Natural Language Processing Report

Xin-Yuan Huang, Tian-Yi Lin, and Mu-Chu Lee
b03902124, b02902062, b01902082

1 Introduction

This project is divided into 2 phases. In phase1, we need to detect whether a sentence has redundancy or not. In phase2, every sentence has redundancy. But there is only one consecutive segments that is redundant. Our goal is to identify the redundant segment. These 2 phases are quite different, so we used entirely different method in solving them. For unification, we choose Stanford Segmenter and Stanford POS Tagger to do the segmentation and POS tagging.

2 Phase I

Roughly speaking, our methods can be categorized into 2 ideas. One only uses information from training data. We extract features from training data and construct a model for the testing data to use. The other uses external information, which we mean we do not construct our model based on the training set, but uses the training set for validation and see if the model we constructed works. After then, we combine these 2 ideas, in hope of improving the performance.

2.1 Training Model by Training Dataset

For this idea, we have tried 2 different methods, with some of their variants. The first one is a naive method that uses trigram count to do classification. After segmentation, we add 2 start and 2 end symbol at front and back of each sentence. So “把挫折就变成动力” will become “<s> <s> 把挫折就变成动力 <\s> <\s>” . Now given a testing sentence, for every trigram in the sentence, we calculate the occurrence in redundant sentence (`RED`) and non-redundant sentence (`NRED`). If the sum of count in `RED` is more than the sum of count in `NRED`, we say this sentence is `RED`, and vice versa. However, we have slightly changed the condition to:

For some integer K, when “count in `RED`” $>$“count in `NRED`” $-K$

This modification gives us a better recall, but it will also lower the precision. Thus we need to take some care when applying this method.

Now we evaluate how this method works in testing data. We first random shuffle the training data, then do 5-fold cross validation on it. Surprisingly, we got an accuracy 11%. This is a extremely low accuracy, but at the same time, it will be extremely high if we reverse the answer. After some investigation, we notice that it is due to the fact that most of our training data can be paired, e.g. “一到周末他就要带我去钓鱼” is `NRED`, but “一到每个周末他就要带我去钓鱼” is `RED`. These 2 sentences are very similar, but one is `RED`, one is `NRED`. Using cross validation with only random shuffle, it is very likely that one of them is in the train set, while the other is in the testing set. With the method we have mentioned, most trigram in the test sentence will appear in RED(if test is `NRED`) at least one time. So the prediction made by our model will be `RED`, but the true answer should be `NRED`.

To fix this problem, we sort the training data lexicographically, and cut them into 5 chunks consecutively. By this way, most of the pairs are will appear in the same chunk. Then, we do 5-fold cross validation by the chunks we constructed. We calculate the mean μ and the standard deviation σ of the accuracy and F1 score. Performance for the above method (w/ $K = 0$) is $0.5032(\mu_{AC}) \pm 0.0136(\sigma_{AC})$, $0.3509(\mu_{F1}) \pm 0.0449(\sigma_{F1})$. (Note that we will use this notation for the following evaluations.) We found out that this is not far from random guessing. It may be caused by the fact that most of the trigram in the newly seen sentence doesn’t appear in training data. In fact, about $\frac{4}{5}$ of the testing sentences, their trigram are not in the training set.

Thus we decided to try some variants of this method:

1. Word Bigram (w/ $K = 0$)
   Performance : 0.5486 ± 0.0217, 0.5735 ± 0.0277.
   Since the problem of entirely unseen trigram is solved, the performance is improved. However, some information are lost after we used this method. In an extreme case, if we degrade this case into unigram, the accuracy will drop back as if random guess.

2. Chinese Character(Char) Trigram (w/ $K = 0$)
   Performance : 0.5442 ± 0.0197, 0.5179 ± 0.0361.
   The motivation for this is to capture things like “却甚至”, “了 <\s><\s>”. But they can also be captured by word bigram: “却甚至”, “了 <\s>”. Thus it is not as good as the first method, word bigram.

3. POS Trigram (w/ $K = 0$)
   Performance : 0.5680 ± 0.01077, 0.6285 ± 0.0093.
   Since it does not have many unseen trigrams like Word Trigram, it can capture longer relations than Word Bigram. It is the best of all previous methods. It has high recall, and the precision is also high. So we have more chance in modifying K. (Since we will lower the precision with bigger K, with better recall) By choosing $K = 12$, we have performance of 0.5534 ± 0.0109, 0.6678 ± 0.0054. Which is better than giving all output 1(where F1 score= 0.6667).

The previous method is unnatural as we need to lower the precision manually for getting higher F1-score. We come up with another approach that uses the idea of rule-based methods. We try to extract rules that indicate redundancy. If the testing sentence conforms to some rule, we consider it to be `RED`. But figuring out the template of the rules isn’t something trivial. After some brainstorming, we come up of an idea which is based on the previous method but more compact. We extract POS Trigrams and Word Bigrams that is highly relates to `RED`. We have the rule set as “Whether this gram appeared in the testing sentence or not.” The rule is simple, but efficient.

To extract gram that relates to `RED`, we use the statistical method taught in class. Consider two categorical random variables X and Y. X correspond to whether this sentence is `RED` or not. Y correspond to whether a particular gram appeared or not. Then we use χ^2 Test to calculate the χ^2 of each POS Trigrams and Word Bigrams. The rules used the k highest χ^2 grams. The performance for this method is 0.519 ± 0.0127, 0.6676 ± 0.0113, which is quite bad. We discovered that due to some unknown reason, some gram actually collocate with `NRED`(rather than `RED`). That is, most occurrence of such gram appear in sentence that is not `RED`. Such gram also has high χ^2, since X and Y are still highly dependent. We neglect such grams, and the performance becomes 0.5544 ± 0.0144, 0.6723 ± 0.0136. As it has intrinsically high recall, and obtaining good precision, it got the best performance within this section.

2.2 Solving by External Information

We now seek for methods that uses the main idea: is this sentence a "good sentence" to classify whether a sentence is `RED` or not. We will give an easy way to determine whether a sentence is a good sentence or not in this section.

Tools we used in this project are Stanford Segmenter and Stanford POS Tagger.

The Stanford Segmenter for Chinese is based on using a well-known technique in the field of NLP, Conditional Random Field(CRF). The probability assigned to a label sequence for a particular sequence of characters by CRF is by:

$$P_\lambda(Y|X) = \frac{1}{Z(X)} exp(\sum_{c \in C} \sum_k \lambda_k f_k(Y_c, X, c))$$

Where Y is the label sequence for the sentence, X is the sequence of unsegmented characters, $Z(X)$ is a normalization term, $f(k)$ is a feature function, and c indexes into characters in the sequence being labeled.

Following we describe the procedures by using the segmenter and POS tagger to classify whether a sentence is `RED` or not.

1. We first try to use Stanford segmenter to see if we can identify whether a sentence is `RED`.

   To identify whether a sentence is `RED` or not. We check if there are any redundant characters in a sentence. We do not consider redundant words here, since we have not yet segmented the sentence into words yet, we do not have a standard that tells us which segment is a word.

   Our procedure is simple. We check if one sentence's probability to "appear" is greater than others. We gain a probability after segmenting a sentence into some words. Let the original sentence be S, and the sentence of removing the ith character is S'_i. And let $P_s(S)$ be the probability of segmenting S into some combination of words.

   We then compare if $P_s(S) \geq P_s(S'_i), \forall i$.

   If yes, we take S as `NRED`, else `RED`. However, this method doesn't work well apparently. The probability on this is based on relative relationship.

2. We then try to use Stanford POS Tagger's calculated probability. The main idea by using Stanford POS Tagger is that we believe sentences that are `RED` should be more "unnatural". For example, we think probability that a sequence appears to be "N+N+N" should be much more lower than "N+V+N". Therefore, we segment S and $S'_i, \forall i$, and gain the tag from the POS tagger. We then compare if the tag's probability of S is greater than all S'_i. If not, we classify S as `RED`, else `NRED`. However, we still don't get good performance simply by this method.

2.3 Training model by Blending

Though the performance of the POS tagger in 2.2 is not good, we still try to combine the previous 2 ideas. Maybe the overall performance will be better as more information are added. Since we need to use the given training data explicitly, the evaluation for this method is the same as 2.1.

The best method for 2.1 is the rule-based method. So we think of a way to generalise that method and to make usage of the score calculated in 2.2. This blended method work as follows:

1. We consider more type of rules(grams). In addition to POS trigram and Word bigram, we include special trigrams of the form: POS/Word/POS and Word/POS/Word, e.g. “是 AD 是” or “VV 了 NN”.

2. We extract grams with high χ^2, similar to what we have done before. Notice that we do not neglect grams that collocate with `NRED`, we will come back to it later.

3. For each sentence, we count the number of occurrence for all the extracted grams to create a high dimensional vector (bag-of-word method). Then we add four dimension at the end of each vectors, which correspond to the score (per word probability obtained in 2.2) of the original sentence, and the three highest score (from high to low) after deleting one Chinese character from the original sentence. Thus our data can be written as $(y_i, \boldsymbol{x}_i), i = 1, \cdots, l$, where y_i is 1 if the sentence is `RED` and -1 if `NRED`.

4. Since it can be viewed as a binary classification problem, we use logistic regression (solved in primal) to train a binary classifier. Which is to solve for the following optimization problem.

$$\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \log(1 + e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}).$$

   In prediction, if $\boldsymbol{w}^T\boldsymbol{x}_i \geq 0$, we consider the sentence be `RED`, vice versa. Here, we used LIBLINEAR to solve this classification problem.

The performance for the blended method is 0.5948 ± 0.0167, 0.5700 ± 0.0210. It has the highest accuracy among all of our methods. But the F1-score is not high, since the optimization problem maximise accuracy rather than F1-score. In order to enhance the F1-score, we modify the minimization problem stated above using `-wi` option in LIBLINEAR, which can give different C for different class. We give higher C for `RED` class to make recall higher. By using `-w1 3`, the performance is 0.5498 ± 0.0160, 0.6461 ± 0.0136.

Now we come back to the issue mentioned in (2). We know that $\boldsymbol{w}$ will be automatically tuned to have the best performance. By picking $\boldsymbol{w}_i = 0, \forall i$ corresponding to the grams collocating with `NRED`, it would be the same as neglecting them. And by choosing $\boldsymbol{w}_i < 0$, we can also make use of the fact that they collocate with `NRED`. Thus, it is actually better not to remove them when extracting the rules(grams) in (2).

The blended method can be shown to be a superset of the rule-based method (RBM). Suppose $\boldsymbol{w}_i$ is very large $\forall i$ corresponding to the grams extracted in RBM, and all other $\boldsymbol{w}_i = 0$. Then if one of such rules appeared in the testing sentence, the sentence would be labeled as `RED`. Which acts the same as RBM. As $\boldsymbol{w}$ is tuned to have optimal performance, the blended method should work better than RBM. Due to this theoretical reason, we submit the result predicted by the blended method. But actually RBM yields better performance due to it’s higher recall, while the blended method only has slightly higher precision than RBM. We noticed it after phase 1 ended. So we believe, if we use classifiers that maximize F1-score, the blended method would have better performance than RBM.

3 Phase II

We categorized our ideas, as we did in 2. For evaluation, since the training data is not paired, we random shuffle the data, then do 5-fold cross validation.

3.1 Training Model by Training Dataset

If we consider each Chinese character as an item of a sequence, we can take this problem as an sequence labelling problem, `RED` or `NRED` for each item, with some constraint: `RED` should be a continuous segment in the sequence. But to extract effective feature such as POS, we must do segmentation. Thus we define an item to be a word, and we define a word to be `RED` if it’s intersection with `RED` segment is nonempty. For example: In “我主修英文系”, the redundant segment is “系”, so the word: “英文系” would be labelled as `RED`.

There are many known statistical method in solving sequence labelling problem. One of the most popular method frequently used in NLP is “Conditional Random Field” (CRF). In particular, We used linear-chain CRF. It can be seen as the sequence version of Logistic Regression or the conditional version of Hidden Markov Model(HMM). It is similar to Maximum Entropy Markov Model(MEMM). Both of them are discriminative model, in oppose to HMM. The major difference is in how the exponential term is normalized, MEMM is normalized locally (for each item, which yields some deficiency), while CRF is normalized globally. Thus we think CRF is more suitable to work with.

We use CRFsuite to train a CRF model. The feature extracted for each item(call it X) includes the word and the POS of X’s surrounding. We choose a window of 2 as the definition of surrounding for optimal performance, i.e. 5 words total with X at the centre. Notice that these features are binary, e.g. “Is previous of X’s word = 我?”. To make use of some simple morphological information, we add a special type of binary feature like this: “Is the next item of X similar to X?”. And we consider 2 item to be similar if they have at least 1 character in common (Recall an item is a segment of character). For example, “容易接上上网” can be easily identified that 上 is `RED`.

On the first attempt, we discovered that there is a huge problem. For most of the sentences, CRF predicts the optimal sequence to be all `NRED`. This result is reasonable, since 93.1% of the sentences in “p2.train” has only 1 `RED` item (`RED` character segment is inside an item). We also noticed that there is 77.7% in “p2.train” such that the `RED` segment coincides with exactly 1 item(word). Inspired by the observation above, we do the following to predict `RED` segment:

1. If there is no `RED` labelled in the sentence by CRF: We pick the item with the lowest marginal probability of being `NRED`. And predict this item (which is a segment of Chinese character) as the `RED` segment.

2. If there is exactly one item labelled as `RED` in the sentence: We simply predict that item as the `RED` segment of the testing sentence.

3. If there are multiple items labelled as `RED`: There is a constraint that `RED` segment must be continuous. So for simplicity, we predict the item with the highest marginal probability of being `RED` to be the `RED` segment.

The accuracy(AC) for this method is 0.3599, and the macro-average F1-score(MF1) is 0.4068.

When all the label is `NRED`, the marginal probability we used is actually the same as the probability obtained by ordinary Logistic Regression(LR) with no regularization term. Thus there would be some problem of overtraining that lower the performance. We investigate 2 options to achieve better result.

1. We simply degenerate into using normal LR. So we label the word with highest probability of being `RED`, to be `RED`. But we also use grid search to find the best regularization parameter C. This method yields a slightly better performance of AC= 0.3697 and MF1= 0.4102. Since degree2 polynomial kernel

(deg2 poly) is frequently used in NLP, for example, the paper by Nivre et al.(2006) shows that using deg2 poly to train a dependency parser yields good result. We have also tried using deg2 poly to train a probabilistic classifier. But since the training time is too long, doing one cross-validation takes about 3 4 hours, we just use the default C without grid search. The performance is slightly worse with AC= 0.3498 and MF1= 0.3988.

2. We continue using CRF. But we label those `NRED` item with it' s POS and everything else remains the same. Although POS has been used in the feature, under this modification, CRF would not reduce to normal LR when no `RED` appears. Since the probability will be conditioned on what the previous item' s POS is, so more information is encoded in the underlying structure. The performance for this method is AC= 0.4018 and MF1= 0.4461. It is the best method in 3.1, and we had used this method as our Phase 2 solution.

3.2 Solving by External Information

The other idea about Phase 2 is quite simple:

For a sequence of redundant words, if we remove the redundant words in it, the sentence will be more ”coherent and smooth” than the original sentence. But how can we measure how ”coherent and smooth” a sentence is? We claim that a sentence without redundant word has higher probability occurrence. So we tend to use Google n-gram to estimate the probability of a sentence. Google n-gram is a set of dozens of files recording the number of occurrence of each n-gram, and the size is extremely large (for example, the size of trigram is up to 532GB). In order to make the calculation faster, we did some preprocessing. First of all, we deleted all records about punctuations and alphabets. And then sum up the records of each years into a single record. Last, we designed a interactive interface for query, so that we can calculate probability of any sentence in just few milliseconds.

At first, we simply leave one Chinese word out each time, and choose the sentence with highest probability in n-gram as answer. The result of this method is extremly bad, we have AC= 0.1199, and the MF1= 0.1379. We discovered that this method tends to leave the part of unknown words out, such as leaving out ” 念” in ” 價值觀念” to make it become ” 價值觀”. To fix this problem, we decided to limit it to leave out ”true redundant word” only. To find the ”true redundant word”, we pick all redundant words in the training data, and choose the words that occurs at least twice as ”true redundant word”. When we limit the choice to the ”true redundant words”, AC= 0.1806, and MF1= 0.1985. However, due to the bad performance, we didn't adopt this method in our final submission. We investigated the results thoroughly and concluded that the reasons may be:

1. The difference of the source of google n-gram and training/testing data. They might not be chosen from the same distribution.

2. The problem about unknown words of n-gram which is mentioned above.

With the two reasons above, we think that external n-gram database isn't so suitable for this problem.