Evaluating common sense using artificial intelligence systems

Pere Ayats Marsal
Purdue University
Research Assistant report
07/01/2018 - 05/05/2018
Supervised by Professor Dan Goldwasser
Computer Science Department

0. Index

1. Introduction

This past semester I have been working with **Professor Dan Goldwasser and PhD student I-Ta Lee** for the completion of my final undergraduate thesis. Their research is in the **Natural Language Processing** field and one of its possible descriptions could be the evaluation of machine common sense using artificial intelligence systems.

They have been building a model capable of solving a **multiple choice quiz** involving different kinds of questions such as predicate, entity, pronoun and discourse type questions. This kind of quiz is built removing words of an article and the main task of the model is to rebuilt it given several options. To do so, the model has to be trained in order to **emulate common sense and machine intelligence**.

My task as research assistant has been to help them with several issues regarding the multiple choice quizzes and other topics. This is a summary of some of the tasks performed by me and by no means does this represent the whole effort put in the final product and article.

2. State of the Art

Even though computers have outperformed us on technical calculations and tasks, common sense has always been a topic of interest for Artificial Intelligence. We define it as **reasoning concerning particular knowledge** about mundane objects, events or actions and it has turned out to be really hard to capture.

One approach to this problem is to **create a database extensive enough** that could emulate human knowledge and hopefully also emulate their common sense. Examples of this would be **Cyc**[1] (Lenat and Guha 1989; Lenat 1995) **or ConceptNet/AnalogySpace**[2] (Speer, Havasi, and Lieberman 2008).

However, how can determine that such models reach our desired common sense? As we said, it is a difficult topic to argue since it has no clear outcome. Psychologists often use what we could call **intelligence tests** to deal with the same problems with humans. These allow them to compare the results with a well-thought standard. That is exactly our goal with the multiple choice quizzes that we designed.

[1] Lenat, D. B. 1995. *Cyc: a large-scale investment in knowledge infrastructure*.
[2] Speer, R.; Havasi, C.; and Lieberman, H. 2008. *AnalogySpace: reducing the dimensionality of common sense knowledge*.

3. Objectives of the project

- Understanding what **common sense** means regarding artificial intelligence and machine learning.
- Designing **a format of quiz** in order to evaluate the degree of machine intelligence of a model.
- Implementing tools to facilitate the creation of the quizzes and their format.
- Understanding the use and application of **word embeddings** in a real life scenario.
- Implementing **baselines** that use word embeddings to solve the quizzes that we have previously designed.

4. Generating questions and answers

4.1. Generating questions and their possible answers

For this task, we were given a set of **articles** from the Wall Street Journal with some resources involving **coreference chains** (all mentions of one entity in the text) and **discourse relations**. The format of each article was the following:

- Article
- List of all the coreference chains in the article
- List of all the discourse relations in the article

In the following parts we will provide strategies to generate questions that require a certain machine intelligence to solve and their possible answers. In annex 1, we will provide real examples of the strategies created.

4.1.1. Strategies to generate interesting questions

We need to establish rules to generate all the candidates that could become questions. Most of the following rules start with an item from the list of discourse relations provided. That is because it is more likely to get an interesting question if there is some kind of relation between two sentences.

1) We choose a discourse relation and **remove the discourse marker**.

2) We choose a discourse relation that has a pronoun in it. If its coreference chain has more entities, we **remove the pronoun**.

3) We choose a coreference chain that has a pronoun and some other entities and we **highlight the pronoun**.

4) We choose a coreference chain that has more than one entity and we **remove one occurence of an entity** making sure that it is not a pronoun (second rule).

5) We choose a discourse relation that has a predicate in it and we **remove the predicate**.

We will learn soon enough that strategy #3 disrupts the format of the multiple choice quiz and cannot be used.

4.1.2. Strategies to generate interesting answers

After establishing some rules to generate questions, we need to provide choices for the user to pick from. They need to be related somehow to the correct answer in order to be interesting to solve. The following strategies are the ones used to generate possible answers for every type of question explained in the last part:

1) We provide **discourse markers of a different type** from the correct one (cause/effect, comparison, contrast, addition…).

2) We provide **random pronouns**.

3) We provide **entities from other coreference** chains in the article making sure that they are not pronouns.

4) We provide **entities from other coreference** chains in the article making sure that they are not pronouns.

5) We provide **other predicates from the same article**.

4.2. Discourse span annotator

As we have seen in the last part, most of the questions are generated using the **discourse relations** of the article. Which means that we need a tool precise enough to detect all the relations of an article. That is why we implemented one.

We were given some training data and a set of **tokenized articles**. First of all, we used the training data to generate a **list of all the connectives/discourse markers** found. This is a very helpful list since it has all the connectives and its types.

Now, the algorithm just has to go through all the sentences checking if they have any connective of the list. To do so, we generate **all the possible combinations of words in a sentence** since some connectives involve multiple words:

```
comb = [sentence['tokens'][r:s] for r, s in itertools.combinations(range(len
(sentence['tokens'])+1), 2)]
```

```
comb = [" ".join([y['word'] for y in x]) for x in comb]
```

Then, we check if any of this combinations is a connective. If so, we save all the information about the connective:

```
for c in comb:
       if c in marker_list:
              doc['discourse'].append({
                     'connective': c,
                     'type': unfiltered_json['connectives'][c],
                     'arg1': (i-1) if i > 0 else i,
                     'arg2': i
              })
```

In annex 2, we will provide the full code for this discourse span annotator.

5. Implementation of baselines to solve questions

In order to test out the difficulty of the questions and compare it to the model implemented by Professor Goldwasser and I-Ta Lee, we implemented three baselines that can solve all the question types using **word embeddings**. All three baselines follow the same behaviour even though they use different libraries and models.

To use the context as our decision-making criteria, having a sentence with a question, we first generate the word embeddings for its **previous and next sentences**. Then, we replace the question tag with **every possible option** and we compute the resulting word embeddings.

We need to find the option with the **highest similarity** between the current and the previous sentence and between the current and the next sentence. That is going to be our solution. We use the **cosine similarity** between the word embeddings to make our decision.

5.1. GloVe model

GloVe (Global Vectors for Word Representation) is an unsupervised learning algorithm for obtaining **vector representations for words**. We downloaded a set of pre-trained word vectors of dimension 300. We treat them as an array where you can access the vector using **its word as the key**.

```
emb = words[w.lower()] if w.lower() in words else None
        if emb is not None:
            embeddings.append(emb)
```

To generate the word embedding of a sentence, we compute the **average vector of all the words** that appear in the sentence.

```
word = np.sum(embeddings, axis=0)
sent_emb = (word / len(embeddings))
```

5.2. Word2Vec library

For this baseline we use the **Python Word2Vec library**. In order to use this library, we also need to load a pre-trained model that contains all the word vectors. The recommended one for this library is **Google's trained model** that uses **Google News** data. We first need to load the model.

```
model = KeyedVectors.load_word2vec_format(word2vec_file, binary=True)
```

After that, we follow the same steps as the previous baseline. This algorithm has ended up being **the most efficient** and the one that **performs better** of the three baselines implemented.

5.3. Skip-Thoughts library

For this baseline we use the **Python Skip-Thoughts library.** This library has already its own models that you can download. We load the model and initialize the encoder and we are ready to go.

```
model = skipthoughts.load_model()
encoder = skipthoughts.Encoder(model)
```

In this case we have a function that returns the **word embedding of an entire sentence** so we do not need to compute the average vector.

```
embeddings = encoder.encode(sent_to_emb)
```

6. Automatization of answer generation

As we have seen in part 2.1.2, we have some strategies to generate all the possible answers for a certain question. We implemented an algorithm that generate them automatically as long as we feed it questions.

We were given a set of articles with questions in a certain format in order to generate options for them. We implemented an algorithm for the predicate type questions and another one for the discourse type questions.

We later found out that some of the strategies previously specified do not require a level of machine intelligence high enough to be interesting. In the next parts we explain all the modifications.

6.1. Predicate type questions

For this type of questions, instead of using other predicates of the same article as we said before, we need to find **the siblings of the right answer**. That ensures us that most of the options could also fit the question in another scenario.

To do so, we need to compute something called **synset**. We use the Python NLTK library. However, we need to **lemmatize the correct predicate** before computing it because precision problems can be found otherwise. To lemmatize all the predicates and get them back to their initial form, the Python Pattern library is used.

```
tense = en.tenses(answer_text)[0]
lemma = en.lemma(answer_text)
synset = wn.synset(lemma+".v.01")
synset_embedding = model[synset.lemma_names()[0]] if synset.lemma_names()[0]
in model else None
parent = synset.hypernyms()[0]

all_siblings = parent.hyponyms()
all_siblings = sorted(all_siblings, key=lambda elem: sorting_siblings(
elem,synset_embedding))

for s in all_siblings:
        if len(options) == 5:
                break
        option = en.conjugate(s.lemma_names()[0],**TENSES_DICT[tense])
        if option not in options:
                options.append(option)
```

However, the synset does not always provide us with four siblings or more. In this case, we need to have a backup strategy. We use word embeddings and the Word2Vec library again.

```
if len(options) < 5:
   options = [answer_text]
   options.extend(get_similar_options(answer_text, 4))
```

6.2. Discourse type questions

For this type of questions, we **reuse the list of connectives** created in part 2.2 but filtering the ones that are composed of other existing discourse markers. That way we get more relevant options and the decision is way more challenging.

As we said in part 2.1.2, the algorithm chooses connectives from a different type of the correct answer. So, we get the list of all the connectives that do not have the same type as the right answer.

```
if answer_text in disc_mark['connectives']:
        type_candidates = []
        for s in disc_mark['connectives'][answer_text]:
            type_candidates = list(set(disc_mark['senses'][s]).
union(type_candidates))

        candidates = [(dc,get_sim(dc,answer_text)) for dc in
disc_mark['connectives'] if dc != answer_text and dc not in type_candidates]
        candidates = sorted(candidates, key=lambda a: a[1], reverse=True)
else:
        candidates = sorted([(dc,get_sim(dc,answer_text)) for dc in
disc_mark['connectives'] if dc != answer_text], key=lambda a: a[1],
reverse=True)
```

Again, we use **word embeddings and the Word2Vec library** to get the most similar connectives to the answer.

7. Conclusion

It has been an amazing experience to work with such talented professionals and have the opportunity to learn more about this interesting field that is Natural Language Processing. Artificial Intelligence and NLP are fields that have always kept me interested but I never took the steps to actually get involved until now. It has been the right decision since it has also been useful to discover how the research works in the United States.

I cannot wait to hear about all the updates of this research and keep acquiring more knowledge and experience. Thank you so much for the opportunity.

8. Bibliography

- Lai, G., Xie, Q., Liu, H., Yang, Y. and Hovy, E. (2017). RACE: Large-scale ReAding Comprehension Dataset From Examinations. Pittsburgh, PA, USA: Language Technologies Institute, Carnegie Mellon University. Available at: https://arxiv.org/pdf/1704.04683.pdf.

- The PDTB Research Group (2007). The Penn Discourse Treebank 2.0 Annotation Manual. Philadelphia, PA, USA: University of Pennsylvania. Available at: https://www.seas.upenn.edu/~pdtb/PDTBAPI/pdtb-annotation-manual.pdf.

- Chen, D., Bolton, J. and Manning, C. (2016). A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task. Berlin, Germany: Association for Computational Linguistics. Available at: https://www.aclweb.org/anthology/P16-1223.

- Xue, N., Tou, H., Pradhan, S., Rutherford, A., Webber, B., Wang, C. and Wang, H. (2016). CoNLL 2016 Shared Task on Multilingual Shallow Discourse Parsing. Berlin, Germany: Association for Computational Linguistics. Available at: http://www.aclweb.org/anthology/K16-2001.

- Goldberg, Y. and Levy, O. (2014). word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. Available at: https://arxiv.org/pdf/1402.3722.pdf.

- Jurafsky, D. (2017). Language Modeling: Introduction to N-grams. Available at: https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf.

- Pennington, J., Socher, R. and Manning, C. (2014). GloVe: Global Vectors for Word Representation. Available at: https://nlp.stanford.edu/projects/glove/.

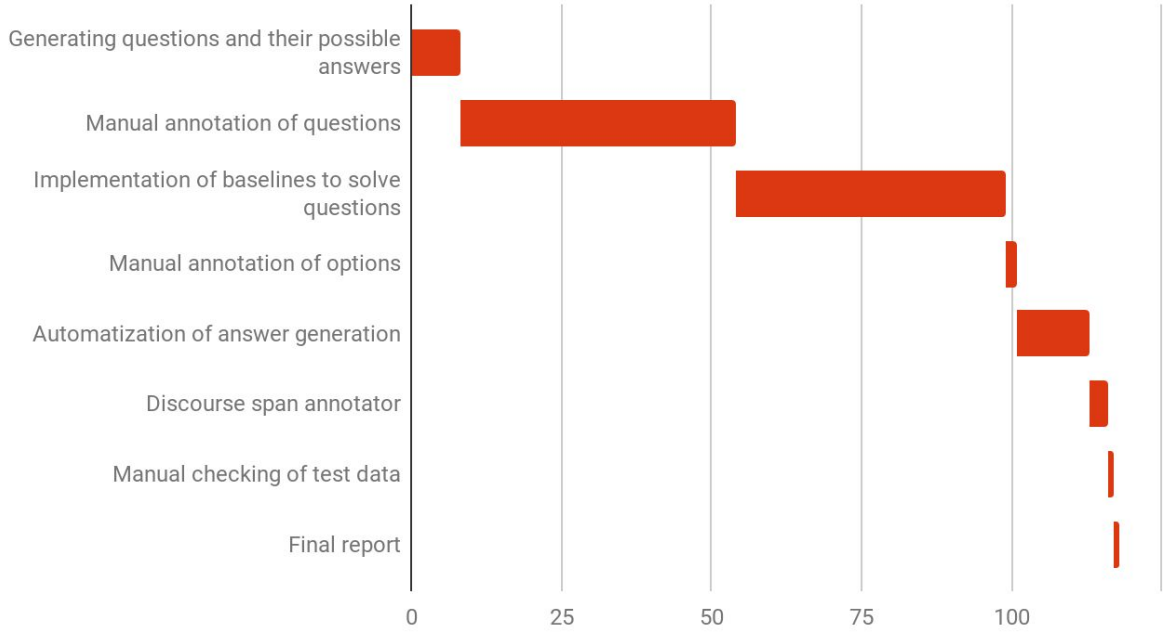
- Google Code Archive (2013). word2vec. Available at: https://code.google.com/archive/p/word2vec/.

- Řehůřek, R. (2009). models.word2vec – Deep learning with word2vec. Available at: https://radimrehurek.com/gensim/models/word2vec.html.

- Kyros, J. (2017). Sent2Vec encoder and training code from the paper "Skip-Thought Vectors". Available at: https://github.com/ryankiros/skip-thoughts.

❖ NLTK Project (2017). WordNet Interface. Available at: http://www.nltk.org/howto/wordnet.html.

9. Annex 1: Gantt Diagram



10. Annex 2: Examples of strategies used to generate questions

10.1. First Article

Some U.S. allies are complaining that President Bush is pushing conventional - arms talks too quickly, creating a risk that negotiators will make errors that could affect the security of Western Europe for years. Concerns about the pace of the Vienna talks -- which are aimed at the destruction of some 100,000 weapons , as well as major reductions and realignments of troops in central Europe -- also are being registered at the Pentagon. **(1)** has called for an agreement by next September at the latest. **(2)** some American defense officials believe the North Atlantic Treaty Organization should take more time to examine the long - term implications of the options being considered.

For one thing, Pentagon officials, who asked not to be identified, worry that the U.S. will have a much tougher time persuading Europeans to keep some short-range nuclear weapons on their soil once Soviet armored forces are thinned out.

At the same time, they **(3)** that a reduction of NATO forces under a treaty will increase the possibility of a conventional Soviet attack unless the West retains a residual force of nuclear weapons in Europe.

Allies concerned about the deadline include the British, French and smaller NATO allies, some of whom don't have adequate staffs to provide quick answers to the questions being raised by what generally are considered the most complex arms-control talks ever attempted. So far, no ally has complained openly, preserving the impression that NATO is in line with the Bush position that a quick agreement bringing Soviet conventional forces down to parity with NATO is the West's top bargaining priority. But even though NATO negotiators have only 10 months left under the Bush timetable, they are still wrestling over such seemingly fundamental questions as “What is a tank ?”. Five of the six categories of weapons under negotiation haven't even been defined. Tanks currently are defined as armored vehicles weighing 25 tons or more that carry large guns. The Soviets complicated the issue by offering to include light tanks, which are as light as 10 tons.

Oleg A. Grinevsky, the chief Soviet negotiator in the conventional-arms talks, argued that this would mean the Soviets would have to destroy some 1,800 tanks, while the U.S. would lose none because it has no light tanks in Europe. But the issue is stickier than it seems.

France, Britain and Italy all have light tanks they would like to keep out of the talks. And some U.S. Army analysts worry that the proposed Soviet redefinition is aimed at blocking the U.S. from developing lighter, more transportable, high-technology tanks. Defining combat aircraft is even tougher.

The Soviets insisted that aircraft be brought into the talks, then argued for exempting some 4,000 Russian planes because they are “solely defensive”. NATO hasn't budged from its insistence that any gun-carrying plane has offensive capability. The dispute over that issue, according to one U.S. official, is a “potential treaty stopper, ” and only President Bush and Soviet leader Mikhail Gorbachev may be able to resolve it. Accounting problems raise more knotty issues.

Greece and Turkey, for example, are suspected of overstating their arsenals in hopes that they can emerge from the arms-reduction treaty with large remaining forces to deter each other. Other nations aren't sure how many weapons they have in their own arsenals.
“It's just going to be sloppy, both on our side and theirs the Warsaw Pact 's,” says one NATO analyst.

So far, neither the Bush administration nor arms-control experts in Congress seem moved by arguments that these problems may take more time to thrash out than President Bush has allowed. They argue that the bigger danger would be that the West would delay action so long that the Soviets might back away from the current conciliatory attitude. “So what if you miss 50 tanks somewhere?” asks Rep. Norman Dicks Wash., a member of the House group that visited the talks in Vienna. “The bottom line is that if we can get that Warsaw Pact superiority brought down to parity, we ought to keep pressing ahead as quickly as possible. I worry more about things becoming so unraveled on the other side that they might become unable to negotiate.”

1. a) Mr. Bush b) Vienna c) NATO

2. a) Whenever b) Such as c) But

3. a) contend b) keep out c) miss

Correct answers: 1.a, 2.c, 3.a

10.2. Second Article

The Manville Personal Injury Settlement Trust said it is considering several ways to ease a liquidity crunch that could include the sale of Manville Corp. to a third party. In a filing with the Securities and Exchange Commission, the majority holder of Manville acknowledged that the cash portion of its initial funding of $765 million will be depleted next year, and that alternative sources of funds will be necessary to meet its obligations. **(1)**, which was created as part of Manville's bankruptcy-law reorganization to compensate victims of asbestos-related diseases, ultimately expects to receive $2.5 billion from Manville, but its cash flow from investments has so far lagged behind its payments to victims.

Spokespersons for both the trust and the company refused to comment on whether any talks with a possible acquirer of Manville had actually taken place. The trust is considering a sale of its Manville holdings, but Manville has the right of first refusal on any sales of its stock held by the trust.

Manville, a forest and building products concern, has offered to pay the trust $500 million for a majority of Manville's convertible preferred stock. Manville and the trust are discussing the offer, but no decision has been made. The filing also said the trust is considering a sale of Manville securities in the open market; an extraordinary dividend on the common stock; or a recapitalization of Manville.

The Soviet Union's jobless rate is soaring to 27% in some areas, Pravda said. It said the situation is caused by efforts to streamline bloated factory payrolls. Unemployment has reached 27.6% in Azerbaijan, 25.7% in Tadzhikistan, 22.8% in Uzbekistan, 18.8% in Turkmenia, 18% in Armenia and 16.3% in Kirgizia, the Communist Party newspaper said. All are non-Russian republics along the southern border of the Soviet Union, and all but Kirgizia have reported rioting in the past six months. The newspaper said it is past time for the Soviet Union to create unemployment insurance and retraining programs like those of the West.

Pravda gave no estimate for overall unemployment but said an “Association of the Unemployed” has cropped up that says the number of jobless is 23 million Soviets, or 17% of the workforce. An 11-week dispute involving Australia's 1,640 domestic pilots has slashed airline earnings and crippled much of the continent's tourist industry. “The only people who are flying are those who have to,” said Frank Moore, chairman of the Australian Tourist Industry Association. He added: “How is a travel agent going to sell a holiday when **(2.he)** can not guarantee a return flight?” Transport giant TNT, which owns half of one of the country's two major domestic carriers, said the cost of the dispute had been heavy, cutting TNT's profits 70% to $12 million in the three months to Sept. 30.

Brazilian financier Naji Nahas, who was arrested on Monday after 102 days in

hiding, is likely to be interrogated next week by the Brazilian judiciary. Mr. Nahas, who single-handedly provoked a one-day closure of Brazil's stock markets in June when he failed to honor a debt of $31.1 million owed to his brokers , yesterday blamed his predicament on the president of the Sao Paulo stock exchange; a few days before Mr. Nahas' failure, the exchange raised the required margin on stock-margin transactions.

China's parliament ousted two Hong Kong residents from a panel drafting a new constitution for the colony. The two, Szeto Wah and Martin Lee, were deemed unfit because they had condemned China's crackdown on its pro-democracy movement. The committee is formulating Hong Kong's constitution for when it reverts to Chinese control in 1997, and Chinese lawmakers said the two can only return **(3)** they "abandon their antagonistic stand against the Chinese government and their attempt to nullify the Sino-British joint declaration on Hong Kong."

1. a) The offer	b) The trust	c) A travel agent
2. a) the trust	b) a travel agent	c) overall unemployment
3. a) if	b) specifically	c) but

Correct answers: 1.b, 2.b, 3.a

10.3. Third Article

Banca Nazionale del Lavoro said **(1)** potential losses from lending to Iraq could reach 1.175 trillion lire ($872 million), marking the bank's first quantification of potential costs of unauthorized lending by its Atlanta branch. **(2)** previously reported that its Georgia branch had taken on loan commitments topping $3 billion without the Rome-based management's approval.

State-owned BNL, Italy's largest bank, has filed charges against the branch's former manager, Christopher Drogoul, and a former branch vice president, alleging fraud and breach of their fiduciary duties. BNL also said that its board had approved "after an in-depth discussion," a letter to the Bank of Italy outlining measures the state-owned bank has taken or plans to take to improve controls on its foreign branches. The central bank had ordered BNL to come up with a suitable program by yesterday.

Bank of Italy has also ordered BNL to shore up **(3.its)** capital base to account for potential foreign loan losses, and the Rome bank has outlined a 3 trillion

lire capital-raising operation. BNL was unable to elaborate on what measures were planned by the bank to improve controls on its branches abroad.

1. a) their b) its c) she

2. a) the Bank of Italy b) Rome c) BNL

3. a) Bank of Italy b) Rome bank c) BNL

Correct answers: 1.b, 2.c, 3.c

10.4. Fourth Article

James River Corp., Richmond, Va., said it acquired the tissue operations of Buhrmann-Tetterode N.V. of the Netherlands for about $77 million. The Dutch unit, known as Celtona B.V., is a leading maker of consumer and away-from-home tissue products for the Benelux region.

(1) the acquisition **(2)** production assets of Invercon Papermils, a maker of household tissue products for the U.K. and Ireland. The combined operations had 1988 revenue of about $100 million.

James River, a maker of pulp, paper and plastic products, already has interests in tissue businesses in France, Spain, Italy and Turkey. The company said **(3)** plans to form European ventures with Italian and Finnish companies. The Celtona operations would become part of those ventures.

1. a) As soon as b) But c) In addition,

2. a) acquired b) includes c) would become

3. a) James River Corp. b) The Celtona c) The combined operations

Correct answers: 1.c, 2.b, 3.a

10.5. Fifth Article

Vernon E. Jordan was elected to the board of this transportation services concern. **(1)** has served as executive director of the United Negro College Fund, director of the Voter Education Project of the Southern Regional

Council and attorney-consultant to the U.S. Office of Economic Opportunity. **(2.His)** election **(3)** Ryder 's board to 14 members.

1. a) Ryder’s board b) this transportation services concern c) Mr. Jordan

2. a) Vernon E. Jordan b) Ryder’s c) Ryder’s board

3. a) has served b) was elected c) increases

Correct answers: 1.c, 2.a, 3.c

11. Annex 3: Discourse span annotator raw code

```
import json
import itertools
import sys
import os
import logging
import argparse
import time


def get_arguments(argv):
    parser = argparse.ArgumentParser(description='Discourse marker annotator')
    parser.add_argument('articles_folder', metavar='ARTICLES_FOLDER',
                        help='where to get the articles')
    parser.add_argument('output_folder', metavar='OUTPUT_FOLDER',
                        help='output file in json')

    parser.add_argument('-v', '--verbose', action='store_true', default=False,
                        help='show info messages')
    parser.add_argument('-d', '--debug', action='store_true', default=False,
                        help='show debug messages')
    args = parser.parse_args(argv)
    return args


def bin_config(get_arg_func):
    # get arguments
    args = get_arg_func(sys.argv[1:])

    # set logger
    logger = logging.getLogger()
    if args.debug:
        logger.setLevel(logging.DEBUG)
    elif args.verbose:
        logger.setLevel(logging.INFO)
    else:
        logger.setLevel(logging.ERROR)

    formatter = logging.Formatter('[%(levelname)s][%(name)s] %(message)s')
    try:
        if not os.path.isdir(args.output_folder):
            os.mkdir(args.output_folder)
        fpath = os.path.join(args.output_folder, 'log')
    except:
        fpath = 'log'
    fileHandler = logging.FileHandler(fpath)
    fileHandler.setFormatter(formatter)
    logger.addHandler(fileHandler)

    consoleHandler = logging.StreamHandler()
```

```
    consoleHandler.setFormatter(formatter)
    logger.addHandler(consoleHandler)
    return args


def main():
    new_docs = {}

    fnames = [f for f in os.listdir(args.articles_folder) if f.endswith(".json")]
    unfiltered_json = json.load(open('unfiltered_discourse_markers.json'))
    marker_list = unfiltered_json['connectives'].keys()
    t_start = time.time()
    for fn in fnames:
        fpath = os.path.join(args.articles_folder, fn)
        logging.info("loading {}...".format(fpath))
        doc = json.load(open(fpath, "r"))
        did = fn.split(".")[0]
        doc['discourse'] = []

        # go through all the sentences in the document
        for i, sentence in enumerate(doc["sentences"]):

            comb = [sentence['tokens'][r:s] for r, s in
itertools.combinations(range(len(sentence['tokens'])+1), 2)]
            comb = [" ".join([y['word'] for y in x]) for x in comb]
            for c in comb:
                if c in marker_list:
                    doc['discourse'].append({
                        'connective': c,
                        'type': unfiltered_json['connectives'][c],
                        'arg1': (i-1) if i > 0 else i,
                        'arg2': i
                    })
                    print doc['discourse']


        # dump questions
        fpath = os.path.join(args.output_folder, "{}.json".format(did))
        logging.info("dumping {}...".format(fpath))
        json.dump(doc, open(fpath, "w"))

    logging.info("process questions: {} s".format(time.time()-t_start))


if __name__ == "__main__":
    args = bin_config(get_arguments)
    main()
```

12. Annex 4: Baselines raw code

```
import os
import sys
import pandas as pd
import numpy as np
import csv
import json
import cPickle as pkl
import re
from nltk.tokenize import sent_tokenize
from collections import defaultdict
from itertools import chain


# def get_w(words, w):
    # if w.lower() in words.index.values:
        # return np.array(words.loc[w.lower()])
    # else:
        # return -1


def get_sentence_embedding(sentence, option):
    new_sen = sentence + " " + option
    embeddings = []
    missing_emb = 0
    total_emb = 0
    for w in new_sen.split(" "):
        # emb = get_w(words, w)
        emb = words[w.lower()] if w.lower() in words else None
        if emb is not None:
            embeddings.append(emb)
        else:
            missing_emb += 1
        total_emb += 1
    if len(embeddings) == 0:
        dim = words[words.keys()[0]].shape[0]
        sent_emb = np.random.uniform(low=-1.0/dim, high=1.0/dim, size=dim)
    else:
        word = np.sum(embeddings, axis=0)
        sent_emb = (word / len(embeddings))

    return sent_emb, missing_emb, total_emb


def get_similarity(embed_one, embed_two):
    return (np.dot(embed_one, embed_two) / (np.linalg.norm(embed_one) *
np.linalg.norm(embed_two)))


def print_solutions(solutions, correct, total, missing_emb, total_emb):
```

```
    for x in solutions:
        print "Q" + str(x[0]) + ": " + x[1] + " [Similarity: " + str(x[2]) + "]"
    print "Correct answers: " + str(correct) + " Total questions: " + str(total)
    print "Missing embeddings: " + str(missing_emb) + " Total embeddings: " +
str(total_emb)


def load_glove(fpath):
    words = {}
    with open(fpath, 'r') as fr:
        for line in fr:
            line = line.rstrip("\n")
            sp = line.split(" ")
            emb = [float(sp[i]) for i in range(1, len(sp))]
            assert len(emb) == 300
            words[sp[0]] = np.array(emb, dtype=np.float32)
    return words


def process_one_file(fpath):
    article = json.load(open(fpath, "r"))
    sentences = sent_tokenize(article["question_text"])
    solutions = []
    correct, missing_emb, total_emb = 0, 0, 0
    correct_per_type = defaultdict(int)
    total_per_type = defaultdict(int)

    question_num = 1
    for i, sent in enumerate(sentences):
        while True:
            question_str = "__({})__".format(question_num)
            if question_str not in sent:
                break

            sent = re.sub("\_\_\(\d+\)\_\_ ", "", sent, 1)

            prev_emb, prev_me, prev_te = get_sentence_embedding(sentences[i-1], "")
if i-1 >= 0 else (None, 0, 0)
            next_emb, next_me, next_te = get_sentence_embedding(sentences[i+1], "")
if i+1 < len(sentences) else (None, 0, 0)
            missing_emb += prev_me + next_me
            total_emb += prev_te + next_te

            max_similarity = 0.
            max_option = ""
            max_ind = 0
            for j, opt in enumerate(article["options"][question_num-1]):
                cur_emb, cur_me, cur_te = get_sentence_embedding(sentences[i], opt)
                missing_emb += cur_me
                total_emb += cur_te
                sim = 0
                if prev_emb is not None:
                    sim += get_similarity(prev_emb, cur_emb)
                if next_emb is not None:
```

```
                    sim += get_similarity(next_emb, cur_emb)
                if sim > max_similarity:
                    max_similarity, max_option, max_ind = sim, opt, j

            solutions.append((question_num, max_option, max_similarity))
            if max_ind == article["answers"][question_num-1]:
                correct += 1
                correct_per_type[int(article["question_types"][question_num-1])] +=
1

            total_per_type[int(article["question_types"][question_num-1])] += 1
            question_num += 1

    # I-Ta: this part is very slow, so I re-write it.
    #       I keep it here incase you wanna debug it.
    #       you can compare the speed of these two methods.
    #       Also, the answer is not always in the zero index
    #       we might shuffle it.
    # ToDo: remove below
    # for sInd, s in enumerate(sentences):
        # print sInd
        # while "__(" in s:
            # start = s.find("__(")
            # end = s.find(")__")
            # offset = end - start
            # qString = ""
            # for o in range(offset - 3):
                # qString += s[s.find("__(") + o + 3]
            # questionNumber = int(qString)
            # s = s.replace("__(" + qString + ")__", "")

            # prev_emb = None
            # next_emb = None

            # if sInd > 0:
                # prev_emb = get_sentence_embedding(sentences[sInd-1], "")
            # if sInd < len(sentences) - 1:
                # next_emb = get_sentence_embedding(sentences[sInd+1], "")

            # max_similarity = 0.
            # max_option = ""
            # max_ind = 0

            # for oInd, option in enumerate(article["options"][questionNumber-1]):
                # current_emb = get_sentence_embedding(sentences[sInd], option)
                # sim = 0.
                # if prev_emb is not None:
                    # sim += get_similarity(prev_emb, current_emb)
                # if next_emb is not None:
                    # sim += get_similarity(next_emb, current_emb)

                # if sim > max_similarity:
                    # max_similarity, max_option, max_ind = sim, option, oInd
```

```
            # solutions.append((questionNumber, max_option, max_similarity))
            # if max_ind == 0:
                # correct += 1
    print_solutions(solutions, correct, len(solutions), missing_emb, total_emb)
    return correct, len(solutions), missing_emb, total_emb, correct_per_type,
total_per_type


def main():
    #files = [f for f in os.listdir(question_folder) if f.endswith(".json")]
    files = json.load(open('annotated_docs.json'))[question_folder]

    total_correct, total_questions, missing_emb, total_emb = 0, 0, 0, 0
    correct_per_type = defaultdict(int)
    total_per_type = defaultdict(int)
    for f in files:
        did = ".".join(f.split(".")[:-1])
        fpath = os.path.join(question_folder, f)

        cnt_correct, cnt_questions, cnt_missing, cnt_total, cntc_per_type,
cntt_per_type = process_one_file(fpath)
        total_correct += cnt_correct
        total_questions += cnt_questions
        missing_emb += cnt_missing
        total_emb += cnt_total

        for k, v in cntc_per_type.items():
            if k == 0:
                correct_per_type['DISCOURSE'] += v
            elif k == 1:
                correct_per_type['ENTITY'] += v
                correct_per_type['ENTITY_PRED'] += v
            elif k == 2:
                correct_per_type['ENTITY'] += v
                correct_per_type['ENTITY_PRED'] += v
            else:
                correct_per_type['PRED'] += v
                correct_per_type['ENTITY_PRED'] += v
        for k, v in cntt_per_type.items():
            if k == 0:
                total_per_type['DISCOURSE'] += v
            elif k == 1:
                total_per_type['ENTITY'] += v
                total_per_type['ENTITY_PRED'] += v
            elif k == 2:
                total_per_type['ENTITY'] += v
                total_per_type['ENTITY_PRED'] += v
            else:
                total_per_type['PRED'] += v
                total_per_type['ENTITY_PRED'] += v

    print("\n===========================================")
    print("total_correct={}, total_questions={}".format(total_correct,
total_questions))
```

```
    print("accuracy={}".format(float(total_correct) / total_questions))
    print("missing_embeddings={}, total_embeddings={}".format(missing_emb,
total_emb))
    print("missing_rate={}".format(float(missing_emb) / total_emb))
    for k, v in correct_per_type.items():
        print("type={}, total_correct={}, total_questions={},
accuracy={}".format(k,v,total_per_type[k],float(v)/total_per_type[k]))
    print("============================================")


if __name__ == "__main__":
    glove_file = sys.argv[1]
    question_folder = sys.argv[2]
    #question_file = sys.argv[3]
    #words = pd.read_table(glove_file, sep=" ", index_col=0, header=None,
quoting=csv.QUOTE_NONE)
    words = load_glove(glove_file)
    main()
```

Glove model code

```
import os
import sys
import pandas as pd
import numpy as np
import csv
import json
import cPickle as pkl
import re
from nltk.tokenize import sent_tokenize
from gensim.models import KeyedVectors


def get_sentence_embedding(sentence, option):
    new_sen = sentence + " " + option
    embeddings = []
    missing_emb = 0
    total_emb = 0
    for w in new_sen.split(" "):
        emb = model[w] if w in model else None
        if emb is not None:
            embeddings.append(emb)
        else:
            missing_emb += 1
        total_emb += 1
    if len(embeddings) == 0:
        dim = len(model[model.index2word[0]])
        sent_emb = np.random.uniform(low=-1.0/dim, high=1.0/dim, size=dim)
    else:
        word = np.sum(embeddings, axis=0)
        sent_emb = (word / len(embeddings))

    return sent_emb, missing_emb, total_emb
```

```
def get_similarity(embed_one, embed_two):
    return (np.dot(embed_one, embed_two) / (np.linalg.norm(embed_one) *
np.linalg.norm(embed_two)))


def print_solutions(solutions, correct, total, missing_emb, total_emb):
    for x in solutions:
        print "Q" + str(x[0]) + ": " + x[1] + " [Similarity: " + str(x[2]) + "]"
    print "Correct answers: " + str(correct) + " Total questions: " + str(total)
    print "Missing embeddings: " + str(missing_emb) + " Total embeddings: " +
str(total_emb)


def process_one_file(fpath):
    article = json.load(open(fpath, "r"))
    sentences = sent_tokenize(article["question_text"])
    solutions = []
    correct = 0
    missing_emb = 0
    total_emb = 0
    correct_per_type = defaultdict(int)
    total_per_type = defaultdict(int)

    question_num = 1
    for i, sent in enumerate(sentences):
        while True:
            question_str = "__({})__".format(question_num)
            if question_str not in sent:
                break

            sent = re.sub("\_\_\(\d+\)\_\_ ", "", sent, 1)

            prev_emb, prev_me, prev_te = get_sentence_embedding(sentences[i-1], "")
if i-1 >= 0 else (None, 0, 0)
            next_emb, next_me, next_te = get_sentence_embedding(sentences[i+1], "")
if i+1 < len(sentences) else (None, 0, 0)
            missing_emb += prev_me + next_me
            total_emb += prev_te + next_te

            max_similarity = 0.
            max_option = ""
            max_ind = 0
            for j, opt in enumerate(article["options"][question_num-1]):
                cur_emb, cur_me, cur_te = get_sentence_embedding(sentences[i], opt)
                missing_emb += cur_me
                total_emb += cur_te
                sim = 0
                if prev_emb is not None:
                    sim += get_similarity(prev_emb, cur_emb)
                if next_emb is not None:
                    sim += get_similarity(next_emb, cur_emb)

                if sim > max_similarity:
                    max_similarity, max_option, max_ind = sim, opt, j
```

```
            solutions.append((question_num, max_option, max_similarity))
            if max_ind == article["answers"][question_num-1]:
                correct += 1
                correct_per_type[int(article["question_types"][question_num-1])] +=
1

            total_per_type[int(article["question_types"][question_num-1])] += 1
            question_num += 1
    print_solutions(solutions, correct, len(solutions), missing_emb, total_emb)
    return correct, len(solutions), missing_emb, total_emb, correct_per_type,
total_per_type


def main():
    files = json.load(open('annotated_docs.json'))[question_folder]

    total_correct, total_questions, missing_emb, total_emb = 0, 0, 0, 0
    correct_per_type = defaultdict(int)
    total_per_type = defaultdict(int)

    for f in files:
        did = ".".join(f.split(".")[:-1])
        fpath = os.path.join(question_folder, f)

        cnt_correct, cnt_questions, cnt_missing, cnt_total, cntc_per_type,
cntt_per_type = process_one_file(fpath)
        total_correct += cnt_correct
        total_questions += cnt_questions
        missing_emb += cnt_missing
        total_emb += cnt_total

        for k, v in cntc_per_type.items():
            if k == 0:
                correct_per_type['DISCOURSE'] += v
            elif k == 1:
                correct_per_type['ENTITY'] += v
                correct_per_type['ENTITY_PRED'] += v
            elif k == 2:
                correct_per_type['ENTITY'] += v
                correct_per_type['ENTITY_PRED'] += v
            else:
                correct_per_type['PRED'] += v
                correct_per_type['ENTITY_PRED'] += v
        for k, v in cntt_per_type.items():
            if k == 0:
                total_per_type['DISCOURSE'] += v
            elif k == 1:
                total_per_type['ENTITY'] += v
                total_per_type['ENTITY_PRED'] += v
            elif k == 2:
                total_per_type['ENTITY'] += v
                total_per_type['ENTITY_PRED'] += v
            else:
```

```
                total_per_type['PRED'] += v
                total_per_type['ENTITY_PRED'] += v

    print("\n=============================================")
    print("total_correct={}, total_questions={}".format(total_correct,
total_questions))
    print("accuracy={}".format(float(total_correct) / total_questions))
    print("missing_embeddings={}, total_embeddings={}".format(missing_emb,
total_emb))
    print("missing_rate={}".format(float(missing_emb) / total_emb))
    for k, v in correct_per_type.items():
        print("type={}, total_correct={}, total_questions={},
accuracy={}".format(k,v,total_per_type[k],float(v)/total_per_type[k]))
    print("=============================================")


if __name__ == "__main__":
    word2vec_file = sys.argv[1]
    question_folder = sys.argv[2]
    model = KeyedVectors.load_word2vec_format(word2vec_file, binary=True)
    main()
```

Word2Vec library

```
import os
import sys
import pandas as pd
import numpy as np
import csv
import json
import skipthoughts
import cPickle as pkl
import re
from nltk.tokenize import sent_tokenize


def get_similarity(embed_one, embed_two):
    return (np.dot(embed_one, embed_two) / (np.linalg.norm(embed_one) *
np.linalg.norm(embed_two)))


def print_solutions(solutions, correct, total):
    for x in solutions:
        print "Q" + str(x[0]) + ": " + x[1] + " [Similarity: " + str(x[2]) + "]"
    print "Correct answers: " + str(correct) + " Total questions: " + str(total)


def process_one_file(fpath):
    article = json.load(open(fpath, "r"))
    sentences = sent_tokenize(article["question_text"])
    solutions = []
    correct = 0
    correct_per_type = defaultdict(int)
    total_per_type = defaultdict(int)
```

```
    question_num = 1
    for i, sent in enumerate(sentences):
        while True:
            question_str = "__({})__".format(question_num)
            if question_str not in sent:
                break

            prev_emb = True if i-1 >= 0 else False
            next_emb = True if i+1 < len(sentences) else False

            max_similarity = 0.
            max_option = ""
            max_ind = 0
            for j, opt in enumerate(article["options"][question_num-1]):
                sent_to_emb = []
                if prev_emb:
                    sent_to_emb.append(sentences[i-1])
                sent_to_emb.append(re.sub("\_\_\(\d+\)\_\_ ", opt, sent, 1))
                if next_emb:
                    sent_to_emb.append(sentences[i+1])
                embeddings = encoder.encode(sent_to_emb)

                sim = 0
                if prev_emb and next_emb:
                    sim += get_similarity(embeddings[0], embeddings[1])
                    sim += get_similarity(embeddings[1], embeddings[2])
                else:
                    sim += get_similarity(embeddings[0], embeddings[1])

                if sim > max_similarity:
                    max_similarity, max_option, max_ind = sim, opt, j

            solutions.append((question_num, max_option, max_similarity))
            if max_ind == article["answers"][question_num-1]:
                correct_per_type[int(article["question_types"][question_num-1])] +=
1

            total_per_type[int(article["question_types"][question_num-1])] += 1
            sent = re.sub("\_\_\(\d+\)\_\_ ", "", sent, 1)
            question_num += 1

    print_solutions(solutions, correct, len(solutions))
    return correct, len(solutions), correct_per_type, total_per_type


def main():
    files = json.load(open('../annotated_docs.json'))[question_folder]
    total_correct, total_questions = 0, 0
    correct_per_type = defaultdict(int)
    total_per_type = defaultdict(int)
    for f in files:
        did = ".".join(f.split(".")[:-1])
        fpath = os.path.join(question_folder, f)
```

```
        cnt_correct, cnt_questions, cntc_per_type, cntt_per_type =
process_one_file("../" + fpath)
        total_correct += cnt_correct
        total_questions += cnt_questions

        for k, v in cntc_per_type.items():
            if k == 0:
                correct_per_type['DISCOURSE'] += v
            elif k == 1:
                correct_per_type['ENTITY'] += v
                correct_per_type['ENTITY_PRED'] += v
            elif k == 2:
                correct_per_type['ENTITY'] += v
                correct_per_type['ENTITY_PRED'] += v
            else:
                correct_per_type['PRED'] += v
                correct_per_type['ENTITY_PRED'] += v
        for k, v in cntt_per_type.items():
            if k == 0:
                total_per_type['DISCOURSE'] += v
            elif k == 1:
                total_per_type['ENTITY'] += v
                total_per_type['ENTITY_PRED'] += v
            elif k == 2:
                total_per_type['ENTITY'] += v
                total_per_type['ENTITY_PRED'] += v
            else:
                total_per_type['PRED'] += v
                total_per_type['ENTITY_PRED'] += v

    print("\n=============================================")
    print("total_correct={}, total_questions={}".format(total_correct,
total_questions))
    print("accuracy={}".format(float(total_correct) / total_questions))
    for k, v in correct_per_type.items():
        print("type={}, total_correct={}, total_questions={},
accuracy={}".format(k,v,total_per_type[k],float(v)/total_per_type[k]))
    print("=============================================")


if __name__ == "__main__":
    question_folder = sys.argv[1]
    model = skipthoughts.load_model()
    encoder = skipthoughts.Encoder(model)
    main()
```

Skip-Thoughts library

13. Annex 5: Answer generation raw code

```
import sys
import os
import logging
import argparse
import json
import time
import nltk
import numpy as np
from nltk.corpus import wordnet as wn
import pattern.en as en
from gensim.models import KeyedVectors
from pattern.en import INFINITIVE, PAST, PRESENT, INDICATIVE, PROGRESSIVE, SG


QTYPE_DISCOURSE_MARKER = 0
QTYPE_ENTITY = 1
QTYPE_COREF_ENTITY = 2
QTYPE_COREF_PREDICATE = 3

TENSES_DICT = {
    ('infinitive', None, None, None, None): dict(tense=INFINITIVE),
    ('past', None, None, 'indicative', 'imperfective'): dict(tense=PAST),
    ('present', None, None, 'indicative', 'progressive'):
dict(tense=PRESENT,mood=INDICATIVE,aspect=PROGRESSIVE),
    ('past', None, None, 'indicative', 'progressive'):
dict(tense=PAST,mood=INDICATIVE,aspect=PROGRESSIVE),
    ('present', 1, 'singular', 'indicative', 'imperfective'):
dict(tense=PRESENT,person=1,number=SG,mood=INDICATIVE),
    ('present', 3, 'singular', 'indicative', 'imperfective'):
dict(tense=PRESENT,person=3,number=SG,mood=INDICATIVE)
}


def get_arguments(argv):
    parser = argparse.ArgumentParser(description='Add options to the entity
questions')
    parser.add_argument('question_folder', metavar='QUESTION_FOLDER',
                        help='config file in json')
    parser.add_argument('output_folder', metavar='OUTPUT_FOLDER',
                        help='output file in json')
    parser.add_argument('word_embedding_model', metavar='WORD_EMBEDDING_MODEL',
                        help='word embedding model')

    parser.add_argument('-v', '--verbose', action='store_true', default=False,
                        help='show info messages')
    parser.add_argument('-d', '--debug', action='store_true', default=False,
                        help='show debug messages')
    args = parser.parse_args(argv)
    return args
```

```
def bin_config(get_arg_func):
    # get arguments
    args = get_arg_func(sys.argv[1:])

    # set logger
    logger = logging.getLogger()
    if args.debug:
        logger.setLevel(logging.DEBUG)
    elif args.verbose:
        logger.setLevel(logging.INFO)
    else:
        logger.setLevel(logging.ERROR)

    formatter = logging.Formatter('[%(levelname)s][%(name)s] %(message)s')
    try:
        if not os.path.isdir(args.output_folder):
            os.mkdir(args.output_folder)
        fpath = os.path.join(args.output_folder, 'log')
    except:
        fpath = 'log'
    fileHandler = logging.FileHandler(fpath)
    fileHandler.setFormatter(formatter)
    logger.addHandler(fileHandler)

    consoleHandler = logging.StreamHandler()
    consoleHandler.setFormatter(formatter)
    logger.addHandler(consoleHandler)
    return args


def sorting_siblings(elem, synset_embedding):
    elem_embedding = model[elem.lemma_names()[0]] if elem.lemma_names()[0] in model
else None
    if elem_embedding is None or synset_embedding is None:
        return 1
    else:
        return 1 - (np.dot(elem_embedding, synset_embedding) /
(np.linalg.norm(elem_embedding) * np.linalg.norm(synset_embedding)))


def get_similar_options(word, number):
    embedding = model[word] if word in model else None
    if embedding is not None:
        return [x[0] for x in model.wv.similar_by_vector(embedding,
topn=number+1)[1:]]
    else:
        return ["" for i in range(number)]


def create_options(answer_text):
    options = [answer_text]
    try:
        tense = en.tenses(answer_text)[0]
```

```
        lemma = en.lemma(answer_text)
        synset = wn.synset(lemma+".v.01")
        synset_embedding = model[synset.lemma_names()[0]] if
synset.lemma_names()[0] in model else None
        parent = synset.hypernyms()[0]

        all_siblings = parent.hyponyms()
        all_siblings = sorted(all_siblings, key=lambda elem:
sorting_siblings(elem,synset_embedding))
        for s in all_siblings:
            if len(options) == 5:
                break
            option = en.conjugate(s.lemma_names()[0],**TENSES_DICT[tense])
            if option not in options:
                options.append(option)

        if len(options) < 5:
            options = [answer_text]
            options.extend(get_similar_options(answer_text, 4))
        return sorted(options), sorted(options).index(answer_text)
    except:
        options.extend(get_similar_options(answer_text, 4))
        return sorted(options), sorted(options).index(answer_text)


def main():
    new_docs = {}

    fnames = [f for f in os.listdir(args.question_folder) if f.endswith(".json")]
    t_start = time.time()
    for fn in fnames:
        # load questions
        fpath = os.path.join(args.question_folder, fn)
        logging.info("loading {}...".format(fpath))
        doc = json.load(open(fpath, "r"))
        did = fn.split(".")[0]

        # go through all the questions in the document
        for i, (qtype, ans_idx, options) in enumerate(zip(doc["question_types"],
doc["answers"], doc["options"])):
            # skip types other than entity questions
            if qtype == QTYPE_DISCOURSE_MARKER or qtype == QTYPE_COREF_ENTITY or
qtype == QTYPE_ENTITY:
                continue

            gold_answer_text = options[ans_idx]
            doc["options"][i], doc["answers"][i] = create_options(gold_answer_text)


        # dump questions
        fpath = os.path.join(args.output_folder, "{}.json".format(did))
        logging.info("dumping {}...".format(fpath))
        json.dump(doc, open(fpath, "w"))
```

```
    logging.info("process questions: {} s".format(time.time()-t_start))


if __name__ == "__main__":
    args = bin_config(get_arguments)
    model = KeyedVectors.load_word2vec_format(args.word_embedding_model,
binary=True)
    main()
```

Predicate type questions

```
import sys
import os
import logging
import argparse
import json
import time
import nltk
import numpy as np
from gensim.models import KeyedVectors



QTYPE_DISCOURSE_MARKER = 0
QTYPE_ENTITY = 1
QTYPE_COREF_ENTITY = 2
QTYPE_COREF_PREDICATE = 3


def get_arguments(argv):
    parser = argparse.ArgumentParser(description='Add options to the entity
questions')
    parser.add_argument('question_folder', metavar='QUESTION_FOLDER',
                        help='config file in json')
    parser.add_argument('output_folder', metavar='OUTPUT_FOLDER',
                        help='output file in json')
    parser.add_argument('word_embedding_model', metavar='WORD_EMBEDDING_MODEL',
                        help='word embedding model')

    parser.add_argument('-v', '--verbose', action='store_true', default=False,
                        help='show info messages')
    parser.add_argument('-d', '--debug', action='store_true', default=False,
                        help='show debug messages')
    args = parser.parse_args(argv)
    return args


def bin_config(get_arg_func):
    # get arguments
    args = get_arg_func(sys.argv[1:])

    # set logger
    logger = logging.getLogger()
    if args.debug:
        logger.setLevel(logging.DEBUG)
```

```
    elif args.verbose:
        logger.setLevel(logging.INFO)
    else:
        logger.setLevel(logging.ERROR)

    formatter = logging.Formatter('[%(levelname)s][%(name)s] %(message)s')
    try:
        if not os.path.isdir(args.output_folder):
            os.mkdir(args.output_folder)
        fpath = os.path.join(args.output_folder, 'log')
    except:
        fpath = 'log'
    fileHandler = logging.FileHandler(fpath)
    fileHandler.setFormatter(formatter)
    logger.addHandler(fileHandler)

    consoleHandler = logging.StreamHandler()
    consoleHandler.setFormatter(formatter)
    logger.addHandler(consoleHandler)
    return args


def get_embedding(sentence):
    embeddings = []
    for s in sentence.split(' '):
        emb = model[s] if s in model else None
        if emb is not None:
            embeddings.append(emb)

    if len(embeddings) == 0:
        dim = len(model[model.index2word[0]])
        sent_emb = np.random.uniform(low=-1.0/dim, high=1.0/dim, size=dim)
    else:
        sent_emb = (np.sum(embeddings, axis=0) / len(embeddings))
    return sent_emb


def get_sim(a, b):
    a_emb = get_embedding(a)
    b_emb = get_embedding(b)
    return np.dot(a_emb, b_emb) / (np.linalg.norm(a_emb) * np.linalg.norm(b_emb))


def create_options(answer_text):
    answer_text = answer_text.lower()
    options = [answer_text]
    candidates = []
    if answer_text in disc_mark['connectives']:
        type_candidates = []
        for s in disc_mark['connectives'][answer_text]:
            type_candidates =
list(set(disc_mark['senses'][s]).union(type_candidates))
```

```
        candidates = [(dc,get_sim(dc,answer_text)) for dc in
disc_mark['connectives'] if dc != answer_text and dc not in type_candidates]
        candidates = sorted(candidates, key=lambda a: a[1], reverse=True)
    else:
        candidates = sorted([(dc,get_sim(dc,answer_text)) for dc in
disc_mark['connectives'] if dc != answer_text], key=lambda a: a[1], reverse=True)

    for c in candidates:
        if len(options) == 5:
            break
        if c[0] not in options:
            options.append(c[0])

    return sorted(options), sorted(options).index(answer_text)


def main():
    new_docs = {}

    fnames = [f for f in os.listdir(args.question_folder) if f.endswith(".json")]
    t_start = time.time()
    for fn in fnames:
        # load questions
        fpath = os.path.join(args.question_folder, fn)
        logging.info("loading {}...".format(fpath))
        doc = json.load(open(fpath, "r"))
        did = fn.split(".")[0]

        # go through all the questions in the document
        for i, (qtype, ans_idx, options) in enumerate(zip(doc["question_types"],
doc["answers"], doc["options"])):
            # skip types other than entity questions
            if qtype == QTYPE_COREF_PREDICATE or qtype == QTYPE_COREF_ENTITY or
qtype == QTYPE_ENTITY:
                continue

            gold_answer_text = options[ans_idx]
            doc["options"][i], doc["answers"][i] = create_options(gold_answer_text)


        # dump questions
        fpath = os.path.join(args.output_folder, "{}.json".format(did))
        logging.info("dumping {}...".format(fpath))
        json.dump(doc, open(fpath, "w"))

    logging.info("process questions: {} s".format(time.time()-t_start))


if __name__ == "__main__":
    args = bin_config(get_arguments)
    model = KeyedVectors.load_word2vec_format(args.word_embedding_model,
binary=True)
    disc_mark = json.load(open('discourse_markers.json'))
    main()
```

Discourse type questions