1.1 (12 points) Implement distributional counting as described above for a provided w, V , and VC. Submit your code.

```
from collections import defaultdict

# function to find indexes of words in a set within an array.
def find_indexes(arr, words_set):
    """Finds all indexes of words in a set within an array using list comprehension."""
    return [(i, w) for i, w in enumerate(arr) if w in words_set]

# function to word vectors.
def word_vector_count(w):
  with open("/vocab-15kws.txt") as f:
    vocab = f.read().split()

  with open("/vocab-5k.txt") as f1:
    vocab_context = f1.read().split()

  V = set(vocab)
  Vc = set(vocab_context)

  my_dict = defaultdict(int)

  with open("/wiki-1percent.txt", "r") as file:
    for line in file:
        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:
                    key = (word, context_word)
                    my_dict[key] += 1



  return my_dict
```

1.2 (6 points) Using vocab-15kws.txt to populate V and vocab-5k.txt to populate VC, use your code to report #(x, y) for the pairs in the following table for w = 3.

```
from collections import defaultdict

target_pairs = {('chicken', 'the'), ('chicken', 'wings'), ('chicago', 'chicago'), ('coffee', 'the'), ('coffee', 'cup'), ('coffee', 'cof

my_dict = word_vector_count(3)

print("{:<20} {:<20}".format('tuple', 'count'))

for key, value in my_dict.items():
    if key in target_pairs:
        print("{:<20} {:<20}".format(str(key), value))
```

```
tuple                count
('coffee', 'the')    95
('chicago', 'chicago') 38
('chicken', 'the')   52
('coffee', 'cup')    10
('chicken', 'wings') 6
('coffee', 'coffee') 4
```

1.2 (6 points) Using vocab-15kws.txt to populate V and vocab-5k.txt to populate VC, use your code to report #(x, y) for the pairs in the following table for w = 6.

```
from collections import defaultdict

target_pairs = {('chicken', 'the'), ('chicken', 'wings'), ('chicago', 'chicago'), ('coffee', 'the'), ('coffee', 'cup'), ('coffee', 'cof

my_dict = word_vector_count(6)
```

```
print("{:<20} {:<20}".format('tuple', 'count'))

for key, value in my_dict.items():
    if key in target_pairs:
        print("{:<20} {:<20}".format(str(key), value))
```

```
tuple                count
('coffee', 'the')    201
('chicago', 'chicago') 122
('chicken', 'the')   103
('coffee', 'cup')    14
('coffee', 'coffee') 36
('chicken', 'wings') 7
```

Function to get word pairs from given datasets i.e. men.txt and simlex-999.txt.

```
import math
from collections import defaultdict

# function to find pair of words and scores from given datasets.
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
```

Function to calculate cosine similarity.

```
import math
from collections import defaultdict

# function to find cosine similarity.
def cosine_similarity(vec1, vec2):

    dot_product = sum(vec1[word] * vec2[word] for word in vec1 if word in vec2)
    norm_vec1 = math.sqrt(sum(value ** 2 for value in vec1.values()))
    norm_vec2 = math.sqrt(sum(value ** 2 for value in vec2.values()))
    if norm_vec1 == 0 or norm_vec2 == 0:
        return 0.0
    return dot_product / (norm_vec1 * norm_vec2)
```

Function to calculate Spearman correlation.

```
from scipy.stats import rankdata

# function to find 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))

    return spearman_rho
```

1.3 (6 points) Using w = 3 (and again using vocab-15kws.txt for V and vocab-5k.txt for VC), evaluate your count-based word vectors using EVALWS and report your results on MEN.

```
import math
from collections import defaultdict

cosine_similarities_men = {}

word_pairs, word_pair_scores = make_word_pairs("/men.txt")

word_vectors = defaultdict(dict)

my_dict = word_vector_count(3)

for (word, context_word), count in my_dict.items():
    word_vectors[word][context_word] = count

for (word1, word2) in word_pairs:
    if word1 in word_vectors and word2 in word_vectors:

        vec1 = word_vectors[word1]
        vec2 = word_vectors[word2]

        similarity = cosine_similarity(vec1, vec2)

        cosine_similarities_men[(word1, word2)] = similarity


spearman_rho = spearman_rank_correlation(cosine_similarities_men, word_pair_scores)
print(f"Spearman's ρ for MEN dataset for w = 3: {spearman_rho}")
```

```
Spearman's ρ for MEN dataset for w = 3: 0.22536738248526467
```

1.3 (6 points) Using w = 3 (and again using vocab-15kws.txt for V and vocab-5k.txt for VC), evaluate your count-based word vectors using EVALWS and report your results on simlex-999.

```
import math
from collections import defaultdict

cosine_similarities_simlex999 = {}

word_pairs, word_pair_scores = make_word_pairs("/simlex-999.txt")

for (word1, word2) in word_pairs:
    if word1 in word_vectors and word2 in word_vectors:

        vec1 = word_vectors[word1]
        vec2 = word_vectors[word2]

        similarity = cosine_similarity(vec1, vec2)

        cosine_similarities_simlex999[(word1, word2)] = similarity


spearman_rho = spearman_rank_correlation(cosine_similarities_simlex999, word_pair_scores)
print(f"Spearman's ρ for simlex-999 dataset for w = 3: {spearman_rho}")
```

```
Spearman's ρ for simlex-999 dataset for w = 3: 0.059538612941463454
```

2.1 (10 points) Extend your implementation to be able to compute IDF-based word vectors using Eq. 1. Using w = 3, vocab-15kws.txt to populate V , and vocab-5k.txt to populate VC, evaluate (EVALWS) your IDF-based word vectors and report your results.

For men dataset

```
import math
from collections import defaultdict

cosine_similarities_men = {}

word_pairs, word_pair_scores = make_word_pairs("/men.txt")

with open("/vocab-5k.txt") as f1:
    vocab_context = f1.read().split()

Vc = set(vocab_context)

total_lines = 0
context_word_line_counts = defaultdict(int)

with open("/wiki-1percent.txt", "r") as file:
    for line in file:
        total_lines += 1
        words = set(line.split())
```

```
        for word in words:
            if word in Vc:
                context_word_line_counts[word] += 1

word_vectors = defaultdict(dict)

my_dict = word_vector_count(3)

for (word, context_word), count in my_dict.items():
    count_context_word = context_word_line_counts[context_word]
    word_vectors[word][context_word] = count*(total_lines/count_context_word)

for (word1, word2) in word_pairs:
    if word1 in word_vectors and word2 in word_vectors:

        vec1 = word_vectors[word1]
        vec2 = word_vectors[word2]

        similarity = cosine_similarity(vec1, vec2)

        cosine_similarities_men[(word1, word2)] = similarity


spearman_rho = spearman_rank_correlation(cosine_similarities_men, word_pair_scores)

# Print the Spearman's ρ result
print(f"Spearman's ρ for MEN dataset for w = 3: {spearman_rho}")
```

```
Spearman's ρ for MEN dataset for w = 3: 0.47297401866377986
```

For simlex-999 dataset

```
import math
from collections import defaultdict

cosine_similarities_simlex999 = {}

word_pairs, word_pair_scores = make_word_pairs("/simlex-999.txt")

for (word1, word2) in word_pairs:
    if word1 in word_vectors and word2 in word_vectors:

        vec1 = word_vectors[word1]
        vec2 = word_vectors[word2]

        similarity = cosine_similarity(vec1, vec2)

        cosine_similarities_simlex999[(word1, word2)] = similarity


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

```
Spearman's ρ for simlex-999 dataset for w = 3: 0.15982142857142856
```