Assignment 10 + Conditional Random Fields

(SNLP tutorial 11)

Vilém Zouhar, Awantee Deshpande, Julius Steuer

6th, 8th July

Organisation

- Check that you have the finalised versions of the tutorial slides (github.com/zouharvi/uds-snlp-tutorial)
- Check if you are eligible for the exam, and register accordingly.
- Project will be released on Friday, expected deadline at the end of August (tentatively 20th Aug), will be specified in the project instructions.
- Next week's tutorial discussion: Open Q&A
- Send a list of questions to me (Teams or Private Piazza Post)
- Discussion of sample exam

Assignment 10

- Exercise 1: Lesk's Algorithm
- Exercise 2: Expectation Maximisation
- Exercise 3: Yarowsky Algorithm

Overview

- Sequence Labelling / Entity Recognition
  - Rule-based
  - HMM
  - Bayesian Network
  - Log-linear 1st Order Sequential Model
  - Linear Chain CRF / CRF
- Model comparison
- Implementations

Sequence Labelling / Entity Recognition

- `My name is Joachim, I live in Saarbrücken, and my matriculation number is 1234.`
- `My name is [Joachim:PERSON], I live in [Saarbrücken:LOC], and my matriculation number is [1234:MATNUM].`
- NER as Sequence labelling:
  X: sequence of words
  Y: labels {`MATNUM`, `PERSON`, `LOCATION`, `NONE`}

Sequence Labelling / Entity Recognition

- My name is Joachim, I live in Saarbrücken, and my matriculation number is 1234.
- My name is [Joachim:PERSON], I live in [Saarbrücken:LOC], and my matriculation number is [1234:MATNUM].
- NER as Sequence labelling:
  X: sequence of words
  Y: labels {MATNUM, PERSON, LOCATION, NONE}

- NER can be reformulated as sequence labelling, which includes also e.g. part of speech tagging
- Given a sentence we want to classify every token.

Rule-based

- Regex substitute:
  `matriculation (number)? (is)? (\d+)` $\rightarrow$ `[\3:mat-num]`
- Gets out of hand quickly:
  `(am|name (is)?) (.*?) (and|\s[.,?])?` $\rightarrow$ `[\3:person]`
- No automated learning

Rule-based

- Regex substitute:
  `matriculation (number)? (is)? (\d+)` → `[\3:mat-num]`
- Gets out of hand quickly:
  `(am|name (is)?) (.*?) (and|\s[.,?])?` → `[\3:person]`
- No automated learning

- The most straightforward solution just uses regex substitution, but that becomes very complex very soon and is also not performant enough, because it does not learn from the data.
- The only advantage is that we know explicitly which rules get applied.
- Still bad in general.

Generative vs. Discriminative Models

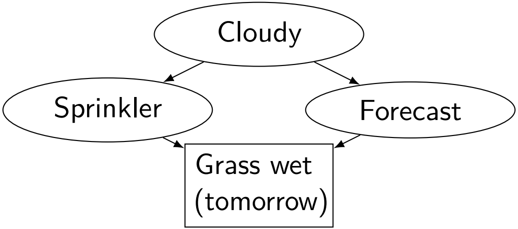
- What's the difference?
- Generative: Model actual distribution of data, learn joint probability and predict conditional probability using Bayes Theorem i.e. predict $P(Y|X)$ using $P(X|Y)$ and $P(Y)$
  e.g. Naive Bayes, HMMs
- Discriminative: Model decision boundary between classes, learn conditional probability directly, estimate parameters for $P(Y|X)$ directly from data
  e.g. MaxEnt Classifier, CRFs

Bayesian Network

- Directed acyclic graph (DAG), $(x \to y) \in E : y$ dependent on x

Local Markov Property

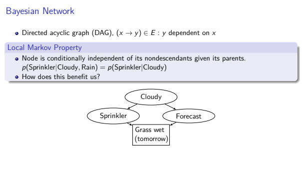
- Node is conditionally independent of its nondescendants given its parents.
  $p(\text{Sprinkler}|\text{Cloudy}, \text{Rain}) = p(\text{Sprinkler}|\text{Cloudy})$
- How does this benefit us?



2021-07-07

Assignment 10 + Conditional Random Fields

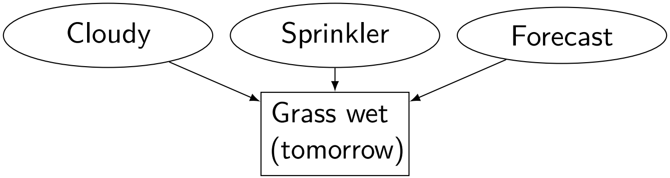
Bayesian Network



- Has to be DAG, otherwise cycles
- It models dependence between variables which can be either latent or observed
- We use it to reason about events of which we know the observed values and we want to know the cause
- From the graph we may for example find out, that it makes no sense to condition *Sprinkler* on *Rain*, because these two variables are independent. It would however be an approximation if we treated *Cloudy* independent of *Grass wet*.

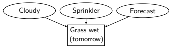
Naïve Bayes

- Assume absolute independence except for the one observed variable
- $p(y = \text{Yes}|x) = p(y_j|x) = \frac{p(x|y_j)p(y_j)}{p(x)} \propto p(x|y_j)p(y_j) \approx p(y_j)\prod_i p(x_i|y_j)$



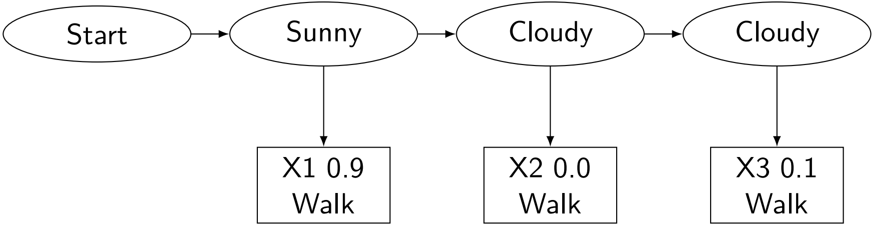
Naïve Bayes

- Assume absolute independence except for the one observed variable
- $p(y = \text{Yes}|x) = p(y_j|x) = \frac{p(x|y_j)p(y_j)}{p(x)} \propto p(x|y_j)p(y_j) \approx p(y_j) \prod_i p(x_i|y_j)$



- In Naïve Bayes we artificially flatten the network so that the observed variable is directly dependent to all causes and there are no other dependencies.
- The formula shows where the approximation is taking place.
- A practical example why this is naïve is that the variable *Rain* is heavily dependent on the *Cloudy* variable but as well on the *Foggy*, which in turn is almost the same thing as *Cloudy*. And if we put both all these in the formula, then we assign higher weight to the concept of *cloudiness* than to *sunniness*.

HMM



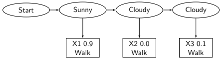
Sketch of HMM structure
observed variable *Walk duration*, latent variable: $Weather \in \{Sunny, Cloudy\}$

$$p(y|x) = \prod_i p(y) \cdot o(y, x_i) \text{ (Naïve Bayes)}$$

$$\Rightarrow$$

$$p(Y|X) = \prod_i a(y_{i-1}, y_i) \cdot o(y_i, x_i) \text{ (HMM)}$$

HMM



Sketch of HMM structure
observed variable *Walk duration*, latent variable: $Weather \in \{Sunny, Cloudy\}$

$$p(y|x) = \prod_i p(y) \cdot o(y, x_i) \text{ (Naïve Bayes)}$$
$$\Rightarrow$$
$$p(Y|X) = \prod_i a(y_{i-1}, y_i) \cdot o(y_i, x_i) \text{ (HMM)}$$

- From bayesian network point of view, HMMs model a structure in which latent variable is connected to another one, which in turn is connected to observed ones.
- These relationships are explicitly modeled by the transition (horizontal) and emission (vertical) functions.

HMM

- Hidden states: {MATNUM, PERSON, LOCATION, NONE}
- Better hidden states: {MATNUM, START+PERSON, INTERNAL+PERSON, END+PERSON, LOCATION, NONE, ...}
- Transitions: MLE from annotated data
- Emission probabilities: MLE from annotated data (+ smoothing)
- $p(Y|X) = \prod_i a(y_{i-1}, y_i) \cdot o(y_i, x_i)$

Questions

- What are the drawbacks of HMMs?

HMM

- Hidden states: {`MATNUM`, `PERSON`, `LOCATION`, `NONE`}
- Better hidden states: {`MATNUM`, `START+PERSON`, `INTERNAL+PERSON`, `END+PERSON`, `LOCATION`, `NONE`, `...`}
- Transitions: MLE from annotated data
- Emission probabilities: MLE from annotated data (+ smoothing)
- $p(Y|X) = \prod_i a(y_{i-1}, y_i) \cdot o(y_i, x_i)$

Questions

- What are the drawbacks of HMMs?

- HMMs seem a better fit for this task, since it captures transition probabilities between latent variables and emission probabilities.
- The probability of the sequence is computed as the product of transitions and observations.
- The probabilities can be estimated using MLE counting + some smoothing
- Side note, HMM is a generative model, because it can model the joint distribution p(y,x)
- In case we don't have annotated data, we may still make use of HMMs by employing the Baum-Welch algorithm.
- The emission probabilities are just distributions over all observable variables and every latent variable gets a unique one. For example in POS tagging, it may be the partial counts, but in speech processing, it's a gaussian mixture.
- We usually require supervised examples to do this MLE counting, but the Baum-Welch algorithm is able to estimate all these probabilities even if we don't know the latent labels.
- The reason for low performance is that the emission probabilities capture only features that dependent only on the current state and we have little control over the features.

Logistic Regression

$$p(y|x) = \frac{\exp(\Phi(y,x))}{\sum_{y'} \exp(\Phi(y',x))}$$

$$\arg\max_y \frac{\exp(\Phi(y,x))}{\sum_{y'} \exp(\Phi(y',x))} = \arg\max_y \exp(\Phi(y,x))$$

Assignment 10 + Conditional Random Fields

└─Logistic Regression

$$p(y|x) = \frac{\exp(\Phi(y,x))}{\sum_{y'} \exp(\Phi(y',x))}$$

$$\arg\max_y \frac{\exp(\Phi(y,x))}{\sum_{y'} \exp(\Phi(y',x))} = \arg\max_y \exp(\Phi(y,x))$$

- Another approach is to assign a score to every sequence and then pick the best one.
- So Phi in this case would just score every possible sequence and by doing softmax we get a conditional probability

Model overview

- Multinomial logistic regression:
  $p(y_j|x) = \frac{exp(Z_j \cdot x)}{\sum_i exp(Z_i \cdot x)}$
- Multiclass naïve Bayes:
  $p(y_j|x) = \frac{p(x|y_j)p(y_j)}{p(x)} \propto p(x|y_j)p(y_j) \approx p(y_j)\prod_i p(x_i|y_j)$

Model overview

- Multinomial logistic regression:
  $p(y_j|x) = \frac{exp(Z_j \cdot x)}{\sum_i exp(Z_i \cdot x)}$
- Multiclass naïve Bayes:
  $p(y_j|x) = \frac{p(x|y_j)p(y_j)}{p(x)} \propto p(x|y_j)p(y_j) \approx p(y_j) \prod_i p(x_i|y_j)$

- Bayes splits input features.
- Why generative? p(x|y_j) is the generative part, while p(y_j|x) is discriminative.

Log-linear 1st Order Sequential Model

- Sequence of hidden states: y, $\{\texttt{MATNUM}, \texttt{PERSON}, \texttt{LOCATION}, \texttt{NONE}\}$
- Observed sequence of variables: x (words)
- Goal: Model $p(y|x)$ for all pairs (x, y)

- $p(y|x) \propto \exp\left\{\sum_i \log a(y_{i-1}, y_i) + \log o(y_i, x_i)\right\}$

- $p(y|x) = \frac{1}{Z(x)} \cdot \exp\left\{\sum_i \log a(y_{i-1}, y_i) + \log o(y_i, x_i)\right\}$

- $p(y|x) = \frac{1}{Z(x)} \cdot \prod_i a(y_{i-1}, y_i) o(y_i, x_i)$

Log-linear 1st Order Sequential Model

- Sequence of hidden states: y, {MATNUM, PERSON, LOCATION, NONE}
- Observed sequence of variables: x (words)
- Goal: Model $p(y|x)$ for all pairs (x, y)
- $p(y|x) \propto \exp\{\sum_i \log a(y_{i-1}, y_i) + \log o(y_i, x_i)\}$
- $p(y|x) = \frac{1}{Z(x)} \cdot \exp\{\sum_i \log a(y_{i-1}, y_i) + \log o(y_i, x_i)\}$
- $p(y|x) = \frac{1}{Z(x)} \cdot \prod_i a(y_{i-1}, y_i) o(y_i, x_i)$

- Looks like HMM.
- This has exactly the same number of parameters but they all model $p(y|x)$ and not $p(x, y)$. This is more ideal for us.

Log-linear 1st Order Sequential Model

Viterbi:

$$\mathit{argmax}\ \ p(y|x) = \mathit{argmax}\ \ \log p(y|x) = \mathit{argmax}\ \ F(y,x) - \log \sum_{y'} \exp F(y',x)$$

$$= \mathit{argmax}\ \ F(y,x)$$

$$\alpha_t(y_j) = \max_i \exp\left(\log \alpha_{t-1}(y_i) + a(y_j,y_i) + o(y_j,x_t)\right)$$

$$\alpha'_t(y_j) = \mathit{argmax}_i\ \ \alpha_{t-1}(y_i) + \exp\left(a(y_j,y_i) + o(y_j,x_t)\right)$$

$O(|Y|^2 \cdot T)$

Log-linear 1st Order Sequential Model

Viterbi:

$$
\begin{aligned}
&argmax\ p(y|x) = argmax\ \log p(y|x) = argmax\ F(y,x) - \log \sum_{y'} \exp F(y',x) \\
&= argmax\ F(y,x) \\
&\alpha_t(y_j) = \max_i \exp \left(\log \alpha_{t-1}(y_i) + a(y_j, y_i) + o(y_j, x_t) \right) \\
&\alpha'_t(y_j) = argmax_i\ \alpha_{t-1}(y_i) + \exp\left(a(y_j, y_i) + o(y_j, x_t) \right)
\end{aligned}
$$

$O(|Y|^2 \cdot T)$

- First we may be interested in just the argmax, for which we need to store the pointers (Viterbi algorithm)
- Build trellis.

Log-linear 1st Order Sequential Model

Forward:

$$\log fw_t(y_j) = \log \sum_i \exp\Big(\log fw_{t-1}(y_i) + a(y_j, y_i) + o(y_j, x_t)\Big)$$

$$Z(X) = \sum_i \exp\Big(\log fw_{|T|-1}(y_i) + a(y_j, y_i) + o(y_j, x_t)\Big)$$

$$\rightarrow$$

$$p(y|x) = \frac{\alpha_{|T|}(y_{:-1})}{Z(x)}$$

$O(|Y|^2 \cdot T)$

Log-linear 1st Order Sequential Model

Forward:

$$\log fw_t(y_j) = \log \sum_i \exp\Big(\log fw_{t-1}(y_i) + a(y_j, y_i) + o(y_j, x_t)\Big)$$

$$Z(X) = \sum_i \exp\Big(\log fw_{|T|-1}(y_i) + a(y_j, y_i) + o(y_j, x_t)\Big)$$

$$\rightarrow$$

$$p(y|x) = \frac{\alpha_{|T|}(y_{:-1})}{Z(x)}$$

$O(|Y|^2 \cdot T)$

- To compute the full conditional probability, we also need the partition function, which we can compute using the forward algorithm.
- Finally, we have the argmax as well as the conditional probability.
- This can also be done using matrix methods TODO

Log-linear 1st Order Sequential Model

- Replace $o(y_j, x_t)$ with $\lambda_1 h_1(y_j, x_t) + \lambda_2 h_2(y_j, x_t) + \dots$
- Same with $a(y_j, y_i) = \lambda'_1 g_1(y_j, y_i) + \lambda'_2 g_2(y_j, y_i) + \dots$
- Why not just $\sum_{\text{feature } f} \lambda_i f_i(y_i, y_j, x_t)$?

Log-linear 1st Order Sequential Model

- Replace $o(y_j, x_t)$ with $\lambda_1 h_1(y_j, x_t) + \lambda_2 h_2(y_j, x_t) + \dots$
- Same with $a(y_j, y_i) = \lambda'_1 g_1(y_j, y_i) + \lambda'_2 g_2(y_j, y_i) + \dots$
- Why not just $\sum_{\text{feature } f} \lambda_i f_i(y_i, y_j, x_t)$?

- o can be any scoring function, does not need to be a distribution like with HMMs
- It can be a sum of other feature functions.
- In fact, this can be generalized even further
- And finally, there is no reason to not allow features to observe the whole sequence, because neither Viterbi nor Forward decoding limits this.

Model overview

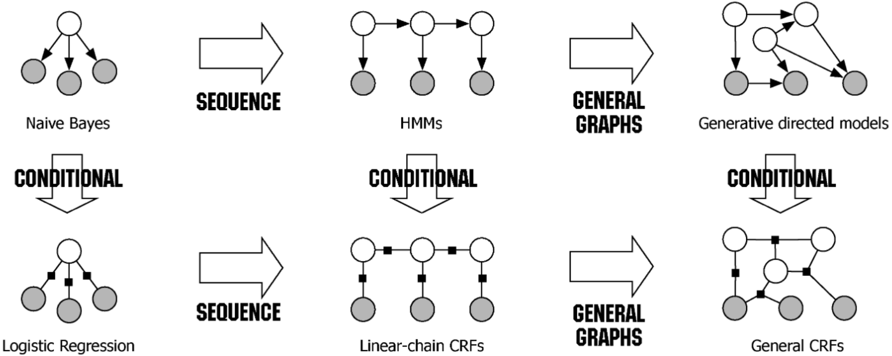


Figure 1: CRF in relation to other models; Source [2]

Model overview

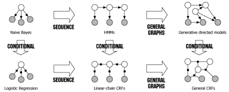


Figure 1: CRF in relation to other models; Source [2]

- There is a system of models with different properties.
- First, there is naive bayes and the conditional version, multinomial logistic regression.
- These model single class predictions. While naive bayes does this generatively, logistic regression uses the scoring mechanism.
- On sequences, we can either have the HMMs or a conditional version, which are linear chain CRFs.
- Finally there are models for which there is no clear correspondence between a latent variable and a single observed one.

HMM → Linear CRF

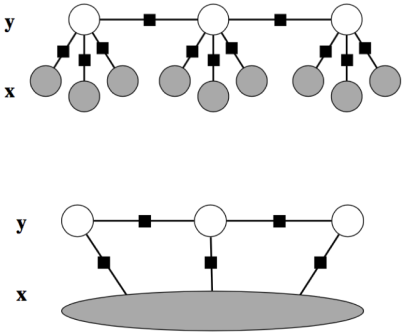


Figure 2: HMM vs. Linear Chain CRF; Source [12]

Question

- What is the difference between HMM and CRF?

Conditional Random Fields

- Factorization to maximal cliques.
- Allow access to a whole clique

Clique

$G = (V, E) \quad C \subseteq V : \forall x, y \in C : (x, y) \in E$

Maximal Clique

$C \subseteq C' \Rightarrow C = C'$

CRF

$p(y|x) = \frac{1}{Z(x)} \prod_{c \in C} \Psi_c(x_c)$

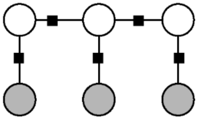


Figure 3: Linear CRF [2]

Conditional Random Fields

- Factorization to maximal cliques.
- Allow access to a whole clique

Clique
$G = (V, E) \quad C \subseteq V : \forall x, y \in C : (x, y) \in E$

Maximal Clique
$C \subseteq C' \Rightarrow C = C'$

CRF
$p(y|x) = \frac{1}{Z(x)} \prod_{c \in C} \Psi_c(x_c)$

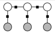


Figure 3: Linear CRF [2]

- We may generalize CRFs to allow access to more data in the feature functions
- This requires the graph to be factorized into maximal cliques on which we define the potential function
- Linear Chain CRFs fulfil these requirements, because they form a chain of latent variables, so maximal cliques are single nodes
- Explain cliques
- There is just a single decomposition into maximal clicque and it creates a factorization of the whole graph

Linear CRF

- Sequence of hidden states: y, {`MATNUM, PERSON, LOCATION, NONE`}
- Observed sequence of variables: x (words)

- $p(y|x) \propto \prod_i \exp\left\{\sum_j \lambda_j f_j(y_{i-1}, y_i, x, i)\right\}$

- $p(y|x) = \frac{1}{Z(x)} \prod_i \exp\left\{\sum_j \lambda_j f_j(y_{i-1}, y_i, x, i)\right\}$

- Features: $f_j(y_{i-1}, y_i, x, i)$
- Parameters: λ
- Clique template: $\{\Psi_i(y_{i-1}, y_i, x, i) | \forall i \in \{1...n\}\}$

Linear CRF

- Sequence of hidden states: y, {MATNUM, PERSON, LOCATION, NONE}
- Observed sequence of variables: x (words)
- $p(y|x) \propto \prod_i \exp\left\{ \sum_j \lambda_j f_j(y_{i-1}, y_i, x, i)\right\}$
- $p(y|x) = \frac{1}{Z(x)} \prod_i \exp\left\{ \sum_j \lambda_j f_j(y_{i-1}, y_i, x, i)\right\}$
- Features: $f_j(y_{i-1}, y_i, x, i)$
- Parameters: λ
- Clique template: $\{\Psi_i(y_{i-1}, y_i, x, i) | \forall i \in \{1...n\}\}$

- From the formulation we can see that it's again a discriminative model.
- The right side is not a probability, but rather a score, so we need to normalize it.
- Z, is the partition function for normalization (just like in softmax)

Linear CRF - Binary Features

$$f_j(y_{i-1}, y_i, x, i) = \begin{cases} 1 & \text{if } \text{cond}_f(y_{i-1}, y_i, x, i) \\ 0 & \text{else} \end{cases}$$

$$f_1(y_{i-1}, y_i, x, i) = \begin{cases} 1 & \text{if } x_{i-2} \text{ is capitalized} \\ 0 & \text{else} \end{cases}$$

$$f_a(y_{i-1}, y_i, x, i) = \begin{cases} 1 & \text{if } y_{i-1} = \texttt{number} \wedge y_t = \texttt{none} \\ 0 & \text{else} \end{cases}$$

$$\lambda_a = a(\text{number}, \texttt{none})$$

$$f_o(y_{i-1}, y_i, x, i) = \begin{cases} 1 & \text{if } y_i = \texttt{number} \wedge x_i = \texttt{<num>} \\ 0 & \text{else} \end{cases}$$

$$\lambda_o = o(\text{number}, \texttt{<num>})$$

Linear CRF - Binary Features

$$f_j(y_{i-1}, y_i, x, i) = \begin{cases} 1 & \text{if } \mathrm{cond}_f(y_{i-1}, y_i, x, i) \\ 0 & \text{else} \end{cases}$$

$$f_1(y_{i-1}, y_i, x, i) = \begin{cases} 1 & \text{if } x_{i-2} \text{ is capitalized} \\ 0 & \text{else} \end{cases}$$

$$f_a(y_{i-1}, y_i, x, i) = \begin{cases} 1 & \text{if } y_{i-1} = \texttt{number} \wedge y_t = \texttt{none} \\ 0 & \text{else} \end{cases}$$

$$\lambda_a = a(\text{number}, \texttt{none})$$

$$f_o(y_{i-1}, y_i, x, i) = \begin{cases} 1 & \text{if } y_i = \texttt{number} \wedge x_i = \texttt{<num>} \\ 0 & \text{else} \end{cases}$$

$$\lambda_o = o(\text{number}, \texttt{<num>})$$

- The feature functions here are indicators, that produce 1 in case of some conditions.
- These conditions have access to the current and the last latent variable, but also to all observed variables and the current position.
- This way we can emulate the log-linear 1st order sequential model by using these indicator functions and setting the corresponding variables.
- The theta parameters are learnable from the data.

Linear Chain CRF - Non-binary Features

$f_w(y_{i-1}, y_i, x, i) = |x_i|$ word length

$f_s(y_{i-1}, y_i, x, i) = |c|$ number of non-alphabetic characters

Questions

- How do we interpret the values of λ_j for the features f_j? ($\lambda_j > 0$, $\lambda_j = 0$, $\lambda_j < 0$?)
- How are λs estimated?
- How many such features can we create?

Linear Chain CRF - Non-binary Features

$f_w(y_{i-1}, y_i, x, i) = |x_i|$ word length
$f_s(y_{i-1}, y_i, x, i) = |c|$ number of non-alphabetic characters

Questions

- How do we interpret the values of λ_j for the features f_j? ($\lambda_j > 0$, $\lambda_j = 0$, $\lambda_j < 0$?)
- How are λs estimated?
- How many such features can we create?

- In CRFs it is common to have an order of thousands features
- Multiple features can be active for a certain sequence and so, CRFs tend to overlap features, which HMMs cannot do.

CRF - Operations

Training:

$$argmax_\lambda \; p(y_D|x_D, \lambda)$$

Interpretation: Given label sequences and inputs, find parameters of the CRF M that maximise $p(y|x, \lambda)$.
Done using gradient methods, Forward-Backward algorithm etc.

Inference (Viterbi):

$$argmax_y \; p(y|x, \lambda)$$

Interpretation: Given input x and CRF M, find optimal y.

Decoding (forward):

$$max_y \; p(y|x, \lambda)$$

Assignment 10 + Conditional Random Fields

└CRF - Operations

CRF - Operations

Training: $argmax_{\lambda}\ p(y_D|x_D, \lambda)$

Interpretation: Given label sequences and inputs, find parameters of the CRF M that maximise $p(y|x, \lambda)$.
Done using gradient methods, Forward-Backward algorithm etc.

Inference (Viterbi): $argmax_y\ p(y|x, \lambda)$

Interpretation: Given input x and CRF M, find optimal y.

Decoding (forward): $max_y\ p(y|x, \lambda)$

- Inference - viterbi
- Decoding - forward
- Training - gradient methods

Linear Chain CRF - Estimating λ

Gradient descent (ascent):

$$\frac{\partial \log p(y|x,\lambda)}{\partial \lambda_i} = \sum_{t=1}^{T} f_i(y_{t-1}, y_t, x, t) - \sum_{y'} \sum_{t=1}^{T} f_i(y'_{t-1}, y'_t, x, t) \cdot p(y'|x)$$

$$\lambda_f \leftarrow \lambda_f + \epsilon \left[\sum_{t=1}^{T} F(y_{t-1}, y_t, x, t) - \sum_{y'} \sum_{t=1}^{T} F(y'_{t-1}, y'_t, x, t) \cdot p(y'|x,\lambda) \right]$$

Limited-memory BFGS (quasi-Newton method)

Linear Chain CRF - Estimating λ

Gradient descent (ascent):

$$\frac{\partial \log p(y|x,\lambda)}{\partial \lambda_i} = \sum_{t=1}^{T} f_i(y_{t-1}, y_t, x, t) - \sum_{y'} \sum_{t=1}^{T} f_i(y'_{t-1}, y'_t, x, t) \cdot p(y'|x)$$

$$\lambda_f \leftarrow \lambda_f + \epsilon \left[\sum_{t=1}^{T} F(y_{t-1}, y_t, x, t) - \sum_{y'} \sum_{t=1}^{T} F(y'_{t-1}, y'_t, x, t) \cdot p(y'|x,\lambda) \right]$$

Limited-memory BFGS (quasi-Newton method)

- As for the parameter estimation, there exists a solution, since the function is concave (negative is convex).
- It can be reached iteratively as no closed-form exists
- Note that we are adding the gradient, that's because we want to maximize the objective function.
- For the actual optimization, limited memory approximation of BFGS (Broyden–Fletcher–Goldfarb–Shanno) algorithm is used.
- It's a quasi Newtonian method, which means that it makes local approximation with the second term of the taylor expansion
- And for that it approximates the inverse of the hessian matrix

Feature selection:

Alternative 1

1. Start with all features.
2. a. If there exists a feature removing which worsens the performance by $< t$, remove it. Repeat 2.
3. b. If not, exit.

Alternative 2

1. Start with no features.
2. a. If there exists a feature adding which improves the performance by $> t$, add it. Repeat 2.
3. b. If not, exit.

Properties

- Hard to setup & train
- Fast inference

Feature selection:

Alternative 1

1. Start with all features.
2. 2. If there exists a feature removing which worsens the performance by $< t$, remove it. Repeat 2.
3. 3. If not, exit.

Alternative 2

1. Start with no features.
2. 2. If there exists a feature adding which improves the performance by $> t$, add it. Repeat 2.
3. 3. If not, exit.

Properties

- Hard to setup & train
- Fast inference

- In practice, one may also wish to use just a limited number of features.
- When adding, it is possible to consider also combining with existing features. Especially for indicator features, it is possible to combine them using boolean operators.
- This can also be done in reverse - remove least useful features.

Linear Chain CRF - Regularization

Objective function:

$\mathcal{L} = \sum_s \log p(y^{(s)}|x^{(s)}, \lambda)$

LASSO:

$\mathcal{L}_{+lasso} = \sum_s \log p(y^{(s)}|x^{(s)}, \lambda) - \lambda_1 \sum_i |\lambda_i|$

Ridge:

$\mathcal{L}_{+ridge} = \sum_s \log p(y^{(s)}|x^{(s)}, \lambda) - \frac{\lambda_2}{2} \sum_i \lambda_i^2$

Elastic net:

$\mathcal{L}_{+elastic} = \sum_s \log p(y^{(s)}|x^{(s)} \lambda) - \frac{\lambda_2}{2} \sum_i \lambda_i^2 - \lambda_1 \sum_i |\lambda_i|$

Code

```
from sklearn_crfsuite import CRF
X_train = [
    [word2features(s, i) for i in range(len(s))]
    for s in train_sents]
y_train = [
    [label for token, postag, label in s]
    for s in train_sents]
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.1, c2=0.1,
    max_iterations=100,
)
crf.fit(X_train, y_train)
```

- Fast Linear Chain CRFs (C): http://www.chokkan.org/software/crfsuite/
- Fast Linear Chain CRFs (C++): https://taku910.github.io/crfpp/

Code

```
from sklearn_crfsuite import CRF
X_train = [
    [word2features(s, i) for i in range(len(s))]
    for s in train_sents]
y_train = [
    [label for token, postag, label in s]
    for s in train_sents]
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.1, c2=0.1,
    max_iterations=100,
)
crf.fit(X_train, y_train)
```

- Fast Linear Chain CRFs (C): http://www.chokkan.org/software/crfsuite/
- Fast Linear Chain CRFs (C++): https://taku910.github.io/crfpp/

c1 = lambda1, c2 = lambda2

Resources

1. Hidden Markov Model: https://web.stanford.edu/~jurafsky/slp3/A.pdf
2. Bayesian Networks: https://www.ics.uci.edu/~rickl/courses/cs-171/0-ihler-2016-fq/Lectures/Ihler-final/09b-BayesNet.pdf
3. Overview: https://www.analyticsvidhya.com/blog/2018/08/nlp-guide-conditional-random-fields-text-classification
4. Very detailed: http://homepages.inf.ed.ac.uk/csutton/publications/crftut-fnt.pdf
5. Academic-level introduction to CRF: https://www.youtube.com/watch?v=7L0MKKfqe98
6. Generalized CRF: https://people.cs.umass.edu/~wallach/technical_reports/wallach04conditional.pdf
7. Accessible introduction: http://pages.cs.wisc.edu/~jerryzhu/cs769/CRF.pdf
8. Forward-backward for CRF: https://www.cs.cornell.edu/courses/cs5740/2016sp/resources/collins_fb.pdf

Resources

9. NER using CRF: https://medium.com/data-science-in-your-pocket/named-entity-recognition-ner-using-conditional-random-fields-in-nlp-3660df22e95c
10. Python code: https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html#let-s-use-conll-2002-data-to-build-a-ner-system
11. Naïve Bayes, HMM, CRF: http://cnyah.com/2017/08/26/from-naive-bayes-to-linear-chain-CRF/
12. Highly Informative Naïve Bayes, HMM, MaxEnt, CRF: https://ls11-www.cs.tu-dortmund.de/_media/techreports/tr07-13.pdf