**inverted\_index\_min=0.8\_max=0.99**

* pickle
* sample of data structure:

with open(“./inverted\_index\_min=0.8\_max=0.99”, “rb”) as f:

inverted\_index = pickle.load(f)

inverted\_index[“fantasi”] = {'1536771': 654, '1095121': 3, '43161998': 45, '41107568': 234, '40094032': 4, '1137060': 934,...}

* Notes:
  + keys are terms; all terms are **lowercase** and **stemmed**
  + values are dictionaries that map **work ids** to **raw** **term frequencies**
  + all terms occur in >=80% of the documents and <= 99% of the documents. It’s very easy to build an inverted index with different thresholds, so let me know if you want to experiment.
* **(min\_tf\_3)inverted\_index\_min=0.65\_max=0.99**
  + exactly the same, but terms that occurred ≤3 times across all the reviews for a work are ignored. this got rid of some of the weirder terms cropping up in our data, which allowed me to lower the min DF threshold to 65%
* **(min\_tf\_5)inverted\_index\_min=0.55\_max=0.99** 
  + the same, but terms that occurred ≤5 times across all reviews for a work are ignored. min DF threshold lowered to 55%

**work\_tf\_idf\_vectors\_min=0.8\_max=0.99\_NORMS**

* pickle
* sample of data structure:

with open("./work\_tf\_idf\_vectors\_min=0.8\_max=0.99", "rb") as f:

tf\_idf\_vectors = pickle.load(f)

tf\_idf\_vectors[“25696480”] = {'NORM': 183.39156245036628,'blew': 1.5149042580583758, 'push': 0.4388879101768791, '2015': 9.963187804758672, 'instanc': 1.366398696164463, ... }

* Notes:
  + every vector has an extra key, NORM that maps to the norm of this vector
  + keys are work ids (string value)
  + values are dictionaries mapping terms to **tf-idf values** (not term frequencies, as in the inverted index!)
    - I’ve been using **math.log(NUM\_WORKS/(1 + DF), 2)** for IDF
      * To get document frequency, I use len(inverted\_index[“<term>”])
      * NUM\_WORKS is 2,761 (or the length of the mapping of work\_tf\_idf\_vectors works too so you don’t have to hard code a value when computing cosine similarity)
  + again this, only includes terms that occur in >=80% of the documents and <= 99% of the documents
* **(min\_tf\_3)work\_tf\_idf\_vectors\_min=0.65\_max=0.99\_NORMS**
  + correspond to  **(min\_tf\_3)inverted\_index\_min=0.65\_max=0.99** above
* **(min\_tf\_5)work\_tf\_idf\_vectors\_min=0.55\_max=0.99\_NORMS**
  + correspond to **(min\_tf\_5)inverted\_index\_min=0.55\_max=0.99** above

**workid\_to\_book\_info**

* pickle
* sample of data structure

with open("./workid\_to\_book\_info", "rb") as f:

book\_info = pickle.load(f)

book\_info[“3360164”] = { 'title': 'Emma',

'url': 'https://www.goodreads.com/book/show/6969.Emma',

'image': 'https://images.gr-assets.com/books/1373627931m/6969.jpg',

'description': "'I never have been in love; it is not my way, or my

nature; and I do not think I ever shall.'\nBeautiful, clever, rich - and single - Emma Woodhouse is perfectly content with her life and sees no need for either love or marriage. Nothing, however, delights her more than interfering in the romantic lives of others. But when she ignores the warnings of her good friend Mr. Knightley and attempts to arrange a suitable match for her protegee Harriet Smith, her carefully laid plans soon unravel and have consequences that she never expected. With its imperfect but charming heroine and its witty and subtle exploration of relationships, Emma is often seen as Jane Austen's most flawless work.\nThis edition includes a new chronology and additional suggestions for further reading.",

'rating': '3.99',

'num\_ratings': 470489,

'author\_ids': ['1265'],

'author\_names': ['Jane Austen']}

* Notes:
  + keys are work ids (string value)

**titles\_authors\_workid\_img\_tuples**

* pickle
* sample of data structure:

with open("./titles\_authors\_workid\_img\_tuples", "rb") as f:

info\_tuples = pickle.load(f)

info\_tuples[<arbitrary\_index>] = ('Emma', ['Jane Austen'], '3360164',

'https://images.gr-assets.com/books/1373627931m/6969.jpg')

* Notes:
  + this is just a list, no particular order. Intended to be a lookup table for user-inputted book titles. Authors are listed because there are repeat titles.
  + authors will always be listed in alphabetical order
* Why isn’t this a hashmap? Since we would be looking for substrings of the keys, idk that it would actually speed us up. Similarly, alphabetizing it might help us do a binary search through the works, but if the user doesn’t input the first chunk of the title, then we have to do a linear search anyway...
* Honestly, it might just be easier to do this lookup using **workid\_to\_book\_info**, but I thought maybe this would be slightly faster