Conditional Random Fields

Andrea Passerini
passerini@disi.unitn.it

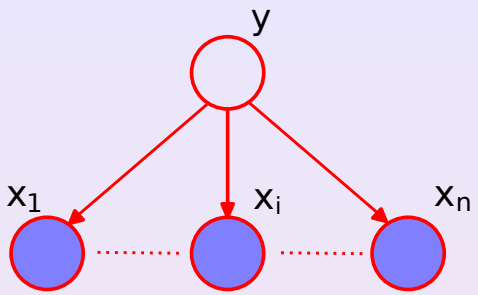
Statistical relational learning

Generative vs discriminative models

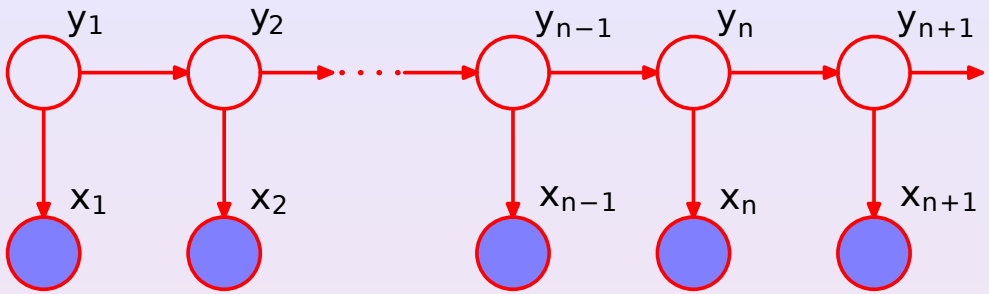
joint distributions

- Traditional graphical models (both BN and MN) model joint probability distributions $p(\boldsymbol{x}, \boldsymbol{y})$
- In many situations we know in advance which variables will be observed, and which will need to be predicted (i.e. $\boldsymbol{x}$ vs $\boldsymbol{y}$)
- Hidden Markov Models (as a special case of BN) also model joint probabilities of states and observations, even if they are often used to estimate the most probable sequence of states $\boldsymbol{y}$ given the observations $\boldsymbol{x}$
- A problem with joint distributions is that they need to explicitly model the probability of $\boldsymbol{x}$, which can be quite complex (e.g. a textual document)

Generative vs discriminative models



Naive Bayes



Hidden Markov Model

generative models

- Directed graphical models are called *generative* when the joint probability decouples as $p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y})$
- The dependencies between input and output are only from the latter to the former: the output *generates* the input
- Naive Bayes classifiers and Hidden Markov Models are both generative models

Generative vs discriminative models

Discriminative models

- If the purpose is choosing the most probable configuration for the output variables, we can directly model the conditional probability of the output given the input: $p(\boldsymbol{y}|\boldsymbol{x})$
- The parameters of such distribution have higher freedom wrt those of the full $p(\boldsymbol{x}, \boldsymbol{y})$, as $p(\boldsymbol{x})$ is not modelled
- This allows to effectively exploit the structure of $\boldsymbol{x}$ without modelling the interactions between its parts, but only those with the output
- Such models are called *discriminative* as they aim at modeling the discrimination between different outputs

Conditional Random Fields (CRF, Lafferty et al. 2001)

Definition

- Conditional random fields are conditional Markov networks:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \sum_{(\mathbf{x},\mathbf{y})_C} (-E((\mathbf{x},\mathbf{y})_C))$$

- The partition function $Z(\mathbf{x})$ is summed only over $\mathbf{y}$ to provide a proper conditional probability:

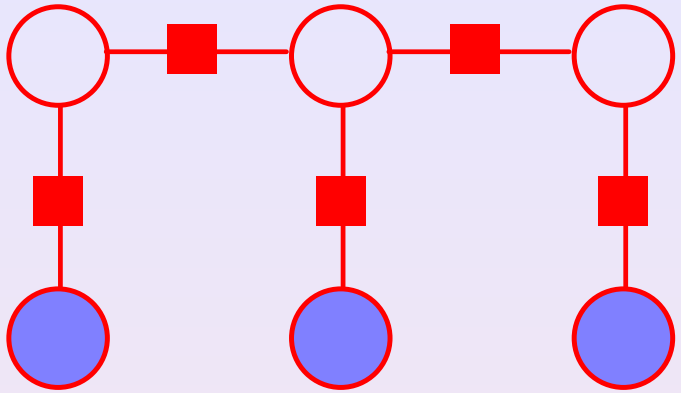
$$Z(\mathbf{x}) = \sum_{\mathbf{y}'} \exp \sum_{(\mathbf{x},\mathbf{y}')_C} \left(-E((\mathbf{x},\mathbf{y}')_C)\right)$$

Feature functions

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \sum_{(\mathbf{x},\mathbf{y})_C} \sum_{k=1}^{K} \lambda_k f_k((\mathbf{x},\mathbf{y})_C)$$

- The negated energy function is often written simply as a weighted sum of real-valued *feature functions*
- Each feature function should capture a certain characteristic of the clique variables

Linear chain CRF



Description (simple form)

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp \sum_t \left(\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}) + \sum_{h=1}^{h} \mu_h f_h(x_t, y_t) \right)$$

- Models the relation between an input and an output sequence
- Output sequences are modelled as a linear chain, with a link between each consecutive output element
- Each output element is connected to the corresponding input.

Linear chain CRF

Description (more generic form)

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

- the linear chain CRF can model arbitrary features of the input, not only identity of the current observation (like in HMMs)
- We can think of $\mathbf{x}_t$ as a vector containing input information relevant for position t, possibly including inputs at previous or following positions
- We can easily make transition scores (between consecutive outputs y_{t-1}, y_t) dependent also on current input x_t

Linear chain CRF

Parameter estimation

- Parameters λ_k of feature functions need to be estimated from data
- We estimate them from a training set of i.i.d. input/output sequence pairs

$$\mathcal{D} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\} \quad i = 1, \ldots, N$$

- each example $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ is made of a sequence of inputs and a corresponding sequence of outputs:

$$\boldsymbol{x}^{(i)} = \{x_1^{(i)}, \ldots, x_T^{(i)}\} \qquad \boldsymbol{y}^{(i)} = \{y_1^{(i)}, \ldots, y_T^{(i)}\}$$

Note

- For simplicity of notation we assume each training sequence have the same length.
- The generic form would replace T with $T^{(i)}$

Parameter estimation

Maximum likelihood estimation

- Parameter estimation is performed maximizing the *likelihood* of the data $\mathcal{D}$ given the parameters $\theta = \{\lambda_1, \ldots, \lambda_K\}$
- As usual to simplify derivations we will equivalently maximize *log-likelihood*
- As CRF model a *conditional* probability, we will maximize *conditional log-likelihood*:

$$\ell(\theta) = \log \prod_{i=1}^{N} p(\boldsymbol{y}^{(i)} | \boldsymbol{x}^{(i)}) = \sum_{i=1}^{N} \log p(\boldsymbol{y}^{(i)} | \boldsymbol{x}^{(i)})$$

Parameter estimation

Maximum likelihood estimation

- Replacing the equation for conditional probability we obtain:

$$\ell(\theta) = \sum_{i=1}^{N} \log \left(\frac{1}{Z(\mathbf{x}^{(i)})} \exp \sum_{t} \sum_{k=1}^{K} \lambda_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) \right)$$

$$= \sum_{i=1}^{N} \sum_{t} \sum_{k=1}^{K} \lambda_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)})$$

Gradient of the likelihood

$$\frac{\partial \ell(\theta)}{\partial \lambda_k} = \underbrace{\sum_{i=1}^{N} \sum_{t} f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)})}_{\tilde{E}[f_k]} - \underbrace{\sum_{i=1}^{N} \sum_{y,y'} \sum_{t} f_k(y, y', \mathbf{x}_t^{(i)}) p_\theta(y, y' | \mathbf{x}^{(i)})}_{E_\theta[f_k]}$$

Interpretation

- $\tilde{E}[f_k]$ is the expected value of f_k under the empirical distribution $\tilde{p}(\mathbf{y}, \mathbf{x})$ represented by the training examples
- $E_\theta[f_k]$ is the expected value of f_k under the distribution represented by the model with the *current value of the parameters*: $p_\theta(\mathbf{y}|\mathbf{x})\tilde{p}(\mathbf{x})$ ($\tilde{p}(\mathbf{x})$ is the empirical distribution of x)

Gradient of the likelihood

Interpretation

$$\frac{\partial \ell(\theta)}{\partial \lambda_k} = \tilde{E}[f_k] - E_\theta[f_k]$$

- The gradient measures the difference between the expected value of the feature under the empirical and model distributions
- The gradient is zero when the model adheres to the empirical observations
- This highlights the risk of overfitting training examples

Parameter estimation

Adding regularization

- CRF often have a large number of parameters to account for different characteristics of the inputs
- Many parameters mean risk of overfitting training data
- In order to reduce the risk of overfitting, we penalize parameters with a too large norm

Parameter estimation

Zero-mean Gaussian prior

- A common choice is assuming a Gaussian prior over parameters, with zero mean and covariance $\sigma^2 I$ (where I is the identity matrix)

$$p(\theta) \propto \exp\left(-\frac{||\theta||^2}{2\sigma^2}\right)$$

  where Gaussian coefficient can be ignored as it's independent of θ
- σ^2 is a free parameter determining how much to penalize feature weights moving away from the zero
- the log probability becomes:

$$\log(p(\theta)) \propto -\frac{||\theta||^2}{2\sigma^2} = -\sum_{k=1}^{K}\frac{\lambda_k^2}{2\sigma^2}$$

Parameter estimation

Maximum a-posteriori estimation

- We can now estimate the maximum a-posteriori parameters:

$$\theta^* = argmax_\theta \ell(\theta) + \log p(\theta) = argmax_\theta \ell_r(\theta)$$

where the regularized likelihood $\ell_r(\theta)$ is:

$$\ell_r(\theta) = \sum_{i=1}^{N} \sum_{t} \sum_{k=1}^{K} \lambda_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2}$$

Parameter estimation

Optimizing the regularized likelihood

- Gradient ascent $\rightarrow$ usually too slow
- Newton's method (uses Hessian, matrix of all second order derivatives) $\rightarrow$ too expensive to compute the Hessian
- Quasi-Netwon methods are often employed:
  - compute an approximation of the Hessian with only first derivative (e.g. BFGS)
  - limited-memory versions exist that avoid storing the full approximate Hessian (size is quadratic in the number of parameters)

Inference

Inference problems

- Computing the gradient requires computing the marginal distribution for each edge $p_\theta(y, y'|\mathbf{x}^{(i)})$
- This has to be computed at each gradient step, as the set of parameters θ changes in the direction of the gradient
- Computing the likelihood requires computing the partition function $Z(\boldsymbol{x})$.
- During testing, finding the most likely labeling requires solving:

$$\mathbf{y}^* = argmax_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$$

Inference algorithms

- All such tasks can be performed efficiently by dynamic programming algorithms similar to those for HMM

Inference algorithms

Analogy to HMM

- Inference algorithms rely on *forward*, *backward* and *Viterbi* procedures analogous to those for HMM
- To simplify notation and highlight analogy to HMM, we will use the formula of CRF with clique potentials:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_t \Psi_t(y_t, y_{t-1}, \mathbf{x}_t)$$

- where the clique potentials are:

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

Inference algorithms

Forward procedure

- The forward variable $\alpha_t(i)$ collects the unnormalized probability of output $y_t = i$ and the sequence of inputs $\{x_1, \ldots, x_t\}$:

$$\alpha_t(i) \propto p(x_1, \ldots, x_t, y_t = i)$$

- As for HMMs, it is computed recursively

$$\alpha_t(i) = \sum_{j \in S} \Psi_t(i, j, \mathbf{x}_t) \alpha_{t-1}(j)$$

- where S is the set of possible values for the output variable

Backward procedure

- The backward variable $\beta_t(i)$ collects the unnormalized probability of the sequence of inputs $\{x_{t+1}, \ldots, x_T\}$ given output $y_t = i$:

$$\beta_t(i) \propto p(x_{t+1}, \ldots, x_T | y_t = i)$$

- As for HMMs, it is computed recursively

$$\beta_t(i) = \sum_{j \in S} \Psi_{t+1}(j, i, \mathbf{x}_{t+1}) \beta_{t+1}(j)$$

Forward/backward procedures

Computing partition function

- Instead of computing $p(\mathbf{x})$, forward (or backward) variables allow to compute the partition function $Z(\mathbf{x})$:

$$p(\mathbf{x}) = \sum_{j \in S} p(\mathbf{x}, y_T = j) \propto \sum_{j \in S} \alpha_T(j) = Z(\mathbf{x})$$

Forward/backward procedures

Computing edge marginals

- Marginal probabilities for edges can be computed as in HMM from forward and backward variables:

$$p(y_t, y_{t-1}|\mathbf{x}) = \frac{p(y_t, y_{t-1}, \mathbf{x})}{p(\mathbf{x})}$$
$$= \frac{\alpha_{t-1}(y_{t-1})\Psi(y_t, y_{t-1}, \mathbf{x}_t)\beta_t(y_t)}{Z(\boldsymbol{x})}$$

Note

- Numerator and denominator are NOT probabilities (they are unnormalized)
- The fraction is a correctly normalized probability

Viterbi decoding

Intuition as in HMM

- Relies on a **max variable** $\delta_t(i)$ containing the unnormalized probability of the best sequence of outputs up to $t-1$ plus output $y_t = i$ and the inputs up to time t:

$$\delta_t(i) = \max_{y_1,\ldots,y_{t-1}} p(y_1, \ldots, y_{t-1}, y_t, x_1, \ldots, x_t)$$

- A **dynamic programming** procedure allows to compute the max variable at time t based on the one at time $t-1$.
- An array ψ allows to keep track of the outputs which maximized each step
- Once time T is reached, a backtracking procedure allows to recover the sequence of ouputs which maximized overall probability.

Viterbi decoding

The algorithm

1. Initialization:
$$\delta_1(i) = \Psi(i, -, \mathbf{x}_1)\ i \in S$$
2. Induction:
$$\delta_t(j) = \max_{i \in S} \delta_{t-1}(i)\Psi(j, i, \mathbf{x}_t),\ j \in S,\ 2 \leq t \leq T$$
$$\psi_t(j) = \text{argmax}_{i \in S}\delta_{t-1}(i)\Psi(j, i, \mathbf{x}_t),\ j \in S,\ 2 \leq t \leq T$$
3. Termination:
$$p^* \propto \max_{i \in S} \delta_T(i)$$
$$y_T^* = \text{argmax}_{i \in S}\delta_T(i)$$
4. Path (output sequence) backtracking:
$$y_t^* = \psi_{t+1}(y_{t+1}^*),\ t = T-1, T-2, \ldots, 1$$

Viterbi decoding

Note

- There is no need for normalization as we are only interested in best output sequence, not its probability

Application example

Biological named entity recognition (Settles, 2004)

- Named entity recognition consists of identifying within a sentence words or sequences of adjacent words belonging to a certain class of interest
- For instance, classes of biological interest could be `PROTEIN,DNA,RNA,CELL-TYPE`

Analysis of $\underbrace{\text{myeloid-associated genes}}_{DNA}$ in $\underbrace{\text{human hematopoietic progenitor cells}}_{CELL-TYPE}$

Biological named entity recognition

Labelling

- For each class of interest, the labeling distinguishes between:
  - the first word in the named entity (e.g. `B-DNA`, with `B` standing for begin)
  - the following words in the named entity (e.g. `I-DNA`, with `I` standing for internal)
- Words not belonging to any class of interest are labelled as `O` (other).

Analysis	of	myeloid-associated	genes	in	human	hematopoietic	progenitor	cells
O	O	B-DNA	I-DNA	O	B-CELL-TYPE	I-CELL-TYPE	I-CELL-TYPE	I-CELL-TYPE

Note

- Labels of adjacent words are strongly correlated $\rightarrow$ ideal for sequential models

Biological named entity recognition

Feature functions: dictionary

- The simplest set of feature functions consists of dictionary entries.
- Each feature would model the observation of a certain word and its class assignment, possibly together to the class assignment of the previous word:

$$f_k(y_t, y_{t-1}, \mathbf{x}_t) = \begin{cases} 1 & \texttt{if } x_t = \texttt{cells} \wedge y_t = \texttt{CELL-TYPE} \\ & \quad \wedge\ y_{t-1} = \texttt{CELL-TYPE} \\ 0 & \texttt{otherwise} \end{cases}$$

- Note that the model will have distinct features for the occurrence of the word `cells` in different labeling contexts
- A higher weight λ_k will be arguably learned for observing the feature in the `CELL-TYPE` labelling context wrt other ones.

Biological named entity recognition

Feature functions: dictionary

- Most dictionary features will be very sparse, with very low occurrence in both training data and novel test ones.
- Very sparse features will be probably receive low or zero weight, also as an effect of regularization.
- Fewer relevant dictionary words (e.g. `cell, protein`, common verbs) should receive higher (positive or negative) weight if found to discriminate between classes in training data.
- Anyhow some properties common to different words could also be found discriminant

Biological named entity recognition

Feature functions: orthographic features

- Capitalization is often associated to named entities more than other words (e.g. `mRNA`)
- Alphanumeric strings are typically used to identify specific proteins, genes in biological databases (e.g. `7RSA`)
- Dashes often appear in complex compound words (e.g. `myeloid-associated`)
- Each such feature can be encoded with a separate function

Biological named entity recognition

Feature functions: orthographic word classes

- A word representation in terms of few relevant orthographic features can be achieved by:
  - replacing any upper case letter with `A`
  - replacing any lower case letter with `a`
  - replacing any digit with `0`
  - replacing any other character with `_`
- Examples:

word	word class
`7RSA`	`0AAA`
`1CIX`	`0AAA`
`F-actin`	`A_aaaaa`
`T-cell`	`A_aaaa`

Biological named entity recognition

Feature functions: neighbouring words

- Features related to $\mathbf{x}_t$ are not limited to characteristics of the word at position t
- For instance, context features can model the identity of the word together to those of the preceding and following ones
- The same can be done for other characteristics of words, such as word classes, presence of dashes
- Information on neighbouring words can be combined in arbitrary way to create features deemed relevant (e.g. a capitalized word preceded by and article as in `the ATPase`)

Biological named entity recognition

Feature functions: semantic features

- When available, semantic information can strongly help in building disambiguating features
- For instance, amino-acids codes are often capitalized or have capitalized initial (e.g. `CYS, His`)
- Such strings could be wrongly identified as named entities of one of the classes.
- An explicit feature representing an amino-acid code could be added to help disambiguation.

Conditional random fields

Parameter tying

- All cliques (y_t, y_{t-1}, x_t) in linear-chain CRF share the same set of parameters λ_k independently of t
- This parameter tying allows to:
  - avoid an explosion of parameters, controlling overfitting
  - apply a learned model to sequences of different length

Conditional random fields

Clique templates

- Parameter tying can be represented by dividing the set of cliques C in a factor graph G into *clique templates*:

$$C = \{C_1, \ldots, C_P\}$$

- all cliques in each clique template C_p share the same parameters θ_p
- linear chain CRF have a single clique template for all (y_t, y_{t-1}, x_t)

Generic conditional random fields

Description

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in C} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p)$$

- the first product runs over clique templates
- the second product run over cliques in a template
- the clique potential share template parameters θ_p
- the partition function is:

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{C_p \in C} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p)$$

Generic conditional random fields

Clique potential

$$\Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p) = \exp\left(\sum_{k=1}^{K(p)} \lambda_{kp} f_{kp}(\mathbf{x}_c, \mathbf{y}_c)\right)$$

- $K(p)$ is the number of feature functions for the template p
- λ_{kp} are the template-dependent weights of the feature functions

Generic conditional random fields

Dependencies between examples

- In linear-chain CRF, we assumed a dataset of i.i.d. examples made of input/output sequences.
- In general, there can be dependencies (thus links) between “examples” in the training set
- The training set can be seen as a single large CRF, possibly made of some disconnected components
- In the case of i.i.d. examples, there would be a disconnected component for each example
- We will thus drop the sum over training examples in discussing parameter estimation (and inference)

Parameter estimation

Conditional log-likelihood

$$\ell(\theta) = \sum_{C_p \in C} \sum_{\Psi_c \in C_p} \sum_{k=1}^{K(p)} \lambda_{kp} f_{kp}(\mathbf{x}_c, \mathbf{y}_c) - \log Z(\mathbf{x})$$

- As for the linear-chain CRF, a regularized conditional log-likelihood can be obtained adding Gaussian priors (or other distributions) on the clique template parameters

Parameter estimation

Gradient of conditional log-likelihood

$$\frac{\partial \ell(\theta)}{\partial \lambda_{kp}} = \sum_{\Psi_c \in C_p} f_{kp}(\mathbf{x}_c, \mathbf{y}_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{y}'_c} f_{kp}(\mathbf{x}_c, \mathbf{y}'_c) p_\theta(\mathbf{y}'_c | \mathbf{x})$$

- The gradient is again the difference between expected values of feature functions under empirical and model distribution respectively.

Inference

Belief propagation

- *Belief propagation* is a generalization of forward-backward procedure. It computes exact inference on tree-structured models
- *Belief propagation* can also be applied on models with cycles (called *loopy belief propagation*).
- Loopy belief propagation is no more exact nor guaranteed to converge, but has successfully been employed as an approximation strategy.

Inference

Junction trees

- a tree-structured representation of any graphical model can be obtained building a *junction tree*
- Nodes in junction trees are clusters of variables in the original tree, each link between a pair of clusters has a separator node with the variables common to both clusters.
- Exact inference can be achieved on junction trees by belief propagation
- The algorithm is exponential in the number of variables in the clusters and is intractable for arbitrary graphs.

Inference

Sampling

- Sampling methods compute approximate inference sampling from the model distribution
- A number of samples for the variables in the model is generated by some random process.
- The probability of a certain configuration of variable values is computed aggregating the samples
- The random process takes time to converge before generating samples from the correct distribution
- This can be quite slow if we need to do inference at each step of training

Applications

Examples

- named-entity recognition: detect in a sentence words or sequences of adjacent words referring to a named-entity and classify the entity
- extract contact information from personal web pages (e.g. name, address, mobile, email)
- perform multi-label classification modelling dependencies between labels
- perform RNA secondary structural alignment
- label images in computer vision

Skip-chain CRF (Sutton and McCallum, 2004)

Motivation

- The same input can appear multiple times in a certain sequence
- Such multiple instances are often likely to share the same label
- In named entity recognition, multiple instances of the same word often refer to the same entity (or class of entities)
- It would be desirable that the model tends to label such multiple occurences consistently

Skip-chain CRF

Modeling long-range dependencies

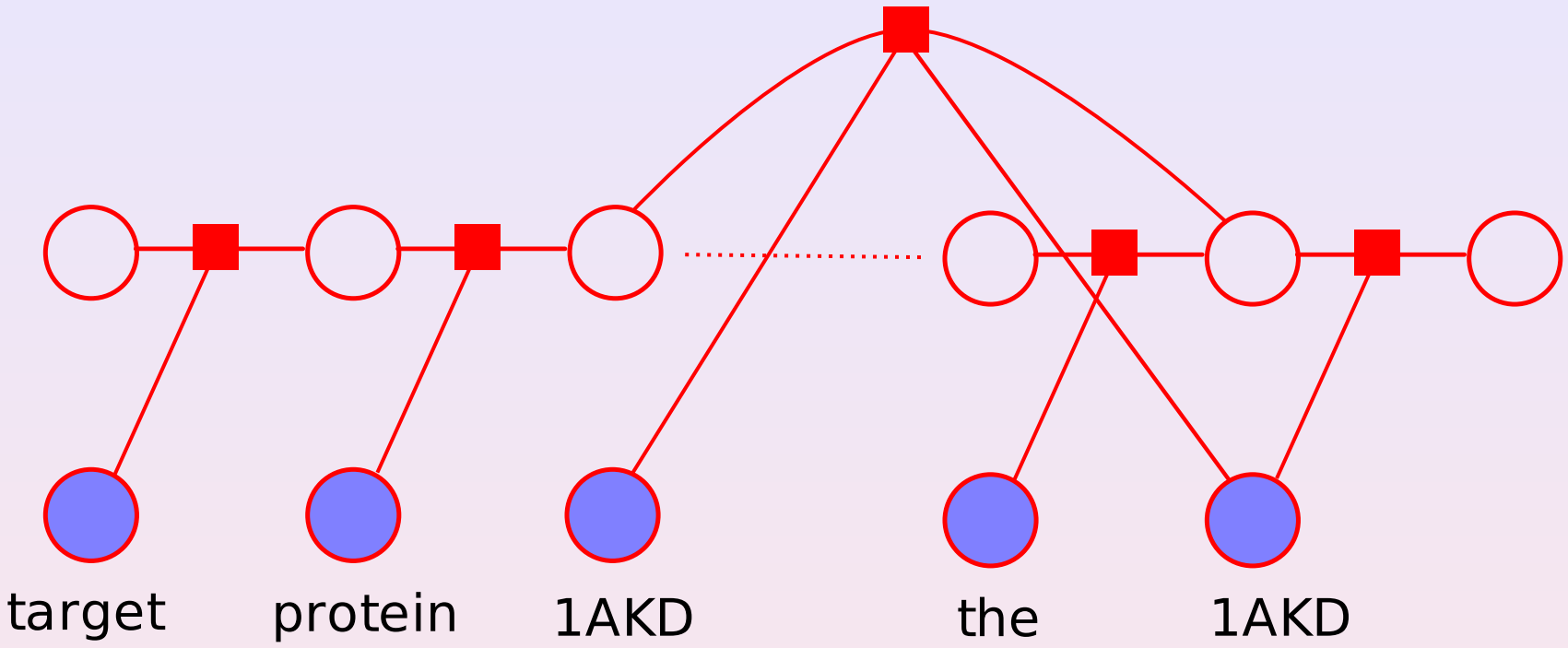
- Multiple instances of the same input can appear at arbitrary distance within the sequence
- A linear-chain model needs to pass information along such distances in order for difference instances to influence their respective labeling decisions
- The problem is in the Markov assumption, that label at time y_t only depends on labels at previous k time instants (with $k = 1$ in linear chains)
- Modeling *long-range dependencies* in such a setting is extremely unlikely

Skip-chain CRF

Adding shortcuts

- A possible approach to address the problem is by adding *shortcut* (or *skip*) links between distant outputs.
- The number of such links should be limited in order to add limited complexity to the model
- Conditional models allow to add links which are *dependent* of the input content
- For instance, it is possible to add links only between outputs with same inputs (i.e. the shared instances)
- It is also possible to add links only for instances which most likely will share the same class (e.g. capitalized words like `7RSA`, but not adjectives like `human`)

Skip-chain CRF



The graphical model

- A skip-chain CRF is a linear chain CRF with the addition of shortcut links between nodes likely to share the same class

Skip-chain CRF

The joint probability

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_t \Psi_t(y_t, y_{t-1}, \mathbf{x}_t) \prod_{(u,v)\in\mathcal{I}} \Psi_{(u,v)}(y_u, y_v, \mathbf{x}_u, \mathbf{x}_v)$$

- $\mathcal{I}$ is the set of related pairs (i.e. entries assumed likely to be from the same class)
- The model has two clique templates:
  - the standard linear-chain template over transitions plus current input:

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}_t) = \exp \sum_k \lambda_{1k} f_{1k}(y_t, y_{t-1}, \mathbf{x}_t)$$

  - a skip-chain template for the related pairs, with their outputs and inputs:

$$\Psi_{(u,v)}(y_u, y_v, \mathbf{x}_u, \mathbf{x}_v) = \exp \sum_k \lambda_{2k} f_{2k}(y_u, y_v, \mathbf{x}_u, \mathbf{x}_v)$$

Skip-chain CRF

Skip-chain features

- Skip-chain features should try to pass information from one input to its related partner
- An effective technique was that of modeling each input feature as a disjunction of the input features at u and v. E.g.:

$$f_{2k}(y_u, y_v, \mathbf{x}_u, \mathbf{x}_v) = \begin{cases} 1 & \texttt{if } ((x_u = \texttt{0AAA} \wedge \\ & \quad x_{u-1} = \texttt{protein}) \vee \\ & \quad (x_v = \texttt{0AAA} \wedge \\ & \quad x_{v-1} = \texttt{protein})) \wedge \\ & \quad y_u = \texttt{PROTEIN} \wedge \\ & \quad y_v = \texttt{PROTEIN} \\ 0 & \texttt{otherwise} \end{cases}$$

Skip-chain CRF

Inference

- Skip links introduce *loops* in the graphical model, making exact inference intractable in general
- Furthemore, loops can be long and overlapping, and maximal cliques in junction trees can be too large to be tractable
- Approximate inference by loopy belief propagation was applied to train skip-chain CRF with effective results

Factorial CRF (Sutton et al, 2004)

Motivation

- Sequential labelling tasks do not necessarily limit to scalar outputs at each time instant
- A sequence of vectors of outputs can represent the desired outcome
- This happens for instance when there is a hierarchy of outputs

Factorial CRF

Example: POS tagging and chunking

- A relevant and hard task in natural language processing is that of automatically extracting the syntactic structure of a sentence
- The first level of such structure consists of assigning the correct part-of-speech (POS) tag to each individual word (e.g. verb, noun, pronoun, adjective)
- A *shallow* parsing of sentences consists of identifying *chunks* of consecutive groups of words representing grammatical units such as noun or verb phrases.

Factorial CRF

Example

- Sentence:

  ```
  He reckons the current account deficit will
  narrow to only L1.8 billion in September.
  ```

- POS tagging:

  ```
  (PRP)He (VBZ)reckons (DT)the (JJ)current
  (NN)account (NN)deficit (MD)will (VB)narrow
  (TO)to (RB)only (L)L (CD)1.8 (CD)billion
  (IN)in (NNP)September (.).
  ```

- Shallow parsing:

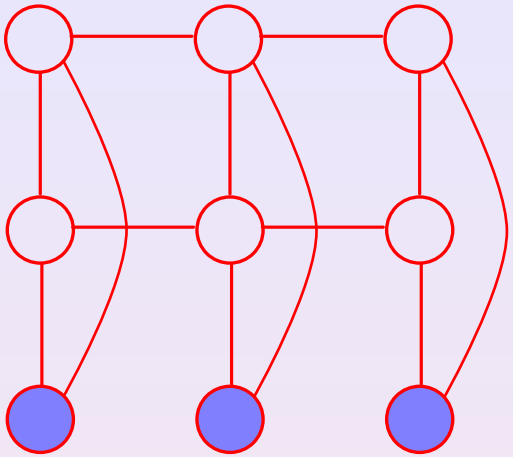
  ```
  [NP He ] [VP reckons ] [NP the current
  account deficit ] [VP will narrow ] [PP to
  ] [NP only L 1.8 billion ] [PP in ] [NP
  September ].
  ```

Factorial CRF

Example: POS tagging and chunking

- POS tagging is often used as a first step for shallow parsing.
- The two tasks can be accomplished in cascade:
  - First each word is labelled with its predicted POS tag
  - Then the sequence of words and POS tags is passed to the shallow parser which identifies the chunks
- However, an error at the first level (POS tagging) will badly affect the performance of the following level (chunking)
- Such *error propagation* effect can be dramatic for multiple levels of labeling.

Factorial CRF



Jointly predicting multiple levels

- A possible solution to address the error propagation issue consists of jointly predicting all levels of the output hierarchy
- Factorial CRF are obtained combining multiple linear-chains CRF, one for each output level
- Input nodes are shared among levels
- Output nodes from one level are link to cotemporal output nodes in the following and previous levels

Factorial CRF

Joint probability

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \prod_{l=1}^{L} \Psi_t(y_{t,l}, y_{t-1,l}, \mathbf{x}_t) \Phi_t(y_{t,l}, y_{t,l+1}, \mathbf{x}_t)$$

- L is the number of levels in the hierarchy
- Ψ_t is the clique template for transitions
- Φ_t is the clique template for cotemporal connections between successive levels

Factorial CRF

Inference

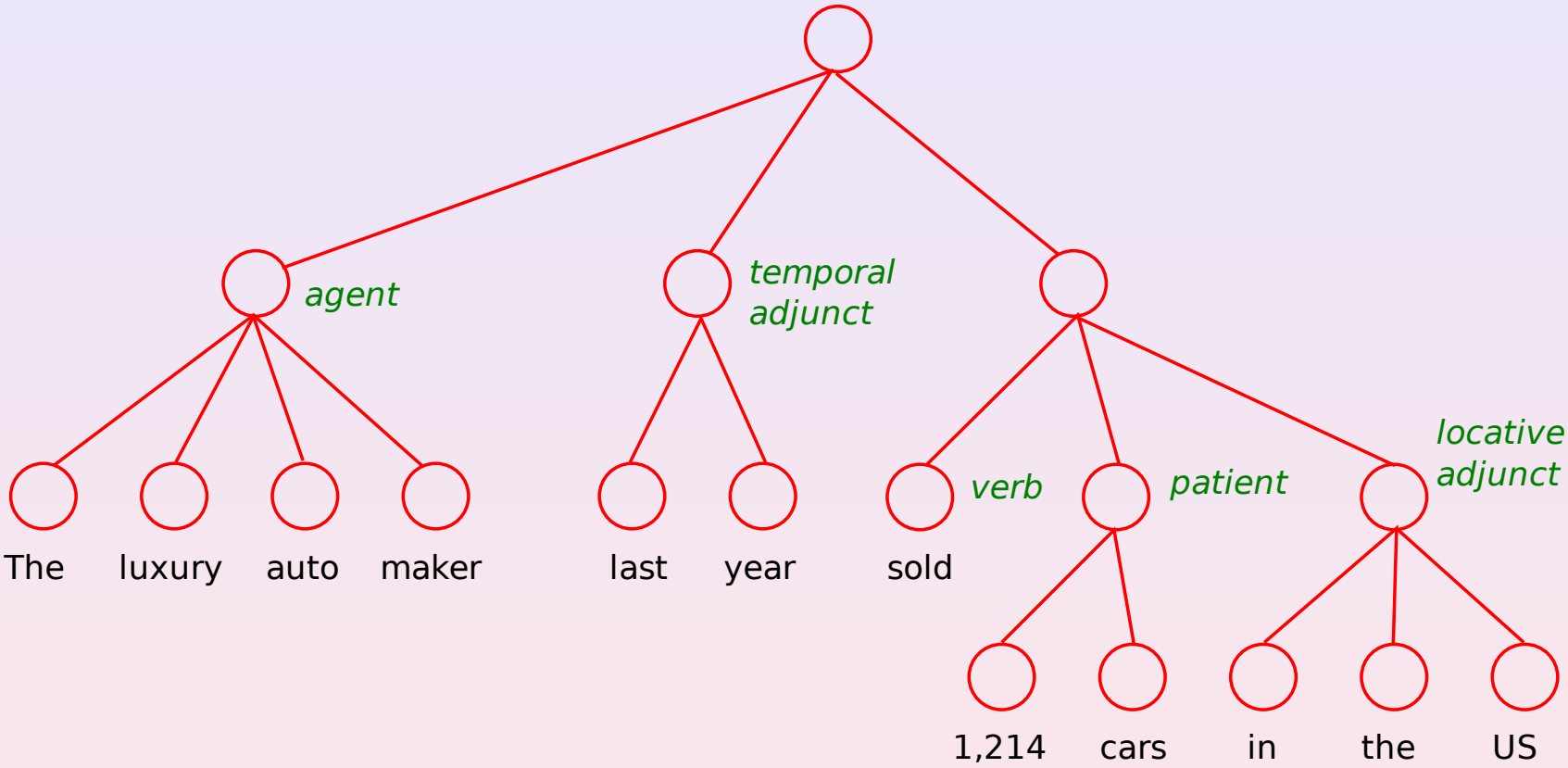
- Cotemporal links introduce *loops* in the graphical model
- Approximate inference by loopy belief propagation was applied to train factorial CRF with effective results

Tree CRF (Cohn and Blunsom, 2005)

Semantic role labelling

- Given a full parse tree, decide which constituents fill semantic roles (agent, patient, etc) for a given verb
- Task is to annotate parse structure with role information
- A tree CRF is constructed according to the structure of the parse tree
- Efficient exact inference can be accomplished by belief propagation

Tree CRF


agent
temporal adjunct
verb
patient
locative adjunct
The
luxury
auto
maker
last
year
sold
1,214
cars
in
the
US

Resources

References

- John Lafferty, Andrew McCallum, and Fernando Pereira, *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*, ICML 2001.
- C. Sutton, A. McCallum, *An Introduction to Conditional Random Fields for Relational Learning*, in Introduction to Statistical Relational Learning, L. Getoor and B. Taskar, eds., the MIT Press, 2007.
- C. Sutton, K. Rohanimanesh, A. McCallum, *Dynamic Conditional Random Fields: Factorized Probabilistic Models for Labeling and Segmenting Sequence Data*, ICML 2004.
- Trevor Cohn and Philip Blunsom, *Semantic Role Labelling with Tree Conditional Random Fields*, CoNLL 2005

Resources

References

- John Lafferty, Andrew McCallum, and Fernando Pereira, *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*, ICML 2001.
- C. Sutton, A. McCallum, *An Introduction to Conditional Random Fields for Relational Learning*, in Introduction to Statistical Relational Learning, L. Getoor and B. Taskar, eds., the MIT Press, 2007.

Resources

Software

- CRF++: Yet Another CRF toolkit (sequence labelling)
  - http://crfpp.sourceforge.net/
- Conditional Random Field (CRF) Toolbox for Matlab (1D chains and 2D lattices)
  - http://www.cs.ubc.ca/~murphyk/Software/CRF/crf.html
- MALLET: Java package for machine learning applications to text. Includes CRF for sequence labelling
  - http://mallet.cs.umass.edu/

Links

- Hanna Wallach page on CRF (includes CRF related publications and software)
  - http://www.inference.phy.cam.ac.uk/hmw26/crf/