Find duplicate questions on StackOverflow by their embeddings

In this assignment you will learn how to calculate a similarity for pieces of text. Using this approach you will know how to find duplicate questions from StackOverflow (https://stackoverflow.com).

Libraries

In this task you will you will need the following libraries:

- StarSpace (https://github.com/facebookresearch/StarSpace) — a general-purpose model for efficient learning of entity embeddings from Facebook
- Gensim (https://radimrehurek.com/gensim/) — a tool for solving various NLP-related tasks (topic modeling, text representation, ...)
- Numpy (http://www.numpy.org) — a package for scientific computing.
- scikit-learn (http://scikit-learn.org/stable/index.html) — a tool for data mining and data analysis.
- Nltk (http://www.nltk.org) — a platform to work with human language data.

Data

The following cell will download all data required for this assignment into the folder `week3/data` .

In [1]:

```python
import sys
sys.path.append("..")
from common.download_utils import download_week3_resources

download_week3_resources()
```

```
File data\train.tsv is already downloaded.
File data\validation.tsv is already downloaded.
File data\test.tsv is already downloaded.
File data\test_embeddings.tsv is already downloaded.
Downloading GoogleNews-vectors-negative300.bin.gz (1.5G) for you, it will take a while...

GoogleNews-vectors-negative300.bin.gz: 100%        1.65G/1.65G [05:22<00:00, 5.11MB/s]
```

Grading

We will create a grader instace below and use it to collect your answers. Note that these outputs will be stored locally inside grader and will be uploaded to platform only after running submiting function in the last part of this assignment. If you want to make partial submission, you can run that cell any time you want.

In [2]:

```python
from grader import Grader
```

In [3]:

```python
grader = Grader()
```

Word embedding

To solve the problem, you will use two different models of embeddings:

- Pre-trained word vectors (https://code.google.com/archive/p/word2vec/) from Google which were trained on a part of Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases. `GoogleNews-vectors-negative300.bin.gz` will be downloaded in `download_week3_resources()` .
- Representations using StarSpace on StackOverflow data sample. You will need to train them from scratch.

It's always easier to start with pre-trained embeddings. Unpack the pre-trained Goggle's vectors and upload them using the function KeyedVectors.load_word2vec_format (https://radimrehurek.com/gensim/models/keyedvectors.html) from gensim library with the parameter *binary=True*. If the size of the embeddings is larger than the avaliable memory, you could load only a part of the embeddings by defining the parameter *limit* (recommended: 500000).

In [4]:

```python
import gensim
```

In [5]:

```python
######### YOUR CODE HERE #############
from gensim.models import KeyedVectors
wv_embeddings = KeyedVectors.load_word2vec_format('./GoogleNews-vectors-negative300.bin', binary=True)
```

How to work with Google's word2vec embeddings?

Once you have loaded the representations, make sure you can access them. First, you can check if the loaded embeddings contain a word:

```
'word' in wv_embeddings
```

Second, to get the corresponding embedding you can use the square brackets:

```
wv_embeddings['word']
```

Checking that the embeddings are correct

To prevent any errors during the first stage, we can check that the loaded embeddings are correct. You can call the function *check_embeddings*, implemented below, which runs 3 tests:

1. Find the most similar word for provided "positive" and "negative" words.
2. Find which word from the given list doesn't go with the others.
3. Find the most similar word for the provided one.

In the right case the function will return the string *These embeddings look good*. Othervise, you need to validate the previous steps.

In [6]:

```
def check_embeddings(embeddings):
    error_text = "Something wrong with your embeddings ('%s test isn't correct)."
    most_similar = embeddings.most_similar(positive=['woman', 'king'], negative=['man'])
    if len(most_similar) < 1 or most_similar[0][0] != 'queen':
        return error_text % "Most similar"

    doesnt_match = embeddings.doesnt_match(['breakfast', 'cereal', 'dinner', 'lunch'])
    if doesnt_match != 'cereal':
        return error_text % "Doesn't match"

    most_similar_to_given = embeddings.most_similar_to_given('music', ['water', 'sound', 'backpack', 'mouse'])
    if most_similar_to_given != 'sound':
        return error_text % "Most similar to given"

    return "These embeddings look good."
```

In [7]:

```
print(check_embeddings(wv_embeddings))
```

```
These embeddings look good.

C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\gensim\models\keyedvectors.py:877: FutureWarning: arrays to s
tack must be passed as a "sequence" type such as list or tuple. Support for non-sequence iterables such as generators i
s deprecated as of NumPy 1.16 and will raise an error in the future.
  vectors = vstack(self.word_vec(word, use_norm=True) for word in used_words).astype(REAL)
```

From word to text embeddings

Task 1 (Question2Vec). Usually, we have word-based embeddings, but for the task we need to create a representation for the whole question. It could be done in different ways. In our case we will use a **mean** of all word vectors in the question. Now you need to implement the function *question_to_vec*, which calculates the question representation described above. This function should work with the input text as is without any preprocessing.

Note that there could be words without the corresponding embeddings. In this case, you can just skip these words and don't take them into account during calculating the result. If the question doesn't contain any known word with embedding, the function should return a zero vector.

In [8]:

```
import numpy as np
```

In [9]:

```
def question_to_vec(question, embeddings, dim=300):
    """
        question: a string
        embeddings: dict where the key is a word and a value is its' embedding
        dim: size of the representation

        result: vector representation for the question
    """
    ######################################
    ######### YOUR CODE HERE #############
    ######################################
    result = np.zeros(dim)
    words = question.split(' ')
    valid_count = 0
    for word in words:
        if word in embeddings:
            result += embeddings[word][:dim]
            valid_count += 1

    result = result/valid_count if valid_count != 0 else result
    return result
```

To check the basic correctness of your implementation, run the function *question_to_vec_tests*.

In [10]:

```python
def question_to_vec_tests():
    if (np.zeros(300) != question_to_vec('', wv_embeddings)).any():
        return "You need to return zero vector for empty question."
    if (np.zeros(300) != question_to_vec('thereisnosuchword', wv_embeddings)).any():
        return "You need to return zero vector for the question, which consists only unknown words."
    if (wv_embeddings['word'] != question_to_vec('word', wv_embeddings)).any():
        return "You need to check the corectness of your function."
    if ((wv_embeddings['I'] + wv_embeddings['am']) / 2 != question_to_vec('I am', wv_embeddings)).any():
        return "Your function should calculate a mean of word vectors."
    if (wv_embeddings['word'] != question_to_vec('thereisnosuchword word', wv_embeddings)).any():
        return "You should not consider words which embeddings are unknown."
    return "Basic tests are passed."
```

In [11]:

```python
print(question_to_vec_tests())
```

```
Basic tests are passed.
```

You can submit embeddings for the questions from the file *test_embeddings.tsv* to earn the points. In this task you don't need to transform the text of a question somehow.

In [12]:

```python
import nltk
nltk.download('stopwords')
from util import array_to_string
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Xiaowei\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [13]:

```python
question2vec_result = []
for question in open('data/test_embeddings.tsv'):
    question = question.strip()
    answer = question_to_vec(question, wv_embeddings)
    question2vec_result = np.append(question2vec_result, answer)

grader.submit_tag('Question2Vec', array_to_string(question2vec_result))
```

```
Current answer for task Question2Vec is: 0.019293891059027776
-0.028727213541666668
0.046056111653645836
0.08525933159722222
0.02430555555555...
```

Now we have a method to create a representation of any sentence and we are ready for the first evaluation. So, let's check how well our solution (Google's vectors + *question_to_vec*) will work.

Evaluation of text similarity

We can imagine that if we use good embeddings, the cosine similarity between the duplicate sentences should be less than for the random ones. Overall, for each pair of duplicate sentences we can generate *R* random negative examples and find out the position of the correct duplicate.

For example, we have the question *"Exceptions What really happens"* and we are sure that another question *"How does the catch keyword determine the type of exception that was thrown"* is a duplicate. But our model doesn't know it and tries to find out the best option also among questions like *"How Can I Make These Links Rotate in PHP", *, *"NSLog array description not memory address"* and *"PECL_HTTP not recognised php ubuntu". *The goal of the model is to rank all these 4 questions (1 *positive* and *R* = 3 *negative*) in the way that the correct one is in the first place.

However, it is unnatural to count on that the best candidate will be always in the first place. So let us consider the place of the best candidate in the sorted list of candidates and formulate a metric based on it. We can fix some *K* — a reasonalble number of top-ranked elements and *N* — a number of queries (size of the sample).

Hits@K

The first simple metric will be a number of correct hits for some *K*:

$$\text{Hits@K} = \frac{1}{N}\sum_{i=1}^{N} [dup_i \in topK(q_i)]$$

where q_i is the i-th query, dup_i is its duplicate, $topK(q_i)$ is the top K elements of the ranked sentences provided by our model and the operation $[dup_i \in topK(q_i)]$ equals 1 if the condition is true and 0 otherwise (more details about this operation could be found here (https://en.wikipedia.org/wiki/Iverson_bracket)).

DCG@K

The second one is a simplified DCG metric (https://en.wikipedia.org/wiki/Discounted_cumulative_gain):

$$\text{DCG@K} = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{\log_2(1 + rank_{dup_i})} \cdot [rank_{dup_i} \le K]$$

where $rank_{dup_i}$ is a position of the duplicate in the sorted list of the nearest sentences for the query q_i. According to this metric, the model gets a higher reward for a higher position of the correct answer. If the answer does not appear in topK at all, the reward is zero.

Evaluation examples

Let's calculate the described metrics for the toy example introduced above. In this case $N = 1$ and the correct candidate for q_1 is *"How does the catch keyword determine the type of exception that was thrown"*. Consider the following ranking of the candidates:

1. *"How Can I Make These Links Rotate in PHP"*
2. *"How does the catch keyword determine the type of exception that was thrown"*
3. *"NSLog array description not memory address"*
4. *"PECL_HTTP not recognised php ubuntu"*

Using the ranking above, calculate *Hits@K* metric for *K = 1, 2, 4*:

- [K = 1] $\text{Hits@1} = \frac{1}{1}\sum_{i=1}^{1} [dup_i \in top1(q_i)] = [dup_1 \in top1(q_1)] = 0$ because the correct answer doesn't appear in the *top1* list.
- [K = 2] $\text{Hits@2} = \frac{1}{1}\sum_{i=1}^{1} [dup_i \in top2(q_i)] = [dup_1 \in top2(q_1)] = 1$ because $rank_{dup_1} = 2$.
- [K = 4] $\text{Hits@4} = \frac{1}{1}\sum_{i=1}^{1} [dup_i \in top4(q_i)] = [dup_1 \in top4(q_1)] = 1$

Using the ranking above, calculate *DCG@K* metric for *K = 1, 2, 4*:

- [K = 1] $\text{DCG@1} = \frac{1}{1}\sum_{i=1}^{1} \frac{1}{\log_2{(1+rank_{dup_i})}} \cdot [rank_{dup_i} \le 1] = \frac{1}{\log_2{(1+rank_{dup_i})}} \cdot [rank_{dup_i} \le 1] = 0$ because the correct answer doesn't appear in the top1 list.
- [K = 2] $\text{DCG@2} = \frac{1}{1}\sum_{i=1}^{1} \frac{1}{\log_2{(1+rank_{dup_i})}} \cdot [rank_{dup_i} \le 2] = \frac{1}{\log_2{3}}$, because $rank_{dup_1} = 2$.
- [K = 4] $\text{DCG@4} = \frac{1}{1}\sum_{i=1}^{1} \frac{1}{\log_2{(1+rank_{dup_i})}} \cdot [rank_{dup_i} \le 4] = \frac{1}{\log_2{3}}$.

Tasks 2 and 3 (HitsCount and DCGScore). Implement the functions *hits_count* and *dcg_score* as described above. Each function has two arguments: *dup_ranks* and *k*. *dup_ranks* is a list which contains *values of ranks* of duplicates. For example, *dup_ranks* is *[2]* for the example provided above.

In [14]:

```python
def hits_count(dup_ranks, k):
    """
        dup_ranks: list of duplicates' ranks; one rank per question; 
                   length is a number of questions which we are looking for duplicates; 
                   rank is a number from 1 to len(candidates of the question); 
                   e.g. [2, 3] means that the first duplicate has the rank 2, the second one — 3.
        k: number of top-ranked elements (k in Hits@k metric)

        result: return Hits@k value for current ranking
    """
    ######################################
    ######### YOUR CODE HERE #############
    ######################################
#     print('dup_ranks, k', dup_ranks, k)
    query_size = len(dup_ranks)
    hits = 0
    for rank in dup_ranks:
        if rank <= k:
            hits += 1
    return hits/query_size
```

Test your code on the tiny examples:

In [15]:

```
def test_hits():
    # *Evaluation example*
    # answers — dup_i
    answers = ["How does the catch keyword determine the type of exception that was thrown"]

    # candidates_ranking — the ranked sentences provided by our model
    candidates_ranking = [["How Can I Make These Links Rotate in PHP",
                           "How does the catch keyword determine the type of exception that was thrown",
                           "NSLog array description not memory address",
                           "PECL_HTTP not recognised php ubuntu"]]
    # dup_ranks — position of the dup_i in the list of ranks +1
    dup_ranks = [candidates_ranking[i].index(answers[i]) + 1 for i in range(len(answers))]

    # correct_answers — the expected values of the result for each k from 1 to 4
    correct_answers = [0, 1, 1, 1]
    for k, correct in enumerate(correct_answers, 1):
        if not np.isclose(hits_count(dup_ranks, k), correct):
            return "Check the function."

    # Other tests
    answers = ["How does the catch keyword determine the type of exception that was thrown",
               "Convert Google results object (pure js) to Python object"]

    # The first test: both duplicates on the first position in ranked list
    candidates_ranking = [["How does the catch keyword determine the type of exception that was thrown",
                           "How Can I Make These Links Rotate in PHP"],
                          ["Convert Google results object (pure js) to Python object",
                           "WPF- How to update the changes in list item of a list"]]
    dup_ranks = [candidates_ranking[i].index(answers[i]) + 1 for i in range(len(answers))]
    correct_answers = [1, 1]
    for k, correct in enumerate(correct_answers, 1):
        if not np.isclose(hits_count(dup_ranks, k), correct):
            return "Check the function (test: both duplicates on the first position in ranked list)."

    # The second test: one candidate on the first position, another — on the second
    candidates_ranking = [["How Can I Make These Links Rotate in PHP",
                           "How does the catch keyword determine the type of exception that was thrown"],
                          ["Convert Google results object (pure js) to Python object",
                           "WPF- How to update the changes in list item of a list"]]
    dup_ranks = [candidates_ranking[i].index(answers[i]) + 1 for i in range(len(answers))]
    correct_answers = [0.5, 1]
    for k, correct in enumerate(correct_answers, 1):
        if not np.isclose(hits_count(dup_ranks, k), correct):
            return "Check the function (test: one candidate on the first position, another — on the second)."

    # The third test: both candidates on the second position
    candidates_ranking = [["How Can I Make These Links Rotate in PHP",
                           "How does the catch keyword determine the type of exception that was thrown"],
                          ["WPF- How to update the changes in list item of a list",
                           "Convert Google results object (pure js) to Python object"]]
    dup_ranks = [candidates_ranking[i].index(answers[i]) + 1 for i in range(len(answers))]
    correct_answers = [0, 1]
    for k, correct in enumerate(correct_answers, 1):
        if not np.isclose(hits_count(dup_ranks, k), correct):
            return "Check the function (test: both candidates on the second position)."

    return "Basic test are passed."
```

In [16]:

```
print(test_hits())
```

```
Basic test are passed.
```

In [17]:

```
def dcg_score(dup_ranks, k):
    """
        dup_ranks: list of duplicates' ranks; one rank per question;
                   length is a number of questions which we are looking for duplicates;
                   rank is a number from 1 to len(candidates of the question);
                   e.g. [2, 3] means that the first duplicate has the rank 2, the second one — 3.
        k: number of top-ranked elements (k in DCG@k metric)

        result: return DCG@k value for current ranking
    """
    ######################################
    ######### YOUR CODE HERE #############
    ######################################
#     print('dup_ranks, k', dup_ranks, k)
    query_size = len(dup_ranks)
    result = 0
    for rank in dup_ranks:
        if rank <= k:
            result += 1/np.log2(1+rank)
    return result/query_size
```

In [18]:

```
def test_dcg():
    # *Evaluation example*
    # answers — dup_i
    answers = ["How does the catch keyword determine the type of exception that was thrown"]

    # candidates_ranking — the ranked sentences provided by our model
    candidates_ranking = [["How Can I Make These Links Rotate in PHP",
                           "How does the catch keyword determine the type of exception that was thrown",
                           "NSLog array description not memory address",
                           "PECL_HTTP not recognised php ubuntu"]]
    # dup_ranks — position of the dup_i in the list of ranks +1
    dup_ranks = [candidates_ranking[i].index(answers[i]) + 1 for i in range(len(answers))]

    # correct_answers — the expected values of the result for each k from 1 to 4
    correct_answers = [0, 1 / (np.log2(3)), 1 / (np.log2(3)), 1 / (np.log2(3))]
    for k, correct in enumerate(correct_answers, 1):
        if not np.isclose(dcg_score(dup_ranks, k), correct):
            return "Check the function."

    # Other tests
    answers = ["How does the catch keyword determine the type of exception that was thrown",
               "Convert Google results object (pure js) to Python object"]

    # The first test: both duplicates on the first position in ranked list
    candidates_ranking = [["How does the catch keyword determine the type of exception that was thrown",
                           "How Can I Make These Links Rotate in PHP"],
                          ["Convert Google results object (pure js) to Python object",
                           "WPF- How to update the changes in list item of a list"]]
    dup_ranks = [candidates_ranking[i].index(answers[i]) + 1 for i in range(len(answers))]
    correct_answers = [1, 1]
    for k, correct in enumerate(correct_answers, 1):
        if not np.isclose(dcg_score(dup_ranks, k), correct):
            return "Check the function (test: both duplicates on the first position in ranked list)."

    # The second test: one candidate on the first position, another — on the second
    candidates_ranking = [["How Can I Make These Links Rotate in PHP",
                           "How does the catch keyword determine the type of exception that was thrown"],
                          ["Convert Google results object (pure js) to Python object",
                           "WPF- How to update the changes in list item of a list"]]
    dup_ranks = [candidates_ranking[i].index(answers[i]) + 1 for i in range(len(answers))]
    correct_answers = [0.5, (1 + (1 / (np.log2(3)))) / 2]
    for k, correct in enumerate(correct_answers, 1):
        if not np.isclose(dcg_score(dup_ranks, k), correct):
            return "Check the function (test: one candidate on the first position, another — on the second)."

    # The third test: both candidates on the second position
    candidates_ranking = [["How Can I Make These Links Rotate in PHP",
                           "How does the catch keyword determine the type of exception that was thrown"],
                          ["WPF- How to update the changes in list item of a list",
                           "Convert Google results object (pure js) to Python object"]]
    dup_ranks = [candidates_ranking[i].index(answers[i]) + 1 for i in range(len(answers))]
    correct_answers = [0, 1 / (np.log2(3))]
    for k, correct in enumerate(correct_answers, 1):
        if not np.isclose(dcg_score(dup_ranks, k), correct):
            return "Check the function (test: both candidates on the second position)."

    return "Basic test are passed."
```

In [19]:

```
print(test_dcg())
```

```
Basic test are passed.
```

Submit results of the functions *hits_count* and *dcg_score* for the following examples to earn the points.

In [20]:

```
test_examples = [
    [1],
    [1, 2],
    [2, 1],
    [1, 2, 3],
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    [9, 5, 4, 2, 8, 10, 7, 6, 1, 3],
    [4, 3, 5, 1, 9, 10, 7, 8, 2, 6],
    [5, 1, 7, 6, 2, 3, 8, 9, 10, 4],
    [6, 3, 1, 4, 7, 2, 9, 8, 10, 5],
    [10, 9, 8, 7, 6, 5, 4, 3, 2, 1],
]
```

In [21]:

```python
hits_results = []
for example in test_examples:
    for k in range(len(example)):
        hits_results.append(hits_count(example, k + 1))
grader.submit_tag('HitsCount', array_to_string(hits_results))
```

```
Current answer for task HitsCount is: 1.0
0.5
1.0
0.5
1.0
0.3333333333333333
0.6666666666666666
1.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1....
```

In [22]:

```python
dcg_results = []
for example in test_examples:
    for k in range(len(example)):
        dcg_results.append(dcg_score(example, k + 1))
grader.submit_tag('DCGScore', array_to_string(dcg_results))
```

```
Current answer for task DCGScore is: 1.0
0.5
0.8154648767857288
0.5
0.8154648767857288
0.3333333333333333
0.5436432511904858
0.7103099178...
```

First solution: pre-trained embeddings

We will work with predefined train, validation and test corpora. All the files are tab-separated, but have a different format:

- *train* corpus contains similar sentences at the same row.
- *validation* corpus contains the following columns: *question*, *similar question*, *negative example 1*, *negative example 2*, ...
- *test* corpus contains the following columns: *question*, *example 1*, *example 2*, ...

Validation corpus will be used for the intermediate validation of models. The test data will be necessary for submitting the quality of your model in the system.

Now you should read *validation* corpus, located at `data/validation.tsv`. You will use it later to evaluate current solution.

In [23]:

```python
def read_corpus(filename):
    data = []
    for line in open(filename, encoding='utf-8'):
        data.append(line.strip().split('\t'))
    return data
```

In [24]:

```python
######### YOUR CODE HERE #############
validation = read_corpus('data/validation.tsv')
```

In [25]:

```python
from sklearn.metrics.pairwise import cosine_similarity
```

We will use cosine distance to rank candidate questions which you need to implement in the function *rank_candidates*. The function should return a sorted list of pairs *(initial position in candidates list, candidate)*. Index of some pair corresponds to its rank (the first is the best). For example, if the list of candidates was *[a, b, c]* and the most similar is *c*, then *a* and *b*, the function should return a list *[(2, c), (0, a), (1, b)]*.

Pay attention, if you use the function *cosine_similarity* from *sklearn.metrics.pairwise* to calculate similarity because it works in a different way: most similar objects has greatest similarity. It's preferable to use a vectorized version of *cosine_similarity* function. Try to compute similarity at once and not use list comprehension. It should speed up your computations significantly.

In [26]:

```python
def rank_candidates(question, candidates, embeddings, dim=300):
    """
        question: a string
        candidates: a list of strings (candidates) which we want to rank
        embeddings: some embeddings
        dim: dimension of the current embeddings

        result: a list of pairs (initial position in the list, question)
    """

    ######################################
    ######### YOUR CODE HERE #############
    ######################################
    ques_vec = question_to_vec(question, embeddings, dim)
    cand_vecs = np.array([question_to_vec(candidate, embeddings, dim) for candidate in candidates])
    cs_similarity = cosine_similarity(ques_vec.reshape(1, -1), cand_vecs)
    sorted_idx = np.argsort(cs_similarity).reshape(-1)[::-1]
    return [(idx, candidates[idx]) for idx in sorted_idx]
```

Test your code on the tiny examples:

In [27]:

```python
def test_rank_candidates():
    questions = ['converting string to list', 'Sending array via Ajax fails']
    candidates = [['Convert Google results object (pure js) to Python object',
                   'C# create cookie from string and send it',
                   'How to use jQuery AJAX for an outside domain?'],
                  ['Getting all list items of an unordered list in PHP',
                   'WPF- How to update the changes in list item of a list',
                   'select2 not displaying search results']]
    results = [[(1, 'C# create cookie from string and send it'),
                (0, 'Convert Google results object (pure js) to Python object'),
                (2, 'How to use jQuery AJAX for an outside domain?')],
               [(0, 'Getting all list items of an unordered list in PHP'),
                (2, 'select2 not displaying search results'),
                (1, 'WPF- How to update the changes in list item of a list')]]
    for question, q_candidates, result in zip(questions, candidates, results):
        ranks = rank_candidates(question, q_candidates, wv_embeddings, 300)
        if not np.all(ranks == result):
            return "Check the function."
    return "Basic tests are passed."
```

In [28]:

```python
test_rank_candidates()
```

Out[28]:

```
'Basic tests are passed.'
```

Now we can test the quality of the current approach. Run the next two cells to get the results. Pay attention that calculation of similarity between vectors takes time and this calculation is computed approximately in 10 minutes.

In [29]:

```python
wv_ranking = []
for line in validation:
    q, *ex = line
    ranks = rank_candidates(q, ex, wv_embeddings)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

In [30]:

```python
for k in [1, 5, 10, 100, 500, 1000]:
    print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg_score(wv_ranking, k), k, hits_count(wv_ranking, k)))
```

```
DCG@   1: 0.214 | Hits@   1: 0.214
DCG@   5: 0.268 | Hits@   5: 0.316
DCG@  10: 0.285 | Hits@  10: 0.367
DCG@ 100: 0.323 | Hits@ 100: 0.557
DCG@ 500: 0.355 | Hits@ 500: 0.814
DCG@1000: 0.375 | Hits@1000: 1.000
```

If you did all the steps correctly, you should be frustrated by the received results. Let's try to understand why the quality is so low. First of all, when you work with some data it is necessary to have an idea how the data looks like. Print several questions from the data:

In [31]:

```python
for line in validation[:3]:
    q, *examples = line
    print(q, *examples[:3])
```

```
How to print a binary heap tree without recursion? How do you best convert a recursive function to an iterative one? Ho
w can i use ng-model with directive in angular js flash: drawing and erasing
How to start PhoneStateListener programmatically? PhoneStateListener and service Java cast object[] to model WCF and Wh
at does this mean?
jQuery: Show a div2 when mousenter over div1 is over when hover on div1 depenting on if it is on div2 or not it should
act differently How to run selenium in google app engine/cloud? Python Comparing two lists of strings for similarities
```

As you can see, we deal with the raw data. It means that we have many punctuation marks, special characters and unlowercased letters. In our case, it could lead to the situation where we can't find some embeddings, e.g. for the word "grid?".

To solve this problem you should use the functions *text_prepare* from the previous assignments to prepare the data.

In [32]:

```
from util import text_prepare
```

Now transform all the questions from the validation set:

In [33]:

```
prepared_validation = []
for line in validation:
    ######### YOUR CODE HERE #############
    prepared_line = [text_prepare(sentence) for sentence in line]
    prepared_validation.append(prepared_line)
```

Let's evaluate the approach again after the preparation:

In [34]:

```
wv_prepared_ranking = []
for line in prepared_validation:
    q, *ex = line
    ranks = rank_candidates(q, ex, wv_embeddings)
    wv_prepared_ranking.append([r[0] for r in ranks].index(0) + 1)
```

In [35]:

```
for k in [1, 5, 10, 100, 500, 1000]:
    print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg_score(wv_prepared_ranking, k),
                                              k, hits_count(wv_prepared_ranking, k)))
```

```
DCG@   1: 0.316 | Hits@   1: 0.316
DCG@   5: 0.383 | Hits@   5: 0.444
DCG@  10: 0.400 | Hits@  10: 0.496
DCG@ 100: 0.434 | Hits@ 100: 0.665
DCG@ 500: 0.457 | Hits@ 500: 0.841
DCG@1000: 0.473 | Hits@1000: 1.000
```

Now, prepare also train and test data, because you will need it in the future:

In [36]:

```
def prepare_file(in_, out_):
    out = open(out_, 'w')
    for line in open(in_, encoding='utf8'):
        line = line.strip().split('\t')
        new_line = [text_prepare(q) for q in line]
        print(*new_line, sep='\t', file=out)
    out.close()
```

In [37]:

```
######################################
######### YOUR CODE HERE #############
######################################
prepare_file('data/train.tsv', 'data/train_prepared.tsv')
prepare_file('data/test.tsv', 'data/test_prepared.tsv')
```

Task 4 (W2VTokenizedRanks). For each question from prepared *test.tsv* submit the ranks of the candidates to earn the points. The calculations should take about 3-5 minutes. Pay attention that the function *rank_candidates* returns a ranking, while in this case you should find a position in this ranking. Ranks should start with 1.

In [38]:

```
from util import matrix_to_string
```

In [39]:

```
w2v_ranks_results = []
######### YOUR CODE HERE #############
prepared_test_data = 'data/test_prepared.tsv'
for line in open(prepared_test_data):
    q, *ex = line.strip().split('\t')
    ranks = rank_candidates(q, ex, wv_embeddings, 300)
    ranked_candidates = [r[0] for r in ranks]
    w2v_ranks_results.append([ranked_candidates.index(i) + 1 for i in range(len(ranked_candidates))])

grader.submit_tag('W2VTokenizedRanks', matrix_to_string(w2v_ranks_results))
```

```
Current answer for task W2VTokenizedRanks is: 95        94      7       9       64      37      32      93      24
100     98      17      60      6       97      49      70      38      42      96      30      21      2       65
67      45      27      26      57      62      11      88      56      66      7...
```

Advanced solution: StarSpace embeddings

Now you are ready to train your own word embeddings! In particular, you need to train embeddings specially for our task of duplicates detection. Unfortunately, StarSpace cannot be run on Windows and we recommend to use provided [docker container (https://github.com/hse-aml/natural-language-processing/blob/master/Docker-tutorial.md)](https://github.com/hse-aml/natural-language-processing/blob/master/Docker-tutorial.md) or other alternatives. Don't delete results of this task because you will need it in the final project.

How it works and what's the main difference with word2vec?

The main point in this section is that StarSpace can be trained specifically for some tasks. In contrast to word2vec model, which tries to train similar embeddings for words in similar contexts, StarSpace uses embeddings for the whole sentence (just as a sum of embeddings of words and phrases). Despite the fact that in both cases we get word embeddings as a result of the training, StarSpace embeddings are trained using some supervised data, e.g. a set of similar sentence pairs, and thus they can better suit the task.

In our case, StarSpace should use two types of sentence pairs for training: "positive" and "negative". "Positive" examples are extracted from the train sample (duplicates, high similarity) and the "negative" examples are generated randomly (low similarity assumed).

How to choose the best params for the model?

Normally, you would start with some default choice and then run extensive experiments to compare different strategies. However, we have some recommendations ready for you to save your time:

- Be careful with choosing the suitable training mode. In this task we want to explore texts similarity which corresponds to *trainMode = 3*.
- Use adagrad optimization (parameter *adagrad* = *true*).
- Set the length of phrase equal to 1 (parameter *ngrams*), because we need embeddings only for words.
- Don't use a large number of *epochs* (we think that 5 should be enough).
- Try dimension *dim* equal to 100.
- To compare embeddings usually *cosine similarity* is used.
- Set *minCount* greater than 1 (for example, 2) if you don't want to get embeddings for extremely rare words.
- Parameter *verbose* = *true* could show you the progress of the training process.
- Set parameter *fileFormat* equals *labelDoc*.
- Parameter *negSearchLimit* is responsible for a number of negative examples which is used during the training. We think that 10 will be enought for this task.
- To increase a speed of training we recommend to set *learning rate* to 0.05.

Train StarSpace embeddings for unigrams on the train dataset. You don't need to change the format of the input data. Just don't forget to use prepared version of the training data.

If you follow the instruction, the training process will take about 1 hour. The size of the embeddings' dictionary should be approximately 100 000 (number of lines in the result file). If you got significantly more than this number, try to check all the instructions above.

In [40]:

```
######### TRAINING HAPPENING HERE #############
```

Assume in the `data` directory, works on Windows with docker using WSL 2 as the backend

```
# Pull docker image
docker pull akashin/coursera-aml-nlp

# Create docker container and run interactive shell, map current directory to `/aml`
docker run -it -p 8080:8080 --name aml -v "$PWD":/aml akashin/coursera-aml-nlp

# Run below command in the docker container
starspace train -trainFile train_prepared.tsv -model textsimilarity -trainMode 3 -adagrad true -ngrams 1 -epoch 5 -dim 100 -similarity cosine -minCount 2 -verbose true -fileFormat labelDoc -negSearchLimit 10 -lr 0.05
```

And now we can compare the new embeddings with the previous ones. You can find trained word vectors in the file *[model_file_name].tsv*. Upload the embeddings from StarSpace into a dict.

In [41]:

```
######### YOUR CODE HERE #############
ss_file = 'data/textsimilarity.tsv'
starspace_embeddings = {}

for line in open(ss_file):
    word, *vector = line.split('\t')
    starspace_embeddings[word] = np.array(vector).astype(float)
```

In [42]:

```
ss_prepared_ranking = []
for line in prepared_validation:
    q, *ex = line
    ranks = rank_candidates(q, ex, starspace_embeddings, 100)
    ss_prepared_ranking.append([r[0] for r in ranks].index(0) + 1)
```

In [43]:

```
for k in [1, 5, 10, 100, 500, 1000]:
    print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg_score(ss_prepared_ranking, k),
                                               k, hits_count(ss_prepared_ranking, k)))
```

```
DCG@   1: 0.516 | Hits@   1: 0.516
DCG@   5: 0.615 | Hits@   5: 0.701
DCG@  10: 0.633 | Hits@  10: 0.757
DCG@ 100: 0.664 | Hits@ 100: 0.905
DCG@ 500: 0.674 | Hits@ 500: 0.980
DCG@1000: 0.676 | Hits@1000: 1.000
```

Due to training for the particular task with the supervised data, you should expect to obtain a higher quality than for the previous approach. In additiion, despite the fact that StarSpace's trained vectors have a smaller dimension than word2vec's, it provides better results in this task.

Task 5 (StarSpaceRanks). For each question from prepared *test.tsv* submit the ranks of the candidates for trained representation.

In [44]:

```
starspace_ranks_results = []
######### YOUR CODE HERE #############
prepared_test_data = 'data/test_prepared.tsv'
for line in open(prepared_test_data):
    q, *ex = line.strip().split('\t')
    ranks = rank_candidates(q, ex, starspace_embeddings, 100)
    ranked_candidates = [r[0] for r in ranks]
    starspace_ranks_results.append([ranked_candidates.index(i) + 1 for i in range(len(ranked_candidates))])

grader.submit_tag('StarSpaceRanks', matrix_to_string(starspace_ranks_results))
```

```
Current answer for task StarSpaceRanks is: 53   91      61      73      31      94      99      29      39      36
50      7       32      74      48      80      8       76      90      84      25      27      15      56      98
17      70      87      59      5       38      26      79      81      1...
```

Please, **don't remove** the file with these embeddings because you will need them in the final project.

Authorization & Submission

To submit assignment parts to Cousera platform, please, enter your e-mail and token into variables below. You can generate token on this programming assignment page. **Note:** Token expires 30 minutes after generation.

In [45]:

```
STUDENT_EMAIL = 'lxwvictor@gmail.com'
STUDENT_TOKEN = 'cjaHaGmH7ZcxFV5n'
grader.status()
```

```
You want to submit these parts:
Task Question2Vec: 0.019293891059027776
-0.028727213541666668
0.046056111653645836
0.08525933159722222
0.02430555555555...
Task HitsCount: 1.0
0.5
1.0
0.5
1.0
0.3333333333333333
0.6666666666666666
1.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1....
Task DCGScore: 1.0
0.5
0.8154648767857288
0.5
0.8154648767857288
0.3333333333333333
0.5436432511904858
0.7103099178...
Task W2VTokenizedRanks: 95      94      7       9       64      37      32      93      24      100     98      17
60      6       97      49      70      38      42      96      30      21      2       65      67      45      27
26      57      62      11      88      56      66      7...
Task StarSpaceRanks: 53 91      61      73      31      94      99      29      39      36      50      7       32
74      48      80      8       76      90      84      25      27      15      56      98      17      70      87
59      5       38      26      79      81      1...
```

If you want to submit these answers, run cell below

In [46]:

```
grader.submit(STUDENT_EMAIL, STUDENT_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

In [ ]:

```

```