I am implementing BM25, DirichletLM, and Pivoted Normalization for my search engine homework. My unit tests are failing with small floating-point differences. Can you carefully check my implementation against the given formulas and suggest where the normalization or averaging may be off? It looks like your implementation is very close. The small differences come from how you normalize the document length and how you apply logarithms.For BM25, make sure you are using mean\_document\_length computed from the unfiltered token count (before stopwords or low-frequency removals).For DirichletLM, ensure that you are adding the normalized document length term outside the query-term loop. That term is independent of the query and should be included once per document.For Pivoted Normalization, double-check that the denominator uses 1 - b + b \* (doc\_length / avgdl) exactly as written in the formula, without any additional scaling.If you fix these, your scores should align with the autograder within the required tolerance.

I have written a BasicInvertedIndex and PositionalInvertedIndex class for indexing documents. My autograder test shows that BM25 average document length is slightly off. Can you review my Indexer.create\_index function and suggest corrections so that the mean document length matches the expected formula?

The main issue is in how you compute mean\_document\_length. Currently, you are updating it incrementally inside add\_doc. This causes it to be based on filtered tokens. The autograder expects you to use the total token count before filtering stopwords or frequency thresholds.To fix this:Do a first pass to count all tokens (before filtering) and store total\_token\_count.After all documents are added, computeindex.statistics['mean\_document\_length'] = (

index.statistics['total\_token\_count'] / index.statistics['number\_of\_documents'

This should resolve the BM25 test mismatchI am trying to run my Python homework on the GreatLakes HPC cluster at the University of Michigan. I get a "numpy dtype size changed" error when running with the default Python module. How can I set up a conda environment or load compatible Python and spacy/nltk libraries to make my code run?

This error happens because the system Python module and SciPy/NumPy builds are incompatible. On GreatLakes, the solution is to create your own Conda environment. For examplemodule load python3.11-anaconda/2024.02conda create -n si650 python=3.11

conda activate si6conda install numpy scipy nltk spacy matplotlib tqdm

python -m spacy download en\_core\_web\_smThen run your code with python problem3.py inside this environment. This avoids the system SciPy mismatch.I am comparing tokenizer runtimes (SplitTokenizer, RegexTokenizer, SpaCyTokenizer) for 1000 Wikipedia documents. When I run on GreatLakes, I get "NameError: spacy not defined". Can you show me how to modify document\_preprocessor.py so that spacy is imported only when SpaCyTokenizer is used?Yes — the issue is that import spacy is not at the top of the file, so the symbol is missing. To fix it, move the import into the class:

class SpaCyTokenizer(Tokenizer):

def \_\_init\_\_(self):

import spacy

self.spacy = spacy

self.nlp = spacy.load("en\_core\_web\_sm") def tokenize(self, text: str) -> list[str]:

return [token.text for token in self.nlp(text)]This way, spacy is only imported if SpaCyTokenizer is instantiated. It avoids errors when SpaCy is not installed and fixes the NameError.Please write a well-structured version of my homework function with clear comments following the provided formula, so that the implementation is easy to debug and consistent with the homework instructions.

Here’s a cleaned-up version of DirichletLM.score with comments:

def score(self, docid, doc\_word\_counts, query\_word\_counts):

"""

Dirichlet Language Model scoring function.

Formula:

sum\_{w in q} c\_q(w) \* log( (c\_d(w) + mu \* P\_ml(w)) / (|d| + mu) )

+ length normalization term

"""

dl = self.index.get\_doc\_metadata(docid).get("length", 1)

collection\_length = self.index.statistics["stored\_total\_token\_count"]

score = 0.0

for term, qf in query\_word\_counts.items():

cd = doc\_word\_counts.get(term, 0)

cf = self.index.statistics["vocab"].get(term, 0)

p\_ml = cf / collection\_length if collection\_length > 0 else 0.0prob = (cd + self.mu \* p\_ml) / (dl + self.mu)

if prob > 0:

score += qf \* np.log(prob) # Normalized length term

avdl = self.index.statistics["mean\_document\_length"]

if avdl > 0:

score += np.log(dl / avdl)return scoreThis directly matches the provided formula, and places the length-normalization term outside the loop, as required.