```
#%% md
3.1 (8 points) Implement the capability of computing PMIs. Use your code to calculate PMIs for w = 3 when using vocab-15kws.txt to populate V and vocab-5k.txt to populate VC. Note that since we are using different vocabularies for center words and context words, pmi(a, b) will not necessarily equal pmi(b, a) (though they will be similar). (If there is a word in V that has no counts, the numerator and denominator for all of its PMI values will be zero, so you can just define all such PMIs to be zero.) For center word x = “coffee”, print the 10 context words with the largest PMIs and the 10 context words with the smallest PMIs. Print both the words and the PMI values
#%%
from scipy.stats import rankdata

# function to compute Spearman correlation.
def spearman_rank_correlation(predicted_similarities, actual_scores):

    predicted = [predicted_similarities[pair] for pair in actual_scores if pair in predicted_similarities]
    actual = [actual_scores[pair] for pair in actual_scores if pair in predicted_similarities]

    predicted_ranks = rankdata(predicted)
    actual_ranks = rankdata(actual)

    n = len(predicted_ranks)
    if n == 0:
        return None

    rank_differences_squared = [(pred_r - act_r) ** 2 for pred_r, act_r in zip(predicted_ranks, actual_ranks)]
    spearman_rho = 1 - (6 * sum(rank_differences_squared)) / (n * (n**2 - 1))

```

```
22     return spearman_rho
23 #%%
24 from collections import defaultdict
25 import math
26 
27 def find_indexes(arr, words_set):
28     return [(i, w) for i, w in enumerate(arr) if w in
    words_set]
29 
30 
31 with open("/vocab-15kws.txt") as f:
32     vocab = f.read().split()
33 
34 with open("/vocab-5k.txt") as f1:
35     vocab_context = f1.read().split()
36 
37 V = set(vocab)
38 Vc = set(vocab_context)
39 
40 
41 my_dict = defaultdict(int)
42 
43 
44 with open("/wiki-1percent.txt", "r") as file:
45     for line in file:
46 
47         words = line.split()
48         length = len(words)
49 
50         word_indexes = find_indexes(words, V)
51 
52 
53         for idx, word in word_indexes:
54             ind_l = max(0, idx - 3)
55             ind_H = min(length, idx + 4)
56             context_window = words[ind_l:idx] + words
   [idx + 1:ind_H]
57             for context_word in context_window:
58                 if context_word in Vc:
59                     key = (word, context_word)
60                     my_dict[key] += 1
```

```
61
62 # To add #(x,y) for all x and y.
63 total_count_N = sum(my_dict.values())
64
65 # To compute joint probabilities of (x,y).
66 joint_probabilities = defaultdict(float)
67 for (word, context_word), count in my_dict.items():
68     joint_probabilities[(word, context_word)] =
   count / total_count_N
69
70 # To compute partial probability of x.
71 partial_probability_x = defaultdict(float)
72 for word in V:
73     partial_probability_x[word] = sum(
   joint_probabilities[(word, context_word)] for
   context_word in Vc)
74
75 # To compute partial probability of x.
76 partial_probability_y = defaultdict(float)
77 for context_word in Vc:
78     partial_probability_y[context_word] = sum(
   joint_probabilities[(word, context_word)] for word
   in V)
79
80 # To calculate PMI.
81 pointwise_mutual_information = defaultdict(float)
82 for (word, context_word), joint_prob in
   joint_probabilities.items():
83     partial_prob_x = partial_probability_x[word]
84     partial_prob_y = partial_probability_y[
   context_word]
85
86     if joint_prob > 0 and partial_prob_x > 0 and
   partial_prob_y > 0:
87         pointwise_mutual_information[(word,
   context_word)] = math.log2(joint_prob / (
   partial_prob_x * partial_prob_y))
88     else:
89         pointwise_mutual_information[(word,
   context_word)] = 0
90
```

```
 91 coffee_pmi = {context_word: pmi for (word,
    context_word), pmi in pointwise_mutual_information.
    items() if word == 'coffee'}
 92
 93 sorted_pmi = sorted(coffee_pmi.items(), key=lambda
    item: item[1], reverse=True)
 94
 95 print("Highest PMI context words for 'coffee':-")
 96 for word, pmi in sorted_pmi[:10]:
 97     print(f"{word}:-> {pmi}")
 98
 99 print("\nLowest PMI context words for 'coffee':-")
100 for word, pmi in sorted_pmi[-10:]:
101     print(f"{word}:-> {pmi}")
102
103 #%% md
104 3.2 (6 points) Now, define word vectors using PMI.
    That is, the word vector for a word x ∈ V has an
105 entry for each word y ∈ VC with value given by pmi(x
    , y). As above, use w = 3, vocab-15kws.txt
106 to populate V , and vocab-5k.txt to populate VC.
    Evaluate (EVALWS) your PMI-based word vectors
107 and report your results.
108
109 #%%
110 from collections import defaultdict
111 import math
112
113 word_vectors = defaultdict(dict)
114
115 # Load word vectors with PMI value.
116 for (word, context_word), pmi in
    pointwise_mutual_information.items():
117     word_vectors[word][context_word] = pmi
118 #%% md
119 For men dataset
120 #%%
121 from collections import defaultdict
122 import math
123
124 word_pairs = []
```

```
125
126 word_pair_scores = {}
127
128 # To find word pairs and scores from given dataset.
129 with open("/men.txt") as f3:
130     next(f3)
131
132     for line in f3:
133         word1, word2, score = line.split()
134         pair = (word1, word2)
135         word_pairs.append(pair)
136         word_pair_scores[pair] = float(score)
137 #%%
138 import math
139 from collections import defaultdict
140
141 # Function to compute cosine similarity.
142 def cosine_similarity(vec1, vec2):
143
144     dot_product = sum(vec1[word] * vec2[word] for
    word in vec1 if word in vec2)
145
146     norm_vec1 = math.sqrt(sum(value ** 2 for value
    in vec1.values()))
147
148     norm_vec2 = math.sqrt(sum(value ** 2 for value
    in vec2.values()))
149
150     if norm_vec1 == 0 or norm_vec2 == 0:
151         return 0.0
152
153     return dot_product / (norm_vec1 * norm_vec2)
154
155 cosine_similarities_men = {}
156
157 for (word1, word2) in word_pairs:
158     if word1 in word_vectors and word2 in
    word_vectors:
159
160         vec1 = word_vectors[word1]
161         vec2 = word_vectors[word2]
```

```
162
163         similarity = cosine_similarity(vec1, vec2)
164         cosine_similarities_men[(word1, word2)] =
    similarity
165 #%%
166 spearman_rho = spearman_rank_correlation(
    cosine_similarities_men, word_pair_scores)
167 print(f"Spearman's ρ for MEN dataset for w = 3: {
    spearman_rho}")
168 #%% md
169 For simlex-999 dataset
170 #%%
171 word_pairs = []
172 word_pair_scores = {}
173
174 with open("/simlex-999.txt") as f4:
175     next(f4)
176     for line in f4:
177         word1, word2, simlex999 = line.split()
178         pair = (word1, word2)
179         word_pairs.append(pair)
180         word_pair_scores[pair] = float(simlex999)
181 #%%
182 import math
183 from collections import defaultdict
184
185 cosine_similarities_simlex999 = {}
186
187 for (word1, word2) in word_pairs:
188     if word1 in word_vectors and word2 in
    word_vectors:
189         vec1 = word_vectors[word1]
190         vec2 = word_vectors[word2]
191
192         similarity = cosine_similarity(vec1, vec2)
193
194         cosine_similarities_simlex999[(word1, word2
    )] = similarity
195 #%%
196 spearman_rho = spearman_rank_correlation(
    cosine_similarities_simlex999, word_pair_scores)
```

```
197 print(f"Spearman's ρ for simlex-999 dataset for w =
    3: {spearman_rho}")
```