Discriminative Sequence Labeling

Natural Language Processing

Daniel Ortiz Martínez, David Buchaca Prats

Table of Contents

Introduction

Generative and Discriminative Machine Learning

- Let us suppose we want to determine the language that someone is speaking
- **Generative approach**: learn to speak in a set of candidate languages and use this knowledge to determine the language being used
- **Discriminative approach**: learn to discern between features that characterize the candidate languages without learning to speak in any of them

Taxonomy of Machine Learning Models

- **Generative models**:
  - Estimate joint probability distribution
  - Generative since they contain all the necessary information to generate synthetic data points
  - Popular example: HMMs
- **Discriminative models**:
  - Directly estimate posterior probabilities
  - Do not attempt to model underlying probability distributions
  - Focused on performance
  - Popular example: classic neural networks

Discriminative Sequence Labeling

Sequence Labeling

- Given a sequence of observations/feature vectors, determine an appropriate label/state for each observation
- Any single observation comes from a finite set Σ
- Any single label comes from a finite set Λ
- Prediction errors avoided by considering observation-to-state and state-to-state relations

Generative Sequence Labeling

- Sequence labeling tasks can be tackled following a generative approach
- One well known implementation alternative would be HMMs:

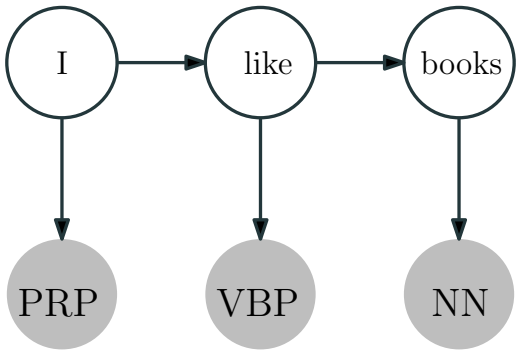
$$P(X = x_1, \ldots, x_N, Y = y_1, \ldots, y_N) = P_{\text{init}}(y_1|\text{start}) \cdot \prod_{i=1}^{N-1} P_{\text{trans}}(y_{i+1}|y_i) \cdot P_{\text{final}}(\text{stop}|y_N) \cdot \prod_{i=1}^{N} P_{\text{emiss}}(x_i|y_i)$$

- After estimating model probabilities, sequence labels can be obtained using Viterbi decoding:

$$y^* = \arg\max_{y \in \Lambda^N} P(Y = y_1, \ldots, y_N | X = x_1, \ldots, x_N)$$

Penn Treebank Example

- The Penn Treebank dataset is widely used in NLP research (https://aclanthology.org/J93-2004/)
- The task consists of annotating each word with its Part-Of-Speech tag



Penn Treebank Example: Tagset

CC	Coordinating conjunction	PRP$	Possessive pronoun
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Adverb, comparative
EX	Existential there	RBS	Adverb, superlative
FW	Foreign word	RP	Particle
IN	Preposition or subordinating conjunction	SYM	Symbol
JJ	Adjective	TO	to
JJR	Adjective, comparative	UH	Interjection
JJS	Adjective, superlative	VB	Verb, base form
LS	List item marker	VBD	Verb, past tense
MD	Modal	VBG	Verb, gerund or present participle
NN	Noun, singular or mass	VBN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non-3rd person singular present
NNP	Proper noun, singular	VBZ	Verb, 3rd person singular present
NNPS	Proper noun, plural	WDT	Wh-determiner
PDT	Predeterminer	WP	Wh-pronoun
POS	Possessive ending	WP$	Possessive wh-pronoun
PRP	Personal pronoun	WRB	Wh-adverb

Penn Treebank Example

- Penn Treebank tagset distinguishes singular and plural for nouns but not for determiners

Penn Treebank Example

- Penn Treebank tagset distinguishes singular and plural for nouns but not for determiners
- Consider tagging of the next two phrases:
  - "this fish"
  - "these fish"

Penn Treebank Example

- Penn Treebank tagset distinguishes singular and plural for nouns but not for determiners
- Consider tagging of the next two phrases:
  - "this fish"
  - "these fish"
- **Question**: can these phrases be correctly tagged using an HMM?

Disadvantages of Generative Sequence Labeling

- HMMs can only capture two phenomena:
  - Observation-to-state relations via emission probabilities $P_{\text{emiss}}(x_i|y_i)$
  - Local state-to-state relations via transition probabilities $P_{\text{trans}}(y_i|y_{i-1})$
- It would be desirable to incorporate general domain knowledge
- However, this is difficult when using generative models
  - Correct model decomposition
  - Computationally costly training and decoding processes

Discriminative Sequence Labeling: Motivation

- Let us imagine that we are asked to predict the label for a single observation x_i using two methodologies:
  - **Linear classifiers**: able to use arbitrarily defined features but unable to exploit knowledge about state dependencies
  - **HMMs**: able to capture state dependency information but unable to incorporate arbitrarily defined features
- Could it be possible to have the best of both worlds?

Discriminative Sequence Labeling: Motivation

- Let us imagine that we are asked to predict the label for a single observation x_i using two methodologies:
  - **Linear classifiers**: able to use arbitrarily defined features but unable to exploit knowledge about state dependencies
  - **HMMs**: able to capture state dependency information but unable to incorporate arbitrarily defined features

- Could it be possible to have the best of both worlds?

  Let us review some preliminary concepts...

Linear Classifiers

- In a classification problem, we are given the input X and want to find out which category it belongs to in a given label set Λ
- X is often represented as a feature vector $\phi(X)$
- Basic idea: to have a set of weight vectors $\{W_1, ..., W_{|\Lambda|}\}$, each one in the same dimension as $\phi(X)$
- Making predictions:

$$\hat{y} = \arg\max_y W_y^T \phi(X)$$

Linear Classifiers

- The model can alternatively be represented by means of
  - $W = (W_1, W_2, ..., W_{|\Lambda|})^T$
  - $\phi(X, y) = (\overrightarrow{0}_1, ..., \overrightarrow{0}_{y-1}, \phi(X), \overrightarrow{0}_{y+1}, ..., \overrightarrow{0}_{|\Lambda|})^T$
    where $\overrightarrow{0}_i$ is the i-th null vector (with same dimension as W_i)

- New prediction formula:

$$\hat{y} = \arg\max_y W^T \phi(X, y)$$

Linear Classifiers

- The weights of linear classifiers are typically learned from labeled data
  - $D \rightarrow \{(X^1, y^1), ..., ((X^M, y^M)\}$
- There are many algorithms for learning a linear classifier
- For its relevance to this content, we will review the *perceptron algorithm*

The Perceptron Algorithm

```
Perceptron(E, D, φ(·), r)
# E → number of epochs, D → dataset
# φ(·) → feature vector, r → learning rate
    W = 0
    for i = 1 to E
        for all X^(j), y^(j) in D
            y* = arg max_y W^T φ(X^(j), y)
            if y* ≠ y^(j)
                W ← W + r(φ(X^(j), y^(j)) − φ(X^(j), y*))
```

HMM as a Linear Model

- Starting from the HMM probability definition:

$$P(X, Y) = P_{\text{init}}(y_1 | \text{start}) \cdot \prod_{i=1}^{N-1} P_{\text{trans}}(y_{i+1} | y_i) \cdot$$
$$P_{\text{final}}(\text{stop} | y_N) \cdot \prod_{i=1}^{N} P_{\text{emiss}}(x_i | y_i)$$

- The log-probability is:

$$\log P(X, Y) = \log P_{\text{init}}(y_1 | \text{start}) + \sum_{i=1}^{N-1} \log P_{\text{trans}}(y_{i+1} | y_i) +$$
$$\log P_{\text{final}}(\text{stop} | y_N) + \sum_{i=1}^{N} \log P_{\text{emiss}}(x_i | y_i)$$

HMM as a Linear Model

- Let us examine the previous expression introducing a set of indicator functions[a]:

$$I_{[e]} = \begin{cases} 1, & e \text{ is true} \\ 0, & e \text{ is false} \end{cases}$$

[a] An indicator function is a function that maps booleans to 0 or 1

HMM as a Linear Model

- Introducing the indicator functions in the log-probability expression:

$$\log P(X, Y) = \sum_{y} \log P_{\text{init}}(y|\text{start}) I_{[y_1=y]} + \sum_{i} \sum_{y} \sum_{y'} \log P_{\text{trans}}(y'|y) I_{[y_i=y, y_{i+1}=y']} +$$
$$\sum_{y} \log P_{\text{final}}(\text{stop}|y) I_{[y_N=y]} + \sum_{i} \sum_{x} \sum_{y} \log P_{\text{emiss}}(x|y) I_{[x_i=x, y_i=y]}$$

- Reordering terms:

$$\log P(X, Y) = \sum_{y} \log P_{\text{init}}(y|\text{start}) I_{[y_1=y]} + \sum_{y} \sum_{y'} \log P_{\text{trans}}(y'|y) \sum_{i} I_{[y_i=y, y_{i+1}=y']} +$$
$$\sum_{y} \log P_{\text{final}}(\text{stop}|y) I_{[y_N=y]} + \sum_{x} \sum_{y} \log P_{\text{emiss}}(x|y) \sum_{i} I_{[x_i=x, y_i=y]}$$

HMM as a Linear Model

- Certain terms can be interpreted as counts:
  - $\sum_i I_{[y_i=y, y_{i+1}=y']}$: number of times transition $y \to y'$ occurs in Y
  - $\sum_i I_{[x_i=x, y_i=y]}$: number of times x is emitted from state y in (X, Y)

- Resulting in the following expression:

$$\log P(X, Y) = \sum_y \log P_{\text{init}}(y|\text{start}) I_{[y_1=y]} + \sum_y \sum_{y'} \log P_{\text{trans}}(y'|y) \cdot C_{Y}(y, y') +$$
$$\sum_y \log P_{\text{final}}(\text{stop}|y) I_{[y_N=y]} + \sum_x \sum_y \log P_{\text{emiss}}(x|y) \cdot C_{X,Y}(x, y)$$

HMM as a Linear Model

- The log-probability can be expressed in matrix form as follows:

$$\log P(X, Y) = \begin{bmatrix} \log P_{\text{init}}(\cdot) \\ \text{vec}(\log P_{\text{trans}}(\cdot)) \\ \text{vec}(\log P_{\text{emiss}}(\cdot)) \\ \log P_{\text{final}}(\cdot) \end{bmatrix}^T \begin{bmatrix} e_{y_1} \\ \text{vec}(C_Y) \\ \text{vec}(C_{X,Y}) \\ e_{y_N} \end{bmatrix}$$

where e_{y_i} represents the y_i-th standard basis and vec concatenates matrix columns into a single vector

HMM as a Linear Model

- Log-probability is calculated in terms of two vectors:
  - Left vector: contains model parameters, does not depend on X or Y
  - Right vector: describes X and Y, knows nothing about the model

HMM as a Linear Model

- Log-probability is calculated in terms of two vectors:
  - Left vector: contains model parameters, does not depend on X or Y
  - Right vector: describes X and Y, knows nothing about the model

- This is a linear model!
  - The left vector can be seen as the weight vector W
  - The right vector can be seen as the feature vector $\Phi(X, Y)$

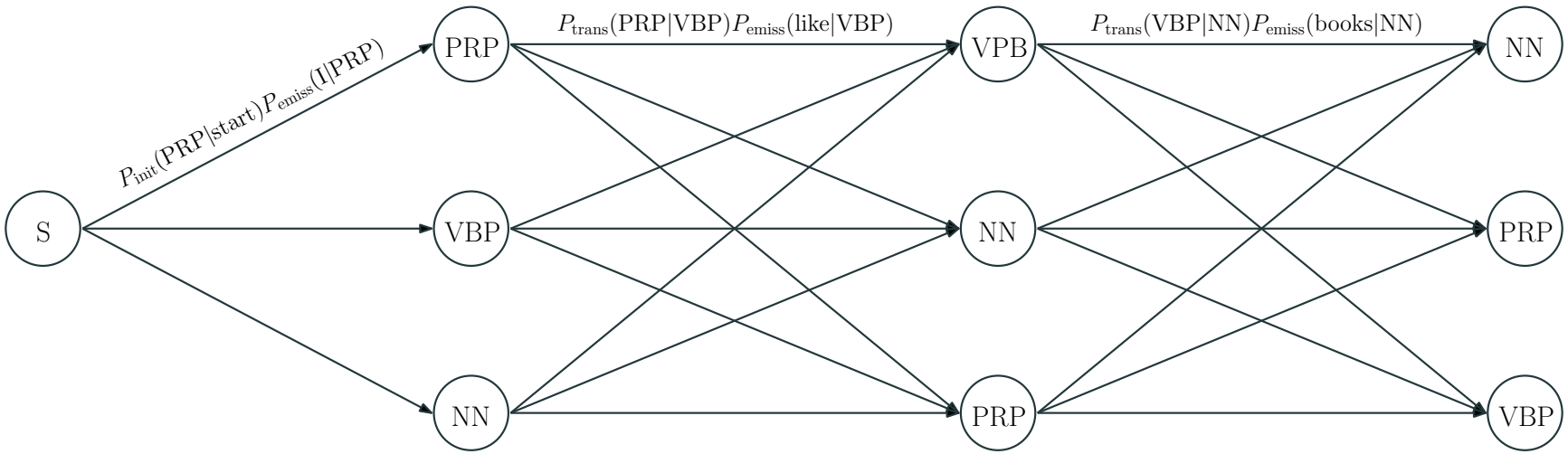
$$\log P(X, Y) = W^T \Phi(X, Y)$$

Viterbi Algorithm for HMM as a Linear Model

- Given a sequence X, we will be interested in generating the labels Y
- With conventional HMMs this is formalized as:

$$Y^* = \arg\max_Y P(Y|X)$$

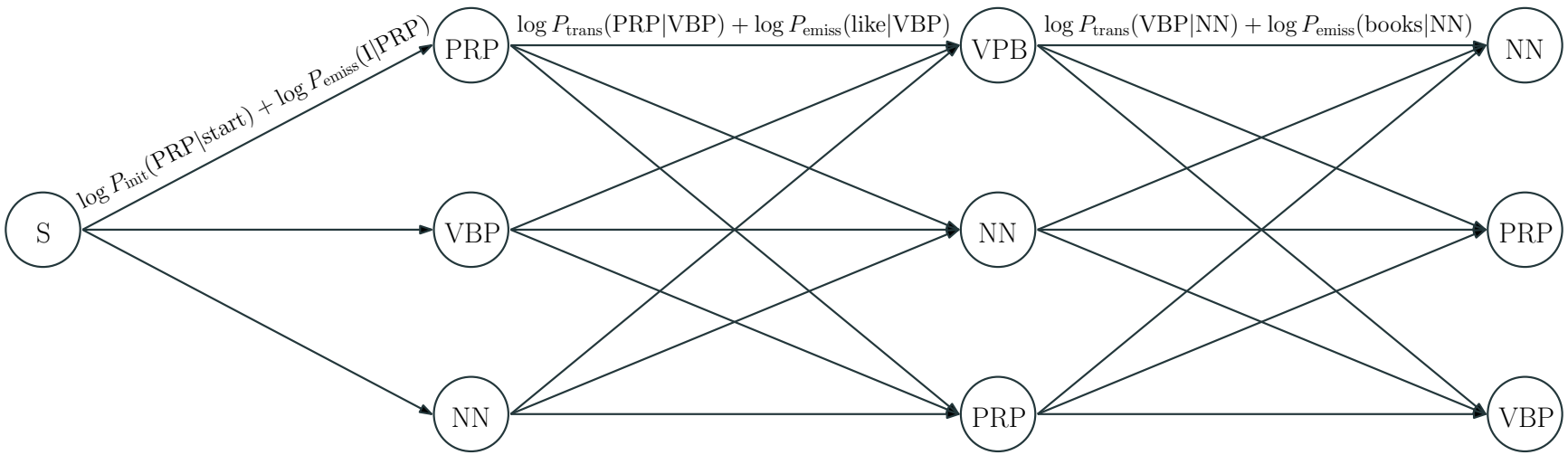
- Let us consider the trellis generated by the Viterbi algorithm for the input "I like books"[b]:



[b] $P_{\text{final}}(\text{stop}|\text{NN})$ omitted at last edge for simplicity

Viterbi Algorithm for HMM as a Linear Model

- Would the Viterbi algorithm be valid for the HMM as a linear model?
- Taking logs in the previous graph and replacing products by sums:



Viterbi Algorithm for HMM as a Linear Model

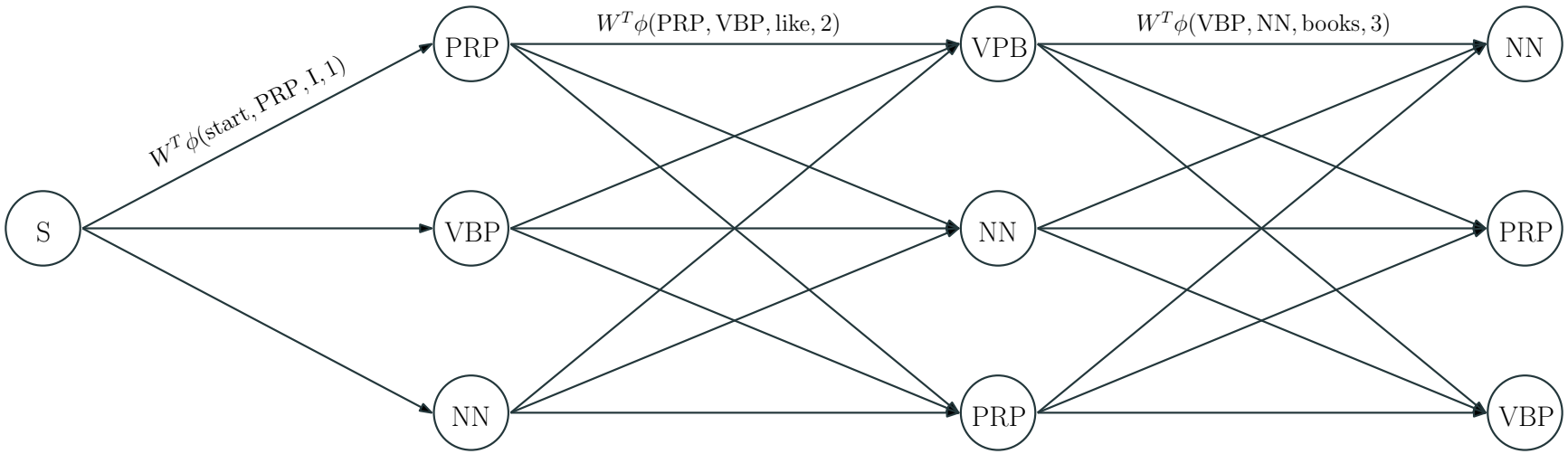
- The terms in the previous graph are included in the weight vector W
- Introducing some notation:
  - (y, y', x, i): edge connecting y with y' with emission of x at position i
  - $\phi(y, y', x, i)$: features of edge (y, y', x, i)
- $\phi(y, y', x, i)$ can be defined so as to verify[c]:

$$\Phi(X, Y) = \sum_{i=1}^{N} \phi(y_{i-1}, y_i, x_i, i)$$

[c] $y_0 \equiv \text{start}$

Viterbi Algorithm for HMM as a Linear Model

- Viterbi algorithm only requires replacing edge's probabilities by $W^T \phi(y_{i-1}, y_i, x_i, i)$:



Extending HMMs

- HMMs are a special case of linear model
- Viterbi algorithm works as long as we can map the feature vector onto each edge
- We can define features beyond those used in HMMs
- Constraints imposed over the features $\phi(\cdot)$:
  - $\phi(\cdot)$ should return a real-valued vector
  - $\phi(\cdot)$ should be a function of (y, y', X, i)

Extending HMMs: Learning the Weights

- We have seen that it is possible to define more general features
- However, the labeling system would be incomplete without a mechanism to estimate the value of the weight vector
- One way to learn the weights is the so-called *structured perceptron algorithm*

The Structured Perceptron

The Structured Perceptron

- The structured perceptron is a learning algorithm for structured prediction
- Structured prediction refers to working with objects that have a structure (e.g. sequences) as opposed to discrete or real values
- Strongly based on the perceptron algorithm

The Structured Perceptron

```
StructuredPerceptron(E, D, φ(·), r)
# E → number of epochs, D → dataset
# Φ(·) → feature vector, r → learning rate
    W = 0
    for i = 1 to E
        for all X^(j), Y^(j) in D
            Y* = arg max_Y W^T Φ(X^(j), Y)
            if Y* ≠ Y^(j)
                W ← W + r(Φ(X^(j), Y^(j)) − Φ(X^(j), Y*))
```

The Structured Perceptron for Sequence Labeling

- The previous algorithm can be straightforwardly instantiated for its use in sequence labeling
- X and Y will be sequences of words and labels, respectively
- $\Phi(X, Y)$ can be defined as a sum of HMM-like features, $\phi(y, y', X, i)$
- The $\arg\max$ operation can be implemented by means of the Viterbi algorithm

HMM Features

- The following features can be easily mapped to the different HMM distributions:

Conditions	
$y = \text{start}, y' = c$, $i = 1$	Initial features
$y = c$, $y' = \tilde{c}$	Transition features
$y' = c$, $i = N$	Final features
$y' = c$, $X = w$	Emission features

Beyond HMM Features

- Let us consider P_{set} and S_{set} to be the set of prefixes and suffixes set by the user

Conditions	
$X = w$, w is uppercased, $y' = c$	Upper case features
$X = w$, w contains digit, $y' = c$	Digit features
$X = w$, w contains hyphen, $y' = c$	Hyphen features
$X = w$, $w[0 : i] \in P_{\text{set}} \forall i \in \{1, 2, 3\}$, $y' = c$	Prefix features
$X = w$, $w[-i : 1] \in S_{\text{set}} \forall i \in \{1, 2, 3\}$, $y' = c$	Suffix features

Beyond HMM Features

- Let us consider a situation where prefixes can be useful
- Note that organic compounds contain specific prefixes, if we were tagging chemistry texts this knowledge could help us improve accuracy

#Carbons	Name
1	**meth**ane
2	**eth**ane
3	**prop**ane
4	**but**ane
5	**pent**ane
6	**hex**ane
7	**hept**ane
8	**oct**ane
9	**nona**ane
10	**dec**ane
11	**undec**ane
12	**dodec**ane

How Can We Treat Such Features?

- It does not seem very reasonable to manually create a feature that captures a relationship between a word and a label (for each word in the vocabulary)
  - Nevertheless we can use a feature mapper for this
  - Each position of the feature vector might capture a different feature

- Examples of POS tagging features:

Feature	**Type**	**Conditions**
prev_tag:noun::verb	Transition	$y = c$ and $y' = \tilde{c}$
id:The::det	Emission	$y' = c$ and $X = w$
id:will::verb	Emission	$y' = c$ and $X = w$

Feature Mapper

- A `feature_mapper` will store a dictionary `feature_dict`: $C^* \longrightarrow \mathbb{N}$ where C is the alphabet.
- Keys are strings that encode a feature
- Values are the positions of the features in the feature vector

```
{'init_tag:adp': 0,
 'id:In::adp': 1,
 'id:an::det': 2,
 'prev_tag:adp::det': 3,
 ...
}
```