Sentence selection for question answering

(NLP&SSL Project)

Tudor Buzu, AI2

Introduction. Sentence selection is an important subtask of the question answering (QA) problem. The objective of this subtask is to identify the sentence from a large text or paragraphs, that contains the correct answer for a given question.

Example. Below is an example taken from the SQuAD[1]. On the left are some questions for the corresponding reading paragraph on the right. The correct answers are colored in red/blue and the sentences containing these answers are marked with bold and underline. The goal of the project is to find just the sentences with the correct answer, not the shortest fragment of the text.

What causes precipitation to fall?
gravity

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail... **Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

Datasets. There are different available datasets that can be used to tackle the question answering and reading comprehension problems. The most known datasets are the following:

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.

2. **WikiQA**[2]
   - a dataset for sentence selection;
   - includes questions for which there are no correct sentences;
   - 3,047 questions sampled from Bing query logs;
   - 29,258 sentences in the dataset, where just 1,473 are answer sentences;
   - the data is split in training (70%), development (10%) and test (20%) set.

Evaluation. For evaluation we can use the following metrics:

1.) **Accuracy.** Sentence selection can be seen as a classification problem, where the correct answer has label 1 and wrong answer – label 0.
2.) **P@k, MAP, MRR.** From an IR point of view, the methods/models for sentence selection can be evaluated by the top k sentence candidates for correct answer. These metrics evaluate the relative rank of correct answer in the candidate sentences of a question.

State-of-the-art. Deep learning solutions achieve state-of-the-art results for this problem.

1. **Deep learning for answer sentence selection**[4]
   - Binary classification problem;
   - Formulation:
     - Each data point is a triple (q_i, a_{ij} , y_{ij});
     - q_i – the ith question;
     - a_{ij} – the jth candidate answer of question i;
     - y_{ij} = 1 if the jth answer of question i is correct, 0 otherwise.

   This solution assumes that correct answers have high semantic similarity to questions. The probability of a answer being correct is:

$$p(y = 1|\mathbf{q}, \mathbf{a}) = \sigma(\mathbf{q}^T \mathbf{M}\, \mathbf{a} + b)$$

   where ***q*** and ***a*** are vector representations of question and answer, ***M*** – transformation matrix, ***b*** – bias.

   The idea is that they try to "generate" an question q` = M a, using a candidate answer, and then compute the similarity between generated question q` and given question q by their dot product.

   There are 2 models used to represent question and answers:

   a.) Bag-of-words model
   - Words: pretrained word embeddings (d=50);
   - Sentences (both question and answer): the average of word embeddings from the sentence;

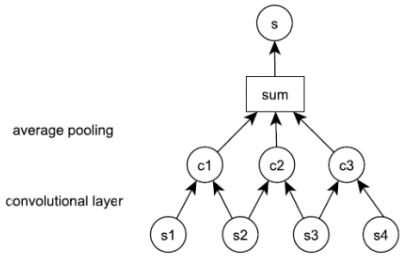
$$\mathbf{s} = \frac{1}{|\mathbf{s}|} \sum_{i=1}^{|\mathbf{s}|} \mathbf{s}_i$$

   b.) Bigram model
   - use a CNN to capture the information from bigrams;
   - convolutional filter combines adjacent words:

$$\mathbf{c}_i = \tanh(\mathbf{t} \cdot \mathbf{s}_{i:i+1} + b)$$

   - the idea is to learn how to combine the embedding of the words in the bigram;

- the architecture of the CNN:



$$\mathbf{s} = \sum_{i=1}^{|\mathbf{s}|-1} \tanh(\mathbf{T}_L\, \mathbf{s}_i + \mathbf{T}_R\, \mathbf{s}_{i+1} + \mathbf{b})$$

For evaluation the authors use MAP and MRR. All experiments are done on TREC Answer Sentence Selection - a dataset created from the TREC QA track (8-13). The summary of the answer sentence selection dataset is presented below.

Data	# Questions	# QA Pairs	% Correct	Judgement
TRAIN-ALL	1,229	53,417	12.0	automatic
TRAIN	94	4,718	7.4	manual
DEV	82	1,148	19.3	manual
TEST	100	1,517	18.7	manual

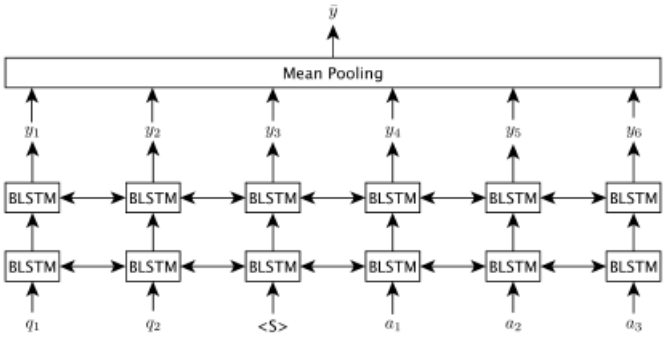
The obtained results are good.

Model	MAP	MRR
TRAIN		
unigram	0.5387	0.6284
bigram	0.5476	0.6437
unigram + count	0.6889	0.7727
bigram + count	**0.7058**	**0.7800**
TRAIN-ALL		
unigram	0.5470	0.6329
bigram	0.5693	0.6613
unigram + count	0.6934	0.7677
bigram + count	**0.7113**	**0.7846**

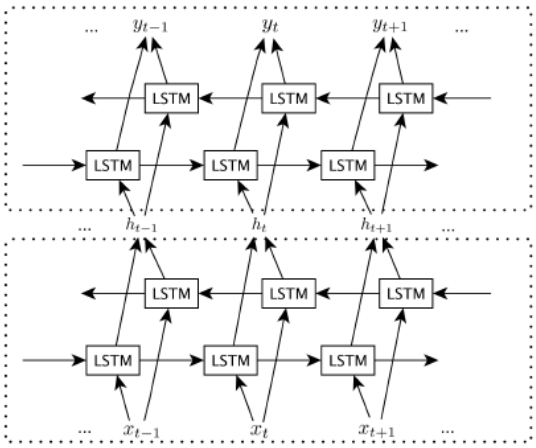
The best models match state-of-the-art results.

System	MAP	MRR
Baselines		
Random	0.3965	0.4929
Word Count	0.5707	0.6266
Wgt Word Count	0.5961	0.6515
Published Models		
Wang et al. (2007)	0.6029	0.6852
Heilman and Smith (2010)	0.6091	0.6917
Wang and Manning (2010)	0.5951	0.6951
Yao et al. (2013)	0.6307	0.7477
Severyn and Moschitti (2013)	0.6781	0.7358
Yih et al. (2013) – LR	0.6818	0.7616
Yih et al. (2013) – BDT	0.6940	**0.7894**
Yih et al. (2013) – LCLR	**0.7092**	0.7700
Our Models		
TRAIN bigram + count	0.7058	0.7800
TRAIN-ALL bigram + count	**0.7113**	**0.7846**

2. **A Long Short-Term Memory model for answer sentence selection in question answering**[5]
   - A combination of the stacked BLSTM relevance model and keywords matching;



   - the words of input sentences: pretrained word embeddings(word2vec model);
   - all the contextual information across the entire sequence (both question and answer sentences) is taken into consideration;
   - a big advantage: use both the previous and future context, by processing the data from two directions with two separate hidden layers;



Author solution achieve state-of-the-art results using a combination of kewords matching baseline score (BM25) and pooling values of three-Layer BLSTM outputs.

Reference	**MAP**	**MRR**
Yih et al. (2013) – Random	0.3965	0.4929
Wang et al. (2007)	0.6029	0.6852
Heilman and Smith (2010)	0.6091	0.6917
Wang and Manning (2010)	0.5951	0.6951
Yao et al. (2013)	0.6307	0.7477
Severyn and Moschitti (2013)	0.6781	0.7358
Yih et al. (2013) – BDT	0.6940	0.7894
Yih et al. (2013) – LCLR	0.7092	0.7700

Features	MAP	MRR
BM25	0.6370	0.7076
Single-Layer LSTM	0.5302	0.5956
Single-Layer BLSTM	0.5636	0.6304
Three-Layer BLSTM	0.5928	0.6721
Three-Layer BLSTM + BM25	**0.7134**	**0.7913**

Technologies/Tools. For implementation part, I will use Python with the following libraries: numpy (arrays), nltk (tokenization, stemming, lemmatization), gensim (word2vec model), sklearn (classification), tensorflow.

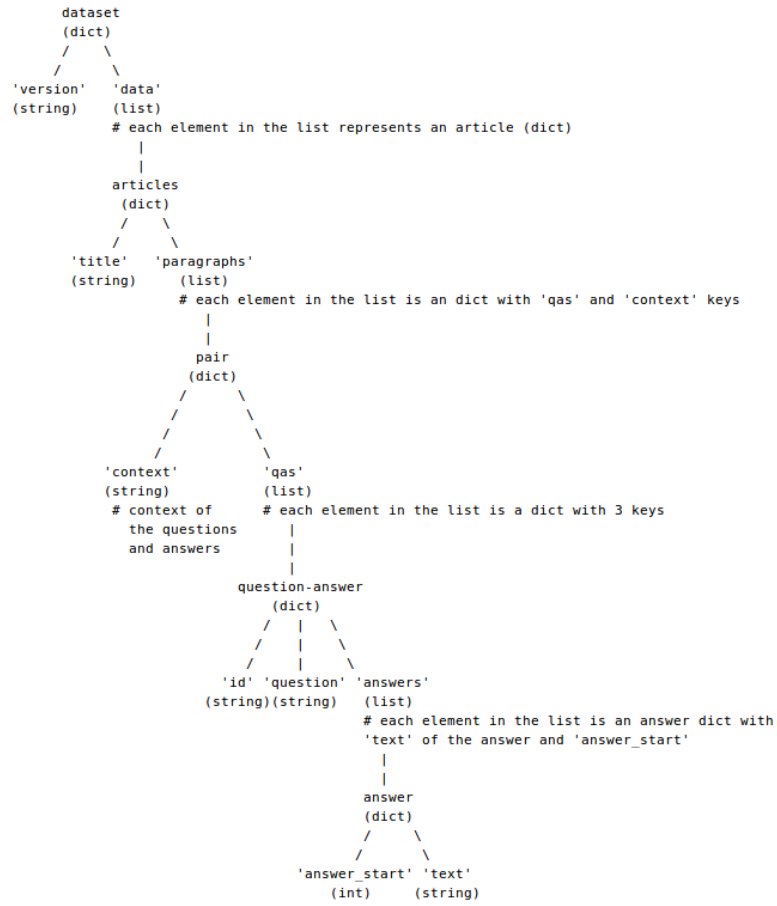
First version of the system

(Milestone 3)

Introduction. For the first version of the system I work with the SQuAD and I propose the following steps:

1.) Analysis for structure of the dataset
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
4.) Results

1.) Analysis for structure of the dataset. First of all, we need to analyze the structure of the dataset. After this step we can go forward and work wittingly.



2.) Data preprocessing. I preprocess data using nltk in the following way:

(a) Context/passage sentence tokenization;
(b) Building labels for correct answers;
(c) Word tokenization;
(d) Lemmatization;
(e) Stopwords elimination.

As a conclusion, the most of candidate answers are in the range [0, 5]. This means that we have small paragraphs with 1-5 sentences.

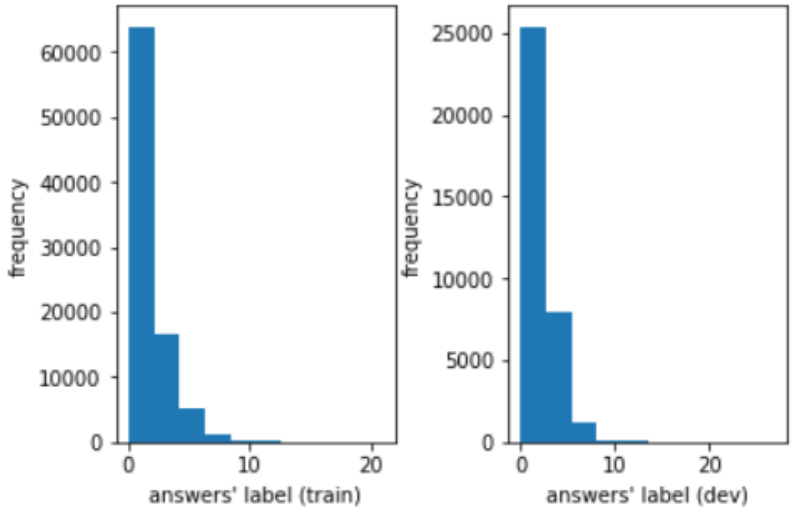


Figure 1. Distribution of answers' labels

3.) Solutions. For this stage of the project I proposed 3 solutions:

3.1) Random approach. A random permutation for candidate answers is generated. This method gives weak results: Prec@1 ~ 0.25 and MAP ~ 0.5.

3.2) String kernels. I will try to use string kernels at word level. I tested different types of kernel:

3.2.1) Spectrum kernel. This kernel is defined like the sum of products between frequencies of question and candidate answer common words. Results: Prec@1 ~ 0.65 and MAP ~ 0.8.

```
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
```

3.2.2) Presence kernel. This kernel is defined like the number of common words in question and candidate answer. Results: Prec@1 ~ 0.77 and MAP ~ 0.86.

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

    return len(vocab_inters)
```

3.2.3) Intersection kernel. This kernel is defined like the sum of minimum frequencies of common words in question and candidate answer. Results: Prec@1 ~ 0.77 and MAP ~ 0.86.

```
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
```

3.3) 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 words from answer. I worked with nltk, wordnet to get synonym sets for words and compute similarities between them. Results: Prec@1 ~ 0.74 and MAP ~ 0.83.

To compute the similarity among words I tested 2 similarity scores:

path_similarity: Return a score denoting how similar two word senses are, based on the shortest path that connects the senses in the is-a (hypernym/hypnoym) taxonomy. The score is in the range 0 to 1.

wup_similarity: Wu-Palmer Similarity: Return a score denoting how similar two word senses are, based on the depth of the two senses in the taxonomy and that of their Least Common Subsumer (most specific ancestor node).

4.) Results. For the first version of the syste, the proposed solutions were evaluated using Prec@1 (average of precisions at 1) and MAP (mean average precision) as metrics. The obtained results are presented below in the table. The best results, for both train and dev set, are achieved with presence kernel.

	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	0.657	0.793	0.677	0.802
Presence kernel	***0.762***	***0.856***	***0.793***	***0.871***
Intersection kernel	0.759	0.854	0.790	0.869
Sentence similarity (path_similarity)	0.717	0.826	0.750	0.841
Sentence similarity (wup_similarity)	0.707	0.819	0.741	0.835

Table 1. Results for the first version of the system

Final version of the system

(Milestone 4)

A summary for the final version of the system and new proposed solutions for the 4th milestone are presented in the list below:

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

In the 3rd milestone I forgot to use preprocessed data. I repeated the experiments with preprocessed data and I updated the 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

Table 2. Updated results for the first version of the system

For this milestone I tried the following methods:

Logistic regression. I trained a logistic regression classifier using word embeddings from a sentence. A sentence is the average of its word embeddings and the used pretrained word embeddings are trained on Google News Dataset.

For training I use the difference, dot product, min, max of sum of question and candidate answer sentence embeddings. All results are presented below.

	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

Table 3. Results for logistic regression approach

BM25. This is 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 that I used here are from string kernels + LR (word embeddings) + BM25. The results improve:

- ❖ train: Prec@1 ~ 0.804 and MAP ~ 0.883.
- ❖ dev: Prec@1 ~ 0.838 and MAP ~ 0.898.

A summary for all results are shown in the below table. The best results are obtained when we combine previous methods.

	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**

Table 4. Summary of all results

During the implementation I used git as a version control system.

The repository of the project can be found in my github account (Tudor67) here:
https://github.com/Tudor67/Sentence-Selection-For-Question-Answering

Steps to follow

1. Clone or download the project;
2. Create the folder `./data/squad/`;
3. Copy the [train and dev set of SQuAD][1] in `./data/squad/`;
4. Run the kernel `./src/data_preprocessing.ipynb`;
   After this step, in `./data/squad/` will be created the following files:
   * _train-v1.1-preprocessed.json_;
   * _dev-v1.1-preprocessed.json_;
5. Run the kernel `./src/baseline.ipynb`.
   Here are the first results using simple approaches.
6. Create the folder `./word_embeddings/`;
   Download and copy in this folder pretrained word embeddings:
https://code.google.com/archive/p/word2vec/

Run the kernel `./src/classification.ipynb`.

Here are the results using LogisticRegression for pretrained word embeddings.

6. Run the kernel `./src/bm25.ipynb`.

Here are the results using bm25 ranking function.

8. Run the kernel `./src/combined_methods.ipynb`.

Here are the last results using a combination of previous methods.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev and Percy Liang, "SquAD: 100,000+ questions for machine comprehension of text". In *Proceedings of the Conference on EMNLP*, 2016.

[2] Yi Yang, Wen-tau Yih and Cristopher Meek, "WikiQA: a challenge dataset for open-domain question answering". In *Proceedings of the Conference on EMNLP*, 2015.

[3] Mihai Masala, Stefan Ruseti and Traian Rebedea, "Sentence selection with neural networks using string kernels". In *KES*, 2017.

[4] Lei Yu, Karl Moritz Hermann, Phil Blunsom and Stephen Pulman, "Deep learning for answer sentence selection". In *NIPS*, 2014.

[5] Di Wang and Eric Nyberg, "A Long Short-Term Memory model for answer sentence selection in question answering". In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, 2015.