CS114 Assignment 6

author: Ziyu Liu ziyuliu@brandeis.edu

Most requirement writings for the write-up can be found inside the ipython notebook.

SO I will just copy paste them here:

Task1:

ppmi vs original count matrix's representation of "dogs"

```
ppmi[word_dict['dogs']]
```

```
array([2.4841, 0.    , 0.    , 0.    , 0.    , 3.112 , 1.7882])
```

```
count_matrix[word_dict['dogs']]
```

```
array([9., 0., 0., 0., 0., 3., 1.])
```

Here we can see that the ppmi vector gives a higher weight to "bite" and "like" over "the".

This is right because "the" can (in both real English and this small corpus) appear together with **almost any noun** while the later two verbs are more informative on the meaning of the word "dog".

Distance calculated by using ppmi

```
calc_dist('women', 'men', ppmi)
calc_dist('women', 'dogs', ppmi)
calc_dist('dogs', 'men', ppmi)
calc_dist('feed', 'like', ppmi)
calc_dist('feed', 'bite', ppmi)
calc_dist('like', 'bite', ppmi)
```

```
The distance between women and men is 1.107
The distance between women and dogs is 4.328
The distance between dogs and men is 4.128
```

```
The distance between feed and like is 2.334
The distance between feed and bite is 5.299
The distance between like and bite is 3.624
```

We can see the distances calculated agree with our intuition. Words with similar meaning do have smaller distance.

Distance calculated by using SVD-reduced ppmi

```
calc_dist('women', 'men', reduced_ppmi)
calc_dist('women', 'dogs', reduced_ppmi)
calc_dist('dogs', 'men', reduced_ppmi)
calc_dist('feed', 'like', reduced_ppmi)
calc_dist('feed', 'bite', reduced_ppmi)
calc_dist('like', 'bite', reduced_ppmi)
```

```
The distance between women and men is 0.851
The distance between women and dogs is 2.704
The distance between dogs and men is 2.185
The distance between feed and like is 2.052
The distance between feed and bite is 5.101
The distance between like and bite is 3.558
```

We can see the reduced ppmi matrix still keep the information needed.

Task 2

Test the accuracy of both approaches

```
from scipy.spatial.distance import cosine, euclidean
google_cosine_count, google_euclidean_count, ppmi_cosine_count,
ppmi_euclidean_count = 0, 0, 0, 0
for index, row in synonym_dataset.iterrows():
    word = row['given_word']
    choices = [row['choice' + num] for num in list('12345')]
    correct_answer = row['correct_answer']

    google_cosine_dists = [google_get_dist(word, candidate, cosine) for candidate
in choices]
    google_cosine_answer = choices[np.argmin(google_cosine_dists)]
    if google_cosine_answer == correct_answer:
        google_cosine_count += 1
    google_euclidean_dists = [google_get_dist(word, candidate, euclidean) for
candidate in choices]
    google_euclidean_answer = choices[np.argmin(google_euclidean_dists)]
    if google_euclidean_answer == correct_answer:
```

```
            google_euclidean_count += 1

        ppmi_cosine_dists = [ppmi_get_dist(word, candidate, cosine) for candidate in
choices]
        ppmi_cosine_answer = choices[np.argmin(ppmi_cosine_dists)]
        if ppmi_cosine_answer == correct_answer:
            ppmi_cosine_count += 1
        ppmi_euclidean_dists = [ppmi_get_dist(word, candidate, euclidean) for
candidate in choices]
        ppmi_euclidean_answer = choices[np.argmin(ppmi_euclidean_dists)]
        if ppmi_euclidean_answer == correct_answer:
            ppmi_euclidean_count += 1


print("Google's accuracy on the dataset:           %.3f using cosine, %.3f using
euclidean"\
      %(google_cosine_count / 1000, google_euclidean_count / 1000))
print("Classic Approach's accuracy on the dataset: %.3f using cosine, %.3f using
euclidean"\
      %(ppmi_cosine_count / 1000, ppmi_euclidean_count / 1000))
```

```
Google's accuracy on the dataset:           0.712 using cosine, 0.545 using
euclidean
Classic Approach's accuracy on the dataset: 0.542 using cosine, 0.542 using
euclidean
```

Result

Accuracy	cosine	euclidean
google	0.712	0.545
COMPOSES	0.542	0.542

The SAT Questions

I think the relationship between words can be best represented by the difference between their vectors. This is mentioned in the slides, and is natural because this represents how to 'move' inside the word-vector space to get to another word from a given word.

For the similarity metric, cosine should be better as it will not be affected by the length of the vector. This is especially critical as we are calculating this metric on the difference vectors. Actually I tested using l2(euclidean) distance here, and got a 0.328 accuracy, much lower than the **0.440** I got using cosine.

```
passed_question = 0
correct_count = 0
```

```
for question in sat_questions:
    correct_answer = ord(question[-1]) - ord('a')
    sample1, sample2 = question[0]
    candidates = question[1:-1]
    dists = [similarity(sample1, sample2, word1, word2) for word1, word2 in
candidates]
    if any([x == np.inf for x in dists]):
        passed_question += 1
        continue
    answer = np.argmin(dists)
    if answer == correct_answer:
        correct_count += 1

print("With %d out of %d questions including out-of-voc words are passed, %d out
of %d are answered correctly.\nFinal accuracy is %.3f"\
      %(passed_question, len(sat_questions), correct_count, len(sat_questions) -
passed_question,\
        correct_count / (len(sat_questions) - passed_question)))
```

```
With 115 out of 374 questions including out-of-voc words are passed, 114 out of
259 are answered correctly.
Final accuracy is 0.440
```