```
#%% md
4.1 (8 points) Evaluate the word vectors (EVALWS) corresponding to the three ways of computing vectors (counts, IDF, and PMI), three values of w (1, 3, and 6), and two context vocabularies
(vocab-15kws.txt and vocab-5k.txt). For all cases, use vocab-15kws.txt for V . Report the
results (there should be 36 correlations in all) and describe your findings. What happens as window
size changes for different methods of creating word vectors? What happens when context vocabulary
changes? Why do you think you observe the trends you see? Do you see the same trends for MEN and
SimLex or do they differ?

For Count method
#%%
from collections import defaultdict
import math
from scipy.stats import rankdata

# Function to find indices for middle word.
def find_indexes(arr, words_set):
    return [(i, w) for i, w in enumerate(arr) if w in words_set]

# Function to find count for word vectors.
def word_vector_count(w, vocab, vocab_context, lines):
    V = set(vocab)
    Vc = set(vocab_context)

    my_dict = defaultdict(int)

    for line in lines:
        words = line.split()
        length = len(words)
        word_indexes = find_indexes(words, V)

        for idx, word in word_indexes:
            ind_l = max(0, idx - w)
```

```
            ind_h = min(length, idx + w + 1)
            context_window = words[ind_l:idx] + words
[idx + 1:ind_h]

            for context_word in context_window:
                if context_word in Vc:
                    my_dict[(word, context_word)] +=
1

    return my_dict
#%%
from collections import defaultdict
import math
from scipy.stats import rankdata
# Function to deduce word pairs and their scores from
 given dataset.
def make_word_pairs(path):
  word_pairs = []

  word_pair_scores = {}
  with open(path) as f3:
      next(f3)
      for line in f3:
          word1, word2, score = line.split()
          pair = (word1, word2)
          word_pairs.append(pair)

          word_pair_scores[pair] = float(score)

  return word_pairs, word_pair_scores
#%%
from collections import defaultdict
import math
from scipy.stats import rankdata

# Function to calculate cosine similarity.
def cosine_similarity(vec1, vec2):
    dot_product = sum(vec1[word] * vec2.get(word, 0)
for word in vec1)
    norm_vec1 = math.sqrt(sum(value ** 2 for value in
 vec1.values()))
```

```
 69     norm_vec2 = math.sqrt(sum(value ** 2 for value
    in vec2.values()))
 70 
 71     if norm_vec1 == 0 or norm_vec2 == 0:
 72         return 0.0
 73 
 74     return dot_product / (norm_vec1 * norm_vec2)
 75 #%%
 76 from collections import defaultdict
 77 import math
 78 from scipy.stats import rankdata
 79 
 80 
 81 # Function to calculate Spearman correlation.
 82 def spearman_rank_correlation(predicted_similarities
    , actual_scores):
 83     predicted = [predicted_similarities[pair] for
    pair in actual_scores if pair in
    predicted_similarities]
 84     actual = [actual_scores[pair] for pair in
    actual_scores if pair in predicted_similarities]
 85 
 86     if not predicted:
 87         return None
 88 
 89     predicted_ranks = rankdata(predicted)
 90     actual_ranks = rankdata(actual)
 91 
 92     n = len(predicted_ranks)
 93     rank_differences_squared = [(pred_r - act_r) **
    2 for pred_r, act_r in zip(predicted_ranks,
    actual_ranks)]
 94 
 95     return 1 - (6 * sum(rank_differences_squared
    )) / (n * (n**2 - 1))
 96 #%%
 97 from collections import defaultdict
 98 import math
 99 from scipy.stats import rankdata
100 
101 def load_file(path):
```

```
    with open(path, "r") as file:
        return file.read().splitlines()

def count_lines(Vc, lines):
    total_lines = 0
    context_word_line_counts = defaultdict(int)

    for line in lines:
        total_lines += 1
        words = set(line.split())
        for word in words:
            if word in Vc:
                context_word_line_counts[word] += 1

    return total_lines, context_word_line_counts

# Defining context window size, different paths, different context word vocab and different methods.
context_window_sizes = [1, 3, 6]
word_pairs_paths = ["/men.txt", "/simlex-999.txt"]
context_vocab_paths = ["/vocab-15kws.txt", "/vocab-5k.txt"]
methods = {"count", "tf-idf", "pmi"}

wiki_lines = load_file("/wiki-1percent.txt")

for path in word_pairs_paths:
    word_pairs, word_pair_scores = make_word_pairs(path)

    for context_vocab in context_vocab_paths:
        vocab_context = load_file(context_vocab)

        for w in context_window_sizes:
            word_vectors = defaultdict(dict)
            vocab = load_file("/vocab-15kws.txt")
            my_dict = word_vector_count(w, vocab, vocab_context, wiki_lines)
            for (word, context_word), count in my_dict.items():
                word_vectors[word][context_word] =
```

```
count


            cosine_similarities = {pair: cosine_similarity(word_vectors.get(pair[0], {}), word_vectors.get(pair[1], {})) for pair in word_pairs}

            spearman_rho = spearman_rank_correlation(cosine_similarities, word_pair_scores)
            print(f"Spearman's ρ for path = {path}, context_vocab = {context_vocab}, w = {w}, method = count: {spearman_rho}")

#%% md
4.1) For TF-IDF Method
#%%
for path in word_pairs_paths:
    word_pairs, word_pair_scores = make_word_pairs(path)

    for context_vocab in context_vocab_paths:
        vocab_context = load_file(context_vocab)

        for w in context_window_sizes:
            word_vectors = defaultdict(dict)
            vocab = load_file("/vocab-15kws.txt")
            my_dict = word_vector_count(w, vocab, vocab_context, wiki_lines)
            total_lines, count_context_word_lines = count_lines(set(vocab_context), wiki_lines)
            for (word, context_word), count in my_dict.items():
                idf = (total_lines / count_context_word_lines[context_word]) if count_context_word_lines[context_word] > 0 else 0
                word_vectors[word][context_word] = count * idf


```

```
165             cosine_similarities = {pair:
    cosine_similarity(word_vectors.get(pair[0], {}),
    word_vectors.get(pair[1], {}))
166                                     for pair in
    word_pairs}
167 
168             spearman_rho = spearman_rank_correlation
    (cosine_similarities, word_pair_scores)
169             print(f"Spearman's ρ for path = {path},
    context_vocab = {context_vocab}, w = {w}, method =
    tf-idf: {spearman_rho}")
170 #%% md
171 4.1) For PMI method
172 #%%
173 for path in word_pairs_paths:
174     word_pairs, word_pair_scores = make_word_pairs(
    path)
175 
176     for context_vocab in context_vocab_paths:
177         vocab_context = load_file(context_vocab)
178 
179         for w in context_window_sizes:
180             word_vectors = defaultdict(dict)
181             vocab = load_file("/vocab-15kws.txt")
182             my_dict = word_vector_count(w, vocab,
    vocab_context, wiki_lines)
183             total_count_N = sum(my_dict.values())
184             joint_probabilities = {pair: count /
    total_count_N for pair, count in my_dict.items()}
185 
186             partial_prob_x = {word: sum(
    joint_probabilities.get((word, cw), 0) for cw in set
    (vocab_context)) for word in set(vocab)}
187             partial_prob_y = {cw: sum(
    joint_probabilities.get((word, cw), 0) for word in
    set(vocab)) for cw in set(vocab_context)}
188 
189             for (word, context_word), joint_prob in
    joint_probabilities.items():
190               pmi = math.log2(joint_prob / (
    partial_prob_x[word] * partial_prob_y[context_word
```

```
190 ])) if joint_prob > 0 else 0
191                 word_vectors[word][context_word] = pmi
192 
193 
194             cosine_similarities = {pair:
    cosine_similarity(word_vectors.get(pair[0], {}),
    word_vectors.get(pair[1], {}))
195                                    for pair in
    word_pairs}
196 
197             spearman_rho = spearman_rank_correlation
    (cosine_similarities, word_pair_scores)
198             print(f"Spearman's ρ for path = {path},
    context_vocab = {context_vocab}, w = {w}, method =
    pmi: {spearman_rho}")
```