Named Entity Recognition using the Viterbi Algorithm with a Hidden Markov Model and a Maximum Entropy Markov Model

Hope Crisafi

Abstract

Named Entity Recognition is a task where words are classified and tagged, with the goal if identifying what kind of entity, if any, the word represents. The purpose of this project was to create two algorithms that could perform named entity recognition. The first one performed NER as a classification task using the Viterbi algorithm in a Hidden Markov Model, and classified based on word features, and well as the features of its neighboring words. For the second one, I built additional techniques into the Viterbi algorithm on top of the first one. Not only does this one take into account the features of its neighboring words, but it takes into account the predicted tag of the previous word. This determines the most likely sequence of tags given a sentence. This NER model is called a Maximum Entropy Markov Model, that uses logistic regression to compute the most likely label for a word, given the word's features and the label of the previous word.

1 Introduction

The purpose of this project was to explore different techniques that could achieve the best results for named entity recognition. The goal was to try two classifiers on the CoNLL Spanish corpus, one that used a Hidden Markov Model and relatively simple, and another that was more complex, and utilized the Viterbi Algorithm and a Maximum Entropy Markov Model. The purpose of the simpler classifier was to create a foundation for the second classifier, and see how just using simple features could generate a decent F-Score without too much additional work. Some of the word features chosen included if the word was capitalized or contained a hyphen, as these were simple and were able to help the classifier distinguish a lot. Once this was completed, the second one added on to this by taking into account the predicted label for the previous word, along with the features of neighboring words, when predicting a label. I predicted that this technique would be able to classify words much more effectively, and that was indeed the case. This technique proved to be much more effective, as it generated an F-score about 20 points higher than the simple HMM classifier. The Viterbi algorithm was able to account for more than just features, and the MEMM was able to help the prediction accuracy with probability vectors.

2 Features

2.1 Initial Features

The HMM classifier required me to pick some features to aid in classification. I swapped features in and out, and tried to get the best combination that would give me the highest F-Score, which was ideally 60 or above. I found that the best feature combination for this part of the project was using islower, istitle, and isidentifier. islower returned true if all characters in a word are lowercase. istitle returned true if the word follows conventions of a title (if all words start with an upper case letter), and isidentifier, that returns true if it contains alphanumeric letters, numbers, or underscores. There were some other features I tried, including isdigit, isprintable, and isupper, but in the context of this project, they were redundant or not applicable.

2.2 Updated Features

The features mentioned in the previous section worked well for the HMM classification, but they were not sufficient for the MEMM model. I needed to add more features to make the predictions and scores higher. On top of my initial features, I added isupper, isalnum, contains hyphen, and contains apostrophe. the isupper method returned true if all characters are uppercase. The isalnum method returned true if all characters are alphanumeric. The methods contains hyphen and contains apostrophe returned true if the word contained a hyphen or apostrophe. These features worked the best in my

Class	Precision	Recall	F-Score
LOC	55.74%	77.03%	64.68
MISC	29.13%	28.54%	28.83
ORG	55.73%	60.65%	58.08
PER	73.30%	72.34%	72.82
Overall	57.71%	64.35%	60.85

Table 1: Scores for ner.py run on the dev set

Class	Precision	Recall	F-Score
LOC	66.29%	69.83%	68.01
MISC	28.98%	30.09%	29.52
ORG	58.19%	68.79%	63.04
PER	70.68%	83.95%	76.74
Overall	60.64%	68.55%	64.35

Table 2: Scores for ner.py run on the test set

MEMM, which delivered the highest F-Score of 81.52 on the test set. In this case, some of the methods were a little redindant and overlapped, but it was worth it, as they helped boost the total F-Score.

3 Classifier Models

3.1 Hidden Markov Model

The first classifier utilized a Hidden Markov Model (HMM). First, I chose features that I thought would help the classifier distinguish word labels the best, as described in the Features section. Then, the function word2features was called, which generates a list of features for a word in a sentence. Then, these features were turned into vectors of numbers, which were then used to train a logistic regression model. Then, predictions were made based on the logistic regression data, and the predictions were written to a file. The resulting F-Score on the dev set was 61.23. However, I changed the features to be the exact ones I used in my Viterbi model, so I changed the features in this classifier to match this. The second time I ran this, with the updated features, my F-Score dropped slightly to 60.85. On the test set, I initially got 64.47. When I run the test set with my updated Viterbi features, I got an F-Score of 64.35. My initial features included islower and isidentifier. Despite the initial features having a slightly higher F-Score than the features I used in the MEMM Viterbi classifier, I decided to keep the updated features for the sake of consistency.

Class	Precision	Recall	F-Score
LOC	79.63%	79.63%	79.63
MISC	80.00%	60.00%	68.57
ORG	85.71%	88.52%	87.10
PER	77.14%	77.14%	77.14
Overall	88.44%	80.00%	80.71

Table 3: Scores for ner-viterbi.py run on the dev set

Class	Precision	Recall	F-Score
LOC	80.26%	83.56%	81.88
MISC	81.25%	56.52%	66.67
ORG	86.36%	81.90%	84.07
PER	80.36%	81.82%	81.08
Overall	82.95%	80.15%	81.52

Table 4: Scores for ner-viterbi.py run on the test set

3.2 Maximum Entropy Markov Model and Viterbi

The Maximum Entropy Markov Movel (MEMM) build off the HMM described in the previous section. In addition to comparing features of the current and neighboring words, the MEMM takes into account the tag assigned to the previous word. The Viterbi algorithm implemented needed to change slightly to incorporate this. First, it sets up the initial probabilities for each state using the trained MEMM model and the first observation passed in. Then it computes the highest probability for each state for each subsequent observation, by considering the probability of transitioning to a state given the current observations, based on the previous states. It keeps track of the highest probability throughout these iterations (argmax), and then returns it once the function is complete. I used my updated features, as described in section 2.2 in this model. All scores using this model compared to the HMM were significantly higher, as shown in tables 3 and 4.

4 Results

In short, the more features and the more sophisticated the algorithm, the better the classifier performs. There are limitations to performance and accuracy when simple features are used to predict the tag of a word. The MEMM model far outperformed the HMM model by margins of up to 20 percent. In addition to adding more features, taking the previous word's predicted label into account when predicting the current word's label also

showed significant improvement in the classifier's performance. Looking at the data, it seems that the ner classifier was pretty consistent across all categories, as the numbers for each classification tag were pretty close in proximity. However, in the ner-viterbi files, all numbers are close in proximity, except for the MISC classifier tag, which is significnaltly lower than the other tag scores. A follow-up research topic counld be investigating why this problem is unique to that tag and that algorithm, as the other tags and algorithm are relatively consistent.