Using embeddings to find duplicate questions on Stack Overflow

January 10, 2020

1 Find duplicate questions on StackOverflow by their embeddings

Similarity for pieces of text will be calculated to find duplicate questions from StackOverflow.

Data

Predefined train and validation corpora will be used. These tab-separated files have the following formats: + train contains similar sentences on the same row + validation contains columns: question, similar question, negative example 1, negative example 2, etc

Word embedding

To solve the problem of word embedding, two different models of embeddings will be used: + Pre-trained word vectors from Google + Representations using StarSpace on data sample, which needs to be trained from scratch.

1.1 Import pre-trained word vectors from Google

These 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.

[5]:
```
#!wget -c "https://s3.amazonaws.com/dl4j-distribution/
↪GoogleNews-vectors-negative300.bin.gz"
```

[8]:
```
import gensim

from gensim.models import KeyedVectors
wv_embeddings = KeyedVectors.
↪load_word2vec_format('GoogleNews-vectors-negative300.bin.gz', binary=True,␣
↪limit=500000)
```

1.1.1 Check embeddings are correct

[9]:
```
#function to check loaded embeddings are correct: runs 3 tests (1) find most␣
↪similar words for provided negative and positive words
#(2) find which word from given list does not go with others (3) find most␣
↪similar word for provided one
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."

print(check_embeddings(wv_embeddings))
```

```
These embeddings look good.
```

```
/Users/charlottefettes/opt/anaconda3/lib/python3.7/site-
packages/gensim/models/keyedvectors.py:877: FutureWarning: arrays to stack must
be passed as a "sequence" type such as list or tuple. Support for non-sequence
iterables such as generators is 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)
```

1.2 From word to text embeddings

As the task is to find duplicate questions, rather than word-based embeddings, this task requires creation of a representation for the whole question. It could be done in different ways. Here, a mean of all word vectors in the question will be used.

The function question_to_vec calculates the question representation (input text does not require preprocessing).

Words without the corresponding embeddings will be skipped, and will not be taken into account when calculating the result. If the question does not contain any known word with embedding, the function should return a zero vector.

[10]:
```
import numpy as np

def question_to_vec(question, embeddings, dim=300):
    result = np.zeros(dim)
    count = 0
    for word in question.split():
        if word in embeddings:
```

```
            result += embeddings[word]
            count += 1
    return result / count if count != 0 else result
```

[11]:
```
#check correctness of function
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."

print(question_to_vec_tests())
```

```
Basic tests are passed.
```

[12]:
```
import nltk
nltk.download('stopwords')

#def array_to_string(arr):
#    return '\n'.join(str(num) for num in arr)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/charlottefettes/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[12]:
```
True
```

[13]:
```
#question2vec_result = []
#for question in open('test_embeddings.tsv'):
#    question = question.strip()
#    answer = question_to_vec(question, wv_embeddings)
#    question2vec_result = np.append(question2vec_result, answer)
```

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.

1.2.1 Evaluation of text similarity

With Google vectors and question_to_vec function to create a representation of any sentence, this method can be evaluated.

The first metric to be used is Hits@K, which is the number of correct hits for some K (more information here: https://en.wikipedia.org/wiki/Iverson_bracket).

The second metric is DCG@K, a simplified DCG metric (more information here: https://en.wikipedia.org/wiki/Discounted_cumulative_gain)

The functions for these two metrics are below. Each function has two arguments: dup_ranks (a list which contains values of ranks of duplicates) and k.

[14]:
```
def hits_count(dup_ranks, k):
    return sum(rank <= k for rank in dup_ranks) / len(dup_ranks)
```

[15]:
```
#text hits_count function
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."
```

[16]:
```
print(test_hits())
```

```
Basic test are passed.
```

[17]:
```
def dcg_score(dup_ranks, k):
    return sum(1 / (np.log2(1 + rank)) for rank in dup_ranks if rank <= k) /␣
↪len(dup_ranks)
```

[18]:
```
#function to test dcg_score function
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."

print(test_dcg())
```

```
Basic test are passed.
```

[19]:
```
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],
]
```

[20]:
```
hits_results = []
for example in test_examples:
    for k in range(len(example)):
        hits_results.append(hits_count(example, k + 1))
```

[21]:
```
dcg_results = []
for example in test_examples:
    for k in range(len(example)):
        dcg_results.append(dcg_score(example, k + 1))
```

1.2.2 First solution: pre-trained embeddings with Google vectors

[22]:
```
def read_corpus(filename):
    data = []
    for line in open(filename, encoding='utf-8'):
        data.append(line.strip().split('\t'))
    return data
```

[23]:
```
validation = read_corpus('data/validation.tsv')
```

Cosine distance will be used to rank candidate questions. This will be used in the below function, which should return a sorted list of pairs. Pair index corresponds to a candidate's rank in terms of how similar it is.

[25]:
```
from sklearn.metrics.pairwise import cosine_similarity
```

[26]:
```
def rank_candidates(question, candidates, embeddings, dim=300):
    qv = question_to_vec(question, embeddings, dim)[np.newaxis, :]
    cvs = np.array([question_to_vec(candidate, embeddings, dim) for candidate␣
 ↪in candidates])
    sims = cosine_similarity(qv, cvs)[0]
    idxs = np.argsort(sims)[::-1]
    return [(i, candidates[i]) for i in idxs]
```

[27]:
```
#function to test rank_candidates
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."

print(test_rank_candidates())
```

```
Basic tests are passed.
```

The quality of the current approach will now be tested.

[28]:
```
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)
```

[29]:
```
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: 0.209 | Hits@   1: 0.209
DCG@   5: 0.263 | Hits@   5: 0.311
DCG@  10: 0.279 | Hits@  10: 0.360
DCG@ 100: 0.316 | Hits@ 100: 0.548
DCG@ 500: 0.349 | Hits@ 500: 0.807
DCG@1000: 0.369 | Hits@1000: 1.000
```

The quality of results is low. To ascertain why this is, a number of questions will be printed.

[30]:
```
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? How 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 What 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 shown, it is dealing with the raw data, with numerous punctuation marks, special characters and uppercase letters. The likely result is that for many words the varying ways that a word is appearing means it cannot find the embedding so it is assumed that it does not have an embedding.

To solve this issue, the raw data will be processed and prepared.

[31]:
```
import re
from nltk.corpus import stopwords

REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
GOOD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
STOPWORDS = set(stopwords.words('english'))

def text_prepare(text):
    text = text.lower()
    text = REPLACE_BY_SPACE_RE.sub(' ', text)
    text = GOOD_SYMBOLS_RE.sub('', text)
    text = ' '.join([x for x in text.split() if x and x not in STOPWORDS])
    return text.strip()
```

[32]:
```
prepared_validation = []
for line in validation:
    prepared_validation.append([text_prepare(text) for text in line])
```

The performance on validation will now be reevaluated.

[33]:
```
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)
```

[34]:
```
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: 0.305 | Hits@   1: 0.305
DCG@   5: 0.375 | Hits@   5: 0.438
```

```
DCG@  10: 0.392 | Hits@  10: 0.489
DCG@ 100: 0.425 | Hits@ 100: 0.656
DCG@ 500: 0.447 | Hits@ 500: 0.830
DCG@1000: 0.465 | Hits@1000: 1.000
```

[35]:
```
#prepare train set as for validation
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()
```

[36]:
```
prepare_file('data/train.tsv', 'data/train_prepared.tsv')
prepare_file('data/validation.tsv', 'data/validation_prepared.tsv')
```

1.2.3 Second solution: StarSpace embeddings

Instead of using pretrained Google vectors, here word embeddings can be trained specially for the task of duplicates detection.

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 word embeddings are generated 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.

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).

To obtain the best parameters, extensive experiementation should be conducted. For purposes here, the following parameters will be used: + To explore texts similarity, trainMode = 3 is required + Adagrad optimisation (parameter adagrad = true). + Length of phrase set equal to 1 (parameter ngrams), as embeddings required for words only + 5 epochs + Dimension (dim) equal to 100 + To compare embeddings, cosine similarity will be used + minCount > 1 to avoid getting embeddings for very rare words + Parameter verbose = true to show progress of the training process + Parameter fileFormat set equal to labelDoc + Parameter negSearchLimit (responsible for number of negative examples used during training) set to 10 + Learning rate of 0.05 to increase speed of training

Requirements for Starspace (https://github.com/facebookresearch/StarSpace):

Need to install Boost library and specify the path of boost library in makefile in order to run StarSpace. Basically:
wget https://dl.bintray.com/boostorg/release/1.63.0/source/boost_1_63_0.zip
unzip boost_1_63_0.zip
sudo mv boost_1_63_0 /usr/local/bin

Building Starspace on Mac OS or Linux:
git clone https://github.com/facebookresearch/Starspace.git

cd Starspace
make

[80]:
```
#training word embeddings
!starspace train -trainFile data/train_prepared.tsv -model StarSpace_embeddings␣
 ↪\
-trainMode 3 \
-adagrad true \
-ngrams 1 \
-epoch 5 \
-dim 100 \
-similarity cosine \
-minCount 2 \
-verbose true \
-fileFormat labelDoc \
-negSearchLimit 10 \
-lr 0.05
```

```
Arguments:
lr: 0.05
dim: 100
epoch: 5
maxTrainTime: 8640000
validationPatience: 10
saveEveryEpoch: 0
loss: hinge
margin: 0.05
similarity: cosine
maxNegSamples: 10
negSearchLimit: 10
batchSize: 5
thread: 10
minCount: 2
minCountLabel: 1
label: __label__
label: __label__
ngrams: 1
bucket: 2000000
adagrad: 1
trainMode: 3
fileFormat: labelDoc
normalizeText: 0
dropoutLHS: 0
dropoutRHS: 0
useWeight: 0
weightSep: :
Start to initialize starspace model.
Build dict from input file : data/train_prepared.tsv
```

```
Read 12M words
Number of words in dictionary:  95058
Number of labels in dictionary: 0
Loading data from file : data/train_prepared.tsv
Total number of examples loaded : 999740
Initialized model weights. Model size :
matrix : 95058 100
Training epoch 0: 0.05 0.01
Epoch: 100.0%  lr: 0.040000  loss: 0.042678  eta: 0h4m  tot: 0h1m3s
(20.0%).7%)m19s  (5.8%)9.6%)m38s  (11.5%)s  (11.7%)39s  (11.8%)h0m39s
(11.8%)42s  (12.7%)m42s  (12.8%)68.9%  lr: 0.043083  loss: 0.048635  eta: 0h4m
tot: 0h0m45s  (13.8%)17.3%)19.4%)
 ---+++               Epoch    0 Train error : 0.04287124 +++---
Training epoch 1: 0.04 0.01
Epoch: 100.0%  lr: 0.030000  loss: 0.013140  eta: 0h3m  tot: 0h2m4s  (40.0%)
(20.6%)s  (22.7%)35.3%  lr: 0.036547  loss: 0.013506  eta: 0h3m  tot: 0h1m24s
(27.1%)m29s  (28.6%)31.6%)74.5%  lr: 0.032563  loss: 0.013238  eta: 0h3m  tot:
0h1m49s  (34.9%)53s  (36.3%)1m55s  (37.1%)s  (39.7%)
 ---+++               Epoch    1 Train error : 0.01320188 +++---
Training epoch 2: 0.03 0.01
Epoch: 100.0%  lr: 0.020000  loss: 0.009539  eta: 0h2m  tot: 0h3m5s  (60.0%).1%)
(40.8%)44.8%)26s  (47.7%)h2m27s  (48.0%) (49.2%)%)51.7%)41s  (52.4%)
(52.4%)54.1%)2m57s  (57.5%)%)
 ---+++               Epoch    2 Train error : 0.00940394 +++---
Training epoch 3: 0.02 0.01
Epoch: 100.0%  lr: 0.010000  loss: 0.007770  eta: <1min   tot: 0h4m4s  (80.0%)s
(63.7%) (64.6%)68.2%)69.2%)60.9%  lr: 0.013734  loss: 0.007777  eta: 0h1m  tot:
0h3m40s  (72.2%)s  (75.9%)75.9%)0h1m  tot: 0h4m0s  (78.7%)93.7%  lr: 0.010681
loss: 0.007712  eta: 0h1m  tot: 0h4m0s  (78.7%)m  tot: 0h4m0s  (78.8%)79.2%)0h1m
tot: 0h4m3s  (79.6%)s  (79.8%)
 ---+++               Epoch    3 Train error : 0.00773823 +++---
Training epoch 4: 0.01 0.01
Epoch: 100.0%  lr: 0.000000  loss: 0.006805  eta: <1min   tot: 0h5m2s
(100.0%)(81.7%)24.1%  lr: 0.007728  loss: 0.006568  eta: <1min   tot: 0h4m18s
(84.8%)%)%)s  (92.7%)
 ---+++               Epoch    4 Train error : 0.00685289 +++---
Saving model to file : StarSpace_embeddings
Saving model in tsv format : StarSpace_embeddings.tsv
```

1.2.4 Compare performance of embeddings from the two solutions on validation set

[81]:
```
starspace_embeddings = {}
for line in open('StarSpace_embeddings.tsv', encoding='utf-8'):
    word, *vec = line.strip().split('\t')
    starspace_embeddings[word] = np.array(vec, dtype=np.float)
```

[82]:
```
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)
```

[83]:
```
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: 0.520 | Hits@   1: 0.520
DCG@   5: 0.618 | Hits@   5: 0.701
DCG@  10: 0.637 | Hits@  10: 0.759
DCG@ 100: 0.667 | Hits@ 100: 0.907
DCG@ 500: 0.677 | Hits@ 500: 0.981
DCG@1000: 0.679 | Hits@1000: 1.000
```

Due to training for the particular task with the supervised data, a higher quality is obtained for the StarSpace embeddings than for the previous approach. Also, despite the fact that StarSpace's trained vectors have a smaller dimension than word2vec's, it provides better results in this task.