August, 2020

Final Project Report

Distributed System Programming

Tal Litman – 203519848 & Guy Ben Haim – 204553820

1. Design

1.1. General Flow:

1.1.1. Go over the Corpus and extract the path between each pair of words given in sentence from the 'Biarcs' dataset, marking each special path as a pattern with a special index in the final vector.

1.1.2. Ignore each path that contains less distinct noun-pairs then dpmin.

1.1.3. Build the features vector for each noun-pair from the patterns for each one and a Boolean describing if the noun-pair is hypernym based on the 'Annotated Set' input.

1.1.4. Train: use Weka software for classification based on the features vector and the 'Annotated Set' to train the classifier.

1.2. Components Of The System:

1.2.1. Class Node – represents a single node in the dependency path tree . Each node consists of:

a. word – the actual word from the Corpus.

b. part-of-speech-tag - the represented tag of the word as described here: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

c. depth-label - is a Stanford-basic-dependencies label as described here: https://nlp.stanford.edu/software/dependencies_manual.pdf

d. head-index - is an integer, pointing to the head of the current token as described here:

all parsed from the 'Biarcs' dataset (e.g. cease/VB/ccomp/0).

1.2.2. Stemmer object – In the node creation phase, for each word in the input we will stem it.

1.2.3. Class DependencyTree – represents the sentence dependency tree for getting the patterns between the different nodes which is the pattern for the noun-pair. It consists of:

a. tree – a simple direct graph whose edges built from node to it's head-index, its purpose is to find the shortest path between 2 words i.e. between 2 nodes.

b. total_count - count of appearances of the noun-pair per pattern as parsed from the 'Biarcs' dataset.

c. Patterns – a list of 'TreePattern's, each one consists of path and total count for a noun pair.

1.2.4. Class NounPair – describes a pair of noun words from some sentence in the Corpus. It consists of the words, Boolean describing if the noun-pair is hypernym and total count.

1.2.5. Map-Reduce components –

In each step, *NounPair* refers to the class describes in 1.2.3

a. Step one:

- Mapper – given the ngram resource, parse each sentence to a dependencyTree and for each path in the tree, emit <pattern as a string*, *NounPair*>.
  *i.e. <noun posTag : depLabel : noun posTag…>, we have excluded the words for getting high DPmin value for most of the patterns.
- Reducer – check for each pattern if there are less distinct noun-pairs then dpmin, if not it won't be considered as a feature. For each pair
  emit <*NounPair*, index of the current pattern in the final vector>.
  in the cleanup method, upload to S3 the total number of patterns (=index) to determine the size of the vector to be created in the next step.

b. Step Two:

- Mapper – given multiple inputs, the <key, value> from the Reducer in step one and the 'annotated set',
  emit < *NounPair with the Boolean inside*, -1> and < *NounPair*, index> which will be sorted by the environment (i.e. the '*compareTo'* method of the NounPair object) so the Reducer will get the *NounPair* with the Boolean value first and then the <key, value> from step one.
- Reducer – build the features vector for each *NounPair*, based on the Boolean variable '*isHypername*' in the *NounPair* object and total count for each index (=coordinate) in vector.
  Emit <pair of words as string, isHypername + featuresVector as string> for the Weka to train on.

1.2.6. Weka ('*classifier.java'* class) –

a. For the results from the map-reduce phase, run the *parseOutput()* method in the '*classifier.java'* class, which converts the results from the map-reduce phase to an '*arff'* file.

b. Run the *analyze()* method for running Weka software with the '*BayesNet* classification algorithm for cross-validation of 10, and prints the results.

2. Communication:

2.1. Number Of Key-value sent from Mappers to Reducers:

2.1.1. Step One - 12631437 (437907960 bytes)

2.1.2. Step Two - 12566529 (317367647 bytes)

3. Results:

   We have tested our algorithm for few DPmin options (and with 2 datasets – 00 and 01), highest values were obtained with DPmin=100:

   3.1. DPmin = 100:

      3.1.1. Precision - 0.743761412051126

      3.1.2. Recall - 0.9873659468194506

      3.1.3. F1 measures - 0.8484236437655821

   3.2. DPmin = 50 (for comparison):

      3.2.1. Precision ~ 0.6901

      3.2.2. Recall ~ 0.97

      3.2.3. F1 measures ~ 0.78

4. Analysis:

   4.1 results examples:

      4.1.1. True Positive –

         4.1.1.1. “1 number”

         4.1.1.2. “1890 period”

         4.1.1.3. “accord distribut”

         4.1.1.4. “accord document”

         4.1.1.5. “1930 decad”

      4.1.2. False Positive –

         4.1.2.1. “1940 type”

         4.1.2.2. “a book”

         4.1.2.3. “two level”

         4.1.2.4. “1950 industri”

         4.1.2.5. “a represent”

      4.1.3. True Negative –

         4.1.3.1. “a bibliography”

         4.1.3.2. “abbei benedict”

         4.1.3.3. “abil korea”

4.1.3.4. “absenc agreement”

4.1.3.5. “absenc deposit”

4.1.4. False Negative –

4.1.4.1. “abolit action”

4.1.4.2. “accept instrument”

4.1.4.3. “access thought”

4.1.4.4. “accommod lodj”

4.1.4.5. “accompany music”

4.2. Explanations for the false classes:

4.2.1. We have printed the vectors for the false classes noun-pairs from the output of the second map-reduce phase and have noticed that most of them ,comparing to the positive classes noun-pairs, have low values and overall vectors are sparse (most of the coordinates were zeros e.g. (0,0,0,0,0,0,1,0,0,0,2)), thus, it’s likely that the classifier doesn’t have enough information to classify those noun-pairs properly.

4.2.2. It can be seen that in most cases of all classes that the shortest path between the words of the *nounPair* includes the postTag ’in’ or ‘to’ which makes false cases in classification.
Those are likely to appear in any nounPair that classified as TP, TN so the features that are represented by those posTags decrease the accuracy of the classifier.

e.g. for the TP *nounPair* “1 number” appears in the dp ‘nn:dep:**in**:prep:nns’
and the FP *nounPair* appears in the dp “nns:pobj:**in**:prep:nns”.