SENTENCE SELECTION FOR QUESTION ANSWERING

TUDOR BUZU, AI2

SENTENCE SELECTION FOR QA

- An important subtask of the question answering (QA) problem;
- The objective: identify the sentence from a large text or paragraphs, that contains the correct answer for a given question.

STRUCTURE OF THE PROJECT

1.) Structure of the dataset (analysis)
2.) Data preprocessing
    2.1) Context/passage sentence tokenization
    2.2) Building labels for correct answers
    2.3) Word tokenization
    2.4) Lemmatization
    2.5) Stopwords elimination
3.) Solutions
    3.1) Random approach
    3.2) String kernels
        3.2.1) Spectrum kernel
        3.2.2) Presence kernel
        3.2.3) Intersection kernel
    3.3) Sentence similarity
    3.4) Logistic regression using pretrained word embeddings
    3.5) BM25
    3.6) Combined methods
4.) Results

STRUCTURE OF THE DATASET (ANALYSIS)

1.) SQuAD (Stanford Question Answering Dataset)[1]

- a new reading comprehension dataset that covers a wide range of topics (from musical celebrities, sports, history to abstract concepts);
- 536 articles (sampled uniformly at random from the top 10,000 articles of English Wikipedia);
- the articles were partitioned randomly into training (80%), development (10%) and test (10%) sets.
- 23,215 paragraphs;
- up to 5 questions on the content of one paragraph;
- the correct answer is a segment of text from the corresponding reading passage.

DATA PREPROCESSING

- Context/passage sentence tokenization;
- Building labels for correct answers;
- Word tokenization;
- Lemmatization;
- Stopwords elimination.

SOLUTIONS

- Random approach;
- String kernels;
  - Spectrum kernel;
  - Presence kernel;
  - Intersection kernel;
- Sentence similarity;
- Logistic regression using pretrained word embeddings;
- BM25;
- Combined methods;

RANDOM APPROACH

- A random permutation for candidate answers is generated;
- This method gives weak results: Prec@1 ~ 0.25 and MAP ~ 0.5.

SPECTRUM KERNEL

- This kernel is defined like the sum of products between frequencies of question and candidate answer common words;

- Results: Prec@1 ~ 0.765 and MAP ~ 0.86.

```python
def spectrum_kernel_value(question_words, sentence_words):
    kernel_value = 0
    vocab_inters = set(question_words).intersection(sentence_words)

    for word in vocab_inters:
        kernel_value += num(word, question_words) * num(word, sentence_words)

    return kernel_value
```

PRESENCE KERNEL

- This kernel is defined like the number of common words in question and candidate answer;
- Results: Prec@1 ~ 0.8 and MAP ~ 0.87.

```python
def presence_kernel_value(question_words, sentence_words):
    kernel_value = 0
    vocab_inters = set(question_words).intersection(sentence_words)

    return len(vocab_inters)
```

INTERSECTION KERNEL

- This kernel is defined like the sum of minimum frequencies of common words in question and candidate answer;

- Results: Prec@1 ~ 0.8 and MAP ~ 0.87.

```python
def intersection_kernel_value(question_words, sentence_words):
    kernel_value = 0
    vocab_inters = set(question_words).intersection(sentence_words)

    for word in vocab_inters:
        kernel_value += min(num(word, question_words), num(word, sentence_words))

    return kernel_value
```

SENTENCE SIMILARITY

- Similarity between a question and a candidate answer is calculated as the average of maximum similarities between synonyms of each word from question and synonyms from answer;
- Results: Prec@1 ~ 0.74 and MAP ~ 0.83.

RESULTS

	Train set		Dev set	
Method \ Evaluation metric	Prec@1	MAP	Prec@1	MAP
Random approach	0.245	0.491	0.261	0.500
Spectrum kernel	0.757	0.856	0.779	0.866
Presence kernel	**0.7853**	**0.8717**	**0.8180**	**0.8866**
Intersection kernel	0.7854	0.8718	0.8181	0.8865
Sentence similarity (path_similarity)	0.717	0.826	0.750	0.841
Sentence similarity (wup_similarity)	0.707	0.819	0.741	0.835

LOGISTIC REGRESSION

- I trained a logistic regression classifier using word embeddings from a sentence;
- I used pretrained word embeddings (Google News Dataset);
- A sentence is the average of its word embeddings;
- Feature vector for training: diff_abs, diff_sqr, dot_product, min, max, sum of question and candidate answer sentence embeddings.

RESULTS

	Train set		Dev set	
Feature vector \ Evaluation metric	Prec@1	MAP	Prec@1	MAP
diff_abs	**0.730**	**0.835**	**0.752**	**0.8444**
diff_sqr	0.727	0.833	0.752	0.8442
dot_product	0.655	0.793	0.677	0.802
min	0.714	0.825	0.739	0.836
max	0.713	0.825	0.739	0.836
sum	0.280	0.524	0.294	0.535

BM25

- A good ranking function used by search engines to rank matching documents according to their relevance to a given search query.
- Given a query *Q*, containing keywords q_1, q_2, ... , q_n, the BM25 score of a document *D* is:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{\text{avgdl}}\right)},$$

Usually, the values of free parameters: $k_1 \in [1.2, 2.0]$ and $b = 0.75$.

- Results:
  - train: Prec@1 ~ 0.777 and MAP ~ 0.869.
  - dev: Prec@1 ~ 0.808 and MAP ~ 0.882.

COMBINED METHODS

- I used the scores from the previous methods and trained again a logistic regression classifier;
- Scores from string kernels + LR (word embeddings) + BM25;
- The results look better:
    - train: Prec@1 ~ 0.804 and MAP ~ 0.883.
    - dev: Prec@1 ~ 0.838 and MAP ~ 0.898.

FINAL RESULTS

	Train set		Dev set	
Method \ Evaluation metric	Prec@1	MAP	Prec@1	MAP
Random approach	0.245	0.491	0.261	0.500
Spectrum kernel	0.757	0.856	0.779	0.866
Presence kernel	0.7853	0.8717	0.8180	0.8866
Intersection kernel	0.7854	0.8718	0.8181	0.8865
Sentence similarity (path_similarity)	0.717	0.826	0.750	0.841
Sentence similarity (wup_similarity)	0.707	0.819	0.741	0.835
LR (word embeddings,feature vector = dif_abs)	0.730	0.835	0.752	0.8444
BM25	0.777	0.869	0.808	0.882
LR(LR(word embeddings) + string kernels + BM25)	**0.804**	**0.883**	**0.838**	**0.898**

Q&A