CS388 Project 1: Sequence Models for Named-Entity Recognition

Kelsey Ball
The University of Texas at Austin
kelseytaylorball@gmail.com

Abstract

Named-entity recognition (NER) is the task of identifying references to named entities in a text. In this project, we implement two sequential models for use in an NER system: a Hidden Markov Model (HMM) and a Conditional Random Field (CRF), and evaluate their performance and speed. We also explore various optimizations to the CRF during feature extraction, training, and inference.

1 Introduction

A named entity is an object denoted by a particular name, such as a person, location, or organization. Named-entity recognition is the task of identifying references to named entities in a text. Names may contain a single word, such as *Walmart*, or multiple words, such as *New Zealand*. The word or words in a name are referred to as a named-entity "chunk"; the goal of an NER system is to label such chunks in a sentence.

1.1 Data

For training and evaluation of our models, we use data from the CoNLL 2003 Shared Task (Tjong Kim Sang and De Meulder, 2003). An example of the data is provided here:

```
EU NNP I-NP B-ORG
rejects VBZ I-VP O
German JJ I-NP B-MISC
call NN I-NP O
to TO I-VP O
boycott VB I-VP O
British JJ I-NP B-MISC
lamb NN I-NP O
. . O O
```

The last column contains the NER tag, which follows a begin-inside-outside (BIO) scheme: the `B-` prefix denotes the beginning of a tag; `I-` denotes the inside; and `O` denotes that the word is not part of a named entity. The suffix (e.g., `-ORG`) denotes the type of entity.

2 Models

We first implement a Hidden Markov Model where the transition probabilities and emission probabilities are computed through maximum-likelihood estimation over the training set. We then implement a Conditional Random Field, which uses fixed, rule-based transition potentials and learns a set of feature weights for emission potentials. We describe each model and their implementation below.

2.1 Hidden Markov Model

A Hidden Markov Model (HMM) models a sequence of "hidden" or latent states which correspond to a sequence of observations, or "emitted" states. In the NER problem setting, the latent states are named-entity tags, and the observations are words. The latent states form a Markov chain, in which future states are conditionally independent of past states given the current state. Further, the model assumes that observations are conditionally independent, given their latent states. These strong assumptions allow us to efficiently learn a generative model using maximum-likelihood estimation where

$$P(\boldsymbol{y}|\boldsymbol{x}) \propto P(y_1)\left[\prod_{i=2}^{n} P(y_i|y_{i-1})\right]\left[\prod_{i=2}^{n} P(x_i|y_i)\right]$$

To predict NER chunks on an unseen sequence, we use the Viterbi algorithm to compute and return the most likely sequence of NER tags.

2.2 Conditional Random Field

Like an HMM, a Conditional Random Field (CRF) models transition probabilities in the latent

sequence, and thus is useful for structured prediction. However, whereas a single observation in an HMM only depends on its latent state, a CRF can model dependencies across observations. Our sequential CRF assigns the conditional probability of a tag sequence given a word sequence using a transition "potential" (or score) and an emission potential:

$$P(\boldsymbol{y}|\boldsymbol{x}) \propto \exp w^T \left[\sum_{i=2}^{n} f_t(y_i|y_{i-1}) + \sum_{i=2}^{n} f_e(y_i, i, \boldsymbol{x}) \right]$$

f_t is a rule-based function that assigns zero probability to illegal transitions and uniform probabilities across legal transitions. f_e produces a feature vector for a word based on lexical features and neighboring words. We only learn weights in w which are associated with the output of f_e.

Like the HMM, the CRF also uses Viterbi decoding to compute the most likely sequence of tags during inference. We train our baseline CRF model by maximizing the conditional log-likelihood of our training data using stochastic gradient ascent with a step-size of 1.0 for 5 epochs.

2.3 Comparison

Table 1 compares the performance of the HMM and baseline CRF model on the development set. The HMM trains much faster, given that maximum-likelihood estimation reduces to counting pairs of words and tags in the training set, and requires no gradient-based optimization. Inference is also faster for the HMM, because emission probabilities are pre-computed during training, whereas the CRF must extract features and evaluate an inner product during inference to compute an emission potential. However, the CRF achieves a significantly higher F1 score.

Model	Dev F1	Train time	Inf. time
HMM	76.89	1.52s	6.79s
CRF	81.75	2501.23s	189.20s

Table 1: HMM vs. CRF.

3 Extension

Our project extension explores speedups to the CRF model. In this section we discuss optimizations to the pre-processing, training, and inference of our CRF model.

3.1 Pre-processing

Feature extraction: The baseline implementation extracts word features for each (word, tag) pair; however, the tag is only used trivially during extraction. To speed up this function, we extract a set of features once per word, then subsequently re-label the feature set for each tag.

We measure the effect of this speedup on pre-processing, during which we cache feature vectors for all (word, tag) pairs in the training set. Our optimization produced a 40.61% reduction in feature-extraction time, representing an absolute reduction of 14.17 seconds.

We also measure the effect of this speedup during inference by measuring the total inference time over the development set. Our optimization produced a 2.19% relative speedup and an absolute reduction of 4.14 seconds, indicating that a relatively small portion of decoding time is dedicated to feature extraction.

Method	Preprocess time	Inf. time
Baseline	34.89s	189.20s
Optimized	20.71s	185.07s

Table 2: Optimized feature extraction.

3.2 Training

Optimizers. In Figure 1, we compare the rate of convergence of training and total train time using different optimizers: a vanilla SGD optimizer with step size = 1.0 and an unregularized Adagrad optimizer with $\eta = 1.0$. Using the unregularized Adagrad optimizer, training converges faster and reaches a much higher F1-score of 88.36.

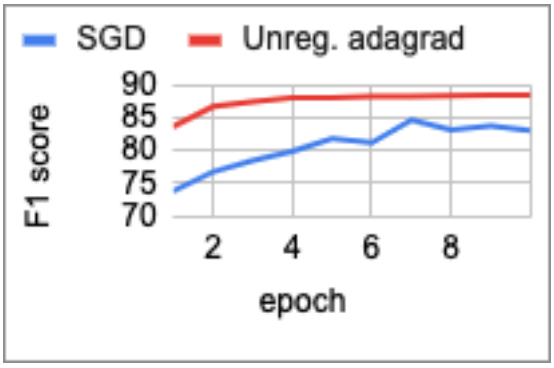

Figure 1: F1 score per optimizer

Gradient update computation. To store the gradient, we make use python's `Counter` collection. Our baseline implementation used the $+$ operator to add sparse feature vectors (also implemented as `Counter`'s) to the gradient; however, we discovered that `Counter.update` provided a faster equivalent operation, producing a $\sim 1\%$ relative reduction in train time over 1 epoch on a small set of 5000 examples.

3.3 Inference

Beam search. Beam search is a generalization of a greedy algorithm in which we maintain a list of the top-k candidates in each step of the search. In the context of Viterbi decoding, we consider the k most likely tags from the previous timestep, rather than considering all possible transitions. By reducing the size of the search space, beam search speeds up inference.

In Table 2, we report the relative speedup for different beam "widths" (i.e., values of k) for total inference time. The baseline model uses Viterbi decoding without beam search, where transitions from all 9 possible previous tags are considered at each timestep. For these experiments, we use our best model (trained with unregularized Adagrad over 10 epochs).

Beam width	Dev F1	Inf. time	Speedup
No beam (k=9)	88.36	161.86s	-
k=3	31.74	154.13s	4.78%
k=5	53.61	159.54s	1.43%
k=7	64.69	161.55s	.19%

Table 3: Relative inference speedup for different beam widths.

Inference time decreases marginally as the beam width decreases; however, the speedup comes at a significant cost to performance.

References

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. *I*ntroduction to the CoNLL-2003 Shared Task: Language Independent Named Entity Recognition. In Proceedings of the Conference on Natural Language Learning. (CoNLL).