Report on Implementation of Highly Reliable SRA System

Xun Xu, Guanhua Chen, Pengfei Li

December 10, 2013

1. Word Sense Based Sentence Semantic Similarity Measurement

1.1 Basic Idea

To measure the similarity between 2 sentences plays a major role in SRA system. But as a unsolved problem in NLP, there are many possible but not prefect approaches to accomplish that. Our team decided to use Word Sense Based Similarity measurement for the reasons listed:

1. Student answers are short corpus focused on the same topic, which makes the traditional latent semantic analyze powerless. However, this feature guarantees the limited size of word senses used in the sentences because they are trying to answer the same question. Therefore, to compare the word senses of each sentences could be a good approach to determine if 2 short sentences are similar.

2. Word Sense Based Similarity is technically possible because Wordnet provides estimated distance between word senses.

Here are the steps for computing semantic similarity between two sentences:

1. First, each sentence is partitioned into a list of tokens.

2. Then find the most appropriate sense for every word in a sentence (Word Sense Disambiguation).

3. Finally, compute the similarity of the sentences based on the similarity of the pairs of words.

1.2 Naive WSD

The simplest way of WSD is to use Stanford parser to create POS tags assisting Wordnet to determine the most possible senses of a word in sentences. But our team disposed this approach because 1) Stanford parser could seriously compromise the performance of our system. Because our system is written in Python and has to use a bridge to communicate with Stanford parser which is written in Java. 2) Most of the POS tags Stanford parser uses is not supported in Wordnet.

Assumption: words belong to similar word senses tend to be tagged with the same POS tag. This assumption is straightforward. For example: if word senses of "play" and "drama" are similar, then "play" is probable a noun, and if word senses of "play" and "perform" are similar, then "play" is probable a verb. And based on this assumption, a naive WSD algorithm only works on assisting calculating semantic similarity has been created:

Algorithm: find most appropriate POS for wordX in sentence X

---

similarity=0.0
bestPOS=None
For wordY in Sentence Y:
–For POS in {Noun,Verb,Adv,Adj}//4 pos tags supported in
–//Wordnet and have greatest semantic influence
–senseX=wordnet.synsets(wordX, pos=POS)
–senseY=wordnet.synsets(wordY, pos=POS)
–if senseX & senseY
—senseDistance=similarity between senseX and senseY
—if senseDistance>similarity
——similarity = senseDistance
——bestPOS=POS
return bestPOS

Evaluation of this naive approach: This algorithm works fine when determining the sense of a word which has a word with similar sense in the other sentence, but does not make any sense on a word having no similar word in the other sentence. Because a word having no similar word in the other sentence will have no positive influence on the similarity of the 2 sentences, so this naive method is able to function in Word Sense Based Sentence Semantic Similarity Measurement.

1.3 Compute the similarity of the sentences based on the similarity of the pairs of words

Building a semantic similarity relative matrix R[m, n] of each pair of word senses, where R[i, j] is the semantic similarity between the most appropriate sense of

word at position i of X and the most appropriate sense of word at position j of Y. Thus, R[i,j] is also the weight of the edge connecting from i to j. Because this matrix stores the similarity of each pair of words from these 2 sentences, so it is actually like a bipartite graph. And therefore we formulate the problem of capturing semantic similarity between sentences as the problem of computing a maximum total matching weight of a bipartite graph, where X and Y are two sets of disjoint nodes. We use the Hungarian method to solve this problem. A simple fast heuristic method is presented as follows:

Algorithm: Compute sentence similarity score

```
Float sentenceSemanticSimilarity(X,Y,POS_X,weightX){
–//initial weight matrix
–R=Float[len(X)][len(Y)];
–for X[i] in X{
—for Y[j] in Y{
——senseX=wordnet.getSense(X[i],POS_X[i]);
——senseY=wordnet.getSense(Y[i],POS_X[j]);
——if (senseX & senseY){
———similarity=senseX.getSimilarity(senseY);
———if (similarity>R[i][j]){
————R[i][j]=similarity;
———}
——}
—}
–}
–return computeSimilaruty(R,weightX);
}
—
Float computeSimilarity(R,weightX){
–ScoreSum = 0;
–foreach (X[i] in X){
—bestCandidate = -1;
—bestScore = -maxInt;
—foreach (Y[j] in Y){
——if (Y[j] is still free && R[i, j] > bestScore){
———bestScore = R[i, j];
———bestCandidate = j;
——}
—}
—if (bestCandidate != -1){
——mark the bestCandidate as matched item.
——scoreSum = scoreSum + bestScore*weightX[i];
—}
–}
–return ScoreSum
}
```

Adjust the weight of each word Because not every word has equal influence on the semantic level of sentence, we have to estimate each word's weight. In this case, we use plsa to extract top 2 topics of all reference answers of that question where X and Y are trying to answer and use that 2 word vectors to compute word weights and apply them in the algorithm above as weightX.

2. Handle special cases

The approach we described by now is only able to handle some most general cases, but it works poorly on special cases like non-demain and contradictory. So we design corresponding algorithm for detecting non-domain and contradictory.

2.1 Contradictory Detection

Contradictory has one or a few of features listed below:

1. Contradictory resulted from negative words: "I *don't* like him"; "Dog is *not* as cute as cat".
2. Contradictory resulted from antonym: "The dog is *good*" vs "The dog is *bad*".
3. Contradictory resulted from different seq. of words: "A *dog* is larger than a *cat*" vs "A *cat* is larger than a *dog*"

For case 1, Stanford parser can find out all negative words and the word it denies. And case 2 can be handled by Wordnet. Finally, bigram is applied to detect case 3. Case 1 and case 2 are better to be handled at the same time because both stanford parser and wordnet are semantic based approaches. And because "bigram" style co-occurance vector based contraditory detection is a probabilistic approach, case 3 supposed to be handled separately.

— Algorithm: Contradictory Detection of case 1&2

```
bool isContradictory(X,Y){
–//count if there are odd number of negative words in
–//X
–xFlag=neg_count(X)%2==1;
–//remove all negative words in X
–X=removeNeg(X);
–yFlag=neg_count(Y)%2==1;
–Y=removeNeg(Y);
–//count the number of pairs of antonyms
–//extracted from each of X and Y
–anFlag=countAntonymPair(X,Y)%2==1;
–//replace antonyms of words in X in Y with their
–//corresponding words in X
–replaceAntonym(X,Y);
–if(anFlag)
—yFlag=not yFlag;
–similarFlag=False;
–POS_X=posTagging(X,Y);
–weightX=plsaModel.generateWeightVector(X);
–if(sentenceSemanticSimilarity(X,Y,POS_X,weightX)>threshold){
—similarFlag=True;
–}
–if ((xFlag!=yFlag) && similarFlag) return True
–return False;
}
```

— Algorithm: Contradictory Detection of case 3

```
Foreach question:
–create bigram of reference answers and correct answers and store the word
pairs in PositiveSet
–create bigram of contradictory answers and store the word pairs in NegativeSet
–ContradictoryFeatureSet=NegativeSet-PositiveSet
–model[question]=ContradictoryFeatureSet
—
bool isContradictory(question,answer){
–bigram=createBigram(answer).key_set();
–for word_pair in bigram{
—if(model[question].contains(word_pair)) return True;
–}
–return False;
}
```

Evaluation: Without special handle of contradictory, only 8% of contradictory can be recognized. But after special handle for contradictory is deployed, this rate has been raised to 42% on test set.

2.2 Non-domain check

In the 5-way classification, non-domain answer is very different from other four classes. Therefore we check it independently. First we check the definition of non_domain: if the student answer expresses a request for help, frustration or lack of domain knowledge - e.g., "I don't know", "as the book says", "you are stupid". From the definition and analysis of all non-domain rating answers, we find that the unigram corpus and bigram corpus of the all possible non-domain answers are limited. We populate the corpus with all non-domain answers and some ordinary conversation sentences(Asking for help or blame). Then we can check if the student answer to be tested is nearly the composition of the unigram corpus and bigram corpus. If so, we say it is non-domain.

Evaluation: The method reaches a relative high score on train set. The recall is 95% and the accuracy is 85%.

3. Implementation details

3.1 Preprocess the dataset

3.1.1 Text Normalization

3.1.1.1 Tokenize

Stopwords and pronunciation should be removed before the train model. Here we use NLTK stopwords corpus and tokenizer package to do this.

3.1.1.2 Stem/Lemmatize

Stem or Lemmatize should be applied before building word matrix as well as word dictionary.

Here we test the performance of Porter Stemmer and WordNet Lemmatizer. The WordNet Lemmatizer performs a litter better, however, it is much slower than stem. At last we decide to choose Porter stemmer.

3.1.1.3 Spell Correct

Spell check is performed before stopwords removal and stemmer. However, it is worth discussing on the tolerance rate of wrong word.

- Word Check: We need to check if a word is correct spelled. Wordnet or NLTK is not fit for this situation. Enchant package is widely used here.
- Edit Distance for possible correction: We assume the edit distance of correct word and raw word is at most 1. There are following possibility to correct a word.
    1. Insert a letter or single quote (e.g. "dont->don't")
    2. delete a letter
    3. replace a letter
    4. transfer position of adjacent letter
- Populate high frequency words list: The above step may produce many possible correction. We use a high frequency words list (1200 words including open/close class words) as priority.

```
change = []
splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
inserts    = [a + c + b for a, b in splits for c in self.alphabet]
deletes    = [a + b[1:] for a, b in splits if b]
replaces   = [a + c + b[1:] for a, b in splits for c in self.alphabet if b]
transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
changeWords = inserts + deletes + replaces + transposes
for word in changeWords:
        if self.spell_dict.check(word) and word in self.hfwords:
                return word
for word in changeWords:
        if self.spell_dict.check(word):
                return word
```

3.2 Training Procedure

1. Preprocess the data
2. Create word weight vector using plsa for each question on its reference answers
3. Create "bigram" style co-occurrence vector for each question
4. Train non-domain detection model
5. For each student answer:
    (a) non-domain detection
    (b) contradictory detection

(c) Apply Word Sense Based Sentence Semantic Similarity Measurement between this answer and each reference answer, use the greatest similarity score as this answer's score

6. Collect answer scores and their correct accuracy, determine the best boundaries of "irrelevant", "partial_correct_incomplete" and "correct"

3.3 Apply this model on test set

1. Preprocess the data
2. For each student answer:
   (a) check if the answer belongs to non-domain, if yes, grade this answer as non-domain, continue on next answer
   (b) check if the answer belongs to contradictory, if yes, grade this answer as contradictory, continue on next answer
   (c) Apply Word Sense Based Sentence Semantic Similarity Measurement between this answer and each reference answer, use the greatest similarity score as this answer's score
   (d) Grade this answer beased on its score and boundaries of "irrelevant", "partial_correct_incomplete" and "correct"

3.4 Test Results

seb test set:

	precision	recall	fmeasure
correct	0.6710240	0.7080460	0.6890380
partially_correct_incomplete	0.3366337	0.4112903	0.3702359
contradictory	0.4361702	0.4100000	0.4226804
irrelevant	0.5871560	0.4383562	0.5019608
non_domain	0.5000000	0.6000000	0.5454545
macroaverage	0.5061968	0.5135385	0.5058739
microaverage	0.5490251	0.5388889	0.5399240

beetle test set:

	precision	recall	fmeasure
correct	0.6627451	0.7752294	0.7145877
partially_correct_incomplete	0.3636364	0.4000000	0.3809524
contradictory	0.5132743	0.3945578	0.4461538
irrelevant	0.0000000	0.0000000	0.0000000
non_domain	0.7666667	0.8518519	0.8070175
macroaverage	0.4612645	0.4843278	0.4697423
microaverage	0.5396726	0.5653846	0.5481903

4. Future works

First of all, find more accurate algorithm to determine word weights in a sentence other than PLSA which has similar performance with simple unigram approach. And instead of apply the same boundaries of score to all the answers, we are sure that training different boundaries for each question could improve performance because we observed different best boundaries of score on different questions.