homework4_p1

August 4, 2020

##

50.040 Natural Language Processing, Summer 2020

Homework 4

Due 31 July 2020, 5pm

Write your student ID and name

ID: 1002961

Name: Wu Tianyu

Students whom you have discussed with (if any): None

0.0.1 Requirements:

- Use Python to complete this homework.
- Please list students with whom you have discussed (if any).
- Follow the honor code strictly.

In this homework, we'll implement `IBM Model 1` using the `expectation-maximization (EM)` algorithm. We need to estimate the translation probabilities $t(f|e)$ on a parallel corpus, where e is a word from the English sentences and f is a word from the corresponding foreign sentences.

Note that there's a constraint for such probabilities:

$$\sum_f t(f|e) = 1, \quad t(f|e) \geq 0 \quad (1)$$

We'll use this constraint when initializing the translation probabilities in subsequent tasks.

0.1 Data

We'll use the English-French parallel corpus under the folder `data/part1`, which contains a set of translation instances. As can be seen below each instance consists of an English-French sentence pair (note that we are translating from French into English, but as we discussed in class, when working on the translation model using IBM model 1, we are interested in generating French from English).

```
Hop in. Montez.
Hug me. Serre-moi dans tes bras !
I left. Je suis parti.
```

The dataset is obtained from MXNET. Please run the provided code below to obtain the preprocessed English sentences and French sentences. Do not perform any further preprocessing.

[1]:
```
import seaborn as sns
import numpy as np
from time import time
from collections import Counter, defaultdict

from matplotlib import pyplot as plt
%matplotlib inline
```

1 Part 1: Statistical Machine Translation [25 points]

[2]:
```
path = 'data/part1/en-fr.txt'
with open(path, 'r', encoding='utf8') as f:
    raw_text = f.read()

#Original code from
#https://www.d2l.ai/chapter_recurrent-neural-networks
def preprocess_nmt(text):
    '''
    Arg:
        text: parallel text, string
    Return:
        out: preprocessed text, string
    '''
    text = text.replace('\u202f', ' ').replace('\xa0', ' ')
    no_space = lambda char, prev_char: (
        True if char in (',', '!', '.') and prev_char != ' ' else False)
    out = [' '+char if i > 0 and no_space(char, text[i-1]) else char
           for i, char in enumerate(text.lower())]
```

```
    out = ''.join(out)
    return out

def tokenize_nmt(text, num_examples = None):
    '''
    Args:
        text: parallel text, string
        num_examples: number of examples to be selected, int
    Returns:
        left: English sentences, list
        right: French sentences, list
    '''
    left, right = [], []
    for i, line in enumerate(text.split('\n')):
        if num_examples and i > num_examples: break
        parts = line.split('\t')
        if len(parts) == 2:
            left.append(parts[0].split(' '))
            right.append(parts[1].split(' '))
    return left, right
```

[3]:
```
#English sentences and corresponding French sentences
#Each sentence has been preprocessed and tokenized
text = preprocess_nmt(raw_text)
english_sents, french_sents = tokenize_nmt(text)
```

[4]:
```
english_sents[:10], french_sents[:10]
```

[4]:
```
([['go', '.'],
  ['hi', '.'],
  ['run', '!'],
  ['run', '!'],
  ['who?'],
  ['wow', '!'],
  ['fire', '!'],
  ['help', '!'],
  ['jump', '.'],
  ['stop', '!']],
 [['va', '!'],
  ['salut', '!'],
  ['cours', '!'],
  ['courez', '!'],
  ['qui', '?'],
  ['ça', 'alors', '!'],
  ['au', 'feu', '!'],
  ['à', "l'aide", '!'],
  ['saute', '.'],
```

```
['ça', 'suffit', '!']])
```

1.0.1 Quesiton 1 (3 points)

1. Implement `word_pairs_in_corpus` which finds out all the possible word pairs (alignments) (e, f) that appear in all the instances of the English-French dataset `english_sents`, `french_sents`. Note that we need to pad each English sentence with the special token "NULL" at the beginning.
2. List down the 10 most frequent pairs.
3. Count the number of unique pairs.

[12]:
```
def word_pairs_in_corpus(en_sents, fr_sents):
    '''
    params:
        en_sents: list[list[str]]
        fr_sents: list[list[str]]
    return:
        align_counts: Dict()--- key: (english_word, french_word), value: counts␣
↪of the word pair in the corpus
    '''
    align_counts = None
    # YOUR CODE HERE
    align_counts = defaultdict(int)
    for en_sent, fr_sent in zip(en_sents, fr_sents):
        for en_word in en_sent:
            for fr_word in fr_sent:
                align_counts[(en_word, fr_word)] += 1
    align_counts = Counter(align_counts)
    # END OF YOUR CODE
    return align_counts
```

[13]:
```
english_sents = [['NULL'] + sent for sent in english_sents]
align_counts = word_pairs_in_corpus(english_sents, french_sents)
align_counts.most_common(10), len(align_counts)
```

[13]:
```
([(('NULL', '.'), 405663),
  (('.', '.'), 136734),
  (('NULL', 'je'), 119463),
  (('NULL', 'de'), 105219),
  (('NULL', '?'), 81360),
  (('NULL', 'pas'), 80958),
  (('NULL', 'que'), 72000),
  (('NULL', 'à'), 63018),
  (('NULL', 'ne'), 60693),
  (('NULL', 'le'), 58461)],
 1402126)
```

[14]:
```
en_vocab = set([item[0] for item in align_counts.keys()])
fr_vocab = set([item[1] for item in align_counts.keys()])
```

[15]:
```
len(en_vocab), len(fr_vocab)
```

[15]: (17430, 29741)

1.0.2 Question 2 (2 points):

Implment the `corpus_log_prob` that computes the log probability of the corpus

[16]:
```
def corpus_log_prob(en_sents, fr_sents, t):
    '''
    params:
        en_sents: list[list[str]]
        fr_sents: list[list[str]]
        t: Dict() --- contains translation probabilities. For example, t[(english_word, french_word)] = p
    return:
        logp: float --- log probability of the corpus
    '''
    logp = 0
    ### YOUR CODE HERE
    # Assume the sentences are properly padded
    for en_sent, fr_sent in zip(en_sents, fr_sents):
        for fr_word in fr_sent:
            logp += np.log(sum([t[(en_word, fr_word)] for en_word in en_sent]))
    # END OF YOUR CODE
    return logp
```

1.1 Hard EM algorithm

1.1.1 Question 3 (10 points)

Based on the word pairs obtained in Q1, implement `Hard EM algorithm` to calculate the translation probabilities $t(f|e)$ on the English-French corpus.

It is possible that in the hard EM algorithm a word $\tilde{e}$ from an English sentence may not be aligned with any word from the corresponding French sentence. In this case, let us set the corresponding probabilities $t(f|\tilde{e}) = \frac{1}{|V_f|}$ where $|V_f|$ is the size of the French vocabulary (in this case, the number of unique French words that ever appear in the training parallel corpus).

1. Implement `init` function which initializes the translation probability dictionary t according to equation (1). You need to use `numpy.random.rand()` in this part.
2. Implement `hard_EM` function which runs one `Expectation/Maximization` iteration.
3. Run the training code

[17]:
```
def init(word_pairs):
    '''
    Use np.random.rand() to initialize translation probabilities t(f|e)
    params:
        word_pairs: List[(str, str)] --- list of word pairs
    return:
        t: Dict(), key: (english_word, french_word), value: the initial␣
 ↪probability t(f|e). For example, t[(a, un)] = 0.5
    '''
    np.random.seed(5)
    t = dict()
    ### YOUR CODE HERE
    possible_pairs = defaultdict(list)
    for en_word, fr_word in word_pairs:
        possible_pairs[en_word].append(fr_word)
    for en_word in possible_pairs:
        num_poss_fr_word = len(possible_pairs[en_word])
        rand_array = np.random.rand(num_poss_fr_word)
        rand_array /= np.sum(rand_array)
        for fr_word, p in zip(possible_pairs[en_word], rand_array):
            t[(en_word, fr_word)] = p
    ### END OF YOUR CODE
    return t
```

[35]:
```
def hard_EM(en_sents, fr_sents, fr_vocab, t):
    '''
    One 'Expectation', 'Maximization' iteration.
    params:
        en_sents: List[List[str]]
        fr_sents: List[List[str]]
        fr_vocab: int --- size of the French vocab
        t: Dict() --- translation probability dictionary from last iteration

    return:
        new_t: Dict() --- updated parameters, dictionary
    '''
    new_t = t
    ### YOUR CODE HERE

    # Expectation
    hard_count = defaultdict(lambda: defaultdict(int))
    for en_sent, fr_sent in zip(en_sents, fr_sents):
        # fix french word to find best english word alignment
        for fr_word in fr_sent:
            candidate_prob = np.zeros(len(en_sent))
            for i, en_word in enumerate(en_sent):
                candidate_prob[i] = t[(en_word, fr_word)]
```

```
            hard_count[en_sent[np.argmax(candidate_prob)]][fr_word] += 1

    # Maximization
    # first update the missing parameters
    for en_word, fr_word in t:
        if en_word not in hard_count or fr_word not in hard_count[en_word]: #␣
↪case that certain combination disappears
            new_t[(en_word, fr_word)] = 1/len(fr_vocab)
    # finally the normal update
    for en_word in hard_count:
        count_en = sum(hard_count[en_word].values())
        for fr_word in hard_count[en_word]:
            count_en_fr = hard_count[en_word][fr_word]
            new_t[(en_word, fr_word)] = count_en_fr/count_en
    ### END OF YOUR CODE

    return new_t
```

[36]:
```
###############################################################
# Randomly initialized the probabilities under the contraint
###############################################################
hard_t = init(list(align_counts.keys()))

###################
# Hard EM training
###################
iteration = 0
while iteration < 10:

    logp = corpus_log_prob(english_sents, french_sents, hard_t)
    hard_t = hard_EM(english_sents, french_sents, fr_vocab, hard_t)
    print('Objective Function:', round(logp, 5))
    iteration += 1
```

```
Objective Function: -5649986.83975
Objective Function: -2928383.93888
Objective Function: -2246054.34811
Objective Function: -2132199.0455
Objective Function: -2102013.95014
Objective Function: -2065188.29695
Objective Function: -2059486.15937
Objective Function: -2054711.51582
Objective Function: -2053155.45736
Objective Function: -2052421.4786
```

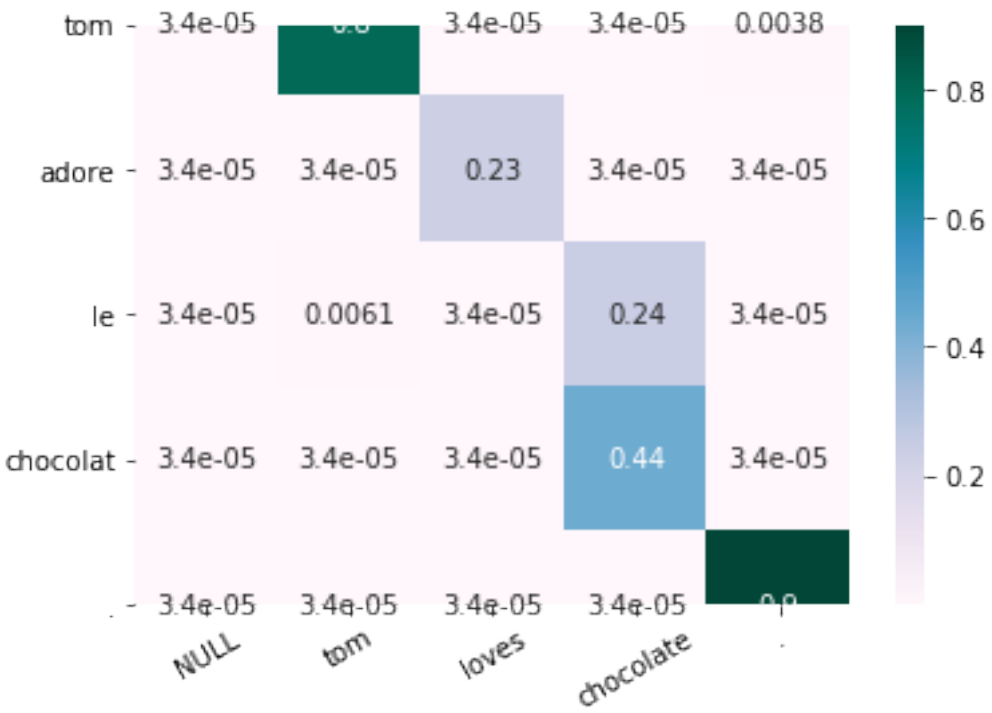
1.1.2 Visualization

Using 2D-heatmap, visualize the translation probability (namely $t(f|e)$) for each of the instances below:

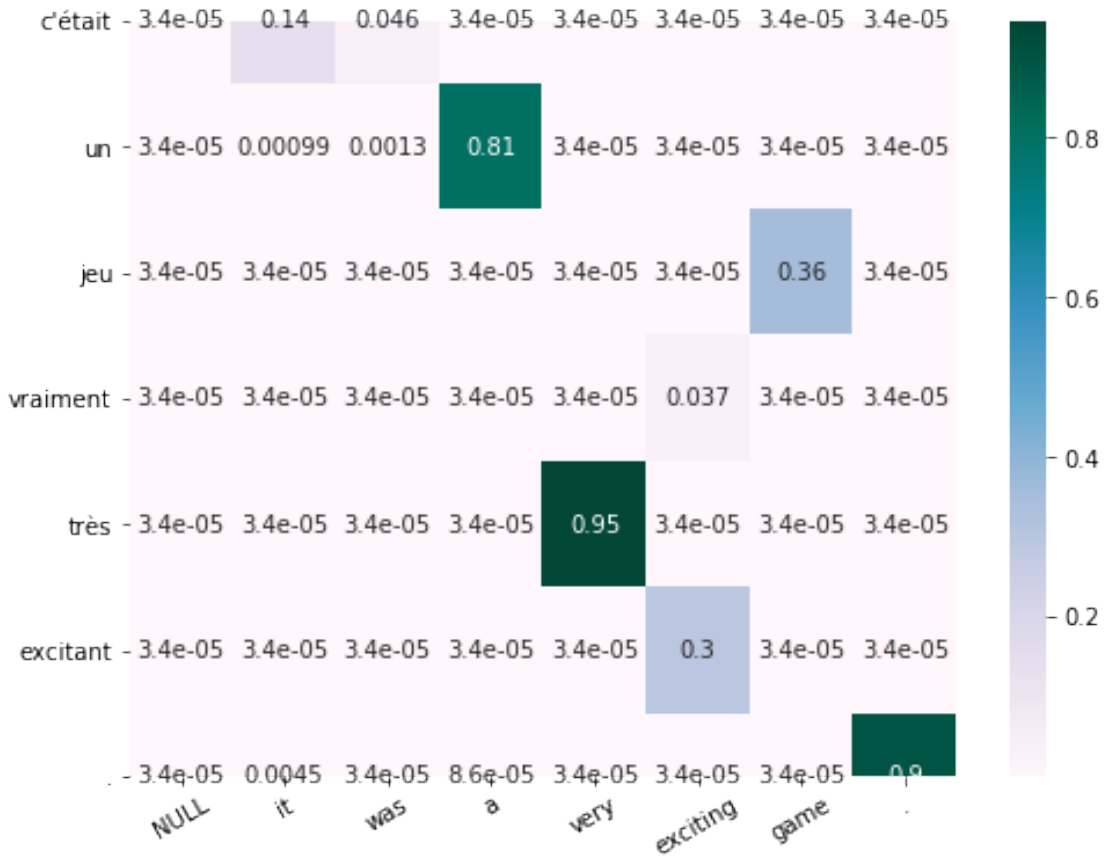
```
NULL tom loves chocolate .   tom adore le chocolat .
NULL it was a very exciting game .   c'était un jeu vraiment très excitant .
```

[37]:
```
def visualize_trans_prob(en, fr, t):
    '''
    Visualize the translation probability of an instance
    '''
    alignments = np.zeros([len(fr), len(en)])
    for i in range(len(fr)):
        for j in range(len(en)):
            alignments[i, j] = t[(en[j], fr[i])]
    sns.heatmap(alignments, cmap='PuBuGn', annot=True)
    _, _ = plt.yticks(np.arange(len(fr))+0.5, fr, rotation=0, fontsize=10)
    _, _ = plt.xticks(np.arange(len(en))+0.5, en, rotation=30, fontsize=10)
```

[38]:
```
en = "NULL tom loves chocolate .".split()
fr = "tom adore le chocolat .".split()
visualize_trans_prob(en, fr, hard_t)
```



[39]:
```
plt.figure(figsize=(8, 6))
en = "NULL it was a very exciting game .".split()
fr = "c'était un jeu vraiment très excitant .".split()
visualize_trans_prob(en, fr, hard_t)
```



1.2 Soft EM algorithm

1.2.1 Question 4 (10 points)

1. Implement `soft_EM` function which runs one `Expectation/Maximization` iteration.
2. Run the training code

[40]:
```
def soft_EM(en_sents, fr_sents, t):
    '''
    params:
        en_sents: English sentences, list
        fr_sents: foreign sentences, list
    return:
        t: updated parameters, dictionary
```

```
    '''
    new_t = t
    ### YOUR CODE HERE

    # Expectation
    soft_count = defaultdict(lambda: defaultdict(int))
    for en_sent, fr_sent in zip(en_sents, fr_sents):
        # fix french word to find best english word alignment
        for fr_word in fr_sent:
            candidate_prob = np.zeros(len(en_sent))
            for i, en_word in enumerate(en_sent):
                candidate_prob[i] = t[(en_word, fr_word)]
            candidate_prob /= np.sum(candidate_prob)
            for i, en_word in enumerate(en_sent):
                soft_count[en_word][fr_word] += candidate_prob[i]

    # Maximization
    for en_word in soft_count:
        count_en = sum(soft_count[en_word].values())
        for fr_word in soft_count[en_word]:
            count_en_fr = soft_count[en_word][fr_word]
            new_t[(en_word, fr_word)] = count_en_fr/count_en
    ### END OF YOUR CODE
    return new_t
```

Let us check the algorithm first using the objective value.

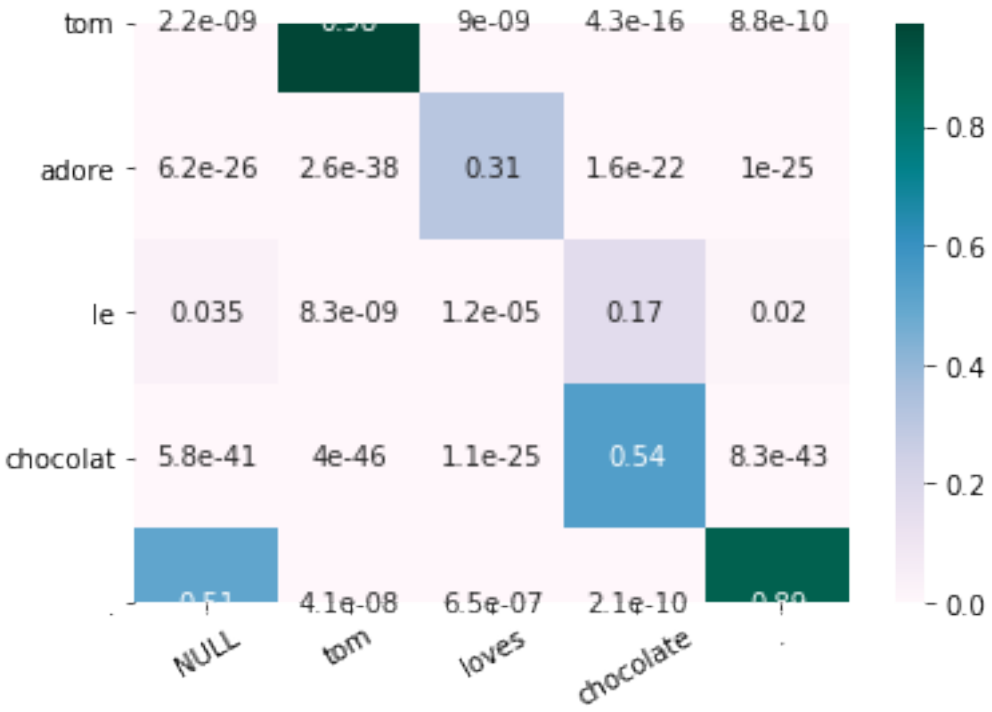
[41]:
```
#################################################################
# Randomly initialized the probabilities under the contraint
#################################################################
soft_t = init(list(align_counts.keys()))

###################
# Hard EM training
###################
iteration = 0
while iteration < 15:

    logp = corpus_log_prob(english_sents, french_sents, soft_t)
    soft_t = soft_EM(english_sents, french_sents, soft_t)
    print('Objective Function:', round(logp, 5))
    iteration += 1
```

```
Objective Function: -5649986.83975
Objective Function: -2653687.48168
Objective Function: -1823981.97094
Objective Function: -1578300.81918
Objective Function: -1496222.98341
```

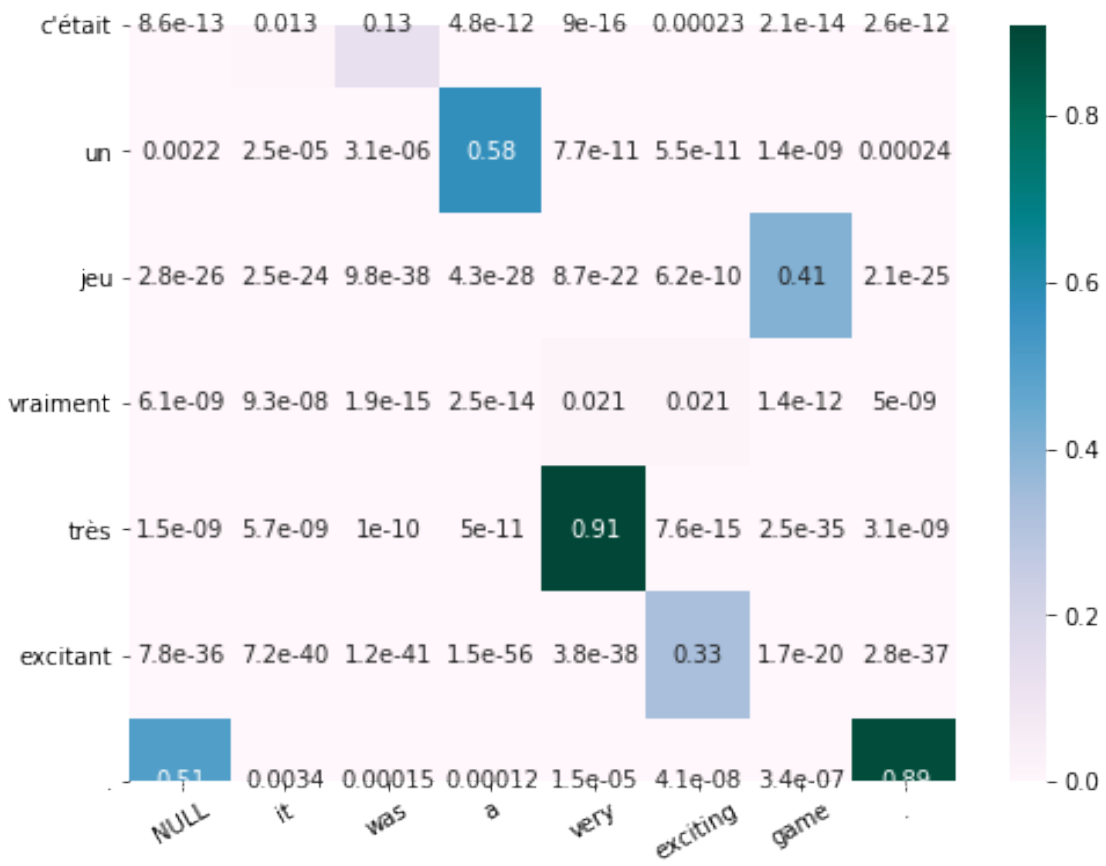
```
Objective Function: -1462402.9808
Objective Function: -1446188.11098
Objective Function: -1437544.25189
Objective Function: -1432513.75957
Objective Function: -1429366.48503
Objective Function: -1427276.2123
Objective Function: -1425816.21761
Objective Function: -1424751.53654
Objective Function: -1423949.94431
Objective Function: -1423331.0484
```

1.2.2 Visualization

[42]:
```
en = "NULL tom loves chocolate .".split()
fr = "tom adore le chocolat .".split()
visualize_trans_prob(en, fr, soft_t)
```



[43]:
```
plt.figure(figsize=(8,6))
en = "NULL it was a very exciting game .".split()
fr = "c'était un jeu vraiment très excitant .".split()
visualize_trans_prob(en, fr, soft_t)
```


c'était
un
jeu
vraiment
très
excitant
.
8.6e-13 0.013 0.13 4.8e-12 9e-16 0.00023 2.1e-14 2.6e-12
0.0022 2.5e-05 3.1e-06 0.58 7.7e-11 5.5e-11 1.4e-09 0.00024
2.8e-26 2.5e-24 9.8e-38 4.3e-28 8.7e-22 6.2e-10 0.41 2.1e-25
6.1e-09 9.3e-08 1.9e-15 2.5e-14 0.021 0.021 1.4e-12 5e-09
1.5e-09 5.7e-09 1e-10 5e-11 0.91 7.6e-15 2.5e-35 3.1e-09
7.8e-36 7.2e-40 1.2e-41 1.5e-56 3.8e-38 0.33 1.7e-20 2.8e-37
0.51 0.0034 0.00015 0.00012 1.5e-05 4.1e-08 3.4e-07 0.89
NULL
it
was
a
very
exciting
game
.
0.8
0.6
0.4
0.2
0.0

[ ]: