POS Tagging using MEMM and CRF++

Parameswari

Maximum Entropy Markov Model (MEMM) for POS Tagging

- **What is MEMM?**
- A **sequence model** used for **Part-of-Speech (POS) tagging**, Named Entity Recognition (NER), etc.
- Predicts a tag for each word by considering the **previous tag and features of the word**.
- Uses **Maximum Entropy (MaxEnt) probability models** instead of fixed probabilities (like in HMMs).

What is Maximum Entropy?

- Maximum Entropy (MaxEnt) is a **probabilistic modeling technique** based on the principle that, given limited information, the probability distribution should be as uniform as possible while still satisfying known constraints.
- It does **not assume any prior knowledge** beyond what is explicitly provided in the data.
- Commonly used in NLP for **classification tasks**, including POS tagging, Named Entity Recognition (NER), and sentiment analysis.

- Given an **input (word + context)**, MaxEnt assigns a probability to each **possible tag**.
- The probabilities are computed using **log-linear models**, ensuring they maximize entropy while fitting the training data.

Formula:

$$P(y|x) = \frac{1}{Z(x)} \exp(\sum_i \lambda_i f_i(x, y))$$

where:

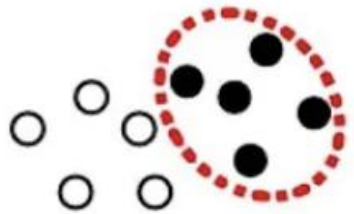
- $P(y|x)$ is the probability of assigning tag y to input x.
- $f_i(x, y)$ are feature functions capturing linguistic properties.
- λ_i are weights learned during training.
- $Z(x)$ is a normalization factor ensuring probabilities sum to 1.

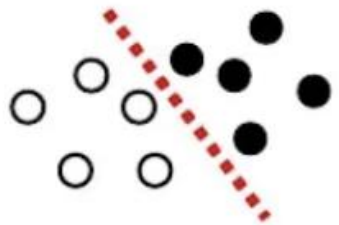
Discriminative Models

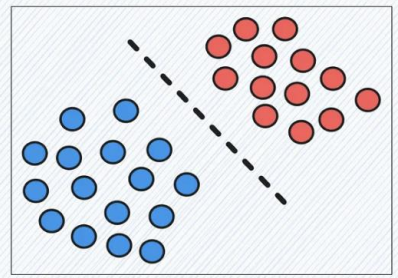
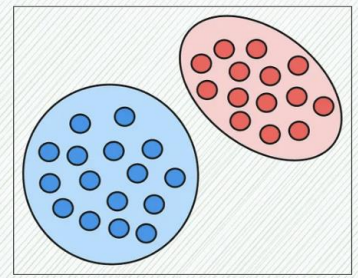
- Directly model **conditional probability** , focusing on **decision boundaries**.
- Examples: **Maximum Entropy (MaxEnt), Conditional Random Fields (CRFs), Support Vector Machines (SVMs)**.
- Used for **classification tasks**.
- Example: MEMM estimates directly without assuming word generation.

Generative vs. Discriminative

- Generative:
  - probabilistic “model” of each class
  - decision boundary:
    - where one model becomes more likely
  - natural use of unlabeled data
- Discriminative:
  - focus on the decision boundary
  - more powerful with lots of examples
  - not designed to use unlabeled data
  - only supervised tasks





Discriminative Models	Generative Models
	
Learns the decision boundary between classes	Learns the input distribution
Maximizes the conditional probability: P(Y|X)	Maximizes the joint probability: P(X, Y)
Directly estimates P(Y|X)	Estimates P(X|Y) to find P(Y|X) using Bayes' rule
Cannot generate new data	Can generate new data
Specifically meant for classification tasks	Typically, they are NOT used to solve classification tasks
Discriminative models don't possess generative properties	Generative models possess discriminative properties
Logistic Regression, Random Forests, SVMs	Hidden Markov Models, Naive Bayes, Gaussian Mixture Models
Neural Networks, Decision Tree, kNN	Gaussian Discriminant Analysis, LDA, Bayesian Networks

How MEMM Works?

- **Steps in POS Tagging with MEMM**
- Look at the **current word** and its **context** (neighboring words, capitalization, suffixes, etc.).
- Consider the **previous word's tag**.
- Use a **Maximum Entropy model** to compute the probability of each possible tag.
- Assign the tag with the **highest probability**.

- **Example Sentence**:
- **"The cat sleeps."**
- "The" → DT (Determiner)
- "cat" → NN (Noun) *(knowing "The" was DT helps)*
- "sleeps" → VB (Verb) *(knowing "cat" was NN helps)*

Feature	HMM (Hidden Markov Model)	MEMM (Maximum Entropy Markov Model)
Probability Type	Transition & Emission Probabilities	Conditional Probabilities
Context	Only **previous tags**	**Previous tags + Word Features**
Feature Flexibility	Fixed features (word-tag probs)	Custom features (capitalization, suffixes, etc.)
Label Bias Problem	**No**	**Yes**

What is the Label Bias Problem?

- MEMMs decide tags **one step at a time**, using only the **current state**.
- If a state has **fewer possible next tags**, it can **dominate predictions unfairly**.
- **Example:**
  - If a rare verb often follows "to," the model may **always** assign it a verb tag, even when incorrect.
- **Solution?** Use **Conditional Random Fields (CRFs)** to model the entire sequence.

- Consider the word **"lead"**. It can be a noun ("a lead pipe") or a verb ("I lead the team").
- If MEMM has seen "lead" more frequently as a **noun**, it might always assign it "NN" (Noun), even when it should be "VB" (Verb).

- MEMMs improve over HMMs by using **word features and conditional probabilities**.
- However, MEMMs suffer from the **label bias problem**.
- **CRFs (Conditional Random Fields)** solve this issue by considering **entire sequences** instead of independent decisions.

CRF++: Yet Another CRF toolkit

- **CRF++** is a simple, customizable, and open source implementation of Conditional Random Fields (CRFs) for segmenting/labeling sequential data.
- It is designed for generic purpose and will be applied to a variety of NLP tasks, such as POS tagging, Named Entity Recognition, Information Extraction and Text Chunking.

Installation

- Download CRF++ from here
- C++ is required to run CRF++.
- To install follow the below steps:
- % ./configure
- % make
- % su
- # make install

Training and testing

- Both the training file and the test file need to be in a particular format for **CRF++** to work properly.
- Generally speaking, training and test file must consist of multiple **tokens**.
- In addition, a **token** consists of multiple (but fixed-numbers) columns.
- Each token must be represented in one line, with the columns separated by white space (spaces or tabular characters). A sequence of token becomes a **sentence**.
- To identify the boundary between sentences, an empty line is put.
- The last column represents a true answer tag which is going to be trained by CRF.

Example training file

- He PRP B-NP
- reckons VBZ B-VP
- the DT B-NP
- current JJ I-NP
- account NN I-NP
- deficit NN I-NP
- will MD B-VP
- narrow VB I-VP
- to TO B-PP
- only RB B-NP
- # # I-NP
- 1.8 CD I-NP
- billion CD I-NP
- in IN B-PP
- September NNP B-NP
- . . O

Template file

- # Unigram
- U00:%x[-2,0]
- U01:%x[-1,0]
- U02:%x[0,0]
- U03:%x[1,0]
- U04:%x[2,0]
- U05:%x[-1,0]/%x[0,0]
- U06:%x[0,0]/%x[1,0]
-
- U10:%x[-2,1]
- U11:%x[-1,1]
- U12:%x[0,1]
- U13:%x[1,1]
- U14:%x[2,1]
- U15:%x[-2,1]/%x[-1,1]
- U16:%x[-1,1]/%x[0,1]
- U17:%x[0,1]/%x[1,1]
- U18:%x[1,1]/%x[2,1]
-

Explanation of template file

- Template file to be prepared before training in advance.
- Each line in the template file denotes one *template*. In each template, special macro *%x[row,col]* will be used to specify a token in the input data.
- *Row* specifies the relative position from the current focusing token and *col* specifies the absolute position of the column.
- Example:
- Input: Data
- He PRP B-NP
- reckons VBZ B-VP
- the DT B-NP << CURRENT TOKEN
- current JJ I-NP
- account NN I-NP

Example …(contd)

template	expanded feature
%x[0,0]	the
%x[0,1]	DT
%x[-1,0]	reckons
%x[-2,1]	PRP
%x[0,0]/%x[0,1]	the/DT
ABC%x[0,1]123	ABCDT123

Template types

- Unigram template: first character, **'U'**
- This is a template to describe unigram features.
- When you give a template "U01:%x[0,1]", CRF++ automatically generates a set of feature functions (func1 … funcN)
- Bigram template: first character, **'B'**
- This is a template to describe bigram features. With this template, a combination of the current output token and previous output token (bigram) is automatically generated.
- When the number of classes is large, **Bigram** templates would produce a tons of distinct features that would cause inefficiency both in training/testing.

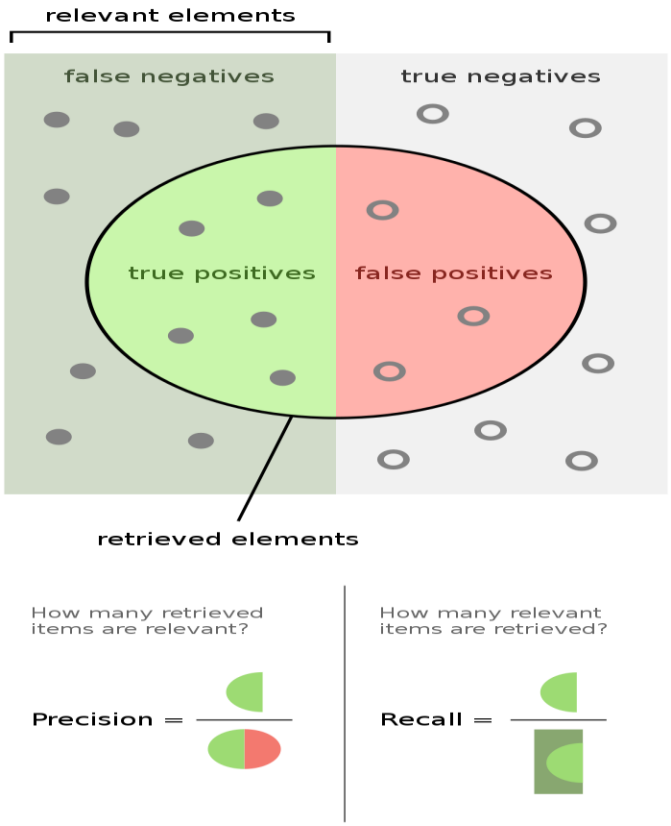
Training and Testing

- For training use the command **crf_learn template_file train_file model_file**
- Template file and train file are pre-requisites and model_file will be generated by the crf++
- For testing use the command **crf_test -m model_file test_files**
- Sample git hub repo for CRF train and test can be found here

Evaluation using Precision, Recall and F1-score

- True/False Positives and Negatives
- **Positive**: The instance is classified as a member of the class the classifier is trying to identify.
  - For example, a classifier looking for **book** as noun would classify **book** as positive (when correct).
- **Negative**: The instance is classified as not being a member of the class we are trying to identify.
  - For example, a classifier looking for book as noun , but it classifies as verb.

Precision, Recall and F1-score ... (contd)



CRFs use handcrafted or automatically extracted features such as:

- **Current word**: x_t
- **Previous and next words**: x_{t-1}, x_{t+1}
- **Prefix/suffix of the word**: E.g., "ing" in "running" suggests a verb.
- **Capitalization**: Indicates proper nouns.
- **Word shape**: E.g., "Xxx" for "John" or "xxx" for "apple."
- **Previous tag**: Helps in modeling transitions between tags.

- Precision

  - *Precision* is a measure of how many of the positive predictions made are correct (true positives).
  - The formula for it is:

$$Precision = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(FP)}$$

Recall / Sensitivity

- Recall is a measure of how many of the positive cases the classifier correctly predicted, over all the positive cases in the data. It is sometimes also referred to as Sensitivity.
- The formula for it is:

$$Recall = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negative(FN)}$$

F1-Score

- F1-Score is a measure combining both precision and recall. It is generally described as the harmonic mean of the two.
- Harmonic mean is just another way to calculate an “average” of values, generally described as more suitable for ratios (such as precision and recall) than the traditional arithmetic mean. The formula used for F1-score in this case is:

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Template

```
# Unigram
U00:%x[-2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[2,0]
U05:%x[-1,0]/%x[0,0]
U06:%x[0,0]/%x[1,0]

U10:%x[-2,1]
U11:%x[-1,1]
U12:%x[0,1]
U13:%x[1,1]
U14:%x[2,1]
U15:%x[-2,1]/%x[-1,1]
U16:%x[-1,1]/%x[0,1]
U17:%x[0,1]/%x[1,1]
U18:%x[1,1]/%x[2,1]

U20:%x[-2,1]/%x[-1,1]/%x[0,1]
U21:%x[-1,1]/%x[0,1]/%x[1,1]
U22:%x[0,1]/%x[1,1]/%x[2,1]
```

```
nagaraju@nagaraju-OptiPlex-Tower-7010:pos$ crf_learn template tt model
CRF++: Yet Another CRF Tool Kit
Copyright (C) 2005-2013 Taku Kudo, All rights reserved.

reading training data:
Done!0.00 s

Number of sentences: 2
Number of features:  1130
Number of thread(s): 20
Freq:                1
eta:                 0.00010
C:                   1.00000
shrinking size:      20
iter=0 terr=0.82353 serr=1.00000 act=1130 obj=27.36044 diff=1.00000
iter=1 terr=0.00000 serr=0.00000 act=1130 obj=12.16156 diff=0.55551
iter=2 terr=0.00000 serr=0.00000 act=1130 obj=5.25254 diff=0.56810
iter=3 terr=0.00000 serr=0.00000 act=1130 obj=5.12053 diff=0.02513
iter=4 terr=0.00000 serr=0.00000 act=1130 obj=5.10065 diff=0.00388
iter=5 terr=0.00000 serr=0.00000 act=1130 obj=5.10005 diff=0.00012
iter=6 terr=0.00000 serr=0.00000 act=1130 obj=5.09992 diff=0.00002
iter=7 terr=0.00000 serr=0.00000 act=1130 obj=5.09990 diff=0.00001
iter=8 terr=0.00000 serr=0.00000 act=1130 obj=5.09990 diff=0.00000

Done!0.03 s
```