Pseudo code

STEP-1: Loading and Preprocessing

* Load Libraries
* Choose the NEWS GROUP & the number of documents
* Load Data
* Remove Headers, Footers and Quotes
* Remove stop words (“and”, “is”, “the” …etc.)
* Remove Punctuations
* Remove numbers – Numbers won’t make sense in the important concepts
* Tokenize each document
* Filter out the documents that are empty, if any.

Bod (Bag of documents) = that contains the list of documents, each document tokenized.

STEP-2: Create TF-IDF and apply SVD

* def unique (bag of docs):
* convert each document into a set and compute a union of all the documents(sets).
* Return the universal set that contains all the unique words

Wordset = unique (bag of docs)

Worddict=create a dictionary (key: unique words, value: 0)

* Def term\_doc\_matrix ():
  + Iterate over zip (bod, worddict)
  + Increment the value of each word when found in the document.
  + Return the worddict dictionary with the updated count values for each unique word.

Docterm = term\_doc\_matrix ()

# Docterm is the dataframe with word counts

* Def term\_freq (worddict, bod):
  + Create empty dictionary tfdict
  + Bodcount = length of bod
  + Tfdict[word] = count/bodcount # normalizing
  + Return tfdict

# compute term freq for all documents tf = word count/total number of documents

* Tfbod = empty list
* For I, j in zip (worddict, bod):
  + Append Term\_freq (I, j) to tfbod
* Def idf (worddict):
  + Create empty dictionary idfdict
  + N = length of worddict
  + Keys of idfdict = keys of worddict, value=0 for all keys
  + For doc in worddict:
    - For word, val in doc.items ():
      * Idfdict[word] +=1 if val is >0

idfdict will give us the count of documents that contain a particular word

* for word, v al in idfdict.items (): # iterating over key and val

#compute log (total num of documents/ documents that contain a particular word)

* idfs = idf(worddict) # applying the function of worddict
* def tfidf (tfbod, idfs):
  + multiply tf with idf for each term.
* Tfidf = computing tfidf for each document using the function tfidf
* Transposing tfidf to get the term document matrix from the document term matrix.
* L, S, R = Applying SVD on this term-document TF-IDF matrix.

STEP-3: Finding the perfect low rank approximation of the TF-IDF matrix

* Def top\_n\_pad\_n (U, S, V, N): #for zero padding and choosing n singular values for further computation
  + t = choose the first n singular values
  + convert the non-selected singular values to 0 and form a diagonal matrix, store as A
  + if ifidf matrix has number of rows > number of columns
    - pad zero rows to A, which is our diagonal singular matrix length(U)-length(S) times.
  + if ifidf matrix has number of columns > number of rows
    - pad zero columns to A, which is our diagonal singular matrix length(V)-length(S) times.
* Return A
* Def reconstruct (u, s, v, n):
  + A = top\_n\_pad\_0(u, s, v, n)
  + Return u\*A\*v # this is the reconstructed matrix using fewer singular values
* Def frobenius (a, a2):
  + Computes the frobenius norm to measure who much the reconstructed matrix varies compared to the original.
  + Return: Square root of sum of each value of a-a2 squared/ Square root of sum of each value of a squared
* Def find\_k ():
  + For I from (1 to length of S):
    - Frobenius (original matrix, reconstruct (u, s, r, i) )
    - RETURN i when the frobenius norm is less than 0.4

STEP-4: OUTPUT MODULES

* Search(q):
  + Strip punctuation from query
  + Convert the query to a lower letters
  + Split the query into individual words and store it as a list
  + Take each word in the query, check if it’s present in the ordered list of all terms. If present replace the word in the terms list with 1 else replace with 0. This will give us the query in a binary form.
  + If none of the words in the query match with our terms print “Keywords don't match with documents"
  + Else:
    - Score=Matrix multiply query with reconstruct(L,S,R,K)

# we find k using the find\_k function

sort the score list and output the ordered list of documents

* Def Print\_concepts (how many concepts, how many words per concept):
  + Terms = X.index # ordered list of words
  + Consider the top n eigen vectors in L , n is the number concepts you need

# L is the U matrix from the SVD output.

* + Each eigen vector has components which corresponds to each word in the documents.
  + Sort the words based on the component weight for each word in each eigen vector.
  + Output: 1st eigen vector represents first concept and the highest weighted terms in the vector represent the most important keywords in the concept.
* Def print\_concept\_docs (how many docs per concept): # this is to print the documents that represent the most important concepts overall.
  + Cols = document numbers
  + Consider the top n eigen vectors in R, n is the number concepts you need

# R is the V matrix from the SVD output.

* + Each eigen vector has components which corresponds to each document.
  + Sort the documents based on the component weight for each document in each eigen vector.
  + Output: 1st eigen vector represents first concept and the highest weighted components in the vector represent the most important documents in the concept.
* Def plot\_docs\_3d (): # To plot the documents in space of 3 most important concepts.
  + Y = (R-transpose) \* (Singular diagonal matrix)

# (V \* S describes the relation between documents (V\*S rows) and the concepts (V\*S columns))

* + Take the first 3 rows of y and plot in 3 dimensions
* def plot\_docs\_2d (): #To plot the documents in the space of 2 most important concepts
  + Y = (R-transpose) \* (Singular diagonal matrix)

# (V \* S describes the relation between documents (V\*S rows) and the concepts (V\*S columns))

* + Take the first 2 rows of y and plot in 2 dimensions