Named Entity Recognition

Manav Bilakhia
Union College CS Department Schenectady NY
bilakhim@union.edu

1 Introduction

Named Entity Recognition is a technique that involves finding and classifying named entities in a piece of text. In simpler terms, it's like identifying and categorizing specific words or phrases that represent people, places, organizations, or other things. This process is similar to part of speech tagging, which assigns a grammatical label to each word in a sentence. However, NER uses a smaller number of tags compared to part of speech tagging, and most words are labeled as "non-entity" or O tags. In other words, NER focuses on a specific set of words or phrases, rather than every word in a sentence. (Jurafsky and Martin, 2019) We obtain our data from the named entity recognition shared task for Spanish and Dutch, which was a part of the Conference on Natural Language Learning (CoNLL) 2002.

The features were specifically selected to present the best results with the Spanish language dataset. The tagging system used in this data consists of four categories: PER for Person, LOC for Location, ORG for Organization, and MISC for miscellaneous named entities. The data is encoded using BIO notation, which means that each named entity tag is either B- for beginning or I- for inside, depending on the position of the word in the named entity.

2 Features

The following list of features were experimented with in order to find the best possible combination.

Process of Elimination: The initial classifier included all the features, both those that were later removed and those that were kept in the final code. The overall FScore of this classifier was 60.64. To improve performance, each feature was excluded one by one, and the resulting change in FScore was observed. Only the features that increased the FScore were retained in the final code, while those that had no effect or decreased the FScore were removed.

Features that made it to the final algorithm:

1. The word itself, labeled with the given offset (o) followed by the string 'word'. The feature came with the starter code.

2. A Boolean value indicating whether the word is all uppercase, labeled with the offset followed by 'is_upper'.

3. A Boolean value indicating whether the word is all lowercase, labeled with the offset followed by 'is_lower'

4. A Boolean value indicating whether the word is title-cased, labeled with the offset followed by 'is_title'

5. A Boolean value indicating whether the word is a digit, labeled with the offset followed by 'is_digit'

6. A Boolean value indicating whether the word ends with 'ing', labeled with the offset followed by 'ends_with_ing'

Features that were experimented with and removed from the algorithm:

1. A Boolean value indicating whether the word ends with 's', labeled with the offset followed by 'ends_with_s'. The feature was pulling the FScore down.

2. A Boolean value indicating whether the word ends with 'able', labeled with the offset followed by 'ends_with_able'. The feature was pulling the FScore down.

3. A Boolean value indicating whether the word ends with 'ly', labeled with the offset followed by 'ends_with_ly'. The feature was pulling the FScore down.

4. If the word contains a hyphen, the function also includes a Boolean value indicating this, labeled with the offset followed by 'word.hyphenated'.The feature was pulling the FScore down.

5. If the word ends with 'mente', the function also includes a Boolean value indicating this, labeled with the offset followed by 'word.adverb'. This feature did not change the FScore.

6. A Boolean value indicating whether the word has an apostrophe, labeled with the offset followed by 'contains_apostrophe'. This feature did not change the FScore.

These features were added/ removed on the basis of how it affected the FScore for the development data while NER was being used as a classifier. The final results of this evaluation are shown in table 1.

3 Basic Evaluation Matrix

This shows the final evaluation matrices for NER implemented as a classification task and for NER as a sequence tagging task.

4 MEMM and The Viterbi Algorithm

A Maximum Entropy Markov Model (MEMM) is a probabilistic model that combines the power of Maximum Entropy and Markov Models to predict the most likely sequence of labels for a given sequence of input observations. In a MEMM, the label of the current observation depends not only on the observation itself but also on the label of the previous observation, which is used as a feature for the current label prediction. The formula used to calculate the probability of the tag sequence is similar to that of an HMM, but instead of multiplying transition and observation probabilities, it multiplies probabilities predicted by the classifier. This allows for more accurate predictions and better handling of complex linguistic structures.

The Viterbi algorithm is a dynamic programming algorithm. It works by creating a trellis, a table with the same number of rows as the number of observations and the same number of columns as the number of possible states. The algorithm iterates over each observation, calculating the probability of each possible state given the observation and the probability of transitioning from the previous state. The maximum probability is chosen as the most likely state for that observation, and the backpointer to the previous state is recorded. This process is repeated until all observations are processed, and the most likely sequence of states can be traced back through the backpointers.

For this project, we have implemented an adapted version of the Viterbi algorithm that decodes based on the output of a Maximum Entropy Markov Model (MEMM). It takes as input a sequence of observations, an instance of the MEMM class, and a boolean flag indicating whether or not to pretty print the trellis. The function first initializes the trellis with the scores and backpointers for the first observation and then fills in the rest of the trellis by iterating over the remaining observations and possible states. The highest-scoring path is then found by tracing back through the backpointers.

Including the previous word's predicted tag in the MEMM model significantly improved the performance of a named entity recognizer. This is because the previous tag provides additional context that can help disambiguate the current word's tag. The Viterbi algorithm uses this additional context to find the most likely sequence of tags for a given sentence. By incorporating the previously predicted tag, the MEMM model is able to capture more complex dependencies between words and tags than a simple Markov model. This can lead to better FScore in assigning tags to words, especially in cases where there are multiple possible tags for a given word. With our datasets, we found that having the previous tags increased our overall FScore from 61% to 68% in the development data evaluation and from 64% to 73% in our testing data as shown by tables 1 2 3 and 4.

5 Honor Code

I affirm that I have carried out my academic endeavors with full academic honesty. [Manav Bilakhia]

References

Dan Jurafsky and James H Martin. 2019. Speech and language processing (3rd (draft) ed.).

Kristina Striegnitz. 2023. Project 3 – named entity recognition.

	Precision	Recall	FScore
accuracy: 94.95%	58.02%	64.63%	61.14
LOC:	55.56%	77.13%	64.60
MISC:	29.88%	28.54%	29.20
ORG:	56.04%	60.88%	58.36
PER:	73.70%	72.91%	73.30

Table 1: The above table shows the evaluation matrix for the development data for NER implemented as a classifier

	Precision	Recall	FScore
accuracy: 96.08%	60.70%	68.49%	64.36
LOC:	65.88%	69.65%	67.71
MISC:	28.82%	29.50%	29.15
ORG:	58.37%	68.71%	63.12
PER:	70.94%	84.35%	77.07

Table 2: The above table shows the evaluation matrix for the testing data for NER implemented as a classifier

	Precision	Recall	FScore
accuracy: 97.12%	67.11%	70.14%	68.59
LOC:	61.35%	80.18%	69.52
MISC:	44.04%	40.67%	42.29
ORG:	67.47%	68.82%	68.14
PER:	81.65%	74.63%	77.98

Table 3: The above table shows the evaluation matrix for the development data for NER implemented with the Viterbi algorithm

	Precision	Recall	FScore
accuracy: 98.03%	71.82%	75.44%	73.58
LOC:	74.12%	75.55%	74.83
MISC:	46.10%	41.89%	43.89
ORG:	72.44%	78.29%	75.25
PER:	77.31%	85.31%	81.11

Table 4: The above table shows the evaluation matrix for the testing data for NER implemented with the Viterbi algorithm