Chinese word segmentation

Chris Venour

Abstract

In this project I implemented the algorithm described in the paper *A Realistic and Robust Model for Chinese Word Segmentation* by H. Chu-Ren, Y. Ting-Shuo, P. Šimon, and H. Shu-Kai. I implemented an xgboost decision tree classifier and stopped training it after 40 epochs, although training could have gone on for longer because the loss function was still decreasing at that point. Results were 94.69% precision and 79.98% recall, yielding an F1 score of 86.71.

1 The code I wrote

I created the following jupyter notebooks:

1. **train-classifier.ipynb**: this notebook shows how the classifier was trained. After training, the classifier and the Chinese alphabet of characters that the classifier uses were saved as pickle files so that they could be used in the two notebooks below.

2. **evaluate-classifier.ipynb**: this notebook loads the trained classifier and the Chinese alphabet (which is indexed in an order the classifier depends on). Once these pickle files have been loaded, the notebook evaluates the trained classifier on the test set.

3. **use-classifier-to-format-some-examples.ipynb**: this notebook loads the trained classifier and its alphabet and compares how the classifier formats some sentences with how humans formatted those same sentences.

2 Training, development and test data

I did not use the whole training set of 745,817 lines of Chinese sentences because I only had so much time to train the classifier. Instead I used the first 90,000 lines of the 'train.txt' file for training and the next 10,000 lines in that file to create a development set. All 1398 lines in the file 'test.txt' were used to build a test set.

My code computes a feature vector for each 4-gram sequence in the data files and this amounted to the following training, development and test set sizes:

1. 996,924 feature vectors in the training set. In other words from the 90,000 lines in 'train.txt' that I earmarked for building the training set, 996,924 4-grams were found and feature vectors for each of those 4-grams were created. The training set consists, therefore, of 996,924 training examples and their labels.

2. the development set consists of 121,960 feature vectors and their labels.

3. the test set contains 17,345 feature vectors and their labels.

3 The classifier

I implemented an xgboost decision tree classifier and wrote all the code in Python 3.6. I experimented with some of the decision tree's parameters, such as the maximum depth a tree can have and the number of epochs/rounds (i.e. the number of training steps on all the data) the classifier is trained for.

A Support Vector Machine or a Neural Network that performs logistic regression would probably also work well for this problem.

Another method of solving the problem, which doesn't implement the ideas found in the paper *A Realistic and Robust Model for Chinese Word Segmentation* by H. Chu-Ren, Y. Ting-Shuo, P. Šimon, and H. Shu-Kai, would be to build a Chinese Language Model (working at the level of characters rather than words) and have it predict, whether a space occurs after a given sequence of characters.

4 How the code for training the classifier works

1. The first 100K lines in the file 'train.txt' were loaded into a list.

2. 90K of these lines were used to create the training set of feature vectors and 10K of the lines were used to create a development set of feature vectors.

3. All the unique Chinese characters within the training set were found and an alphabet was created. Each unique character has a unique index in the alphabet.

4. For each 4-gram found within the training set of Chinese sentences, the following steps were performed:

   (a) for each ABCD 4-gram in a dataset, construct a 5d feature vector <AB, B, BC, C, CD>

   (b) assign a label to that feature vector. A label is either 0 or 1 depending on whether a space appears between the 2nd and 3rd character of the 4-gram

(c) create a **sparse** one-hot vector version of each 5 dimensional integer feature vector i.e. <AB, B, BC, C, CD> and add it to a matrix. The number of unique Chinese characters found in my training and dev set was 4660. That means the length of the bigram part of a one-hot feature vector is 4660*4660 = 21,715,600 entries. The length of the unigram part of the feature vector is 4660 so the one-hot vector used to represent the 5d feature vector of a single 4-gram consists, in total, of 21,715,600 + 4660 = 21,720,260 numbers (0's or 1's). Most of the numbers in the feature vector are 0's and instead of storing all these numbers, sparse one-hot vectors were created and assembled into a sparse matrix using scipy's csr_matrix() function.

5. The steps in line 4 above were also performed to create sparse matrices for the development and test set data.

6. The sparse matrices were converted to a 'dmatrix' format which the xgboost module is able to process.

7. The xgboost model is trained on the sparse training and development set matrices and their labels.

8. The trained model was evaluated on the sparse test set matrix.

5 Some interesting details about my solution

5.1 Using special tokens

I added three special tokens to the alphabet of unique Chinese characters that were found in the training set:

1. **begin-of-sentence token ^** This token/character was added to the front of each line so that a proper feature vector could be built for the first character of a line. For instance the first character in line 8 of the file training.txt is a word with a space after it but there's no character to the left of the character. Without a begin-of-sentence token, a 5d feature vector cannot be built to represent that a space should come after that first char.

2. **end-of-sentence token $** This token has the same functionality as the begin-of-sentence token. The end-of-sentence token was added to the end of each line so that a proper feature vector could be built for the last character of a line.

3. **unk token *** This token was added to the alphabet so that the classifier can handle characters in data (such as the test set) which the classifier hasn't seen before.

5.2 Mixed feature vector

As suggested in the problem description, I construct a 5d feature vector (<AB, B, BC, C, CD> for example) for a ABCD 4-gram. The large 21,720,260 dimensional one-hot feature vector that ultimately gets created, therefore consists of two parts:

1. a one-hot encoding of the bigrams in the 5d vector (AB, BC, and CD in the example).

2. a one-hot encoding of the unigrams in the 5d vector (B and C in our example).

One would think that putting this bigram and unigram information alongside each other in an apparently undifferentiated stream of 0s and 1s would destroy information. But the classifier learns to use this concatenation of bigram and unigram information in a meaningful way and that is kind of extraordinary.

5.3 Counting in base 4660

In previous projects, I've computed values in binary (base 2), decimal (base 10) and hexadecimal (base 16) systems. But in this project I computed values in a base 4660 system (the number of unique Chinese characters found in the training set) in order to find the index of a unique bigram. This was a first for me! See the function findIndexOfBigram() for details.

6 The classifier's performance on the test set

- accuracy: 85.04%
- precision: 94.69%
- recall: 79.98%
- F1 score: 86.71

As the **train-classifier.ipynb** notebook shows, the training error (and development set error) continued to decrease even after 40 epochs but I had to stop training at that point because of time constraints. Scores would be even higher if the classifier was allowed to train for longer.

Using all of the training data, rather than just 1/8 of it as I did, is another obvious way of improving the accuracy of the classifier.