CS388 Project 1: Sequential CRF for NER

Anonymous ACL submission

Abstract

Conditional Random Fields are popular models for Named Entity Recognition task. Named Entity Recognition is where, we would parse through the sentence and find Organisation, Name, Location and Miscellaneous. The first part of this project generates a HMM model using the train data and I implemented Viterbi decoding to identify the named entity from sentences. The second part of the project involves building a sequential CRF model for training and Viterbi decoding for inference. CRF model were chosen for this task because, it is a discriminative model that handles the presence of more number of O's better than HMMs. Finally, for the extension part, I chose 4 other features and gave detailed analysis of the model performance using these features. I collaborated with Neeharika Immaneni for this Project.

1 Introduction

In this project we have used the CoNLL 2003 dataset(Sang and Meulder., 2003) which contains data in column format with the first column having the word, second column with the POS tag and the third column with NER tag. We use all the three columns for our project. For the first tag, the HMM model is generated using the train data such that the state of the HMM is the tag and the emission corresponds to the word for that particular tag. Therefore, given the emission, transition and initial probabilities, I implemented Viterbi decoding to generate the NER tag for each word in the sentence. However, during inference, we would be ignoring the O tag for F1 score computation. The reason being, due to the sheer number of O tag in the data, by default that system would have a very high F1 score.

I implemented the sequential CRF model(C. Sutton and A. McCallum. , 2006) and trained it. To train the network, emission features and transition features need to be extracted from the data. However, since transition features are slow to learn, we extract only the emission features from the data and use the transition features from HMM during decoding. These features are used to compute the Phi based on features and weights. Following this, the marginal probabilities are computed using the forward-backward algorithm. Finally, the gradients are accumulated over all emission features and the weight of the model is updated. During inference, we would again use the Viterbi decoding and find the argmax that maximises the conditional probability to find the tag corresponding to the input word.

In the final part of the project, I have implemented the extension to use various features for CRF training. In the shape of the word, I added checkpunctuation which is an indicator feature for the presence or absence of punctuation in the word. The next feature I used is to determine if the current word is a stop word or not. Stop words refers to the most common words in English that with a very high probability would not be a Noun. In our task, the stop word would not be any one of the named entity to be predicted. In this project, I have used the list of stop words from nltk. The next feature I used is an indicator based on the word belonging to a cluster of POS tags. Even though the original feature extraction uses the POS tags directly, I have established a cluster based on the word belonging to a verb or noun or adverb or adjective followed by distinct cluster for each noun form. This would help the model learn meaningful pos tags compared to the case of having distinct pos tags. The next feature I tried is to increase the n-gram to a size of 5. I chose this to be an optimal n-gram size which would increase the feature size but, it would help the model learn more dependencies around the current word. Fi-

nally, I used the TF-IDF which is a statistical measure to evaluate the importance of a word in the document. In this project, while training, I used an indicator function to find the most important word in the sentence and assign a feature only for that word in the given sentence as TFID-max and for all the other words, it is assigned a feature with the name TFID-not-max. During inference, I just used these pre-computed TFID scores from train data, normalized it and assigned with the TFID-max feature only if the word has a score more than 0.5 and used TFID-not-max feature for all other words. Therefore, this feature helps in identifying the most important word in a sentence based on it's occurrence in the document.

2 Part 1: Viterbi Decoding

In this part of the project, I implemented Viterbi decoding for a HMM model trained to produce the transition probabilities, emission probabilities and initial probabilities. In this implementation, all the probabilities are converted to log probabilities. The data is given to an extract_emission_features function that takes a particular word and adds an index to every time a feature is created. The instructor's code generates features for lexical and pos tags on this word, previous and the current word. Followed by n-gram of 3 for letter in the word, and the word shape with is Upper, is Lower, Capital, is digit. These indicator function are added as a feature with distinct indices across all tags for each word.

In the HMM model training, emission probabilities, transition probabilities and initial probabilities are learned form the train data. These are estimated based on the given transition or emission count divided by the total number over the second axis that corresponds to the next state and word in the corpus. The ProbabilisticSequenceScorer abstraction handles the emission score case where if the word is not present in the corpus, it assigns it to the unknown word case. While implementing, I noticed that if the unknown words were not handled, the F1 score was around 56. Therefore, handling unknown words is essential for the model to decode appropriately. The viterbi decoding algorithm creates viterbi scores and backpointers. The viterbi score is the max of transition probabilities between the current tag and the previous tag with the previous sentences's viterbi score at that tag. The corresponding backpointer is the argmax of the above across possible tags.

After parsing through the entire sentence, the argmax at the final word of the sentence identifies the tag index. This tag index is used through the backpointer in reverse order of the sentence and the corresponding tag for each word is obtained. These tags are then reversed and then returned to compute the final F1 score. The Viterbi implementation gives a F1 score of **76.89** with precision of 85.44 and recall of 69.90.

3 Part 2: CRF Training

In this part, I implemented the training and inference using CRF. The feature extraction step is the same as the HMM task. In training CRF's, the feature weights are of the same size as the length of the feature index and are initialized to zeros. I used Unregularized Adagrad Trainer optimizer given in the starter code. The models are trained for 10 epochs. Initially, using the forward-backward algorithm the alpha and beta scores are computed from the emission scores. Both alpha and beta parameters are initialized to zero. These values are used to compute the marginal probability. The gradients are accumulated over the emission features and it is used to update the feature weight. For faster and inference, I stored the weights after training. Therefore, to run the inference on this model again, I used the trained weight directly to perform decoding and identifying the NER and the corresponding F1 score.

For decoding, I have used the transition log probabilities from the HMM models. However, I have checked for cases where an 'I' appears after 'O' to -np.inf and similarly, if the tags appearing after the I does not match for the previous tag, it is set to -np.inf. After extracting the feature cache, the FeatureBasedSequenceScorer helps in obtaining the emission potentials using the score_indexed_feature function which adds the weight of that feature and returns the score. I used viterbi decoding where, the scores for V and BP matrixes are updated. The BP matrix is parsed through using the argmax obtained for the last word and the tags for the rest of the sentence is generated in the reverse order. The generated tags are then reversed and then given to the LabeledSentence function and hence the F1 score is calculated. The F1 score obtained on the dev dataset was **86.77**, precision was 90.14 and recall is 83.64.

4 Part 3: Extension - Features

In this part, I added a feature to check for punctuation in a given word, check if the current word is a stop word or not, the POS cluster to which the current word belongs to and finally, increasing the n-gram to 5 and a feature based on the TF-IDF score generated from the train data. Every time I add a new feature, it takes almost 2 hours for training and decoding. Also, the time required increases as the number of features added increases.

4.1 Punctuation

This feature is similar to the other word shape features. If a letter of a word has a punctuation, it is added as a word shape feature similar to the original implementation.

4.2 N-gram

I increased the n-gram from 3 to 5. This would help the model to learn the context information better from the current word. However, this would increase the feature size from 3 to 5 for n-grams.

4.3 Stop words

In the nltk package, the most common stop words in English are listed. I used the stop words list from nltk to check if a particular word is stop word or not. This could be a useful feature because, usually, noun would not appear as a common word or any named entity. So, if the current word is 'stop word', it is added as a stop word feature and if it is not, it would be added as a 'not stop word' feature.

4.4 POS clustering

In the list of POS tags from the train data, I clustered them to 7 POS clusters. Given that the named entity would be a noun, I kept the noun entities separate, which formed 4 clusters by itself. The verb variations were added as another cluster, Adverb variations and finally adjectives. If any word's POS does not belong to any one of these clusters, they are as a 'NotInCluster' feature. Given that there are various POS tags, the model may learn better with fewer POS tags that are clustered based on verb, adverb and so on, giving individual cluster to each noun variation since they are important features for a named entity recognizer.

4.5 TF-IDF

TF-IDF referes to term frequency-inverse document frequency. This is a statistical measure to identify the importance of a word in a document. During training, the TF-IDF scores for all sentences in a document were generated. This created a matrix of size: (number of sentences, unique words). The unique words refers to the all the words present in the train set. Therefore, for a given sentence and current word, I checked if that had the maximum TF-IDF score. If the score was maximum for that word, I added it as a TFID_max feature and for all the other words in that sentence, I added it as TFID_not_max feature. During test time, since I do not have access to all the sentences in the test set/ dev set, I used this pre-computed TF-IDF score from the train data. I computed the column sum, meaning, for each word, I added the TF-IDF score across all sentences. I normalized this TF-IDF score. Therefore, now for each word in the train set document, there is a normalized TF-IDF score. For each word in the test set, I checked if it already exist in pre-computed TF-IDF normalized score. If yes, and the score is greater than 0.5, I used the TF-IDF_max score feature and if not, I used the TF-IDF_not_max feature. In the case of the word not belonging to the list of words in TF-IDF score, I used the TF-IDF_not_max features.

5 Conclusion

The sequential CRF implementation predicts the named entity for a given sentence. In the extension part, I used 4 different features over the existing implementation and as shown in table 1, when all the features were used, the F1 score increased from 86.77 to 87.78. However, when the other features were used without the TF-IDF feature, the F1 slightly dropped. This could be because the other features, helped the model better when it was given along with the TF-IDF and n-gram, meaning the TF-IDF and n-gram feature helped the model learn better and gave a better F1 score.

References

C. Sutton and A. McCallum. . 2006. *An introduction to conditional random fields for relational learning.* In L. Getoor and B. Taskar, editors, Introduction to Statistical Relational Learning. MIT Press., Washington, DC.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. *Introduction to the CoNLL-2003 Shared Task: Language- Independent Named Entity Recognition*, volume 1. In Proceedings of the Conference on Natural Language Learning (CoNLL).

Feature name	F1	Precision	Recall
Original features	86.77	90.14	83.64
n-gram: 5	87.7	90.48	85.90
Punctuation	86.77	90.14	83.64
Punctuation + Stop words	86.67	90.09	83.51
Punctuation + stop words + POS clustering	86.65	90.08	83.46
Punctuation + stop words + POS clustering + TF-IDF	87.04	89.63	84.60
Punctuation + stop words + n-gram POS clustering + TF-IDF	**87.78**	89.94	85.73

Table 1: This table lists the features used for CRF with the F1, Precision and Recall scores.