FIT5186 Intelligent Systems Assignment

Synonyms Discovery Using Neural Networks

FANG Yishu (2469****) and XIA Yingying (2469****)

Last Updated: 30 May 2013

Abstract

One critical problem in building datasets of Chinese Linking Open Data (CLOD) is how to discover synonyms from different ontologies. In this work, we have built a neural network model to solve that problem by conducting a series of experiments. Three inputs for the neural network are calculated by three different algorithms we have already developed for a given pair of words, and output will be either true or false. The first algorithm is based on Levenshtein distance, and the other two are based on the co-occurrence of related categories.

1. Introduction

With the development of the Sematic Web, hundreds of datasets have been published under community project of Linking Open Data (LOD) (Berners-Lee & O'Hara, 2013), which is an ideal tool to connect the distributed data across the entire Web. However, native Chinese LOD (CLOD) rarely exists and Chinese information also rarely appears in existing multilingual LOD datasets. So, we try to build a native CLOD dataset from a wide range of heterogeneous Chinese websites. One critical problem in our process is how to find synonyms from different ontologies, since these synonyms will be the basis of ontology merging.

We have developed three different algorithms to estimate whether one word is similar to another (i.e. they are synonyms). Each of those algorithms will generate an estimated value ranging from 0 to 1 based on the two input words (represented as strings). The first algorithm (we call it BOWIsim) is based on Levenshtein distance, the smaller the output is, the more similar they are. As the basis of the next two algorithms, we define related categories of a word as all the words related to that word, for example related categories of 'Audi' may be 'car', 'German', and so on. We build related categories for each word by post that word to a search engine and record all the meaningful words appeared on the web page. Inspecting the similarity of related categories of two words, we develop the second (BORCSsim) and the third algorithm (BORCVsim), the larger the output is, the more similar they are.

But unfortunately, according to previous experiments and analysis, we find that it is hard to achieve the desired effect using single algorithm of these three. As a result, we have built a model with the calculate values of three algorithms as the inputs and true or false as the output.

2. Data Sets

2.1 Pattern preparation

To build this model, we should build a data set for supervised learning. This process involves three steps: finding a set of word pairs as the original data, calculating three algorithm's value of each pair as the input of the network and determine whether each pair of words is really synonym.

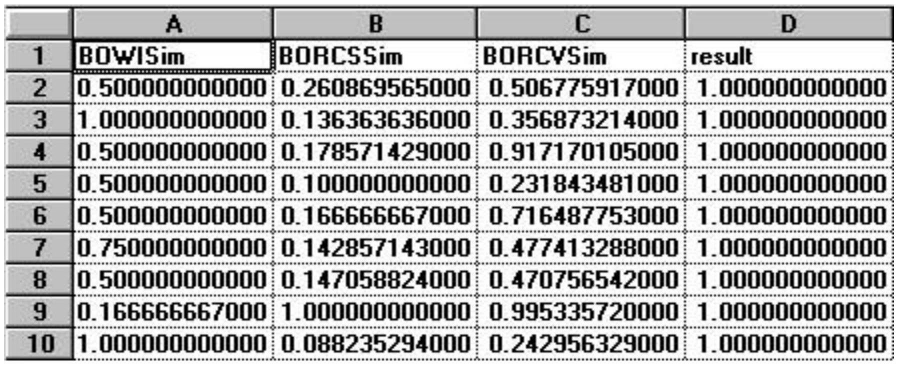
The first is randomly selecting 489 pairs of words from our ontologies. Then, we apply those three algorithms to the 489 pairs to get three values for each pair. And the values will be used as the inputs of the network. Finally, we inspect each pair to determine whether that pair of words is really synonym or not and assign a boolean value to each pair as the desired output of network.

We store all the information in file: "Experiment Data.xlsx". Two tables are contained in that file: Experiment Data (with string) and Experiment Data (pure data). The former one shows all the values including pairs of strings. The latter contains experiment data only.

2.2 Patterns importing to NerualShell 2

We export the pure data table to .csv format, and then use ASCII Files Import Module to import that csv file into PAT file (SIMILAR.PAT).

Here are the first few lines of our pattern file:

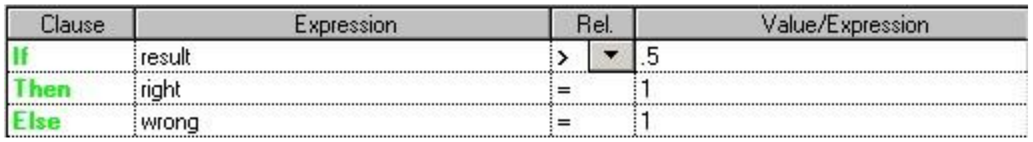


	A	B	C	D
1	BOWISim	BORCSSim	BORCVSim	result
2	0.500000000000	0.260869565000	0.506775917000	1.000000000000
3	1.000000000000	0.136363636000	0.356873214000	1.000000000000
4	0.500000000000	0.178571429000	0.917170105000	1.000000000000
5	0.500000000000	0.100000000000	0.231843481000	1.000000000000
6	0.500000000000	0.166666667000	0.716487753000	1.000000000000
7	0.750000000000	0.142857143000	0.477413288000	1.000000000000
8	0.500000000000	0.147058824000	0.470756542000	1.000000000000
9	0.166666667000	1.000000000000	0.995335720000	1.000000000000
10	1.000000000000	0.088235294000	0.242956329000	1.000000000000

Figure 1: Original patterns

2.3 Data preprocessing

As a classification problem, it’s better to use two output neurons corresponding to two categories. So we need to do build two actual output fields with name ‘true’ and ‘false’ based on the value of result field. If the result field of a pattern is 1 then assign 1 to true field and 0 to the false field. On the contrary, if result field is 0 then assign 0 to true field and 1 to false field. This can be easily done by using **Rules** module. We define and apply a rule:

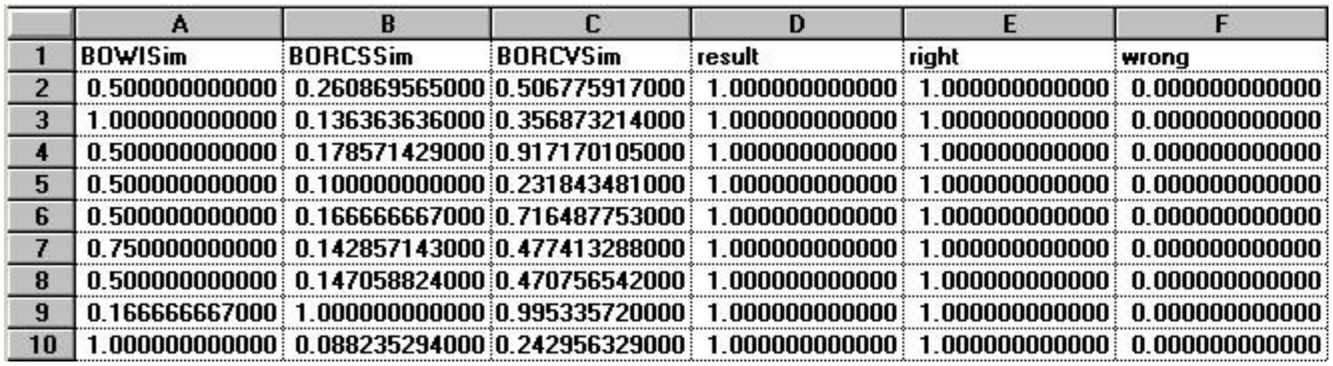


Clause	Expression	Rel.	Value/Expression
If	result	>	.5
Then	right	=	1
Else	wrong	=	1

Figure 2: Rules

This rule is stored in SIMILAR.RLI file.

After preprocessing the first few lines of our pattern file (SIMILAR.PAT) look like this:



	A	B	C	D	E	F
1	BOWISim	BORCSSim	BORCVSim	result	right	wrong
2	0.500000000000	0.260869565000	0.506775917000	1.000000000000	1.000000000000	0.000000000000
3	1.000000000000	0.136363636000	0.356873214000	1.000000000000	1.000000000000	0.000000000000
4	0.500000000000	0.178571429000	0.917170105000	1.000000000000	1.000000000000	0.000000000000
5	0.500000000000	0.100000000000	0.231843481000	1.000000000000	1.000000000000	0.000000000000
6	0.500000000000	0.166666667000	0.716487753000	1.000000000000	1.000000000000	0.000000000000
7	0.750000000000	0.142857143000	0.477413288000	1.000000000000	1.000000000000	0.000000000000
8	0.500000000000	0.147058824000	0.470756542000	1.000000000000	1.000000000000	0.000000000000
9	0.166666667000	1.000000000000	0.995335720000	1.000000000000	1.000000000000	0.000000000000
10	1.000000000000	0.088235294000	0.242956329000	1.000000000000	1.000000000000	0.000000000000

Figure 3: Patterns after preprocessing

2.4 Data analyzing

We can have a brief view of how data is distributed using this figure generated by Weka developed by The University of Waikato (Garner, 1995; Hall et al., 2009):

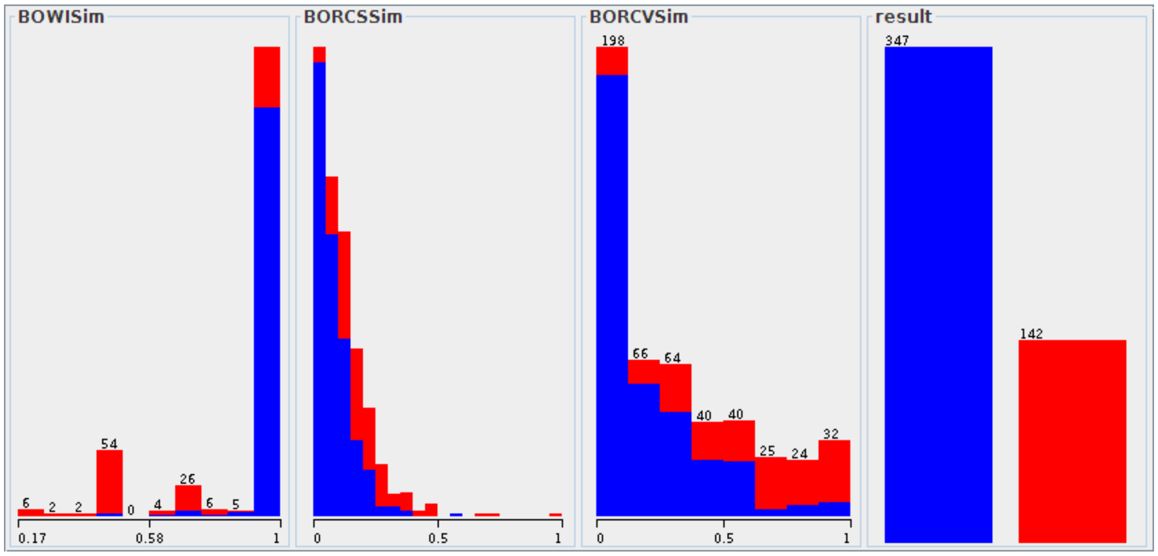


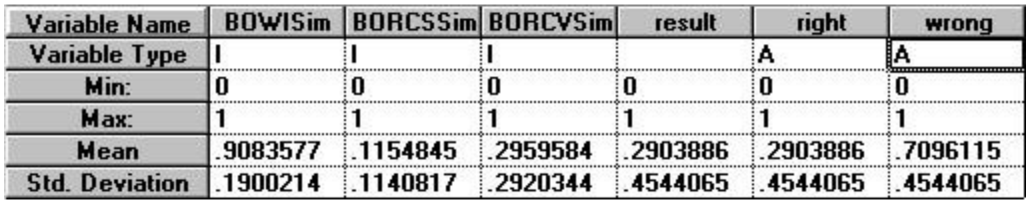
Figure 4: bar chart of all the patterns

The blue portion represents false (with value 0) and red represents true (with value 1). Through this bar chart, we can find that BOWISim is negatively related with result while BORCSSim and BORCVSim are positively related with result.

3. Training Issues

3.1 Define inputs and outputs

The first thing we need to do is defining inputs and actual outputs using module Define Inputs/Outputs. We define BOWISim, BORCSSim, BORCVSim as inputs and define right and wrong as actual outputs.



Variable Name	BOWISim	BORCSSim	BORCVSim	result	right	wrong
Variable Type	I	I	I		A	A
Min:	0	0	0	0	0	0
Max:	1	1	1	1	1	1
Mean	.9083577	.1154845	.2959584	.2903886	.2903886	.7096115
Std. Deviation	.1900214	.1140817	.2920344	.4544065	.4544065	.4544065

Figure 5: Define inputs/outputs

One thing worth mentioning is that we have manually set Min and Max to 0 and 1. As all the algorithm generated values are in the range of [0, 1], the outputs should be either 0 or 1.

3.2 Training set and test set extraction

With inputs and outputs defined, we should then use Test Set Extraction Module to split all patterns into two sets: training set and test set. Here we randomly select 20% data items as the test set. As a result, 392 patterns are stored in SIMILAR.TRN as training set and 97 patterns are stored in SIMILAR.TST as the test set.

3.3 Architecture and parameter selection

Four different architectures have been chosen to do this experiment.

As specified in the assignment specification, we will mainly use back-propagation learning algorithm to train our neural network. Considering that our problem is not very hard, we will use a three layer back-propagate standard connection network. We will conduct 5 experiments with number of hidden neurons set as 5, 10, 22, 30 and 40. Using Back-propagation Training Criteria Module, we can specify some training parameters and stop criteria. The learning rate (c) and momentum rate (α) are fixed at 0.1 and 0.1, respectively. The initial weights are small random numbers around 0.3. The order in which input patterns are presented to the network is also random and the network performance is measured on the test set every 200 epochs. If the test set error has not improved within 2000 epoch, then training was terminated to prevent overtraining.

In spite of the above architecture, we also use another two architectures: Ward net and Probabilistic Neural Network because they are recommended architectures by NerualShell 2 to do prediction and classification, respectively. We just want to have a try and see the differences in performance comparing to the first architecture.

The most classical algorithm to do classification seems to be Logistic Regression. So our fourth architecture will employ this algorithm. For NerualShell 2 only provides a general regression net architecture, we decide to use Weka to do the Logistic Regression.

As a conclusion, four architectures are used:

1) Three layers, standard connection back-propagate network.
2) Three hidden slabs, different activation functions Ward net (back-propagate network).
3) Probabilistic Neural Network (PNN)
4) Logistic Regression (using Weka)

Architecture 1 will be conducted using 5 different numbers of hidden neurons.

All 8 experiments are summarized here:

Experiment Number	Architecture	Number of Neurons
E1	Three layers, standard connection back-propagate network	3-22-2
E2		3-5-2
E3		3-10-2
E4		3-30-2
E5		3-40-2
E6	Ward net	$3\text{-}^{7}{}_{7}\text{-}7\text{-}2$
E7	PNN	3-489-2
E8	Logistic Regression	N/A

Table 1: Experiment list

Experiment 1 to 7 are conducted using NerualShell 2, and I have renamed the FIG file (configuration file), Nxx file (network file, xx is 01 in experiment 1-5, 14 in experiment 6, 11 in experiment 7) to En.FIG and En.Nxx where n is 1-7. Experiment 8 is conducted using Weka. Related files are stored in the folder weka.

4. Results

4.1 Apply models to test set

Having applied those network files to file SIMILAR.TST and the outputs are stored in En.OUT (n is 1-7), we list some statistical information in the following table:

Experiment Number	Right as Right	Wrong as Wrong	Right as Wrong	Wrong as Right	Right			Wrong		
					Precision	Recall	F-measure	Precision	Recall	F-measure
E1	30	56	10	1	96.77%	75.00%	0.85	84.85%	98.25%	0.91
E2	31	54	9	3	91.18%	77.50%	0.84	85.71%	94.74%	0.90
E3	31	56	9	1	96.88%	77.50%	0.86	86.15%	98.25%	0.92
E4	30	56	10	1	96.77%	75.00%	0.85	84.85%	98.25%	0.91
E5	31	54	9	3	91.18%	77.50%	0.84	85.71%	94.74%	0.90
E6	32	53	8	4	88.89%	80.00%	0.84	86.89%	92.98%	0.90
E7	29	56	11	1	96.67%	72.50%	0.83	83.58%	98.25%	0.90
E8	27	97	13	0	100%	67.50%	0.81	81.40%	100%	0.90

Table 2: Experiment result
(Total Number of patterns in test set: 97

Actually right pattern number in test set: 40
Actually wrong pattern number in test set: 57)

In the above table, Right as Right means number of patterns actually right and classified as right, Wrong as Wrong, Right as Wrong and Wrong as Right are defined in the same way. The last six columns show statistical information for right class and wrong class. Precision for Right class is defined as $\frac{\text{number of Right as Right}}{\text{number of classified as Right}}$. Recall is defined as $\frac{\text{number of Right as Right}}{\text{number of actually is Right}}$. F-measured is defined as $\frac{2}{\frac{1}{precision}+\frac{1}{recall}}$. Similarly, the statistical information is defined for Wrong class.

4.2 Analysis

From the above results we can find that Experiment 3 which employs standard back-propagate network with 10 hidden neurons does the best performance, but just gets a slightly improvement than others. This is the contribution graph plotted by Relative Contribution (Strength) Factors Module for Experiment 3:

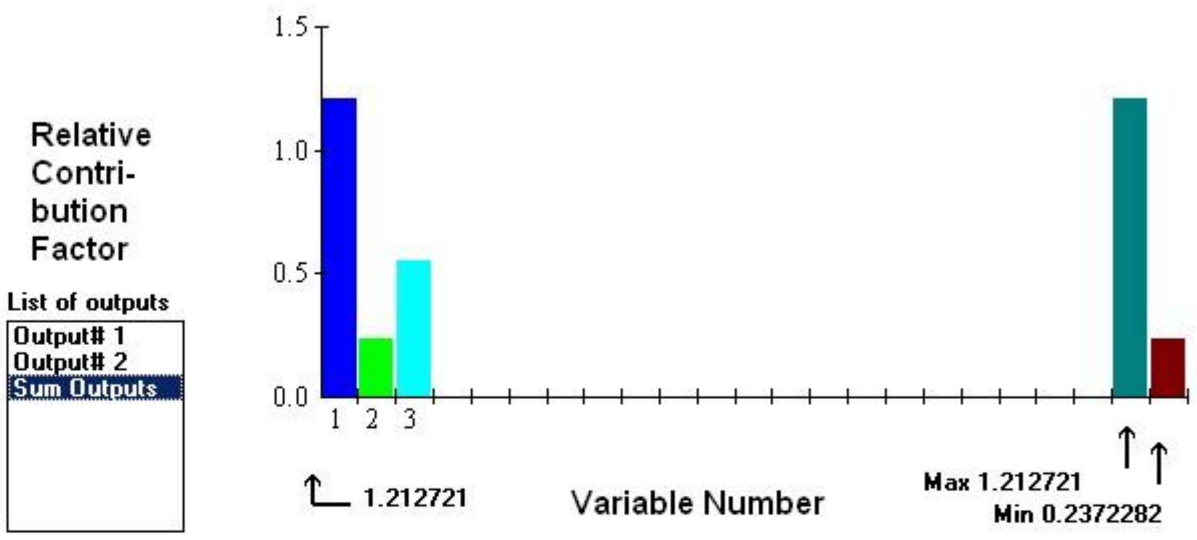


Figure 6: Contribution graph for Experiment 3

From the graph, we can find that BOWISim which based on Levenshtein distance contributes more to the result.

Inspecting the precision and recall of right and wrong, we find that all the networks tend to classify a pattern as wrong rather than wrong, this probably because we provide more wrong patterns than right in training set.

5. Limitations and future works

After carefully analyzing experiment data and results, we find the following four problems and limitations:

1) Levenshtein distance is originally designed for English string match and the performance is not as good as expected when applied to Chinese string. Probably this is because that English words are much longer than Chinese words in the number of characters. An improved way should be found to suit Chinese context.
2) Both the second and third algorithms have used related categories as basis. But the way we get them is very simple and the amount of related categories is not large enough. So a better way should be found to get related categories of a word.
3) In some special case, two words have both similar strings presentation and related categories (i.e. 南京大学 and 东南大学), it is hard to correctly classify them. A more suitable way to classify this kind of pattern is needed.
4) Our model has not made full use of neural network, just use it to make a fitting of three different algorithms. In our future work, we will consider to use related categories to train the neural network directly instead of using the second and the third algorithms.

6. Conclusion

To solve a critical problem in ontology merging, we have built a neural network to classify whether two words are synonyms. Three inputs for the neural network are calculated by three different algorithms for a given pair of words, and output will be either true or false. Experiments have shown that the first algorithm contributes much more than the two others and the network tends to classify patterns to be wrong more than true.

References

Berners-Lee, T., & O'Hara, K. (2013). The read–write Linked Data Web. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 1471-2962.

Garner, S. R. (1995). Weka: The waikato environment for knowledge analysis. *Proceedings of the New Zealand computer science research students conference*, 57-64.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter, 11*(1), 10-18.