Conditional Random Fields

(SNLP tutorial)

Vilém Zouhar

July 6, 2021

Overview

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

Overview

- This part is mostly about sequence labeling and the different approaches which end with conditional random fields.

Sequence Labelling / Entity Recognition

- `My name is V. Zouhar, I live in Saarbrücken and my matriculation number is 1234.`
- `My name is [V. Zouhar:person], I live in [Saarbrücken:loc] and my matriculation number is [1234:mat-num].`
- NER as Sequence labeling:
  X: sequence of words
  Y: labels {`mat-num`, `person`, `location`, `none`}

Sequence Labelling / Entity Recognition

- My name is V. Zouhar, I live in Saarbrücken and my matriculation number is 1234.
- My name is [V. Zouhar:person], I live in [Saarbrücken:loc] and my matriculation number is [1234:mat-num].
- NER as Sequence labeling:
  X: sequence of words
  Y: labels {mat-num, person, location, none}

- NER can be reformulated as sequence labeling, 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.

HMM

- Hidden states: {`mat-num`, `person`, `location`, `none`}
- Better hidden states: {`mat-num`, `START+person`, `INTERNAL+person`, `END+person`, `location`, `none`, `...`}
- Transitions: MLE from annotated data
- Emission probabilities: MLE from annotated data ($+$ smoothing)

HMM: Estimation

- Hidden states: $\pi_1, \pi_2, ..., \pi_N$
- Labels/outputs: $x_1, x_2, ..., x_N$
- Transition probability: $p(\pi_i | \pi_{i-1})$
- Emission probability: $p(x_i | \pi_i)$
- $p(x_1, x_2, ..., x_N, \pi_1, \pi_2, ..., \pi_N) = \prod_i p(\pi_i | \pi_{i-1}) \cdot p(x_i | \pi_i)$
- Decision rule: $\underset{\pi_1, \pi_2, ..., \pi_N}{\arg\max} \left[\prod_i p(\pi_i | \pi_{i-1}) \cdot p(x_i | \pi_i) \right]$

HMM: Estimation

- Hidden states: $\pi_1, \pi_2, ..., \pi_N$
- Labels/outputs: $x_1, x_2, ..., x_N$
- Transition probability: $p(\pi_i|\pi_{i-1})$
- Emission probability: $p(x_i|\pi_i)$
- $p(x_1, x_2, ..., x_N, \pi_1, \pi_2, ..., \pi_N) = \prod_i p(\pi_i|\pi_{i-1}) \cdot p(x_i|\pi_i)$
- Decision rule: $\underset{\pi_1,\pi_2,...,\pi_N}{\arg\max} \left[\prod_i p(\pi_i|\pi_{i-1}) \cdot p(x_i|\pi_i) \right]$

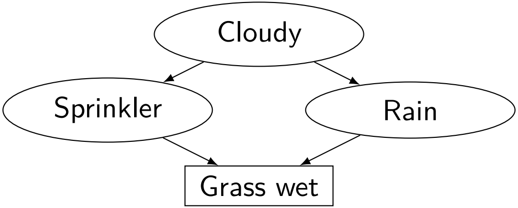
- HMMs seem a better fit for this task, since it captures transition probabilities between latent variables and emission probabilities.
- The probability of the sequence is computed as the product of transitions and observations.
- The probabilities can be estimated using MLE counting + some smoothing
- Side note, HMM is a generative model, because it can model the joint distribution p(y,x)
- In case we don't have annnotated data, we may still make use of HMMs by employing the Baum-Welsch 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 gaussian mixture.
- We usually require supervised examples to do this MLE counting, but the Baum-Welsch 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.

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.
$p(\text{Sprinkler}|\text{Cloudy}, \text{Rain}) = p(\text{Sprinkler}|\text{Cloudy})$

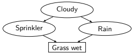


Bayesian Network

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

Local Markov Property

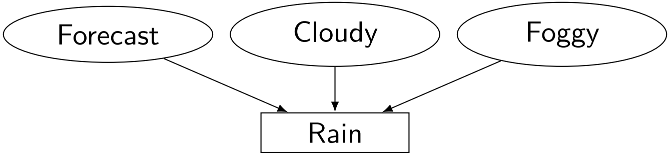
Node is conditionally independent of its nondescendants given its parents.
$p(\text{Sprinkler}|\text{Cloudy}, \text{Rain}) = p(\text{Sprinkler}|\text{Cloudy})$



- 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*.

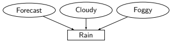
Naïve Bayes

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



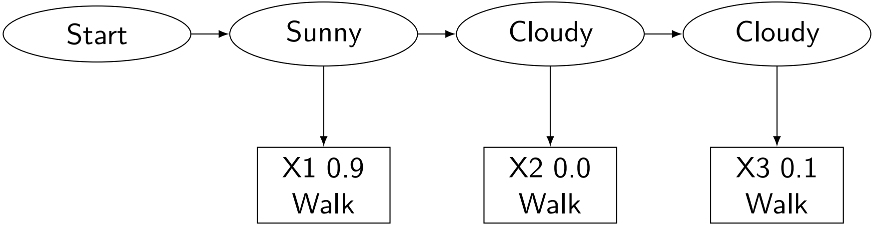
Naïve Bayes

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



- 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 *cloudyness* than to *forecast*.

HMM



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

$$p(\pi|x) = \prod_i p(\pi) \cdot p(x_i|\pi_i) \text{ (Naïve Bayes)}$$

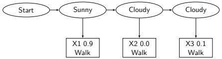
$$\Rightarrow$$

$$p(\pi_1, \pi_2, ..., \pi_N|x) = \prod_i p(\pi_i|\pi_{i-1}) \cdot p(x_i|\pi_i) \text{ (HMM)}$$

Conditional Random Fields

└HMM

HMM



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

$$p(\pi|x) = \prod_i p(\pi) \cdot p(x_i|\pi_i) \text{ (Naïve Bayes)}$$

$$\Rightarrow$$

$$p(\pi_1, \pi_2, ..., \pi_N|x) = \prod_i p(\pi_i|\pi_{i-1}) \cdot p(x_i|\pi_i) \text{ (HMM)}$$

- 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.

Logistic Regression

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

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

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

Log-linear 1st Order Sequential Model

- Sequence of hidden states: y, {`mat-num`, `person`, `location`, `none`}
- Observed sequence of variables: x (words)
- $p(y|x) \propto \exp\left\{ \sum_j \log a(y_{j-1}, y_j) + \log o(y_j, x_j) \right\}$
- $p(y|x) = \frac{1}{Z(x)} \cdot \exp\left\{ \sum_j \log a(y_{j-1}, y_j) + \log o(y_j, x_j) \right\}$
- $p(y|x) = \frac{1}{Z(x)} \cdot \prod_j \{ a(y_{j-1}, y_j) o(y_j, x_j) \}$
- $argmax\ p(y|x) \ldots$

Log-linear 1st Order Sequential Model

- Sequence of hidden states: y, `{mat-num, person, location, none}`
- Observed sequence of variables: x (words)
- $p(y|x) \propto \exp\left\{ \sum_j \log a(y_{j-1}, y_j) + \log o(y_j, x_j) \right\}$
- $p(y|x) = \frac{1}{Z(x)} \cdot \exp\left\{ \sum_j \log a(y_{j-1}, y_j) + \log o(y_j, x_j) \right\}$
- $p(y|x) = \frac{1}{Z(x)} \cdot \prod_j \{ a(y_{j-1}, y_j) o(y_j, x_j) \}$
- $argmax\ p(y|x) \ldots$

- 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:

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

$$= argmax\ \ F(y,x)$$

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

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

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

Log-linear 1st Order Sequential Model

Viterbi:

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

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

- 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:

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

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

$$\rightarrow$$

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

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

Log-linear 1st Order Sequential Model

Forward:

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

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

$$\rightarrow$$

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

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

- 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 $\theta_1 h_1(y_j, x_t) + \theta_2 h_2(y_j, x_t) + \ldots$
- Same with $a(y_j, y_i) = \theta'_1 g_1(y_j, y_i) + \theta'_2 g_2(y_j, y_i) + \ldots$
- Why not just $\sum_{\text{feature } f} \theta_i f_i(y_i, y_j, x_t)$?
- Why not allow $\sum_{\text{feature } f} \theta_i f_i(y_i, y_j, x, t)$?

Log-linear 1st Order Sequential Model

- Replace $o(y_j, x_t)$ with $\theta_1 h_1(y_j, x_t) + \theta_2 h_2(y_j, x_t) + \dots$
- Same with $a(y_j, y_i) = \theta'_1 g_1(y_j, y_i) + \theta'_2 g_2(y_j, y_i) + \dots$
- Why not just $\sum_{\text{feature } f} \theta_i f_i(y_i, y_j, x_t)$?
- Why not allow $\sum_{\text{feature } f} \theta_i f_i(y_i, y_j, x, t)$?

- *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

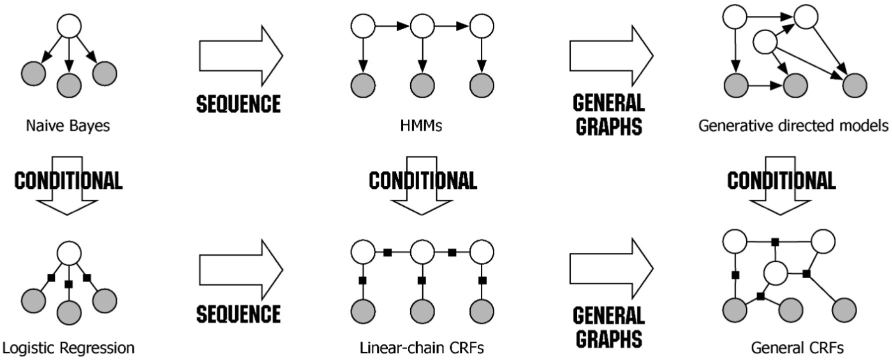


Figure 1: CRF in relation to other models; Source [2]

Model overview

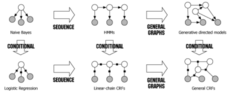


Figure 1: CRF in relation to other models; Source [2]

- 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 $\rightarrow$ Linear Chain CRF

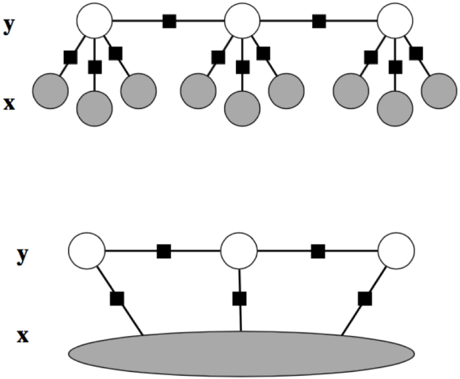


Figure 2: HMM vs. Linear Chain CRF; Source [12]

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.

Linear Chain CRF

- Sequence of hidden states: y, {`mat-num`, `person`, `location`, `none`}
- Observed sequence of variables: x (words)
- $p(y|x) \propto \prod_t \exp\left\{ \sum_{\text{feature } f} \theta_i f_i(y_{t-1}, y_t, x, t) \right\}$
- $p(y|x) = \frac{1}{Z(x)} \prod_t \exp\left\{ \sum_{\text{feature } f_i} \theta_i f_i(y_{t-1}, y_t, x, t) \right\}$
- Features: $f_i(y_{t-1}, y_t, x, t) \geq 0$
- Parameters: θ

Linear Chain CRF

- Sequence of hidden states: y, {`mat-num`, `person`, `location`, `none`}
- Observed sequence of variables: x (words)
- $p(y|x) \propto \prod_t \exp\{\sum_{\text{feature } f} \theta_i f_i(y_{t-1}, y_t, x, t)\}$
- $p(y|x) = \frac{1}{Z(x)} \prod_t \exp\{\sum_{\text{feature } f_i} \theta_i f_i(y_{t-1}, y_t, x, t)\}$
- Features: $f_i(y_{t-1}, y_t, x, t) \geq 0$
- Parameters: θ

- 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 Chain CRF - Features

$$f_i(y_{t-1}, y_t, x, t) = \begin{cases} 1 & \text{if } \text{cond}_f(y_{t-1}, y_t, x, t) \\ 0 & \text{else} \end{cases}$$

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

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

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

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

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

Linear Chain CRF - Features

$$f_i(y_{t-1}, y_t, x, t) = \begin{cases} 1 & \text{if } \mathrm{cond}_f(y_{t-1}, y_t, x, t) \\ 0 & \text{else} \end{cases}$$

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

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

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

$$f_o(y_{t-1}, y_t, x, t) = \begin{cases} 1 & \text{if } y_t = \texttt{number} \land x_t = \texttt{<num>} \\ 0 & \text{else} \end{cases}$$

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

- 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 - Features

$f_w(y_{t-1}, y_t, x, t) = x_t$ word length
$f_s(y_{t-1}, y_t, x, t) = x_t$ number of non-alphabetic characters

Linear Chain CRF - Features

$f_w(y_{t-1}, y_t, x, t) = x_t$ word length
$f_s(y_{t-1}, y_t, x, t) = x_t$ number of non-alphabetic characters

- In CRFs it is common to have an order of thousands features

CRF - Operations

Inference:

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

Decoding:

$$p(y|x, \theta)$$

Training:

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

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

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

Decoding: $p(y|x,\theta)$

Training: $argmax_\theta\ p(y_D|x_D,\theta)$

Linear Chain CRF - Estimating θ

Gradient descent (ascent):

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

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

Limited-memory BFGS (quasi-Newton method)

Linear Chain CRF - Estimating θ

Gradient descent (ascent):

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

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

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

Linear Chain CRF - Regularization

Objective function:

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

LASSO:

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

Ridge:

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

Elastic net:

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

General CRF

- Factorization to maximal clicques.
- Allow access to a whole clicque

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

$p(Y|X) = \frac{1}{Z(X)} \prod_{C \in Y} \Psi_C(X_C)$
$\Psi_C(Y, X) \sum_i \theta_i f_i(Y_{i-1}, Y_i, X, i) \geq 0$

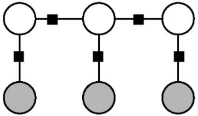


Figure 3: Linear Chain CRF [2]

General CRF

- Factorization to maximal clicques.
- Allow access to a whole clicque

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
$p(Y|X) = \frac{1}{Z(X)} \prod_{C \in Y} \Psi_C(X_C)$
$\Psi_C(Y, X) \sum_i \theta_i f_i(Y_{i-1}, Y_i, X, i) \geq 0$

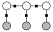


Figure 3: Linear Chain CRF [2]

- We may generalize CRFs to allow access to more data in the feature functions
- This requires the graph to be factorized into maximal clicques on which we define the potential function
- Linear Chain CRFs fulfill 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

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)
```

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)
```

c1 = lambda1, c2 = lambda2

Notes

Feature selection:

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.

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

Notes

Feature selection:

1. Start with all features.
2. 1. If there exists a feature removing which worsens the performance by $< t$, remove it. Repeat 2.
   2. If not, exit.
1. Start with no features.
2. 1. If there exists a feature adding which improves the performance by $> t$, add it. Repeat 2.
   2. 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 ones. Especially for indicator features, it is possible to combine them using boolean operators.
- This can also be done in reverse - remove least useful features.

Homework

TBD

Resources

1. Overview: https://www.analyticsvidhya.com/blog/2018/08/nlp-guide-conditional-random-fields-text-classification
2. Very detailed: http://homepages.inf.ed.ac.uk/csutton/publications/crftut-fnt.pdf
3. NER using CRF: https://medium.com/data-science-in-your-pocket/named-entity-recognition-ner-using-conditional-random-fields-in-nlp-3660df22e95c
4. Forward-backward for CRF: https://www.cs.cornell.edu/courses/cs5740/2016sp/resources/collins_fb.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. Python code: https://sklearn-crfsuite.readthedocs.io/en/latest/tutorial.html#let-s-use-conll-2002-data-to-build-a-ner-system

Resources

9. Fast Linear Chain CRFs (C): http://www.chokkan.org/software/crfsuite/
10. Fast Linear Chain CRFs (C++): https://taku910.github.io/crfpp/
11. Bayesian Networks: https://www.ics.uci.edu/~rickl/courses/cs-171/0-ihler-2016-fq/Lectures/Ihler-final/09b-BayesNet.pdf
12. Naïve Bayes to HMM to CRF: http://cnyah.com/2017/08/26/from-naive-bayes-to-linear-chain-CRF/