinamento os resultados de precisão já chegam a 90% do máximo alcançado no experimento. Isso indica que o preditor pode ser usado com sucesso mesmo com um número reduzido de transcritos completos disponível. Considerando a tecnologia atual, onde os projeto de sequenciamento de genoma são precedidos por projetos de RNAseq, com sequenciamento massivo de transcritos,

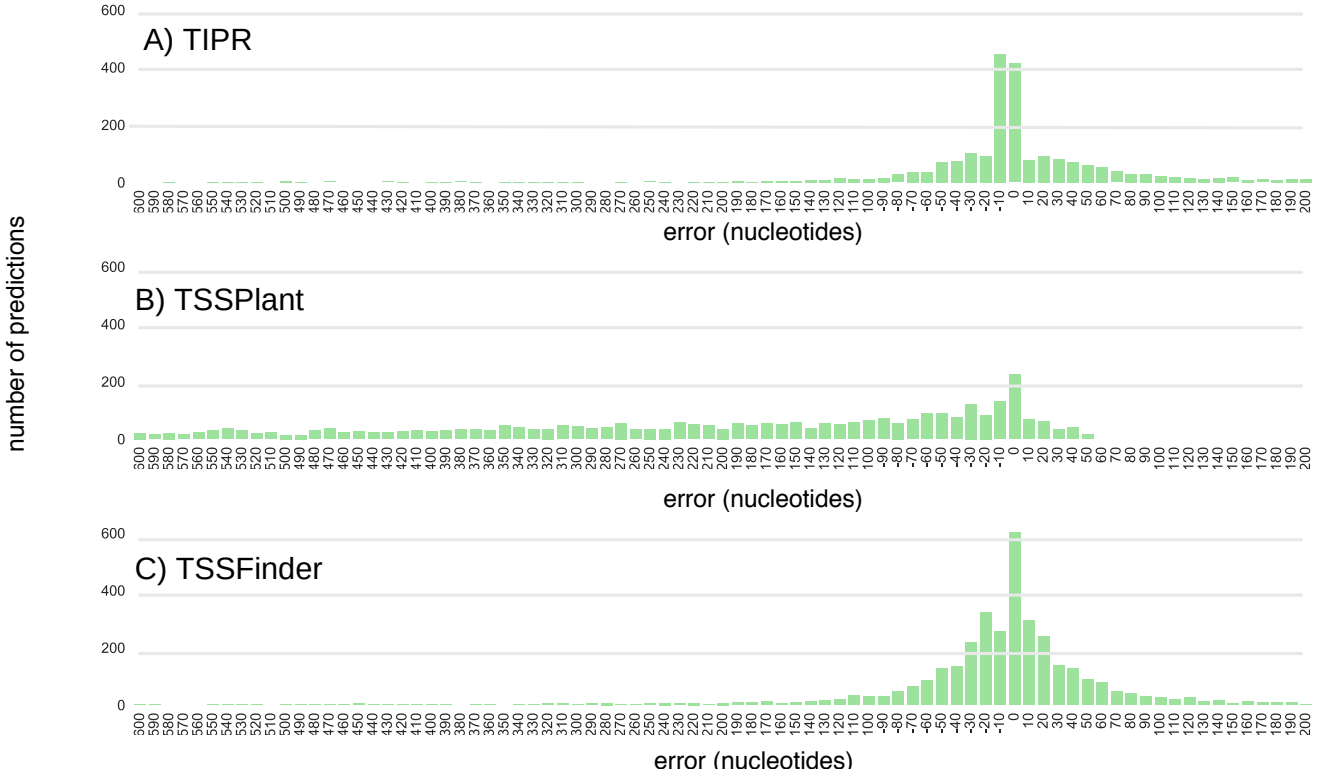

Figura 5.3: *Resultados do procedimento de comparação entre os preditores TSSFinder, TIPR e TSSPlant. Cada barra corresponde a um intervalo de 10 nucleotídeos.*

Tamanho do dataset	Erro máximo 50 (nt)	Erro máximo 100 (nt)	Erro máximo 150 (nt)	Erro máximo 200 (nt)	Erro máximo 250 (nt)
4554	46.8	67.2	78.4	86.8	93.1
2277	45.6	67.1	78.8	87.4	94.0
1138	44.0	64.7	76.7	85.3	92.1
569	42.4	64.5	76.7	86.0	93.5
284	40.6	62.1	74.6	84.0	91.2

Tabela 5.3: *Verificação da queda de acurácia quando o tamanho do dataset de treinamento diminui.*

o TSSFinder torna-se plenamente utilizável para a realização de uma melhor anotação dos genomas.

5.4.1 Tempo de predição

Outro ponto de destaque é que nossa ferramenta faz predições cerca de 30 vezes mais rápido do que o TIPR e o TSSPlant, como mostrado na tabela 5.4

	TSSFinder (1 CPU)	TSSPlant (1 CPU)	TIPR (5 CPUs)	TIPR (1 CPU)
Time	0h12min26s	6h39min48s	274h8min30s	1370h45min12s

Tabela 5.4: *Tempo necessário para predizer 14.750 sequências de* A. thaliana *em um computador Intel(R) Xeon(R) E5-2690 de 2.90GHz, com 32 núcleos e 256Gb RAM*

Capítulo 6

Conclusão e Considerações Futuras

Neste trabalho apresentamos uma ferramenta para a definição de CRFs bem como algoritmos eficientes para rotulação e treinamento. Além disso, elaboramos uma linguagem gráfica simples, baseada em máquina de estados, para facilitar a modelagem e entendimento de CRFs. Mostramos também que é possível construir um preditor de gene utilizando nossa implementação de CRF. E por fim, desenvolvemos um novo preditor de TSS que, além de mais flexível, apresenta resultados superiores aos atuais preditores de TSS.

Como trabalho futuro, podemos diminuir a distância existente entre a linguagem gráfica proposta neste trabalho e a definição do modelo em C++. Um bom exemplo é a versão original do ToPS (Kashiwabara *et al.*, 2013) que utiliza uma linguagem textual, muito próxima da matemática, para especificar modelos.

Com a implementacão do preditor de genes em CRF e os bons resultados do modelo de TSS, os próximos passos serão integrar os dois modelos para tentar resolver um dos principais problemas verificados em nossos testes de predição: erro na localização do sítio de início de tradução. Nos experimentos realizados pelo nosso grupo de pesquisa, boa parte dos erros cometidos pelo MYOP ocorrem durante a identificação do códon de início. Se ancorarmos mais sinais, como o TSS e TATA-box, podemos melhorar esse problema. Além disso, será possível também integrar no modelo de predição de genes informações sobre mapeamento de sequências expressas de mRNA, aumentando a precisão principalmente da predição de sítios de splicing, e eventualmente sinais como mapeamento de experimentos de imunoprecipitação.

Podemos também estender nossa implementação de CRF para modelar diferentes tipos de dependências, como em RNAs, cujos elementos tem dependências arbitrárias entre eles.

Finalmente, a implementação desenvolvida de CRFs para este trabalho não tem sua aplicação restrita ã area de genômica. O arcanbouço ToPS permite o uso de quaisquer alfabetos para as sequências a serem analisadas. Desta maneira esperamos que a aplicabilidade do sistema cobrirá eventualmente todo o espectro de aplicações de Semi-CRFs

Referências Bibliográficas

Abadi *et al.* **(2016)** Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng e Google Brain. TensorFlow: A System for Large-Scale Machine Learning TensorFlow: A system for large-scale machine learning. Em *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, páginas 265–284. ISBN 978-1-931971-33-1. doi: 10.1038/nn.3331. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi. Citado na pág. 44

Abeel *et al.* **(2008a)** Thomas Abeel, Yvan Saeys, Eric Bonnet, Pierre Rouzé e Yves Van de Peer. Generic eukaryotic core promoter prediction using structural features of DNA. *Genome research*, 18(2):310–23. ISSN 1088-9051. doi: 10.1101/gr.6991408. URL http://www.ncbi.nlm.nih.gov/pubmed/18096745http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2203629. Citado na pág. 2

Abeel *et al.* **(2008b)** Thomas Abeel, Yvan Saeys, Pierre Rouzé e Yves Van de Peer. ProSOM: core promoter prediction based on unsupervised clustering of DNA physical profiles. *Bioinformatics (Oxford, England)*, 24(13):24–31. ISSN 1367-4811. doi: 10.1093/bioinformatics/btn172. URL http://www.ncbi.nlm.nih.gov/pubmed/18586720http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2718650. Citado na pág. 2, 53

Abeel *et al.* **(2009)** Thomas Abeel, Yves Van de Peer e Yvan Saeys. Toward a gold standard for promoter prediction evaluation. Em *Bioinformatics*, volume 25. ISBN 1367-4811. doi: 10.1093/bioinformatics/btp191. Citado na pág. 31, 32, 53

Alberts *et al.* **(2014)** Bruce Alberts, Alexander Johnson, Julian Lewis, David Morgan, Martin Raff, Keith Roberts e Peter Walter. *Molecular Biology of the Cell 6e*, volume 6. ISBN 1136844422. doi: 10.1002/1521-3773(20010316)40:6<9823::AID-ANIE9823>3.3.CO;2-C. Citado na pág. 25, 26

Altschul *et al.* **(1990)** Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers e David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410. ISSN 00222836. doi: 10.1016/S0022-2836(05)80360-2. Citado na pág. 28

Angelova *et al.* **(2010)** Mihaela Angelova, Slobodan Kalajdziski e Ljupco Kocarev. Computational Methods for Gene Finding in Prokaryotes. *ICT Innovations 2010, Web Proceedings*, (March 2016): 11–20. doi: ISSN1857-7288. URL http://ictinnovations.org/2010. Citado na pág. 27

Bairoch *et al.* **(2004)** Amos Bairoch, Brigitte Boeckmann, Serenella Ferro e Elisabeth Gasteiger. Swiss-Prot: juggling between evolution and stability. *Briefings in bioinformatics*, 5(1):39–55. ISSN 14675463. doi: 10.1093/bib/5.1.39. Citado na pág. 28

Bajic *et al.* **(2004)** Vladimir B Bajic, Sin Lam Tan, Yutaka Suzuki e Sumio Sugano. Promoter prediction analysis on the whole human genome. *Nature Biotechnology*, 22(11):1467–1473. ISSN 1087-0156. doi: 10.1038/nbt1032. URL http://www.nature.com/doifinder/10.1038/nbt1032. Citado na pág. 2, 53

Baydin *et al.* **(2015)** Atilim Gunes Baydin, Barak A. Pearlmutter e Alexey Andreyevich Radul. Automatic differentiation in machine learning: a survey. *arXiv preprint*, página 28. URL http://arxiv.org/abs/1502.05767. Citado na pág. 44

Bernal *et al.* **(2007)** Axel Bernal, Koby Crammer, Artemis Hatzigeorgiou, Fernando Pereira, CB Burge, S Karlin, M Stanke, S Waack, WH Majoros, M Pertea, SL Salzberg, A Krogh, WH Majoros, SL Salzberg, MQ Zhang, D Kulp, D Haussler, MG Reese, FH Eeckman, MS Gelfand, AA Mironov, PA Pevzner, E Birney, M Clamp, R Durbin, I Korf, P Flicek, D Duan, MR Brent, IM Meyer, R Durbin, SS Gross, MR Brent, A Krogh, C Mathe, MF Sagot, T Schiex, P Rouze, P Flicek, E Keibler, P Hu, I Korf, MR Brent, G Rätsch, S Sonnenburg, J Srinivasan, H Witte, KR Müller, J Lafferty, A McCallum, F Pereira, S Sarawagi, WW Cohen, K Crammer, O Dekel, J Keshet, S Shalev-Shwartz, Y Singer, B Juang, L Rabiner, R Raina, Y Shen, AY Ng, A McCallum, A Kasprzyk, D Keefe, D Smedley, D London, W Spooner, EE Snyder, GD Stormo, M Burset, R Guigo, R Guigo, P Agarwal, JF Abril, M Burset, JW Fickett, S Rogic, AK Mackworth, FB Ouellette, E Keibler, MR Brent, K Crammer, M Collins, SL Salzberg, A Delcher, S Kasif, O White, A Fedorov, L Fedorova, V Starshenko, V Filatov e E Grigor'ev. Global Discriminative Learning for Higher-Accuracy Computational Gene Prediction. *PLoS Computational Biology*, 3(3):e54. ISSN 1553-734X. doi: 10.1371/journal.pcbi.0030054. URL http://dx.plos.org/10.1371/journal.pcbi.0030054. Citado na pág. 31

Bernal *et al.* **(2012)** Axel Bernal, Koby Crammer e Fernando Pereira. Automated gene-model curation using global discriminative learning. *Bioinformatics*, 28(12):1571–1578. ISSN 13674803. doi: 10.1093/bioinformatics/bts176. Citado na pág. 1, 27, 31, 41

Borodovsky e Mcininch (1993) Mark Borodovsky e James Mcininch. GeneMark: Parallel Gene Recognition for Both DNA Strands, 1993. ISSN 00978485. Citado na pág. 30

Brunak *et al.* **(1991)** S Brunak, J Engelbrecht e S Knudsen. Prediction of human mRNA donor and acceptor sites from the DNA sequence. *J Mol Biol*, 220(1):49–65. ISSN 0022-2836 (Print) 0022-2836 (Linking). doi: 0022-2836(91)90380-O[pii]. URL http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=2067018. Citado na pág. 29

Burge (1997) Chris Burge. Identification of genes in human genomic DNA, 1997. Citado na pág. 2, 5, 28, 29, 37

Burge e Karlin (1997) Chris Burge e Samuel Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268(1):78–94. ISSN 00222836. doi: 10.1006/jmbi.1997.0951. Citado na pág. 27, 30

Chan *et al.* **(2016)** William Chan, Navdeep Jaitly, Quoc Le e Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. Em *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, volume 2016-May, páginas 4960–4964. ISBN 9781479999880. doi: 10.1109/ICASSP.2016.7472621. Citado na pág. 1

Chen *et al.* **(2018)** Liang Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy e Alan L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848. ISSN 01628828. doi: 10.1109/TPAMI.2017.2699184. Citado na pág. 23

Chen *et al.* **(2015)** Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang e Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. Relatório técnico. URL http://arxiv.org/abs/1512.01274. Citado na pág. 44

Culotta e Mccallum (2001) Aron Culotta e Andrew Mccallum. Confidence Estimation for Information Extraction. *Proceedings of HLTNAACL 2004 Short Papers on XX HLTNAACL 04*, 64(3):109–112. ISSN 1059910X. doi: 10.3115/1613984.1614012. Citado na pág. 2

DeCaprio *et al.* (2007) David DeCaprio, Jade P. Vinson, Matthew D. Pearson, Philip Montgomery, Matthew Doherty e James E. Galagan. Conrad: Gene prediction using conditional random fields. *Genome Research*, 17(9):1389–1398. ISSN 10889051. doi: 10.1101/gr.6558107. Citado na pág. 1, 2, 30

Dreos *et al.* (2017) Rene Dreos, Giovanna Ambrosini, Romain Groux, Rouaida Cavin Perier e Philipp Bucher. The eukaryotic promoter database in its 30th year: Focus on non-vertebrate organisms. *Nucleic Acids Research*, 45(D1):D51–D55. ISSN 13624962. doi: 10.1093/nar/gkw1069. Citado na pág. 53, 55

Duda *et al.* (2000) Richard O. Duda, Peter E. Hart e David G. Stork. *Pattern Classification*. Wiley-Interscience, 2º ed. Citado na pág. 6

Durbin *et al.* (1998) R Durbin, S Eddy, a Krogh e G Mitchison. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. *Analysis*, página 356. ISSN 0263-6484. doi: 10.1017/CBO9780511790492. URL http://eisc.univalle.edu.co/cursos/web/material/750068/1/6368030-Durbin-Et-Al-Biological-Sequence-Analysis-CUP-2002-No-OCR.pdf. Citado na pág. 17

Ferreira *et al.* (2016) Renato Cordeiro Ferreira, Igor Bonadio e Alan Mitchell Durham. Secretary Pattern: Decreasing Coupling While Keeping Reusability. *Proceedings of the 11th Latin-American Conference on Pattern Languages of Programming*, páginas 14:1–14:11. Citado na pág. 45

Fickett e Tung (1992) J W Fickett e C S Tung. Assessment of protein coding measures. *Nucleic acids research*, 20(24):6441–6450. ISSN 0305-1048. doi: 10.1093/nar/20.24.6441. Citado na pág. 30

Frith *et al.* (2008) Martin C. Frith, Eivind Valen, Anders Krogh, Yoshihide Hayashizaki, Piero Carninci e Albin Sandelin. A code for transcription initiation in mammalian genomes. *Genome Research*, 18(1):1–12. ISSN 10889051. doi: 10.1101/gr.6831208. Citado na pág. 31

Goel *et al.* (2013) Neelam Goel, Shailendra Singh e Trilok Chand Aseri. A comparative analysis of soft computing techniques for gene prediction. *Analytical Biochemistry*, 438(1):14–21. ISSN 00032697. doi: 10.1016/j.ab.2013.03.015. Citado na pág. 27

Gross e Brent (2006) Samuel S. Gross e Michael R. Brent. Using Multiple Alignments to Improve Gene Prediction. *Journal of Computational Biology*, 13(2):379–393. ISSN 1066-5277. doi: 10.1089/cmb.2006.13.379. URL http://www.liebertonline.com/doi/abs/10.1089/cmb.2006.13.379. Citado na pág. 2

Gross *et al.* (2007) Samuel S Gross, Chuong B Do, Marina Sirota, Serafim Batzoglou, C Burge, S Karlin, A Bernal, K Crammer, A Hatzigeorgiou, F Pereira, S Batzoglou, L Pachter, JP Mesirov, B Berger, ES Lander, V Bafna, DH Huson, I Korf, P Flicek, D Duan, MR Brent, SS Gross, MR Brent, M Alexandersson, S Cawley, L Pachter, G Parra, P Agarwal, JF Abril, T Wiehe, JW Fickett, R Guigo, JS Pedersen, J Hein, A Siepel, D Haussler, D Carter, R Durbin, M Arumugam, C Wei, R Brown, M Brent, C Wei, M Brent, E Birney, M Clamp, R Durbin, S Djebali, F Delaplace, H Crollius, M Stanke, S Waack, R Guigo, P Flicek, J Abril, A Reymond, J Lagarde, F Denoeud, S Antonarakis, M Ashburner, V Bajic, E Birney, R Castelo, E Eyras, C Ucla, T Gingeras, J Harrow, T Hubbard, S Lewis, M Reese, C Cortes, V Vapnik, J Lafferty, A McCallum, F Pereira, A Siepel, D Haussler, M Blanchette, WJ Kent, C Riemer, L Elnitski, AF Smit, KM Roskin, R Baertsch, K Rosenbloom, H Clawson, ED Green, M Wang, J Buhler, M Brent, D Karolchik, R Baertsch, M Diekhans, TS Furey, A Hinrichs, YT Lu, KM Roskin, M Schwartz, CW Sugnet, DJ Thomas, E Keibler, M Brent, WJ Kent, DA Benson, I Karsch-Mizrachi, DJ Lipman, J Ostell, DL Wheeler, TMP Team, MR Brent, R Guigo, E Dermitzakis,

P Agarwal, C Ponting, G Parra, A Reymond, J Abril, E Keibler, R Lyle, C Ucla, A Baross, YSN Butterfield, SM Coughlin, T Zeng, M Griffith, OL Griffith, AS Petrescu, DE Smailus, J Khattra, HL McDonald, SJ McKay, M Moksa, RA Holt, MA Marra, JQ Wu, AM Garcia, S Hulyk, A Sneed, C Kowis, Y Yuan, D Steffen, JD McPherson, PH Gunaratne, RA Gibbs, K Nigam, J Lafferty, A McCallum, T Jebara, A Pentland, CB Do, SS Gross, S Batzoglou, CB Do, DA Woods, S Batzoglou, VN Vapnik, A Culotta, D Kulp, A McCallum, D Kulp, D DeCaprio, JP Vinson, MD Pearon, P Montgomery, M Doherty, JE Galagan, SS Gross, O Russakovsky, CB Do, S Batzoglou, M Riedmiller, H Braun, W Vetterling, S Teukolsky, W Press e B Flannery. CONTRAST: a discriminative, phylogeny-free approach to multiple informant de novo gene prediction. *Genome Biology*, 8(12):R269. ISSN 1465-6906. doi: 10.1186/gb-2007-8-12-r269. URL http://genomebiology.biomedcentral.com/articles/10.1186/gb-2007-8-12-r269. Citado na pág. 27

Habibi *et al.* **(2017)** M Habibi, L Weber, M Neves, D L Wiegandt e U Leser. Deep learning with word embeddings improves biomedical named entity recognition. *Bioinformatics*, 33(14):i37–i48. doi: 10.1093/bioinformatics/btx228. URL https://www.ncbi.nlm.nih.gov/pubmed/28881963. Citado na pág. 1

Hammersley e Clifford (1971) John Hammersley e Peter Clifford. Markov fields on finite graphs and lattices. Relatório técnico. Citado na pág. 11, 12

Hebsgaard *et al.* **(1996)** S M Hebsgaard, P G Korning, N Tolstrup, J Engelbrecht, P Rouzé e S Brunak. Splice site prediction in Arabidopsis thaliana pre-mRNA by combining local and global sequence information. *Nucleic acids research*, 24(17):3439–52. ISSN 0305-1048. doi: 6s0185[pii]. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=146109&tool=pmcentrez&rendertype=abstract. Citado na pág. 29

Hsu (2013) Hwei Hsu. *Probability, Random Variables and Random Processes*, volume 53. ISBN 9788578110796. doi: 10.1017/CBO9781107415324.004. Citado na pág. 30

Jose *et al.* **(2016)** S. Jose, P. Nair, V.G. Biju, B.B. Mathew e C.M. Prashanth. Hidden Markov Model: Application towards genomic analysis. Em *Proceedings of IEEE International Conference on Circuit, Power and Computing Technologies, ICCPCT 2016*. ISBN 9781509012770. doi: 10.1109/ICCPCT.2016.7530222. Citado na pág. 1

Kamnitsas *et al.* **(2017)** Konstantinos Kamnitsas, Christian Ledig, Virginia F.J. Newcombe, Joanna P. Simpson, Andrew D. Kane, David K. Menon, Daniel Rueckert e Ben Glocker. Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. *Medical Image Analysis*, 36:61–78. ISSN 13618423. doi: 10.1016/j.media.2016.10.004. Citado na pág. 23

Kashiwabara *et al.* **(2013)** A.Y. Kashiwabara, Í. Bonadio, V. Onuchic, F. Amado, R. Mathias e A.M. Durham. ToPS: A Framework to Manipulate Probabilistic Models of Sequence Data. *PLoS Computational Biology*, 9(10). doi: 10.1371/journal.pcbi.1003234. Citado na pág. 1, 2, 5, 33, 41, 42, 43, 44, 61

Kleffe *et al.* **(1996)** Jürgen Kleffe, Klaus Hermann, Wolfgang Vahrson, Burkhardt Wittig e Volker Brendel. Logitlinear models for the prediction of splice sites in plant pre-mRNA sequences. *Nucleic Acids Research*, 24(23):4709–4718. ISSN 0305-1048. doi: 10.1093/nar/24.23.4709. Citado na pág. 29

Korf *et al.* **(2001)** I. Korf, P. Flicek, D. Duan e M. R. Brent. Integrating genomic homology into gene structure prediction. *Bioinformatics*, 17(Suppl 1):S140–S148. ISSN 1367-4803. doi: 10.1093/bioinformatics/17.suppl{_}1.S140. URL https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/17.suppl_1.S140. Citado na pág. 37

Kudo (2003) Taku Kudo. CRF++ - Yet Another CRF toolkit, 2003. Citado na pág. 33, 40, 46

Kudo *et al.* **(2004)** Taku Kudo, Kaoru Yamamoto e Yuji Matsumoto. Applying Conditional Random Fields to Japanese Morphological Analysis. Em *Proceedings of the Conference on Empirical Methods in Natural Language Processing - EMNLP'04*, volume 4, páginas 230–237. ISBN 978-1-4244-4519-6. doi: 10.1109/ICCSIT.2009.5234727. URL http://www.aclweb.org/anthology/W04-3230. Citado na pág. 1

Kulp *et al.* **(1996)** David Kulp, David Haussler, G Reese e H Eeckman. A Generalized Hidden Markov Model for the Recognition of Human Genes in DNA. *ISMB Proceedings*, páginas 134–142. Citado na pág. 8, 10

Kupiec (1992) Julian Kupiec. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 6(3):225–242. ISSN 10958363. doi: 10.1016/0885-2308(92)90019-Z. Citado na pág. 1

Ladunga (2010) Istvan Ladunga. Computational Biology of Transcription Factor Binding. *Methods in Molecular Biology*, 674(1):143–159. ISSN 19406029. doi: 10.1007/978-1-60761-854-6. URL http://www.springerlink.com/index/10.1007/978-1-60761-854-6. Citado na pág. 53

Lafferty *et al.* **(2001)** John Lafferty, Andrew McCallum e Fernando C N Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning*, 8(June):282–289. ISSN 1750-2799. doi: 10.1038/nprot.2006.61. Citado na pág. 1, 5, 13, 18, 22, 54

Lecun *et al.* **(2015)** Yann Lecun, Yoshua Bengio e Geoffrey Hinton. Deep learning, 2015. ISSN 14764687. Citado na pág. 44

Lodish *et al.* **(2008)** Harvey F Lodish, Arnold Berk, S Lawrence Zipursky, Paul Matsudaira, David Baltimore e Darnell. James. *Molecular Cell Biology*, volume 5. ISBN 0716776014. doi: 10.1016/S1470-8175(01)00023-6. Citado na pág. 25

Long *et al.* **(2015)** Jonathan Long, Evan Shelhamer e Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation ppt. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8828(c):3431–3440. ISSN 10636919. doi: 10.1109/CVPR.2015.7298965. Citado na pág. 1

Mathe *et al.* **(2002)** C. Mathe, Marie-France Sagot, Thomas Schiex e Pierre Rouzé. Current methods of gene prediction, their strengths and weaknesses. *Nucleic Acids Research*, 30(19): 4103–4117. doi: 10.1093/nar/gkf543. Citado na pág. 27, 28

Morales-Cordovilla *et al.* **(2018)** Juan A. Morales-Cordovilla, Victoria Sanchez e Martin Ratajczak. Protein alignment based on higher order conditional random fields for template-based modeling. *PLoS ONE*, 13(6). ISSN 19326203. doi: 10.1371/journal.pone.0197912. Citado na pág. 1

Morton *et al.* **(2015)** Taj Morton, Weng Keen Wong e Molly Megraw. TIPR: Transcription initiation pattern recognition on a genome scale. *Bioinformatics*, 31(23):3725–3732. ISSN 14602059. doi: 10.1093/bioinformatics/btv464. Citado na pág. 28, 32, 53

Narlikar e Ovcharenko (2009) Leelavati Narlikar e Ivan Ovcharenko. Identifying regulatory elements in eukaryotic genomes, 2009. ISSN 14739550. Citado na pág. 32, 53

Ng e Jordan (2002) Andrew Ng e Michael I. Jordan. On generative vs. discriminative classifiers: A comparison of logistic regression and naive bayes. *Proceedings of Advances in Neural Information Processing*, 28(3):169–187. ISSN 13704621. doi: 10.1007/s11063-008-9088-7. Citado na pág. 23

NHGRI (2018) NHGRI. DNA alternative splicing, 2018. URL http://www.genome.gov/Images/EdKit/bio2j_large.gif. Citado na pág. xi, 27

Okazaki (2007) Naoaki Okazaki. CRFsuite - A fast implementation of Conditional Random Fields (CRFs), 2007. Citado na pág. 33, 40, 46

Pearson e Lipman (1988) W. R. Pearson e D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448. ISSN 0027-8424. doi: 10.1073/pnas.85.8.2444. URL http://www.pnas.org/cgi/doi/10.1073/pnas.85.8.2444. Citado na pág. 28

PyTorch Community (2016) PyTorch Community. Tensors and Dynamic neural networks in Python with strong GPU acceleration, 2016. URL https://github.com/pytorch/pytorch. Citado na pág. 44

Rabiner (1989) Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, 1989. ISSN 15582256. Citado na pág. 1, 9

Rogozin e Milanesi (1997) Igor B. Rogozin e Luciano Milanesi. Analysis of donor splice sites in different eukaryotic organisms. *Journal of Molecular Evolution*, 45(1):50–59. ISSN 00222844. doi: 10.1007/PL00006200. Citado na pág. 29

Roy e Todorovic (2017) Anirban Roy e Sinisa Todorovic. Combining bottom-up, top-down, and smoothness cues for weakly supervised image segmentation. Em *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, páginas 7282–7291. ISBN 9781538604571. doi: 10.1109/CVPR.2017.770. Citado na pág. 23

Sarawagi e Cohen (2005) Sunita Sarawagi e William W Cohen. Semi-Markov Conditional Random Fields for Information Extraction. *Advances in Neural Information Processing Systems 17*, páginas 1185–1192. doi: 10.1.1.128.3524. Citado na pág. 14, 18

Sha e Pereira (2003) F Sha e F Pereira. Shallow parsing with conditional random fields. *Proceedings of the 2003 Conference of the North ...*, (June):1071–1079. doi: 10.3115/1073445.1073473. URL http://dl.acm.org/citation.cfm?id=1073473. Citado na pág. 1

Shahmuradov *et al.* (2017) Ilham A. Shahmuradov, Ramzan K. Umarov e Victor V. Solovyev. TSSPlant: A new tool for prediction of plant Pol II promoters. *Nucleic Acids Research*, 45(8). ISSN 13624962. doi: 10.1093/nar/gkw1353. Citado na pág. 32, 53

Smith e Waterman (1981) T. F. Smith e M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197. ISSN 00222836. doi: 10.1016/0022-2836(81)90087-5. Citado na pág. 28

Sonnenburg *et al.* (2006) Sören Sonnenburg, Alexander Zien e Gunnar Rätsch. ARTS: Accurate recognition of transcription starts in human. Em *Bioinformatics*, volume 22. ISBN 1367-4811 (Electronic)\r1367-4803 (Linking). doi: 10.1093/bioinformatics/btl250. Citado na pág. 53

Stanke e Waack (2003) Mario Stanke e Stephan Waack. Gene prediction with a hidden Markov model and a new intron submodel. *Bioinformatics (Oxford, England)*, 19 Suppl 2(suppl 2):215–25. ISSN 1367-4811. doi: 10.1093/BIOINFORMATICS/BTG1080. URL http://www.ncbi.nlm.nih.gov/pubmed/14534192. Citado na pág. 2, 5, 27, 28, 37

Stanke *et al.* (2006) Mario Stanke, Oliver Schöffmann, Burkhard Morgenstern, Stephan Waack, C Burge, M Stanke, S Waack, A Krogh, G Parra, B Enrique, R Guigó, G Parra, P Agarwal, J Abril, T Wiehe, J Fickett, R Guigó, I Korf, P Flicek, D Duan, MR Brent, SS Gross, MR Brent, M Alexandersson, S Cawley, L Pachter, IM Meyer, R Durbin, L Taher, O Rinner, S Gargh, A Sczyrba, B Morgenstern, JS Pedersen, J Hein, A Siepel, D Haussler, MR Brent, R Guigó, RF Yeh, LP Lim, C Burge, A Krogh, E Birney, M Clamp, R Durbin, B Brejova, DG Brown, M Li, T Vinar, JE Allen, M Pertea, SL Salzberg, M Stanke, M Stanke, R Steinkamp, S Waack, B Morgenstern, W Gish, DJ States, R Guigó, P Agarwal, J Abril, M Burset, J Fickett, JE Collins,

ME Goward, CG Cole, LJ Smink, EJ Huckle, S Knowles, JM Bye, DM Beare, I Dunham, B Morgenstern, A Dress, T Werner, M Brudno, M Chapman, B Göttgens, S Batzoglou e B Morgenstern. Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources. *BMC Bioinformatics*, 7(1):62. ISSN 14712105. doi: 10.1186/1471-2105-7-62. Citado na pág. 1, 2, 5

Stanke *et al.* (2008) Mario Stanke, Mark Diekhans, Robert Baertsch e David Haussler. Using native and syntenically mapped cDNA alignments to improve de novo gene finding. *Bioinformatics*, 24(5):637–644. ISSN 13674803. doi: 10.1093/bioinformatics/btn013. Citado na pág. 27, 42

Sutton e Mccallum (2002) Charles Sutton e Andrew Mccallum. An Introduction to Conditional Random Fields for Relational Learning. *Graphical Models*, 7:93. ISSN 14796813. doi: 10.1677/JME-08-0087. Citado na pág. 1, 5, 14, 15, 18, 34

Sutton (2008) Charles a Sutton. Efficient training methods for conditional random fields. (February):219. Citado na pág. 1, 6, 7, 23, 54

Tanaka *et al.* (2008) Tsuyoshi Tanaka, Baltazar A Antonio, Shoshi Kikuchi, Takashi Matsumoto, Yoshiaki Nagamura, Hisataka Numa, Hiroaki Sakai, Jianzhong Wu, Takeshi Itoh, Takuji Sasaki, Ryo Aono, Yasuyuki Fujii, Takuya Habara, Erimi Harada, Masako Kanno, Yoshihiro Kawahara, Hiroaki Kawashima, Hiromi Kubooka, Akihiro Matsuya, Hajime Nakaoka, Naomi Saichi, Ryoko Sanbonmatsu, Yoshiharu Sato, Yuji Shinso, Mami Suzuki, Jun-ichi Takeda, Motohiko Tanino, Fusano Todokoro, Kaori Yamaguchi, Naoyuki Yamamoto, Chisato Yamasaki, Tadashi Imanishi, Toshihisa Okido, Masahito Tada, Kazuho Ikeo, Yoshio Tateno, Takashi Gojobori, Yao-Cheng Lin, Fu-Jin Wei, Yue-ie Hsing, Qiang Zhao, Bin Han, Melissa R Kramer, Richard W McCombie, David Lonsdale, Claire C O'Donovan, Eleanor J Whitfield, Rolf Apweiler, Kanako O Koyanagi, Jitendra P Khurana, Saurabh Raghuvanshi, Nagendra K Singh, Akhilesh K Tyagi, Georg Haberer, Masaki Fujisawa, Satomi Hosokawa, Yukiyo Ito, Hiroshi Ikawa, Michie Shibata, Mayu Yamamoto, Richard M Bruskiewich, Douglas R Hoen, Thomas E Bureau, Nobukazu Namiki, Hajime Ohyanagi, Yasumichi Sakai, Satoshi Nobushima, Katsumi Sakata, Roberto A Barrero, Yutaka Sato, Alexandre Souvorov, Brian Smith-White, Tatiana Tatusova, Suyoung An, Gynheung An, Satoshi OOta, Galina Fuks, Joachim Messing, Karen R Christie, Damien Lieberherr, HyeRan Kim, Andrea Zuccolo, Rod A Wing, Kan Nobuta, Pamela J Green, Cheng Lu, Blake C Meyers, Cristian Chaparro, Benoit Piegu, Olivier Panaud e Manuel Echeverria. The Rice Annotation Project Database (RAP-DB): 2008 update. *Nucleic acids research*, 36(Database issue):1028–33. ISSN 1362-4962. doi: 10.1093/nar/gkm978. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2238920&tool=pmcentrez&rendertype=abstract. Citado na pág. 56

Tolstrup *et al.* (1997) Niels Tolstrup, Pierre Rouzé e Søren Brunak. A branch point consensus from Arabidopsis found by non-circular analysis allows for better prediction of acceptor sites. *Nucleic Acids Research*, 25(15):3159–3163. ISSN 03051048. doi: 10.1093/nar/25.15.3159. Citado na pág. 29

Vinson *et al.* (2007) Jade P Vinson, David DeCaprio, Matthew D Pearson, Stacey Luoma e James E Galagan. Comparative Gene Prediction using Conditional Random Fields. *Advances in Neural Information Processing Systems 19*, páginas 1441–1448. Citado na pág. 1, 41

Wallach (2002) Hanna Wallach. Efficient Training of Conditional Random Fields Master of Science School of Cognitive Science Division of Informatics University of Edinburgh. *Cognitive Science*. Citado na pág. 22

Wallach (2004) Hanna M Wallach. Conditional Random Fields: An Introduction. *Tutorial*, páginas 1–9. Citado na pág. 54

Yamamoto *et al.* (2011) Yoshiharu Y Yamamoto, Yohei Yoshioka, Mitsuro Hyakumachi e Junichi Obokata. Characteristics of core promoter types with respect to gene structure and expression in Arabidopsis thaliana. *DNA research : an international journal*

for rapid publication of reports on genes and genomes, 18(5):333–42. ISSN 1756-1663. doi: 10.1093/dnares/dsr020. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3190954&tool=pmcentrez&rendertype=abstract. Citado na pág. 32, 53

Yang e Cardie (2012) Bishan Yang e Claire Cardie. Extracting Opinion Expressions with semi-Markov Conditional Random Fields. *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, (July): 1335–1345. doi: http://www.aclweb.org/anthology/D12-1122.pdf. Citado na pág. 1

Yao *et al.* **(2014)** Kaisheng Yao, Baolin Peng, Geoffrey Zweig, Dong Yu, Xiaolong Li e Feng Gao. Recurrent Conditional Random Field for Language Understanding. *Icassp 2014*, páginas 4077–4081. ISSN 15206149. doi: 10.1109/ICASSP.2014.6854368. URL http://research.microsoft.com/apps/pubs/default.aspx?id=210167. Citado na pág. 1

Yu e Kobayashi (2003) Shun-zheng Yu e Hisashi Kobayashi. An Efficient Forward – Backward Algorithm for an Explicit-Duration Hidden Markov Model. *IEEE Signal Processing Letters*, 10 (1):11–14. ISSN 1070-9908. doi: 10.1109/LSP.2002.806705. Citado na pág. 1

Zhang e Marr (1993) M. O. Zhang e T. G. Marr. A weight array method for splicing signal analysis. *Bioinformatics*, 9(5):499–509. ISSN 14602059. doi: 10.1093/bioinformatics/9.5.499. Citado na pág. 29

Zhang (2002) Michael Q Zhang. Computational prediction of eukaryotic protein-coding genes. *Nature reviews. Genetics*, 3(9):698–709. ISSN 1471-0056. doi: 10.1038/nrg890. URL http://dx.doi.org/10.1038/nrg890. Citado na pág. 27