Assignment 5
Introduction to Information Retrieval
CS734/834

John Berlin

December 17, 2016

Note

In order to making the computation of the iterations for Hits and Pagerank stop at the desired iteration for section Q.1, I modified the respective algorithm implementations that the NetworkX[1] library provides. The modifications were only done so that they do not throw an exception if convergence was not reached when the stopping point parameter *max_iter* was hit. These changes can be seen in Source Code 12. Also utilized in section Q.1 are three new context classes were created GraphBuilder,AutoSLatexTable, and AutoSaveTwoFileTypes. GraphBuilder creates a new networkx Graph and populates it from the given configuration. AutoSLatexTable is similar to the other autosaver classes discussed in this section for Q4 but it creates a latex table utilizing the tabulate library[2]. AutoSaveTwoFileTypes allows usage of two autosavers from a given configuration containing the required arguments for each.

[1] http://networkx.readthedocs.io/en/networkx-1.10/

[2] https://pypi.python.org/pypi/tabulate

Q.1 Question 10.3

```
Compute five iterations of HITS (see Algorithm 3) and PageRank (see Figure 4.11) on the graph in Figure 10.3. Discuss how the PageRank scores compare to the hub and authority scores produced by HITS
```

Answer

The networkx library was utilized to answer this question and node,edges files used when creating the graphs can be found in the data folder that companies this report. After computing each iteration the respective iterations scores were save to files for use by Source Code 2 to generate the respective iterations graph plot, the python code for this question can be seen in Source Code 1. Please note that the score label for node seven in Figure 1 through Figure 5 has a pagerank of 0.7 throughout. This happens due to the cutting short of the algorithms iterations which was necessary to answer this question.

After the first iteration Figure 1 the scores for each node begin to show who are the good hubs or authorities. Node one and six have interesting values from the hits algorithm. Both nodes authority scores are the highest out of the others likewise with the pagerank score. The other nodes scores show that they are more hubs than authorities except node five which has all three scores present. Iterations two Figure 2 through five Figure 5 show that only nodes five and one have value changes and all other node values stay the same. This leads me to believe that due to the limited size of this graph that it only took two iterations to determine the final graph. When only considering pagerank scores it would be hard to determine who was an authority, a hub or both.

This can be seen in nodes five,three,two and four. Node five who is both an authority and a hub has a pagerank score of 0.11 whereas nodes three, two and four have pagerank scores of 0.07 but node three has a hub score of 0.45 unlike the others who with 0.2. These nodes pagerank score difference from node 5 is only 0.04 which is small enough that without prior knowledge of the graph using pagerank as an indicator for who was a hub or an authority would be hard.

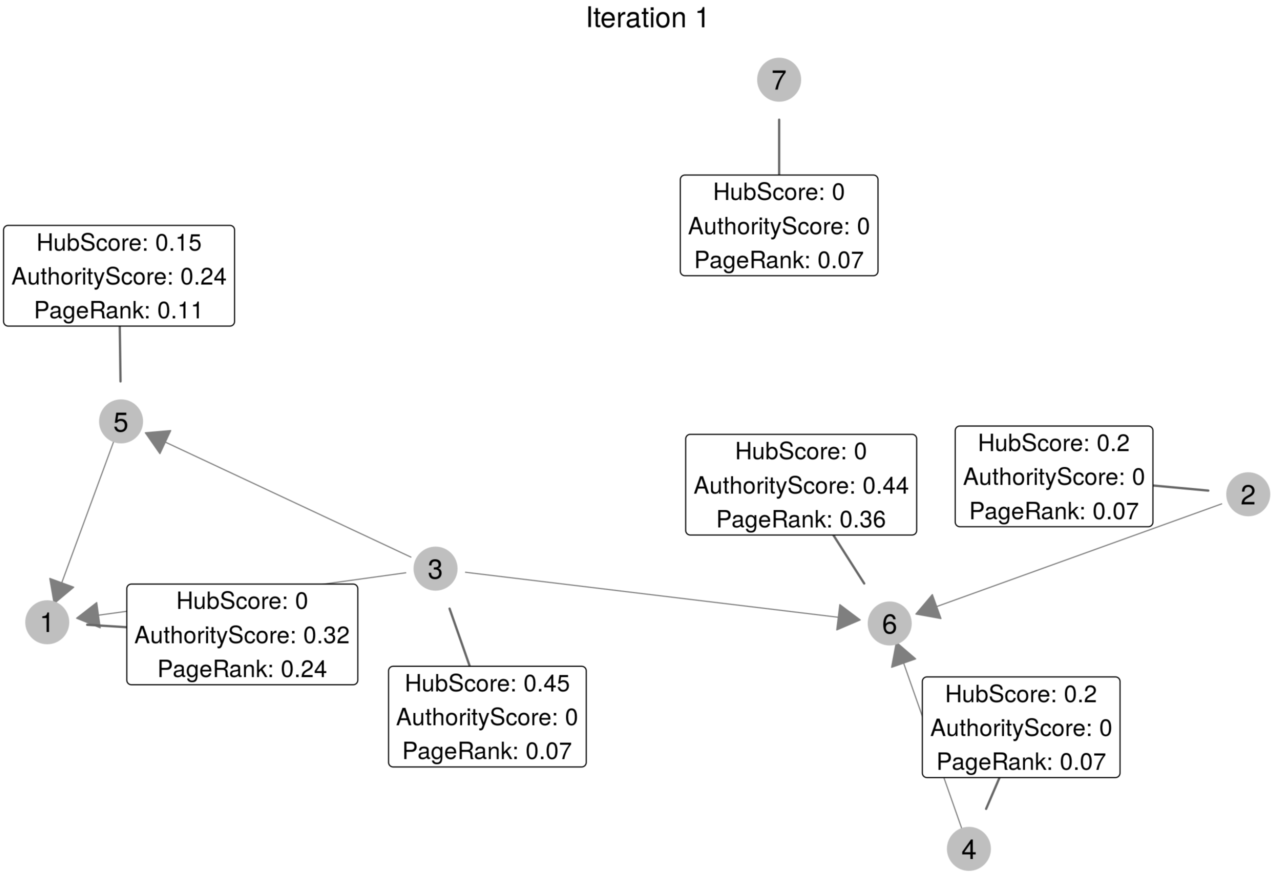


Figure 1: Iteration 1

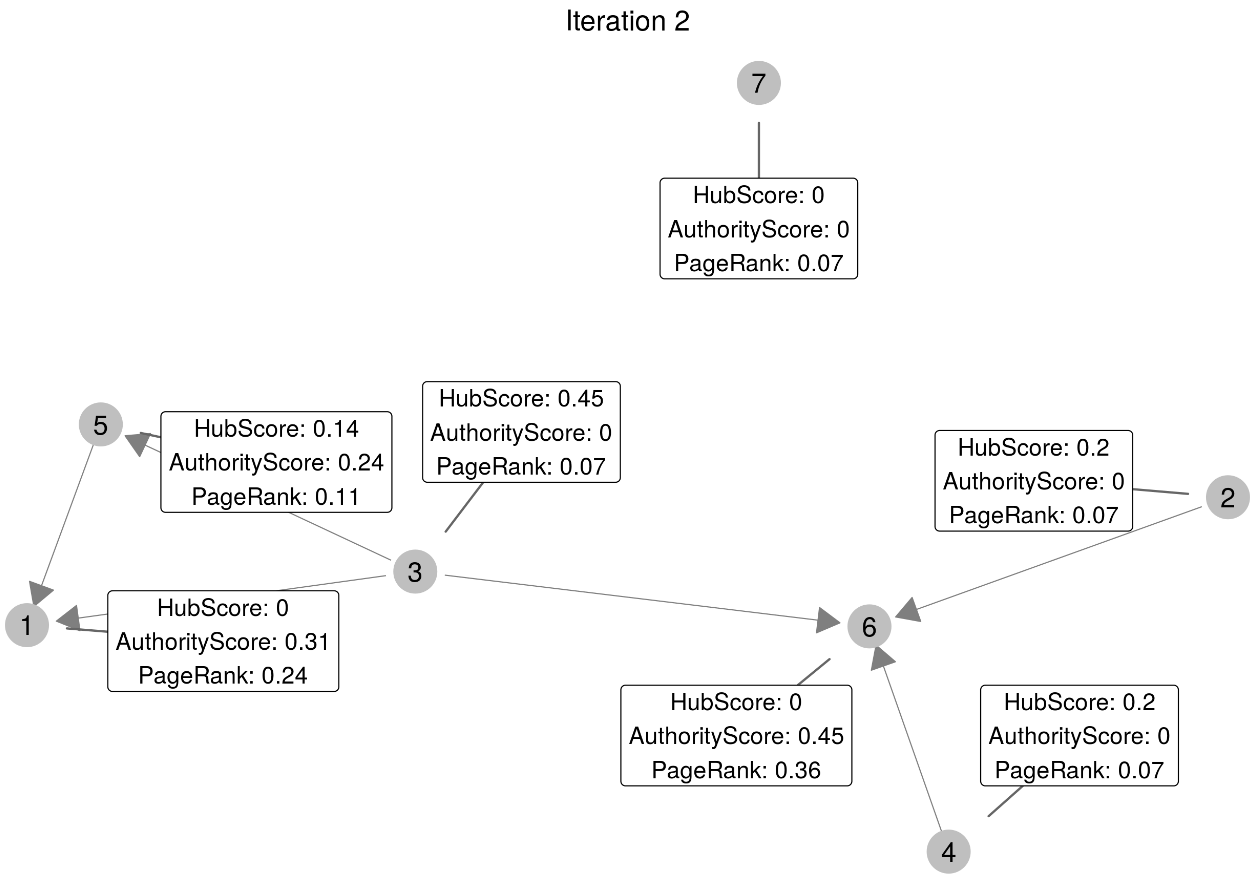


Figure 2: Iteration 1

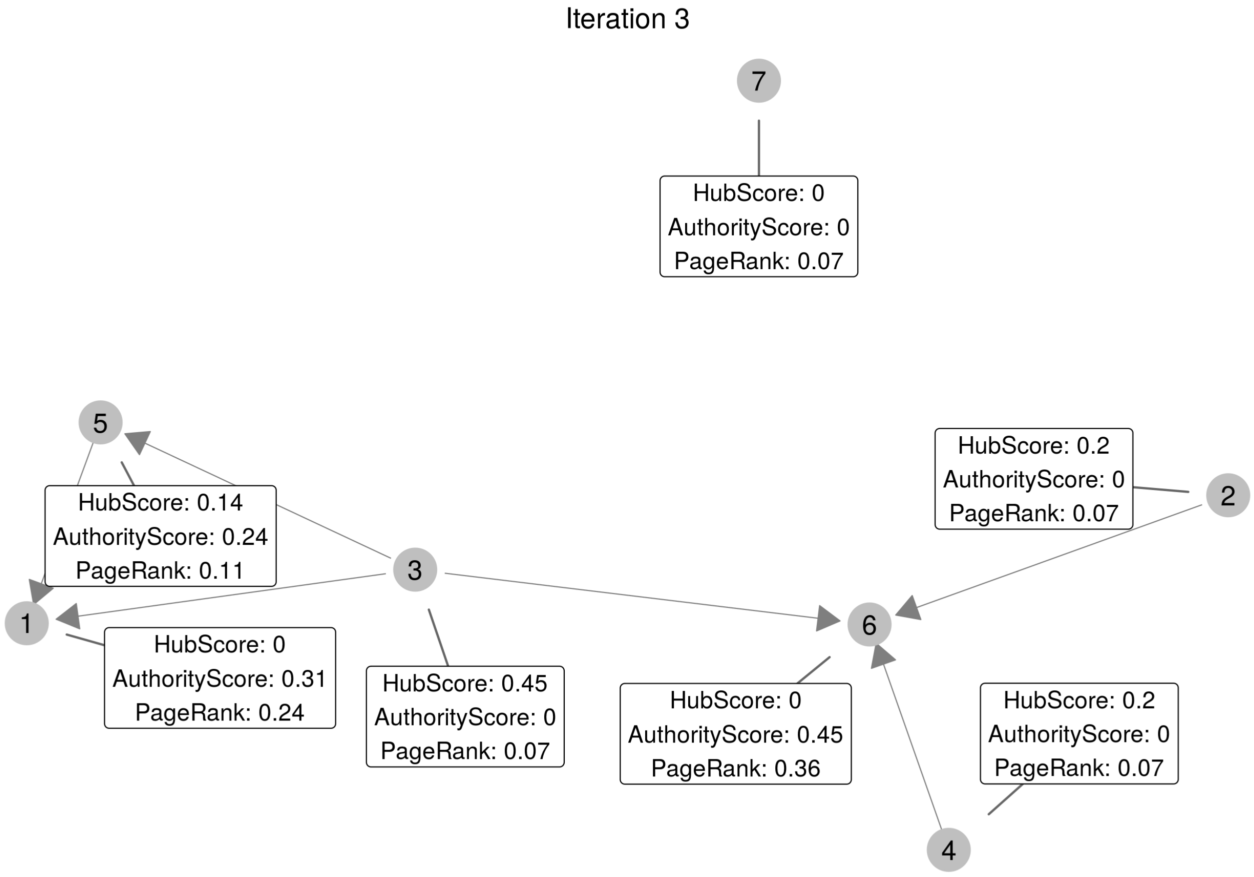


Figure 3: Iteration 3

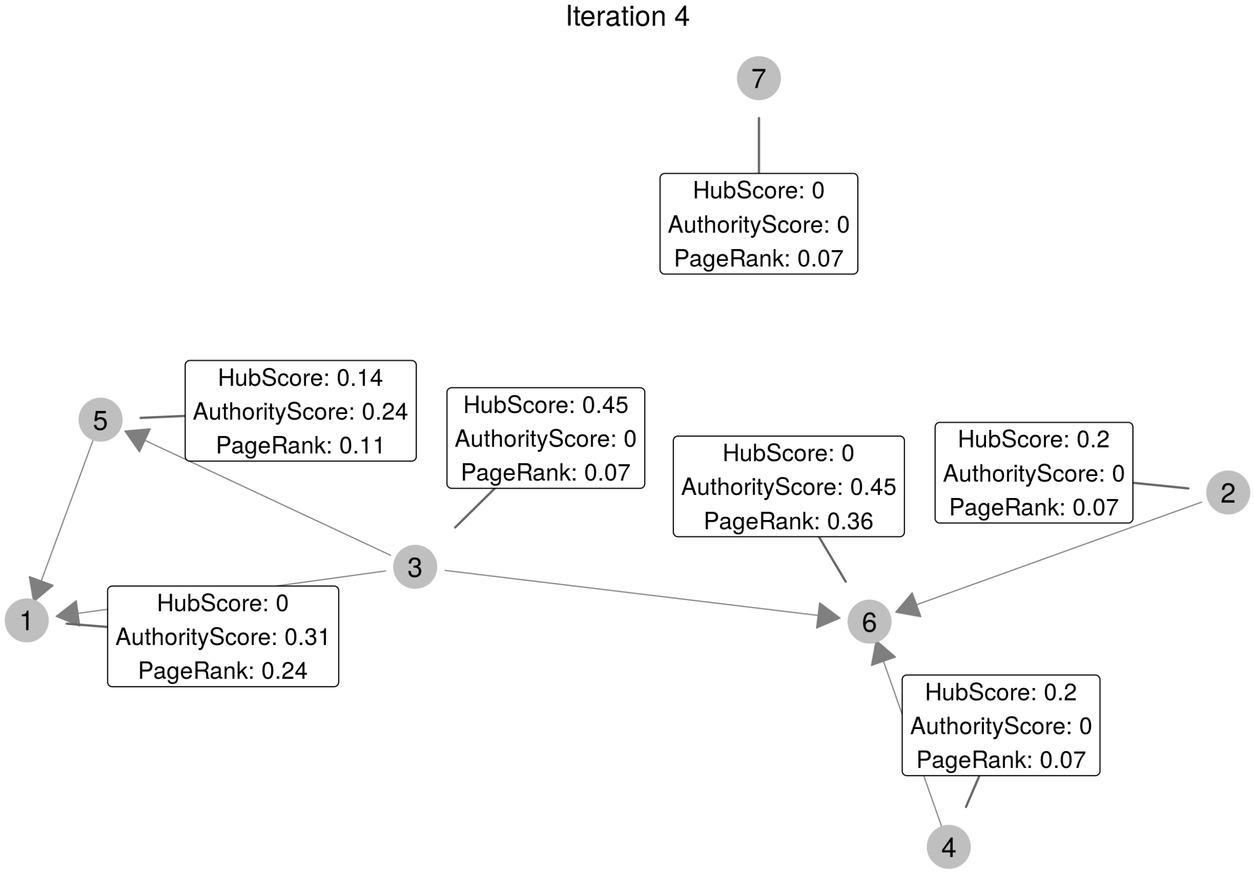


Figure 4: Iteration 4

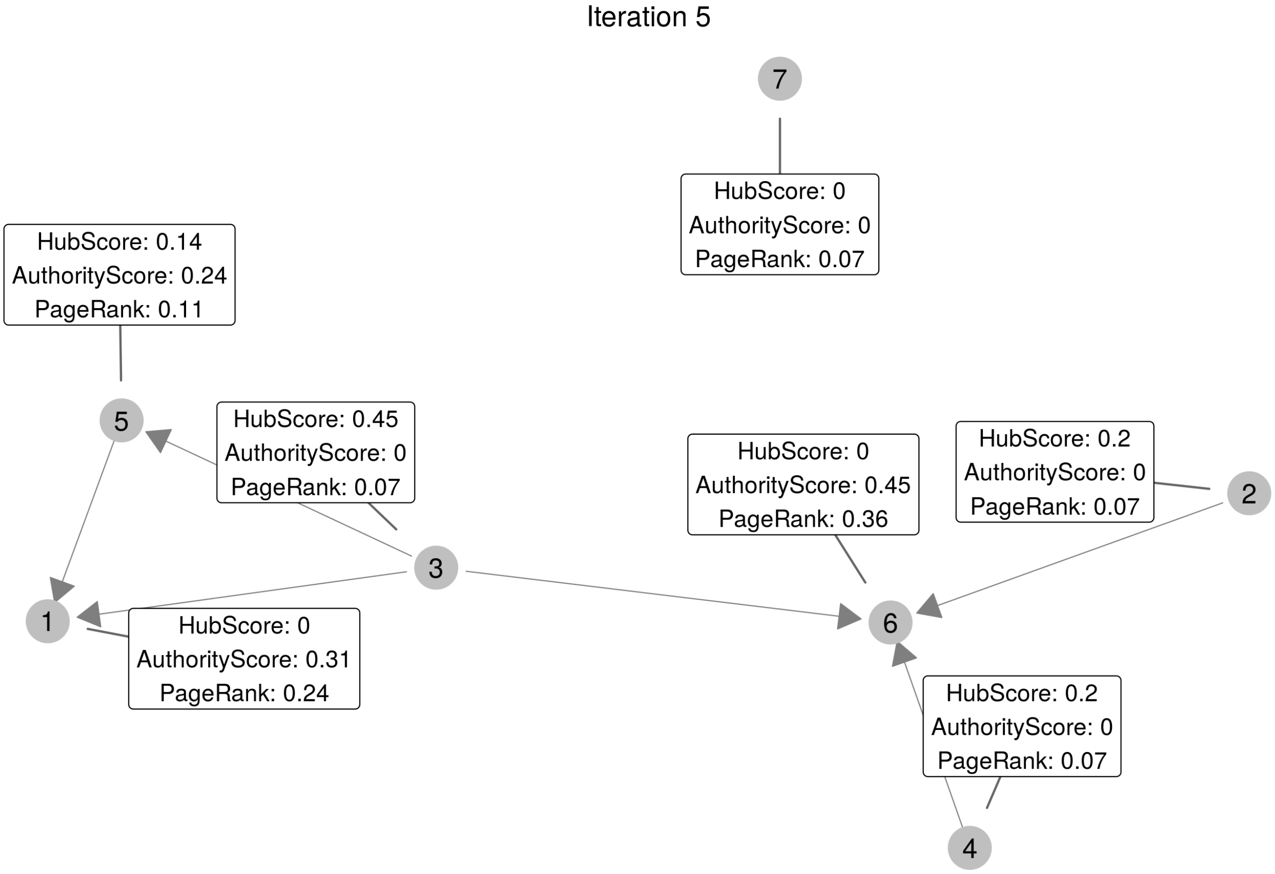


Figure 5: Iteration 5

Source Code 1: Q1 Python

```python
from nx_controled_algos import *
from contextClasses import GraphBuilder, AutoSLatexTable, AutoSaveCsv, AutoSaveTwoFileTypes

node_filep = 'data/graph_nodes.txt'
node_config = {'how_read': 'one_line', 'sep': ' '}
edge_filep = 'data/graph_edges.csv'
edges_config = {'how_read': 'csv', 'csv_mapping': {'node': 'node', 'edge': 'edge'}}

iter_tablep = 'output_files/q1_iteration%d.txt'
iter_csvp = 'output_files/q1_iteration%d.csv'


def do_sort(items):
    return sorted(items, key=lambda x: int(x[0]))


def do_zip(h, a, p):
    return zip(do_sort(h.items()), do_sort(a.items()), do_sort(p.items()))


if __name__ == '__main__':
    with GraphBuilder(node_filep, edge_filep, node_config, edges_config, graph_type='d') as g:
        csv_header = ['iter', 'node', 'hs', 'as', 'pr']
        table_header = ['Iteration', 'Node', 'Hub Score', 'Authority Score', 'PageRank']
        for i in range(1, 6, 1):
            print('iteration %d' % i)
            hubs, authorities = hits_scipy(g, max_iter=i - 1)
            pr = pagerank_scipy(g, max_iter=i)
            args = {'f_clazz': AutoSLatexTable, 'f_filep': iter_tablep % i, 'f_type': list, 'f_fn': table_header,
                    's_clazz': AutoSaveCsv, 's_filep': iter_csvp % i,
                    's_fn': csv_header, 's_type': list}
            with AutoSaveTwoFileTypes(**args) as (ltbl, csv):
                for ((hubn, hs), (authn, auths), (prn, prs)) in do_zip(hubs, authorities, pr):
                    print('node %s, hub score=%s, authority score=%s, pagerank score=%s' % (hubn, hs, auths, prs))
                    csv.append({'iter': i, 'node': hubn, 'hs': hs, 'as': auths, 'pr': prs})
                    ltbl.append([i, hubn, hs, auths, prs])
                print('----------------------------------------\n')
```

Source Code 2: Q1 R

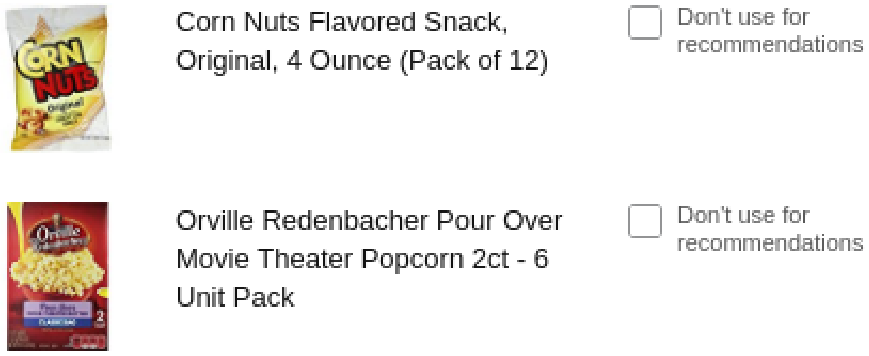
```r
library(network)
library(sna)
library(ggplot2)
library(ggnet)
library(ggrepel)

names <- c('1','2','3','4','5','6','7')
adjm <- read.table('data/adjm.txt',row.names = names,col.names=names,check.names=FALSE)
net <- as.network(as.matrix(adjm), directed = TRUE, loops = FALSE, matrix.type = 'adjacency')
gplot <- ggnet2(net,label=T,arrow.size = 12, arrow.gap = 0.025)

score_labeler <- function(df) {
  it = c()
  for(i in 1:length(df$hs)) {
    hs <- paste('HubScore:',df$hs[i])
    as <- paste('AuthorityScore:',df$as[i])
    pr <- paste('PageRank:',df$pr[i])
    it <- c(it,paste(hs,as,pr,sep = '\n'))
  }
  it
}

for(i in 1:5) {
  iter <- read.csv(paste('output_files/q1_iteration',i,'.csv',sep=''))
  score_labeles <- data.frame(
    x = gplot$data$x,
    y = gplot$data$y,
    labels = score_labeler(iter)
  )
  gplot + labs(title=paste('Iteration',i)) +
    geom_label_repel(data=score_labeles,aes(x,y,label=labels), box.padding = unit(0.5, 'lines'), point.padding =
     ↪  unit(2.6, 'lines'))
  ggsave(paste('images/','q1_iteration',i,'.png',sep=''))
}
```

Q.2 Question 10.6

Find two examples of document filtering systems on the Web. How do they build a profile for your information need? Is the system static or adaptive?

Answer

For my two examples I have choose Amazon and Spotify as both use an adaptive system for making recommendations to me. Figure 6 shows two snack food items I purchased together through Amazon both of which happen to be my all time favorite snack foods.



Corn Nuts Flavored Snack, Original, 4 Ounce (Pack of 12)

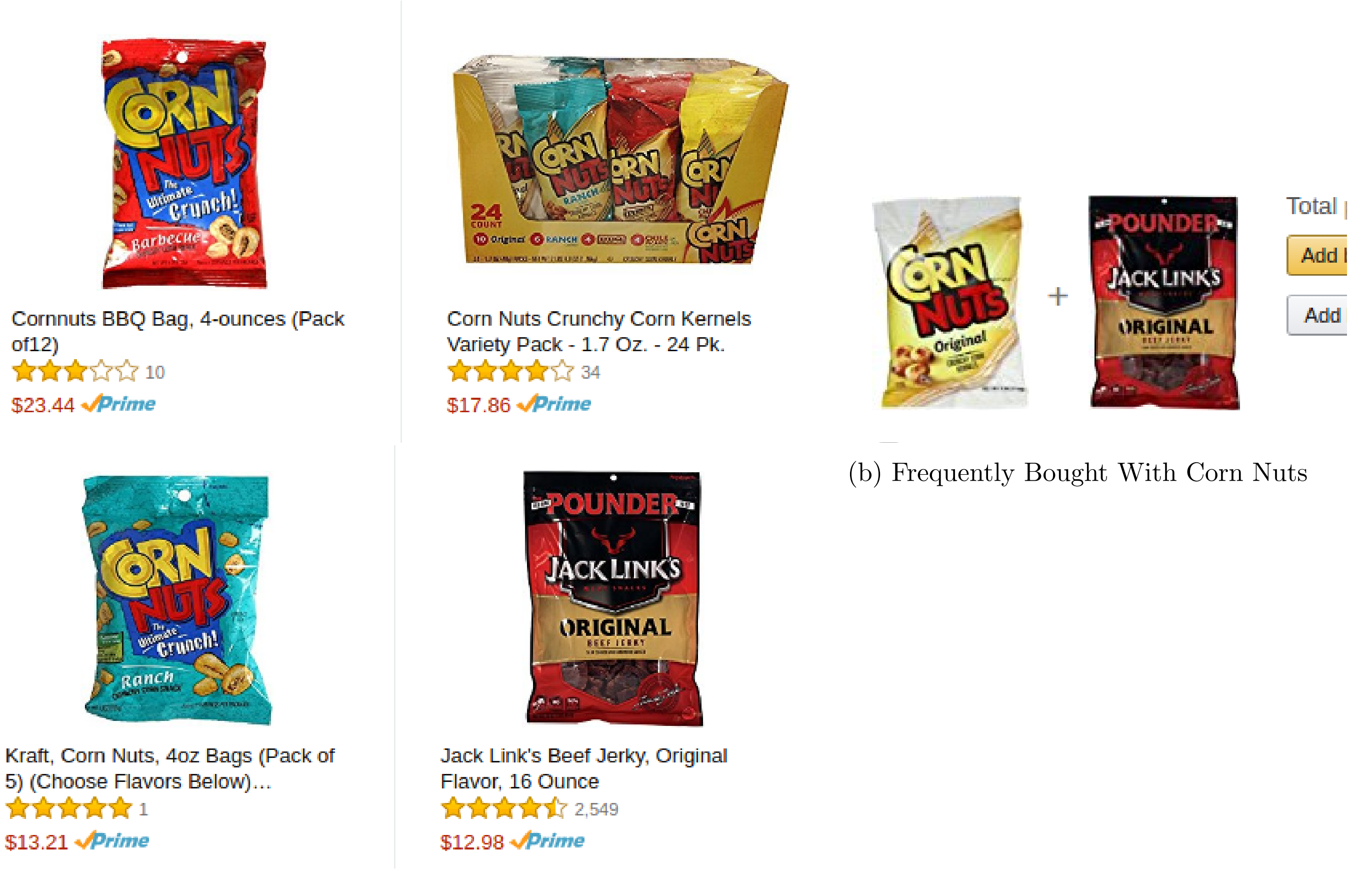
□ Don't use for recommendations

Orville Redenbacher Pour Over Movie Theater Popcorn 2ct - 6 Unit Pack

□ Don't use for recommendations

Figure 6: Purchased Snack Foods

Ever since then I have been getting recommendations for similar snack foods as seen in Figure 7 which shows the first four recommendations. Both of the items I purchased had four and three fourth star ratings which was used in making the correlation to these recommended items Figure 7(a). Three of these recommendation are other Corn Nut products while the fourth is Jerky. This is included because other Amazon customers who bought Corn Nuts also purchased Jack Link's Beef Jerky with it Figure 7(b). This frequently bought item is also recommend for purchase with Corn Nuts when adding it to your cart.



Cornnuts BBQ Bag, 4-ounces (Pack of12)
★★★☆☆ 10
$23.44 ✓Prime

Corn Nuts Crunchy Corn Kernels Variety Pack - 1.7 Oz. - 24 Pk.
★★★★☆ 34
$17.86 ✓Prime

Kraft, Corn Nuts, 4oz Bags (Pack of 5) (Choose Flavors Below)...
★★★★★ 1
$13.21 ✓Prime

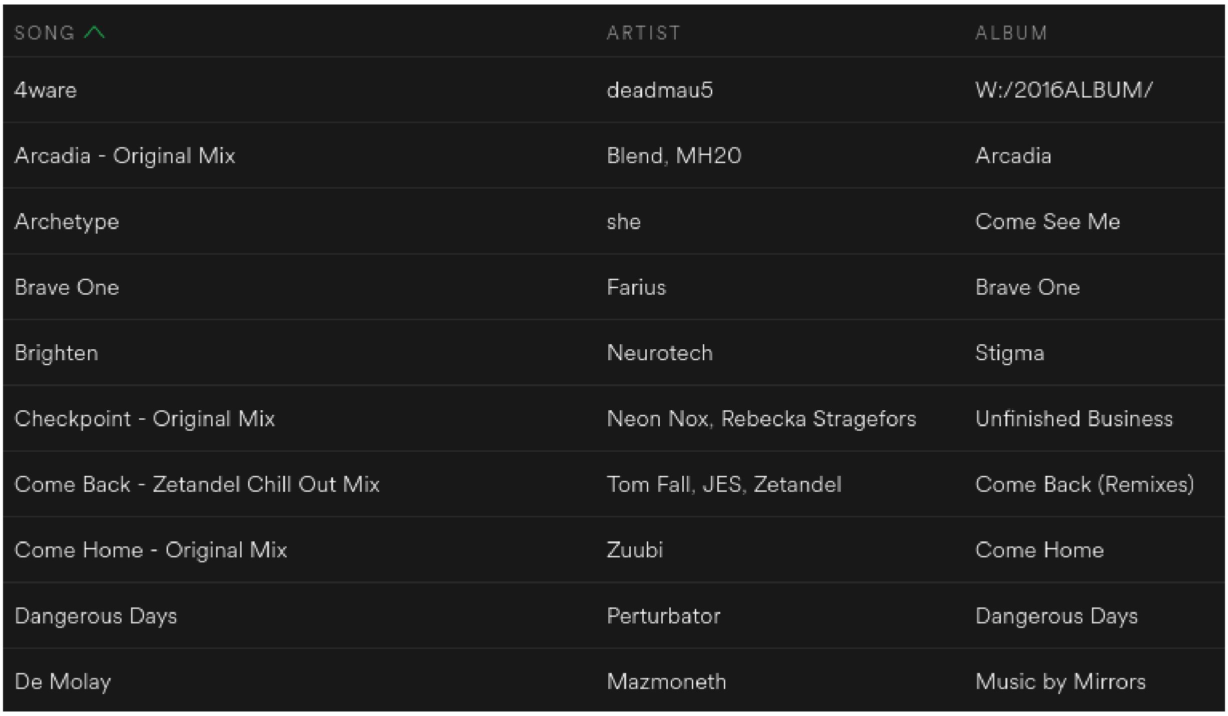
Jack Link's Beef Jerky, Original Flavor, 16 Ounce
★★★★☆ 2,549
$12.98 ✓Prime

(a) First Four Recommendations

(b) Frequently Bought With Corn Nuts

Figure 7: Amazon Snack Food Recommendations

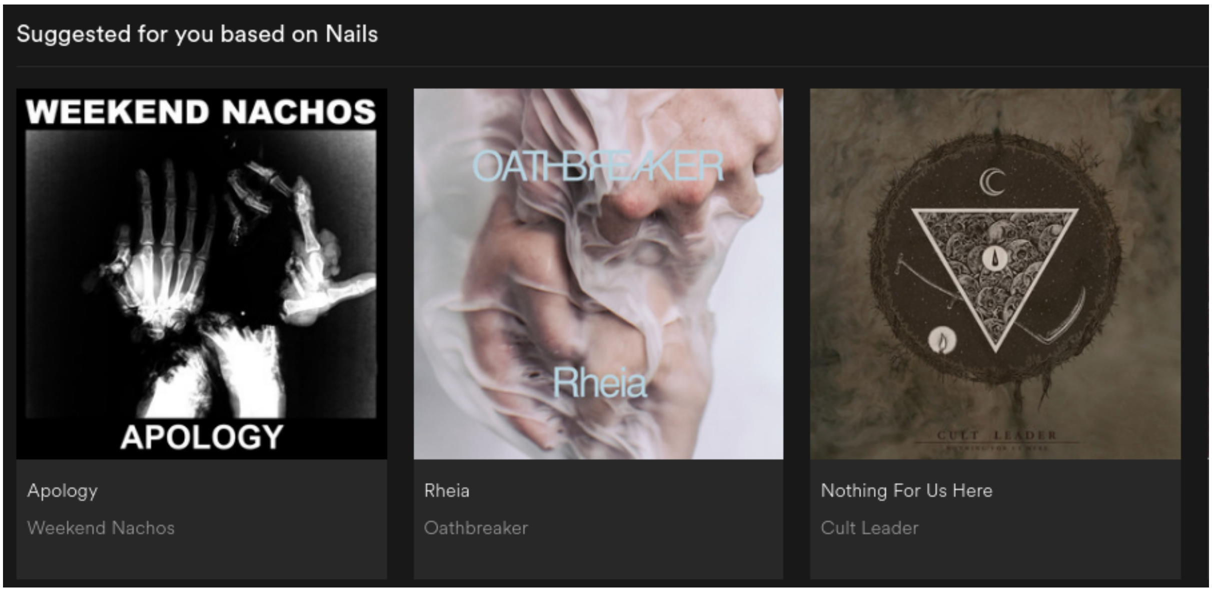
Amazon uses your purchase history and correlates it with purchases from other users who have also bought the same item as you to build its recommendations. My second example Spotify is an online music streaming service that allows for user created playlists, similar artists recommendations, top chart playlists and auto created playlists for a user. Figure 8 shows this weeks auto created playlist for me but most of the songs listed I am not familiar with by name. It was not until I listened to a few that I recognized them as they come from other user created playlists which I subscribe to. They were recommend to me because other users have “liked” the track i.e hit a check mark beside it after which it was saved to their like song list. Most of them were not too bad but the 4ware track by deadmau5 is excellent and one I liked as well after hearing it again via this recommended playlist.



SONG ^	ARTIST	ALBUM
4ware	deadmau5	W:/2016ALBUM/
Arcadia - Original Mix	Blend, MH20	Arcadia
Archetype	she	Come See Me
Brave One	Farius	Brave One
Brighten	Neurotech	Stigma
Checkpoint - Original Mix	Neon Nox, Rebecka Stragefors	Unfinished Business
Come Back - Zetandel Chill Out Mix	Tom Fall, JES, Zetandel	Come Back (Remixes)
Come Home - Original Mix	Zuubi	Come Home
Dangerous Days	Perturbator	Dangerous Days
De Molay	Mazmoneth	Music by Mirrors

Figure 8: Recommended Playlist

Since Spotify uses your listen history in combination with other users plus liked tracks to make recommendations and the recommendation is music the results can be weird at times or way off. For instance Figure 9 which shows the first three recommendations for similar albums based off listening to Nails. Nails is ultra aggressive hardcore/powerviolence band so the recommendations for Weekend Nachos and Cult Leader makes sense as they are in the sludgy crust punk powerviolence spectrums. Now Oathbreaker on the other hand is a post-hardcore band that has some tracks that could be called hardcore punk but that is up for debate. Looks like a bunch of wayward post-hardcorers like Nails and is why Spotify included in this recommendation.



Suggested for you based on Nails

Apology
Weekend Nachos

Rheia
Oathbreaker

Nothing For Us Here
Cult Leader

Figure 9: Suggestions Nails

As much as I would disagree with this recommendation it proves that Spotify is using an adaptive system to generate the recommendations it makes.

Q.3 Question 10.8

```
Implement the nearest neighbor based collaborative filtering algorithm.
Using a publicly available collaborative filtering data set, compare the effective-
ness, in terms of mean squared error, of the Euclidean distance and correlation
similarity.
```

Answer

I chose this question because it is a variation on assignment seven from cs532 Web Sciences and like that question I used the MoveLen dataset. But unlike that question I did not use the code submitted for it nor the code provided from the courses text book Programming Collective Intelligence. Instead I choose to use Pandas[3] which is a high performance data analysis library akin to R. The source code used to generate the similarities can be seen in Source Code 3 and Source Code 5.

The MoveLen dataset contains 100,000 ratings (1-5) from 943 users for 1682 movies where each user has rated at least 20 movies. So computing the similarity for each user for a given k value would take some time as this user based similarity. In order to ensure the computation of k nearest neighbors for an given k value for all users could be done relatively fast, the similarities for all users were computed, i.e 1 user vs 942 others 943 times. This was accomplished by processing 23 "*chunks*" of 41 users 4 chunks at a time in parallel. Then for each user in a chunk the users similarity to every other user was computed using Euclidean distance and Pearson as the similarity measures. Once all chunks had been processed the results were combined into a single `pandas.DataFrame` and serialized to disk. This method of processing is much like the map and reduce paradigm. Instead of distributing the similarity computing function to multiple nodes pythons `multiprocess.Pool` applied that function to each chunk (map) using four different processes on my local machine and then combined (reduce) into a single result by the process which spawned them.

After the one vs all similarity for the users was computed the k nearest neighbors for $k = 10$ was found for both similarity measures and Figure 10 show the mean squared error for both measures.

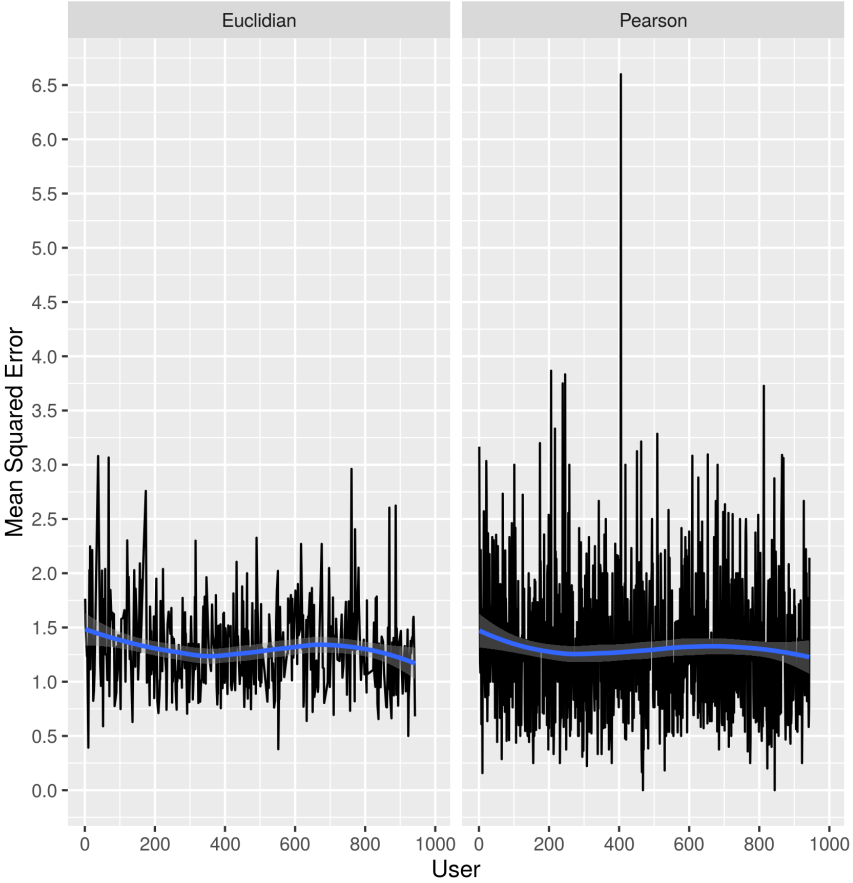


Figure 10: Predicted MSE

[3] http://pandas.pydata.org/

Overall Euclidean distance measure performed the best whereas Pearsons had more extremes. The trend line for both measures shows they behave similarly even tho Pearson had higher MSE values overall. This is more easily seen when considering Figure 11 and Figure 12 which show a histogram (bin width = 50) of the ratings for both measures compared to the predicted values. Pearson, Figure 11 predicted values that were ±1 ranking than the actual. Euclidean distance faired similarly Figure 12 but less so. The reason behind this could be attributed to the fact it is more suited to a clustering algorithm since it is more geared to this kind of problem than a true similarity function.

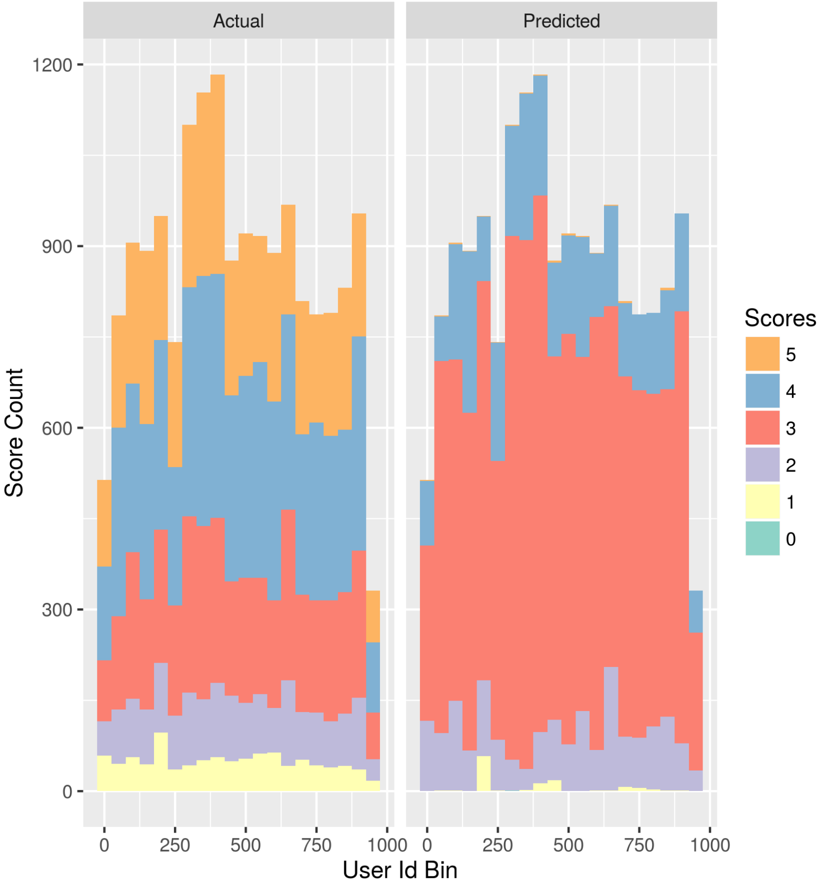


Figure 11: User Prediction Pearson

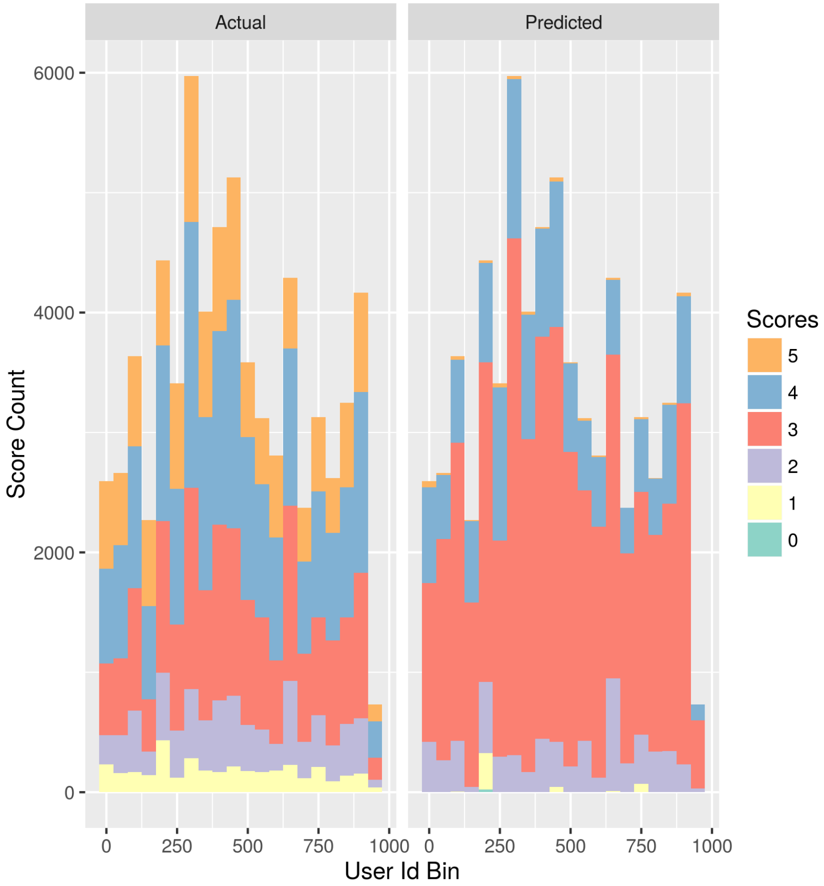


Figure 12: User Prediction Euclidean Distance

Source Code 3: Q2 Compute Similarities

```
import numpy as np
import math
import pandas as pd
import os
from collections import defaultdict
from contextClasses import AutoSaveCsv
from q2_helpers import get_review_df, get_reviewer_sim, get_movies_df
from util import dump_pickle, read_pickle

review_file_pickle = 'pickled/movies_predicted_%s_%d.pickle'
review_file_mean_pickle = 'pickled/movies_predicted_%s_%d.pickle'
review_file_rated_pickle = 'pickled/movies_predicted_rated_%s_%d.pickle'
review_file_rated_mean_pickle = 'pickled/movies_predicted_rated_%s_%d.pickle'

user_pred_rate_csv = 'output_files/user_pred_rated_%s_%d.csv'
user_pred_rate2_csv = 'output_files/user_pred_rated2_%s_%d.csv'
user_pred_rate_mse_csv = 'output_files/user_pred_rated_mse_%s_%d.csv'
user_pred_rate_mse2_csv = 'output_files/user_pred_rated_mse2_%s_%d.csv'


class UserReviewedPred(dict):
    def __missing__(self, key):
        res = self[key] = {}
        return res

    def add_movie(self, mid, mtitle, pred, act):
        self[mid]['mtile'] = mtitle
        self[mid]['pred'] = pred
        self[mid]['act'] = act


class UserPred(dict):
    def __missing__(self, key):
        res = self[key] = {}
        return res

    def add_movie(self, mid, mtitle, pred):
        self[mid]['mtile'] = mtitle
        self[mid]['pred'] = pred


def pick_knn(user_sim, method='pearson_seq', k=10):
    sims = user_sim[user_sim.sim_method == method][np.isfinite(user_sim.sim)].sort_values(by=['sim'])
    return sims.tail(k)


def predict_rated(method='pearson_seq', k=10):
    want = review_file_rated_pickle % (method, k)
    if not os.path.exists(want):
        user_predictions = defaultdict(UserReviewedPred)
        review_df = get_review_df()
        reviewer_sims = get_reviewer_sim(review_df)
        movies = get_movies_df()
        users = review_df.user_id.sort_values().unique()
        for user in users:
            user_reviews = review_df[review_df.user_id == user]
            similar_to_user = reviewer_sims[reviewer_sims.user == user]
            knn = pick_knn(similar_to_user, method=method, k=k)
            user_mean = user_reviews.rating.mean()
            sum_user = 1 / knn.sim.sum()
            neighbors = review_df[review_df.user_id.isin(knn.other_user.unique())]
            reviewed_same = neighbors[neighbors.item_id.isin(user_reviews.item_id)]
            neighbor_reviewed_ids = reviewed_same.item_id.unique()
            for neighbor_review_id in neighbor_reviewed_ids:
                movie_reviewed = reviewed_same[reviewed_same.item_id == neighbor_review_id]
                the_movie = movies[movies.movie_id.isin(movie_reviewed.item_id)]
                movie_reviewed_title = the_movie.movie_title.iat[0]
                neighbor_reviews = neighbors[neighbors.item_id == neighbor_review_id]
                acum = []
                for _, row in neighbor_reviews.iterrows():
                    neighbor_who_did_id = row['user_id']
                    neighbor_who_did_rating = row['rating']
                    neighbor_who_did_mean_rating = review_df[review_df.user_id ==
                     ↪  neighbor_who_did_id].rating.mean()
                    neighbor_who_did_sim_to_cur_u = knn[knn.other_user == neighbor_who_did_id].sim.iat[0]
                    acum.append(
                        neighbor_who_did_sim_to_cur_u * (neighbor_who_did_rating - neighbor_who_did_mean_rating))
                sum_neighbors = sum(acum)
                predicted_score = user_mean + (sum_user * sum_neighbors)
                the_movie_id = the_movie.movie_id.iat[0]
                actual_r = user_reviews[user_reviews.item_id == the_movie_id].rating.iat[0]
                user_predictions[user].add_movie(the_movie_id, movie_reviewed_title, predicted_score, actual_r)
        dump_pickle(user_predictions, want)
        return user_predictions
    else:
        return read_pickle(want)


def predict(method='pearson_seq', k=10):
    want = review_file_pickle % (method, k)
    if os.path.exists(want):
        user_predictions = defaultdict(UserPred)
        review_df = get_review_df()
        reviewer_sims = get_reviewer_sim(review_df)
        movies = get_movies_df()
        users = review_df.user_id.sort_values().unique()
        for user in users:
            user_reviews = review_df[review_df.user_id == user]
            similar_to_user = reviewer_sims[reviewer_sims.user == user]
            knn = pick_knn(similar_to_user, method=method, k=k)
            user_mean = user_reviews.rating.mean()
            sum_user = 1 / knn.sim.sum()
            neighbors = review_df[review_df.user_id.isin(knn.other_user.unique())]
            not_reviewed = neighbors[~neighbors.item_id.isin(user_reviews.item_id)]
            neighbor_reviewed_ids = not_reviewed.item_id.unique()
            for neighbor_review_id in neighbor_reviewed_ids:
                movie_not_reviewed = not_reviewed[not_reviewed.item_id == neighbor_review_id]
                the_movie_not_reviewed = movies[movies.movie_id.isin(movie_not_reviewed.item_id)]
                movie_not_reviewed_title = the_movie_not_reviewed.movie_title.iat[0]
                neighbor_reviews = neighbors[neighbors.item_id == neighbor_review_id]
                acum = []
                for _, row in neighbor_reviews.iterrows():
                    neighbor_who_did_id = row['user_id']
                    neighbor_who_did_rating = row['rating']
                    neighbor_who_did_mean_rating = review_df[review_df.user_id ==
                     ↪  neighbor_who_did_id].rating.mean()
                    neighbor_who_did_sim_to_cur_u = knn[knn.other_user == neighbor_who_did_id].sim.iat[0]
                    acum.append(
                        neighbor_who_did_sim_to_cur_u * (neighbor_who_did_rating - neighbor_who_did_mean_rating))
                sum_neighbors = sum(acum)
                predicted_score = user_mean + (sum_user * sum_neighbors)
                the_movie_id = the_movie_not_reviewed.movie_id.iat[0]
                user_predictions[user].add_movie(the_movie_id, movie_not_reviewed_title, predicted_score)
        dump_pickle(user_predictions, want)
        return user_predictions
    else:
        return read_pickle(want)


def gen_mse_prdr_csv(k=10):
    mse_headers = ['user', 'mse', 'which']
    pre_headers = ['user', 'mid', 'mtitle', 'score', 'which']
    preds_pearson = predict_rated(k=k)
    preds_euclid = predict_rated(method='euclid', k=k)
    want_euclid = user_pred_rate_csv % ('euclid', k)
    want_euclid_mse = user_pred_rate_mse_csv % ('euclid', k)
    want_pearson = user_pred_rate_csv % ('pearson', k)
    want_pearson_mse = user_pred_rate_mse_csv % ('pearson', k)
    if os.path.exists(want_euclid):
        with AutoSaveCsv(list, want_euclid_mse, mse_headers) as mseo, \
                AutoSaveCsv(list, want_euclid, pre_headers) as upe:
            for user, user_preds in preds_euclid.items():
                uprs_len = len(list(user_preds.values()))
                acum = []
                for mid, score in user_preds.items():
                    pred = score['pred']
                    act = score['act']
                    upe.append(
                        {'user': user, 'mid': mid, 'mtitle': score['mtile'], 'score': math.floor(pred),
                         'which': 'predicted'})
                    upe.append(
                        {'user': user, 'mid': mid, 'mtitle': score['mtile'], 'score': act,
                         'which': 'actual'})
                    acum.append((math.floor(pred) - act) ** 2)
                pma_sum = sum(acum)
                mse = ((1 / uprs_len) * pma_sum)
                mseo.append({'user': user, 'mse': mse, 'which': 'Euclidian'})
    if os.path.exists(want_pearson):
        with AutoSaveCsv(list, want_pearson_mse, mse_headers) as mseo, \
                AutoSaveCsv(list, want_pearson, pre_headers) as upe:
            for user, user_preds in preds_pearson.items():
                uprs_len = len(list(user_preds.values()))
                acum = []
                for mid, score in user_preds.items():
                    pred = score['pred']
                    act = score['act']
                    upe.append(
                        {'user': user, 'mid': mid, 'mtitle': score['mtile'], 'score': math.floor(pred),
                         'which': 'predicted'})
                    upe.append(
                        {'user': user, 'mid': mid, 'mtitle': score['mtile'], 'score': act,
                         'which': 'actual'})
                    acum.append((math.floor(pred) - act) ** 2)
                pma_sum = sum(acum)
                mse = ((1 / uprs_len) * pma_sum)
                mseo.append({'user': user, 'mse': mse, 'which': 'Pearson'})


if __name__ == '__main__':
    r_df = get_review_df()
    m_df = get_movies_df()
    rsim_df = get_reviewer_sim(r_df)
    gen_mse_prdr_csv()
```

Source Code 4: Q2 Generate Graphs

```
library(ggplot2)
library(dplyr)
library(scales)
library(ggrepel)
library(RColorBrewer)

source('code/q2_plotter_helper.R')

euclid <- read.csv('output_files/user_pred_rated_euclid_10.csv')
ggplot(euclid,aes(mid)) +
  geom_histogram(aes(fill=as.factor(score)),binwidth = 200) +
  scale_fill_brewer('Scores',breaks = rev(levels(as.factor(euclid$score))),palette='Set3')  +
  facet_wrap(~which,labeller = labeller(which = capitalize)) +
  labs(x = 'Movie Id Bin', y = 'Score Count')

ggsave('images/euclid_predicted_bmovie.png')

ggplot(euclid,aes(user)) +
  geom_histogram(aes(fill=as.factor(score)),binwidth = 50) +
  scale_fill_brewer('Scores',breaks = rev(levels(as.factor(euclid$score))),palette='Set3')  +
  facet_wrap(~which,labeller = labeller(which = capitalize)) +
  labs(x = 'User Id Bin', y = 'Score Count')

ggsave('images/euclid_predicted_buser.png')

pearson <- read.csv('output_files/user_pred_rated_pearson_10.csv')
ggplot(pearson,aes(user)) +
  geom_histogram(aes(fill=as.factor(score)),binwidth = 50) +
  scale_fill_brewer('Scores',breaks = rev(levels(as.factor(euclid$score))),palette='Set3')  +
  facet_wrap(~which,labeller = labeller(which = capitalize)) +
  labs(x = 'User Id Bin', y = 'Score Count')

ggsave('images/pearson_predicted_buser.png')

euclidMSE <- read.csv('output_files/user_pred_rated_mse_euclid_10.csv')
pearsonMSE <- read.csv('output_files/user_pred_rated_mse_pearson_10.csv')

ggplot() +
  geom_line(data=euclidMSE,aes(user,mse)) +
  geom_smooth(data=euclidMSE,aes(user,mse)) +
  geom_line(data=pearsonMSE,aes(user,mse)) +
  geom_smooth(data=pearsonMSE,aes(user,mse)) +
  scale_y_continuous(limits = c(min_y(euclidMSE,pearsonMSE),max_y(euclidMSE,pearsonMSE)),breaks = seq(0,7,by=.5))
   ↪   +
  scale_x_continuous(limits = c(min_x(euclidMSE,pearsonMSE),max_x(euclidMSE,pearsonMSE)+50),breaks =
   ↪   pretty_breaks(n=6)) +
  facet_wrap(~which) +
  labs(x='User',y='Mean Squared Error')

ggsave('images/user_predicted_mse.png')
```

Source Code 5: Q2 Compute Similarities Helper

```
import os
import math
import networkx as nx
import multiprocessing
import numpy as np
import pandas as pd
from itertools import repeat
from sklearn.metrics import pairwise_distances_argmin
from sklearn.cluster import KMeans
from collections import defaultdict
from scipy.spatial.distance import euclidean
from scipy.stats import pearsonr
from util import *

ratings_df_pickle = 'pickled/movie_ratings_df.pickle'
reviewer_sim_pickle = 'pickled/movie_reviewer_sim.pickle'
rating_cluster_pickle = 'pickled/movie_rating_cluster%d.pickle'
movies_df_pickle = 'pickled/movie_movies_df.pickle'


def get_movies_df():
    if not os.path.exists(movies_df_pickle):
        names = ['movie_id', 'movie_title', 'release_date', 'video_release_date', 'IMDb_URL', 'unknown', 'Action',
                 'Adventure', 'Animation', 'Childrens', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
                 ' Film_Noir ','Horror', 'Musical', 'Mystery', 'Romance', ' Sci_Fi ', 'Thriller', 'War',
                  ↪   'Western']
        mov_df = pd.read_csv('data/ml-100k/u.item', sep='|', names=names, encoding='ISO-8859-1')
        dump_pickle(mov_df, movies_df_pickle)
    else:
        mov_df = read_pickle(movies_df_pickle)
    return mov_df


def get_review_df():
    if not os.path.exists(ratings_df_pickle):
        names = ['user_id', 'item_id', 'rating', 'timestamp']
        rev_df = pd.read_csv('data/ml-100k/u.data', sep='\t', names=names)
        dump_pickle(rev_df, ratings_df_pickle)
    else:
        rev_df = read_pickle(ratings_df_pickle)
    return rev_df


def make_new_combined(u1, u2):
    d = {'user_id_x': u1.user_id, 'item_id': u1.item_id, 'rating_x': u1.rating, 'timestamp_x': u1.timestamp,
         'user_id_y': list(repeat(u2.user_id.index[0], u1.user_id.size)), 'rating_y': list(repeat(0,
          ↪   u1.rating.size)),
         'timestamp_y': u1.timestamp}
    return pd.DataFrame(d)


def pearson(rating_u1, rating_u2):
    length = rating_u1.size
    mean_u1 = np.mean(rating_u1)
    mean_u2 = np.mean(rating_u2)
    numerator = sum([(rating_u1.iat[i] - mean_u1) * (rating_u2.iat[i] - mean_u2) for i in range(length)])
    denominator_u1 = math.sqrt(sum([(rating_u1.iat[i] - mean_u1) ** 2 for i in range(length)]))
    denominator_u2 = math.sqrt(sum([(rating_u2.iat[i] - mean_u2) ** 2 for i in range(length)]))
    if (denominator_u1 * denominator_u2) == 0:
        return 0
    correlation = numerator / (denominator_u1 * denominator_u2)
    return correlation


def sim_for_chunk(d):
    print('processing chunk %d' % d['chunk'])
    rdf = d['rdf']
    urs = d['urs']
    u_sims = {'user': [], 'other_user': [], 'sim_method': [], 'sim': []}
    for c_user in urs:
        cu_reviewed = rdf[rdf.user_id == c_user]
        o_uids = rdf[rdf.user_id != c_user].user_id.sort_values().unique()
        for o_uid in o_uids:
            o_u = rdf[rdf.user_id == o_uid]
            c = pd.merge(cu_reviewed, o_u, how='inner', on=['item_id'])
            if len(c.rating_y) == 0:
                c = make_new_combined(cu_reviewed, o_u)

            u_sims['user'].append(c_user)
            u_sims['other_user'].append(o_uid)
            u_sims['sim_method'].append('pearson_seq')
            u_sims['sim'].append(c.rating_x.corr(c.rating_y, method='pearson'))

            u_sims['user'].append(c_user)
            u_sims['other_user'].append(o_uid)
            u_sims['sim_method'].append('pearson_sci')
            p, _ = pearsonr(c.rating_x, c.rating_y)
            u_sims['sim'].append(p)

            u_sims['user'].append(c_user)
            u_sims['other_user'].append(o_uid)
            u_sims['sim_method'].append('pearson_mine')
            u_sims['sim'].append(pearson(c.rating_x, c.rating_y))

            u_sims['user'].append(c_user)
            u_sims['other_user'].append(o_uid)
            u_sims['sim_method'].append('euclid')
            u_sims['sim'].append(euclidean(c.rating_x, c.rating_y))
    print('finished processing chunk %d' % d['chunk'])
    return u_sims


def get_reviewer_sim(rdf):
    if not os.path.exists(reviewer_sim_pickle):
        users = rdf.user_id.sort_values().unique()
        chunks = []
        temp = []
        chunk_c = 1
        for cur_user in users:
            temp.append(cur_user)
            if len(temp) == 41:
                chunks.append({'chunk': chunk_c, 'rdf': rdf.copy(deep=True), 'urs': list(temp)})
                chunk_c += 1
                temp.clear()

        p = multiprocessing.Pool(4)
        ret = p.map(sim_for_chunk, chunks)
        all_sims = {'user': [], 'other_user': [], 'sim_method': [], 'sim': []}
        for it in ret:
            for k, v in it.items():
                all_sims[k].extend(v)
        all_sims = pd.DataFrame(all_sims)
        dump_pickle(all_sims, reviewer_sim_pickle)
    else:
        all_sims = read_pickle(reviewer_sim_pickle)

    return all_sims


def cluster_reviews(n_clusters=20):
    rev_df = get_review_df()
    if not os.path.exists(rating_cluster_pickle % n_clusters):
        users = rev_df.user_id.unique()
        n_users = users.shape[0]
        items = rev_df.item_id.unique()
        n_items = items.shape[0]
        ratings = np.zeros((n_users, n_items))
        for row in rev_df.itertuples():
            ratings[row[1] - 1, row[2] - 1] = row[3]
        k_means = KMeans(n_clusters=n_clusters, n_init=50, max_iter=3000)
        k_means.fit(ratings)
        k_means_cluster_centers = np.sort(k_means.cluster_centers_, axis=0)
        k_means_labels = pairwise_distances_argmin(ratings, k_means_cluster_centers)
        user_in_cluster = defaultdict(list)
        for cluster in range(n_clusters):
            my_members = k_means_labels == cluster
            if np.count_nonzero(my_members) > 0:
                my_members_idxs = my_members.nonzero()
                for user_idx in np.nditer(my_members_idxs):
                    print('user %d is in cluster %d' % (users[user_idx], cluster + 1))
                    user_in_cluster['user'].append(users[user_idx])
                    user_in_cluster['cluster'].append(cluster + 1)
                print('----------------------------------------')
        user_in_cluster = pd.DataFrame(user_in_cluster)
        dump_pickle(user_in_cluster, rating_cluster_pickle % n_clusters)
    else:
        user_in_cluster = read_pickle(rating_cluster_pickle % n_clusters)
    return rev_df, user_in_cluster
```

Source Code 6: Q2 Generate Graphs Helper

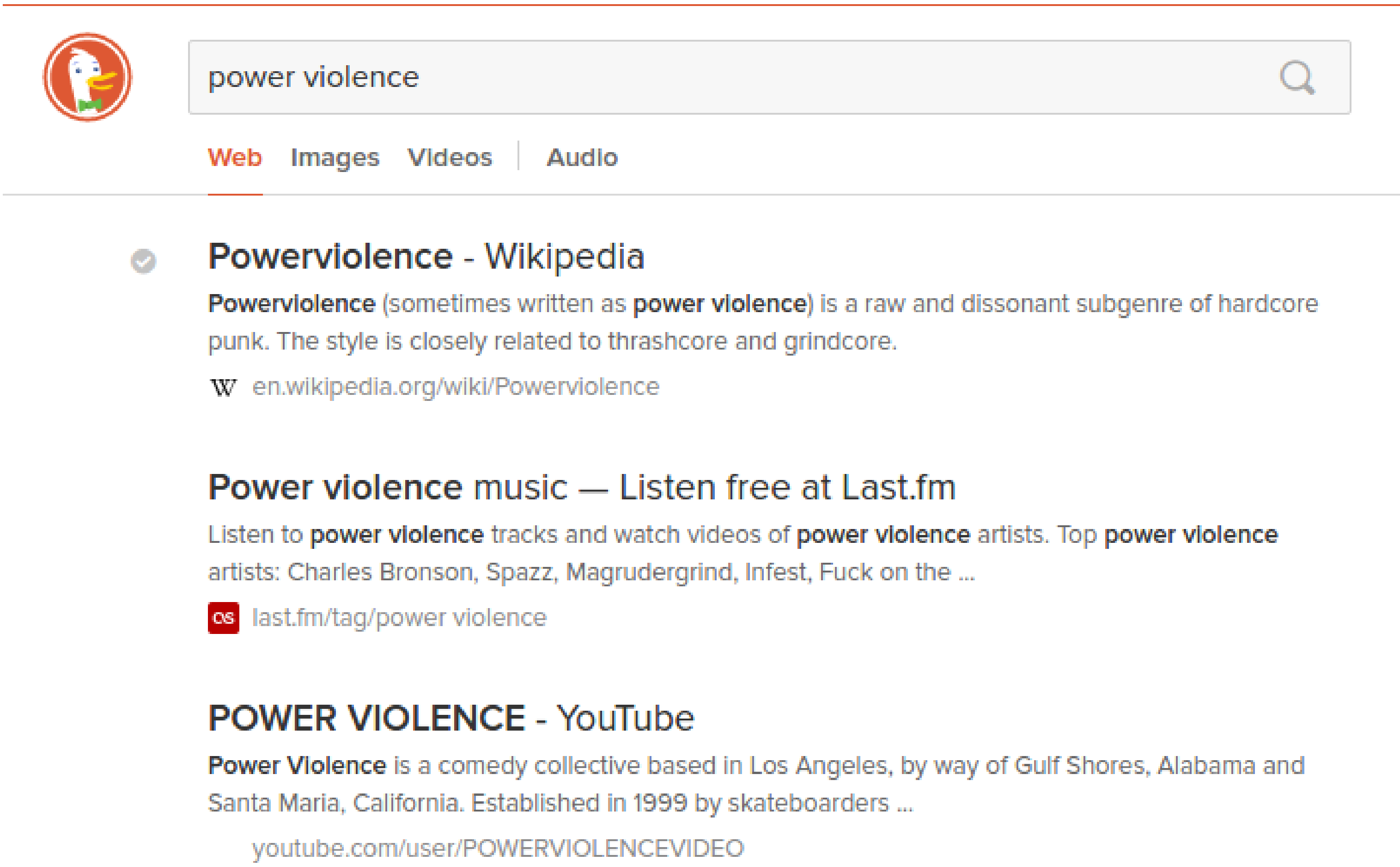
```
capitalize <- function(string) {
  substr(string, 1, 1) <- toupper(substr(string, 1, 1))
  string
}

min_y <- function(e,p) {
  emin <-  min(euclidMSE$mse)
  pmin <-  min(pearsonMSE$mse)
  if (emin < pmin)
    emin
  else
    pmin
}

max_y <- function(e,p) {
  emin <-  max(euclidMSE$mse)
  pmin <-  max(pearsonMSE$mse)
  if (emin > pmin)
    emin
  else
    pmin
}

min_x <- function(e,p) {
  emin <-  min(euclidMSE$user)
  pmin <-  min(pearsonMSE$user)
  if (emin < pmin)
    emin
  else
    pmin
}

max_x <- function(e,p) {
  emin <-  max(euclidMSE$user)
  pmin <-  max(pearsonMSE$user)
  if (emin > pmin)
    emin
  else
    pmin
}
```

Q.4 Question 11.2

```
Does your favorite web search engine use a bag of words representation?
How can you tell whether it does or doesn't?
```

Answer

Unlike the vast majority of the world my favorite search engine is duckduckgo.com because that's so metⒶl but in all seriousness I like it because it is a privacy first search engine and the EFF likes them. All right consider the following query seen in Figure 13.



power violence

Powerviolence - Wikipedia

Powerviolence (sometimes written as **power violence**) is a raw and dissonant subgenre of hardcore punk. The style is closely related to thrashcore and grindcore.

en.wikipedia.org/wiki/Powerviolence

Power violence music — Listen free at Last.fm

Listen to **power violence** tracks and watch videos of **power violence** artists. Top **power violence** artists: Charles Bronson, Spazz, Magrudergrind, Infest, Fuck on the ...

last.fm/tag/power violence

POWER VIOLENCE - YouTube

Power Violence is a comedy collective based in Los Angeles, by way of Gulf Shores, Alabama and Santa Maria, California. Established in 1999 by skateboarders ...

youtube.com/user/POWERVIOLENCEVIDEO

Figure 13: Powerviolence Query

The top three results, along with the others returned by this query, are about the best subgenre of punk powerviolence (pv). I purposely did not use the proper form of pv, as a single word, because in bag of words both "power" and "violence" would be in the "bag" when applying this retrieval model to build the result set. To better understand this consider the following two documents:

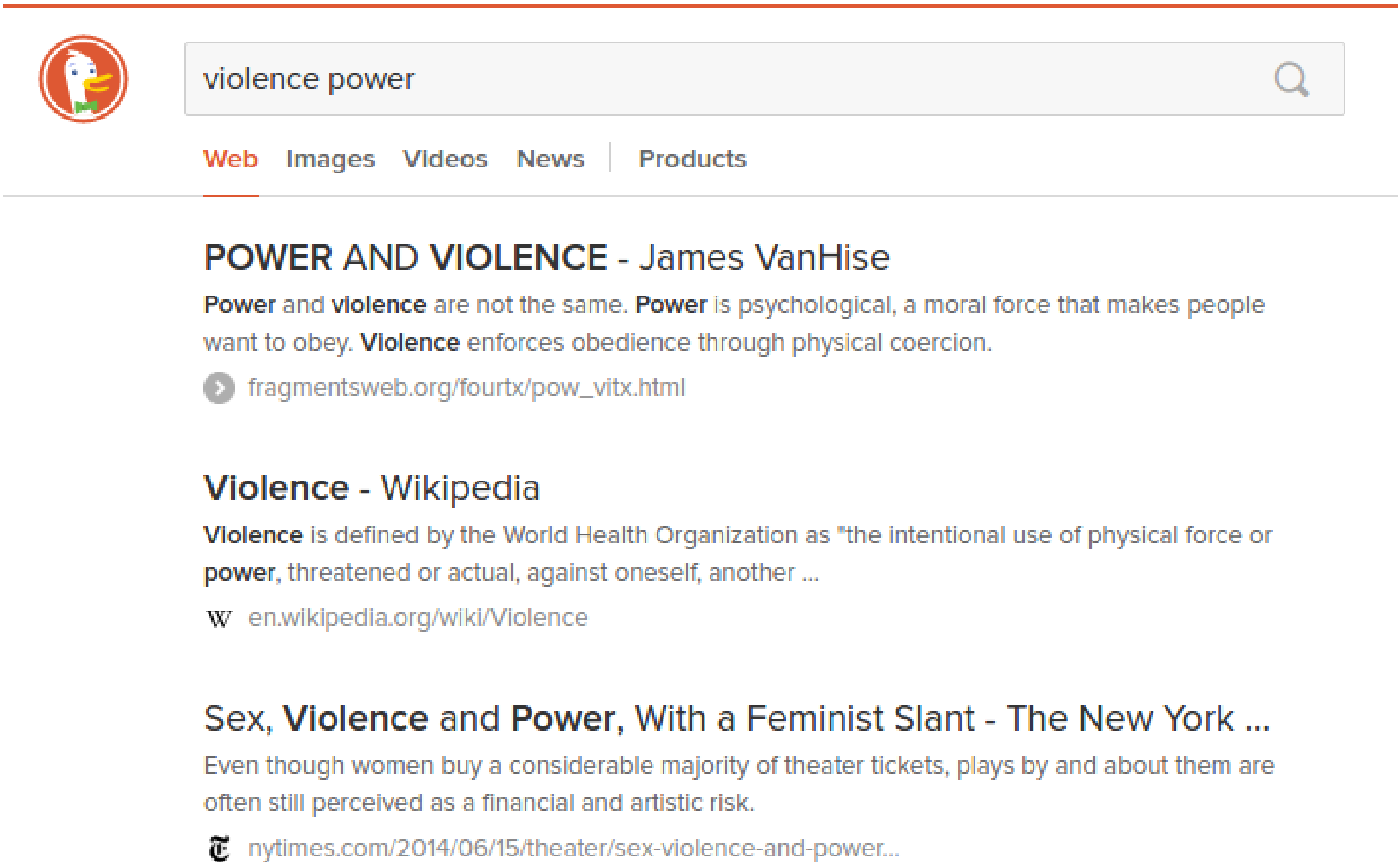
Doc1 Powerviolence (sometimes written as power violence) is a raw and dissonant subgenre of hardcore punk.

Doc2 The violence today at the power plant was extreme

In the bag of words model the two documents would be transformed into:

["Powerviolence", "sometimes", "written", "as", "power", "violence", "is", "a", "raw", "and", "dissonant", "subgenre", "of", "hardcore", "punk", "The", "violence", "today", "at", "the", "power", "plant", "was", "extreme"]

Because of this my query would have **Doc2** returned in the result set which is not desired. To further test this I issued the query *violence power* and the results can be seen in Figure 14.



violence power

POWER AND VIOLENCE - James VanHise

Power and **violence** are not the same. **Power** is psychological, a moral force that makes people want to obey. **Violence** enforces obedience through physical coercion.

fragmentsweb.org/fourtx/pow_vitx.html

Violence - Wikipedia

Violence is defined by the World Health Organization as "the intentional use of physical force or **power**, threatened or actual, against oneself, another ...

en.wikipedia.org/wiki/Violence

Sex, **Violence** and **Power**, With a Feminist Slant - The New York ...

Even though women buy a considerable majority of theater tickets, plays by and about them are often still perceived as a financial and artistic risk.

nytimes.com/2014/06/15/theater/sex-violence-and-power...

Figure 14: Violence Power Query

The top three results for this query do not have anything to do about this subgenre of punk rather the typical expectation for this combination words was returned. But the Wikipedia article for powerviolence was at position ten of the result set with all other results not about powerviolence.

Q.5 Extra Credit Wiki Small

```
Turn in with A5, 8 points available. Download the wikipedia.org URIs from the live web.

summarize the HTTP status codes returned,all 200? Redirections? 404s?
do we still have 6043 documents? or have their been splits & merges
 (see also: wikipedia disambiguation pages)? (1 pt)

compare and contrast the in- and out-links from this collection. the number,
the domains linked to, the HTTP status of the non-wikipedia links in the article
(i.e., are they 200, 404, or something else?). (1 pt)

Compare the least and most popular articles (in terms of in-links) in the 2008 snapshot
vs. the least and most popular articles in your 2015 snapshot. (1 pt)

compare and contrast the anchor text for the articles in the collection:
are we gaining or losing terms? (1 pt)

compare and contrast the sizes of the same pages (i.e., 2008 vs. 2015) in: (2 pts)
• bytes
• total terms
• total unique terms

for all of the pages (i.e., treat each collection as a single unit),
compare the 2008 vs. 2015 snapshots in: (2pts)
• bytes
• total terms
• total unique terms
```

Note

All http requests made while answering this question had a maximum timeout of 5 seconds this was to ensure that a lone url would not hold up the parallelization used to speed up the processing. The main code for this can be seen in Source Code 7, its helper file Source Code 8, the vocab code Source Code 9 and the R code used to generate the graphs Source Code 10.

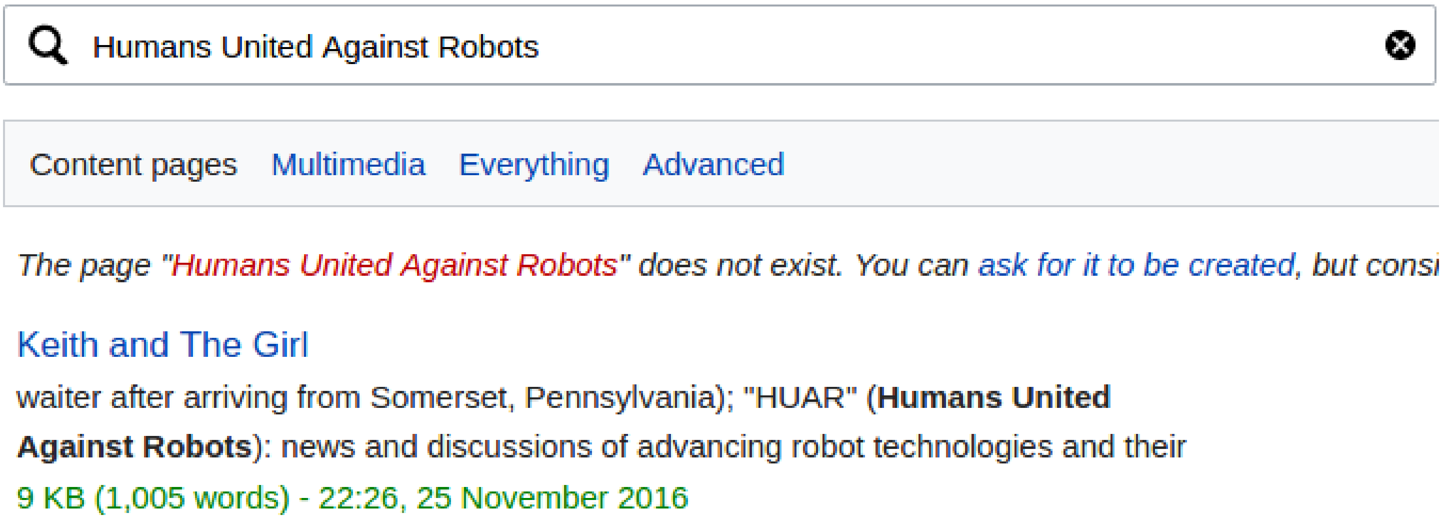
Wiki Small Statuses

The statuses for the 6043 documents can be seen in Table 1.

Table 1: Wiki Small Live Web Counterparts Http Statuses

status	count
200	5800
404	243

As seen in this table the page was either existing or non-existing, those that were not deemed not existing could not be retrieved using the wiki file name (after sensitization) appended to the following url https://en.wikipedia.org/wiki/. On looking up the pages on wikipedia itself some could be found again but the page url had changed but the majority of them had new pages containing the key words from the original. For example Saint-Hilaire_Lange_National_Park now resides at Saint-Hilaire/Lange_National_Park, Humans_United_Against_Robots has been merged into the page Keith and The Girl Figure 15 and the Lakshmana_Swamy page has been renamed to A. Lakshmanaswami Mudaliar but the original pages terms show up in multiple other pages Figure 16.



Humans United Against Robots

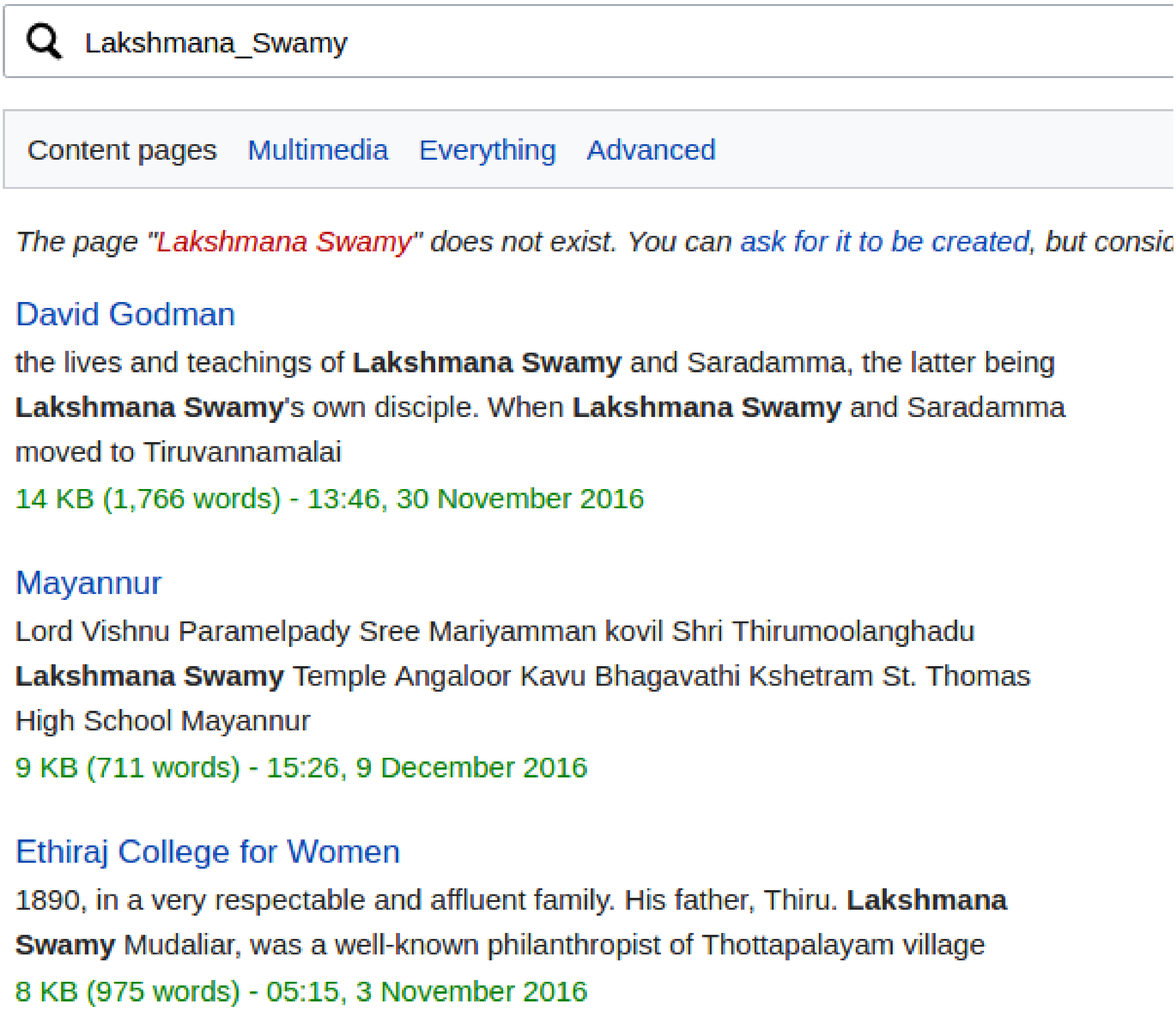
The page "Humans United Against Robots" does not exist. You can ask for it to be created, but consi

Keith and The Girl

waiter after arriving from Somerset, Pennsylvania); "HUAR" (**Humans United Against Robots**): news and discussions of advancing robot technologies and their

9 KB (1,005 words) - 22:26, 25 November 2016

Figure 15: Wsmall 404 Humans_United_Against_Robots



Lakshmana_Swamy

The page "Lakshmana Swamy" does not exist. You can ask for it to be created, but consic

David Godman

the lives and teachings of **Lakshmana Swamy** and Saradamma, the latter being **Lakshmana Swamy**'s own disciple. When **Lakshmana Swamy** and Saradamma moved to Tiruvannamalai

14 KB (1,766 words) - 13:46, 30 November 2016

Mayannur

Lord Vishnu Paramelpady Sree Mariyamman kovil Shri Thirumoolanghadu **Lakshmana Swamy** Temple Angaloor Kavu Bhagavathi Kshetram St. Thomas High School Mayannur

9 KB (711 words) - 15:26, 9 December 2016

Ethiraj College for Women

1890, in a very respectable and affluent family. His father, Thiru. **Lakshmana Swamy** Mudaliar, was a well-known philanthropist of Thottapalayam village

8 KB (975 words) - 05:15, 3 November 2016

Figure 16: Wsmall 404 Lakshmana_Swamy

Wiki Small Live Web Pages Outlink Statuses

There were a total of 80,686 outlinks for the live web pages and of those links 68,338 unique, Table 2 shows the counts for all statuses. I was surprised to find that the live web wiki pages contained links to ip addressed 25 to be exact. An attempt to retrieve them was made but the requests library[4] threw exceptions for all of them so they were skipped. Also to my surprise was the low number timeouts that happened during processing which was only 3,504.

Table 2: Wiki Small Live Web Counterparts Outlink Http Statuses

status	count
ip	25
timedout	3,504
200	42,315
203	1
204	2
300	1
301	12,606
302	3,423
303	2,253
307	29
400	35
401	36
403	242
404	3,963
405	546
406	1
408	1
409	2
410	27
415	2
429	6
500	103
501	5
502	33
503	68
504	1
520	15

During the retrieval of these statuses I noticed a good number of urls were to web archives, so naturally I had to find out how healthy these links are. Table 3 shows the total number of uri-ms for each archive. Please note that archive.org's count is different than web.archive.org's as archive.org pointed to archive collection where as the latter was a true uri-m. The archive with the most dominate presence was web.archive.org with 2,049 uri-ms which is 2.5% of the total out bound links.

Table 3: Web Archives Represented in Live Web Wiki Small Outlinks

Archive	Count	Percent Of Total Non-Wiki Outlinks
archive.org	263	0.325955%
archive.today	1	0.001239%
britishnewspaperarchive.co.uk	5	0.006197%
web.archive.org	2,049	2.539474%
webarchive.loc.gov	2	0.002479%
webarchive.nationalarchives.gov.uk	7	0.008676%
webcitation.org	269	0.333391%

Sadly as seen in Table 4 not all uri-ms are healthy but most of the non 200 are 3xx so their may be hope.

Table 4: Web Archives Outlink Statuses

Archive	Status	Count
archive.org	200	235
archive.org	301	1
archive.org	302	4
archive.org	403	1
archive.org	404	6
archive.today	301	1
britishnewspaperarchive.co.uk	302	5
web.archive.org	200	1,863
web.archive.org	302	78
web.archive.org	400	3
web.archive.org	403	9
web.archive.org	404	22
webarchive.loc.gov	200	2
webarchive.nationalarchives.gov.uk	200	7
webcitation.org	200	258

[4] http://docs.python-requests.org/en/master/

Also only 101 time outs where encountered during the uri-m health check as seen in table Table 5.

Table 5: Web Archives Outlinks Timed Out Count

Archive	Times Connection Timed Out
archive.org	16
web.archive.org	74
webcitation.org	11

Wiki Small Dataset vs Live Web Counterparts File Sizes

Figure 17 shows the file size differences for all 5800 live web counterparts. The difference was calculated by subtracting the live webs file size from the datasets. It must be noted that the dataset versions were pre-sanatized to remove all extra markup (including text not about the article) and given the age differences the dataset versions do not include any internal changes that wikipedia made to the pages representation.

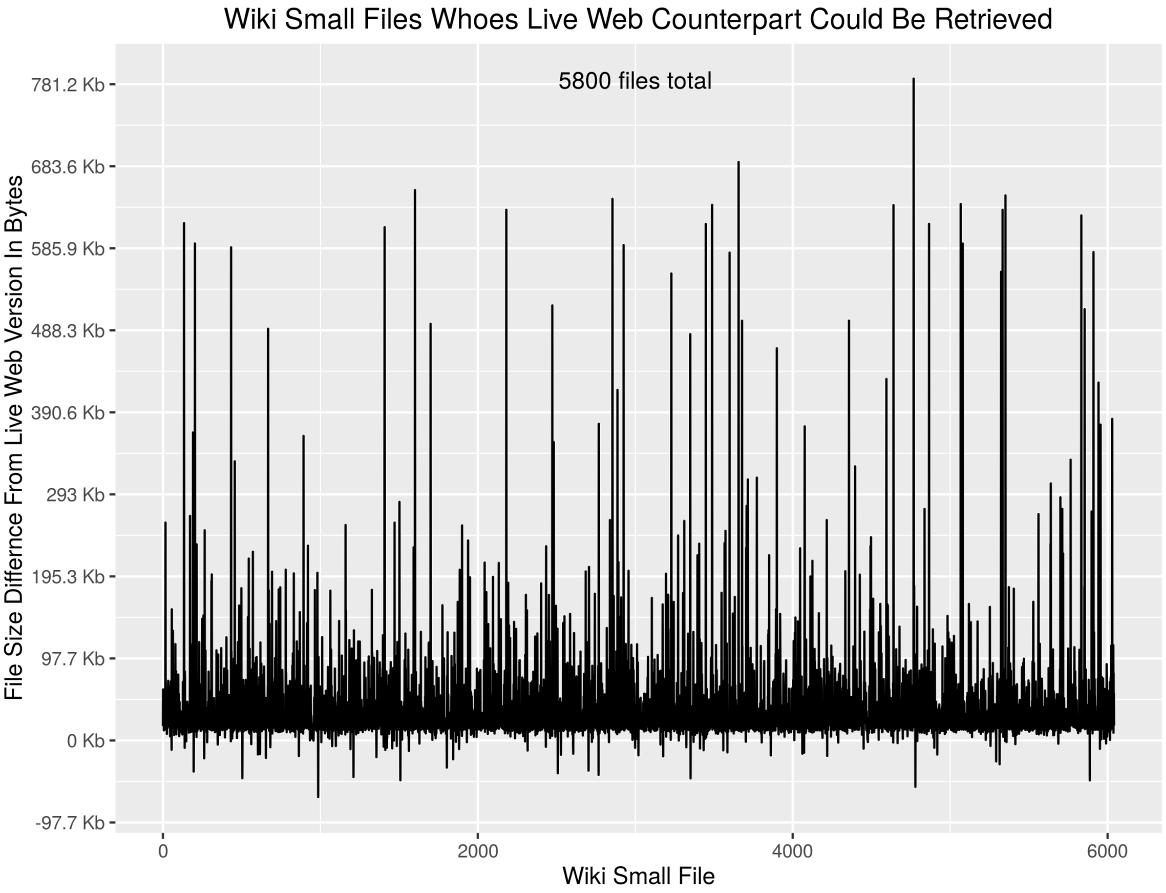


Figure 17: Wiki Small File Differences From Live Web Counterparts

Wiki Small Dataset vs Live Web Counterparts Vocabulary Growth

Since the live web files contain more mark up and text than the datasets I simply plotted the two counts using the code for assignment 2 Figure 18 shows this graph.

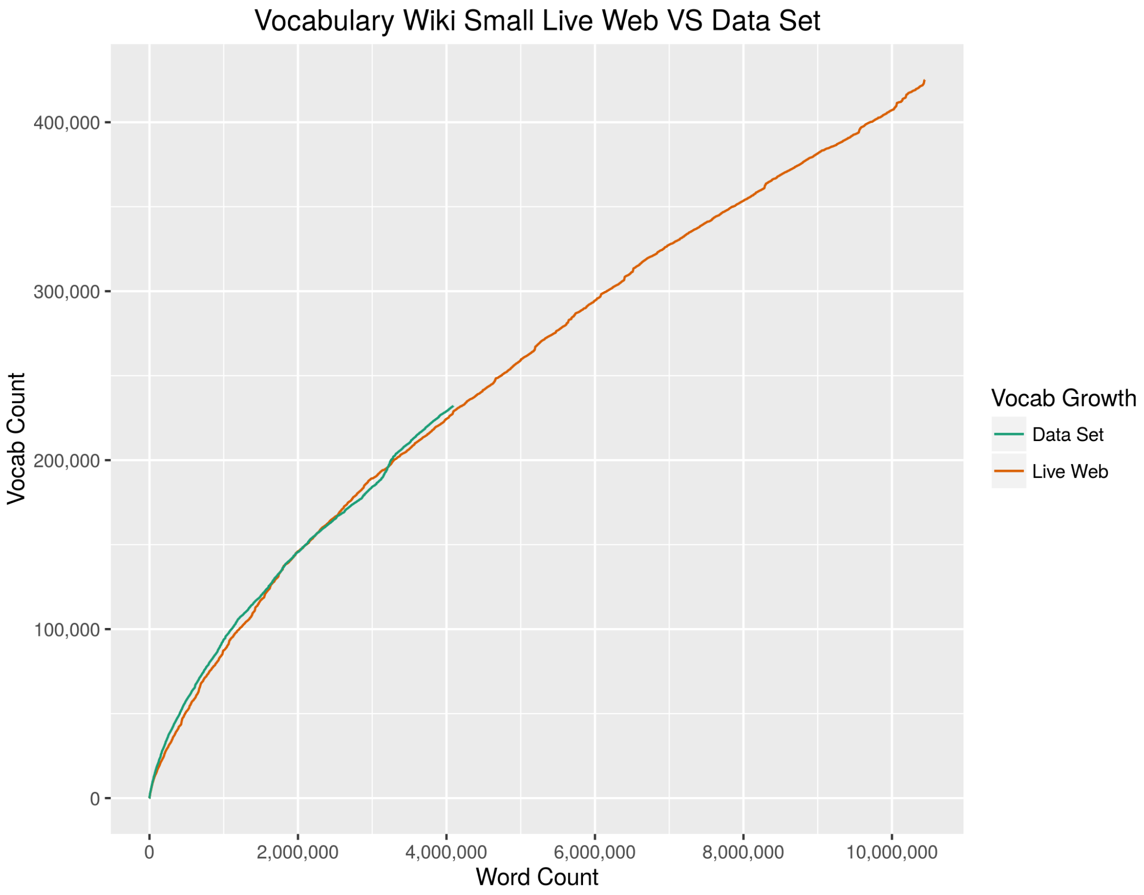


Figure 18: Wiki Small Vocab Growth Comparison

Source Code 7: Wiki Small Code

```
from wiki_helpers import *
from wiki_vocab import *
from contextClasses import AutoSLatexTable


def web_ar(ldf):
    web_ars = check_webarchive(ldf)
    if not os.path.exists('output_files/webarchives.table'):
        if not os.path.exists('pickled/web_ard.pickle'):
            web_ar_counter = web_ars.copy(deep=True)
            web_ar_counter.href = web_ars.href.map(archives_map)
            dump_pickle(web_ar_counter, 'pickled/web_ard.pickle')
        else:
            web_ar_counter = read_pickle('pickled/web_ard.pickle')
        hrefc = ldf[ldf.href.str.contains(r'(www)|(http)')][~ldf.href.str.contains('wiki')].href.count()
        with AutoSLatexTable(list, 'output_files/webarchives.table',
                             ['Archive', 'Count', 'Percent Of Total Outlinks']) as arout:
            for it in web_ar_counter.groupby('href'):
                ar, df = it
                hc = df.href.count()
                arout.append([ar, hc, '{0}%'.format('%f' % ((hc / hrefc) * 100))])
                print(ar, hc, '{0}%'.format('%f' % ((hc / hrefc) * 100)))
    if not os.path.exists('output_files/webarchives_statuses.table'):
        had_status, timed_out = check_ar_outlinks(web_ars)
        with AutoSLatexTable(list, 'output_files/webarchives_statuses.table',
                             ['Archive', 'Status', 'Count', ]) as arout:
            for it in had_status.groupby(by=['archive', 'status']):
                ((ar, stat), df) = it
                c = df.status.count()
                arout.append([ar, stat, c])
                print(ar, stat, c)
        with AutoSLatexTable(list, 'output_files/webarchives_statuses_timeout.table',
                             ['Archive',  'Times Timed Out', ]) as arout:
            for it in had_status.groupby('archive'):
                ar, df = it
                c = df.status.count()
                arout.append([ar,c])
                print(ar, c)

if __name__ == '__main__':
    small_list, _ = read_pickle('pickled/wsmall.pickle')
    tidy = tidy_uris(small_list)
    w_statues = dl_pages(tidy)
    extract_links_count_vocab(wstatues=w_statues)
    link_df = get_link_df()
    extract_links_old(small_list)
    web_ar(link_df)
    old_ldf = get_oldl_df()
    wn_lstats = compute_wnew_link_stats(link_df)
    wo_lstats = old_wsmall_lstats_df()
    di = domain_info(link_df)
    check_wiki_live_web_links()
    statuses_to_csv()
    vocab_small()
    vocab_small_new()
```

Source Code 8: Wiki Small Helper Code

```
import os
import re
import requests
import pandas as pd
import numpy as np
import tldextract
import csv
from tabulate import tabulate
from concurrent.futures import ProcessPoolExecutor
from requests_futures.sessions import FuturesSession
from collections import defaultdict, Counter
from urllib.parse import quote
from contextClasses import BeautifulSoupFromFile
from wiki_vocab import vocab_small_new
from util import read_pickle, dump_pickle

suffixes = ['B', 'KB', 'MB', 'GB', 'TB', 'PB']

size_compare = 'output_files/wsmall_sizes.csv'
status_csv = 'output_files/wsmalln_statuses.csv'
status2_csv = 'output_files/wsmalln_statuses2.csv'

useragents = [
    'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0) Gecko/20100101 Firefox/44.01',
    'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36',
    'Mozilla/5.0 (Linux; Ubuntu 14.04) AppleWebKit/537.36 Chromium/35.0.1870.2 Safari/537.36'
]

base = 'https://en.wikipedia.org/wiki/%s'
wiki_http_link = re.compile('^http(?:s)?:.+$')
match_end = re.compile('.+(_[0-9a-z]{4}\.html)$')

tidied_uirs = 'pickled/titdy_uris.pickle'
wsmall_statues = 'pickled/wsmall_statuses.pickle'
wsmall_statues_ar = 'pickled/wsmall_statuses_ar.pickle'
wsmall_statues_ar2 = 'pickled/wsmall_statuses2_ar.pickle'

wsmall_latest_p = 'wiki_small_latest/%s'

wsmall_latest_links = 'output_files/wsmall_latest.csv'
wsmall_latest_linkdf = 'pickled/wsmall_nlinks.pickle'
wsmall_latest_path = 'wiki_small_latest/%s'
```

```
wsmall_latest_webarchive = 'pickled/wsmall_nweba.pickle'
wsmall_latest_no_wlink = 'pickled/wsmall_no_wiki_links.pickle'
wsmall_latest_no_wlinkd = 'pickled/wsmall_no_wiki_linkswd.pickle'
wsmall_latest_nwl_status = 'pickled/wsmall_no_wiki_links_status.pickle'

wsmall_old_links = 'output_files/wsmall_old_links.csv'
wsmall_old_linkdf = 'pickled/wsmall_old_linksdf.pickle'

archiving_sites = ['http://www.britishnewspaperarchive.co.uk', 'http://webarchive.loc.gov',
                   'https://archive.org/', 'http://www.webcitation.org/',
                   'http://webarchive.nationalarchives.gov.uk', 'https://web.archive', 'http://www.archive.today']

ar_sites = ['britishnewspaperarchive.co.uk', 'webarchive.loc.gov',
            'webcitation.org',
            'webarchive.nationalarchives.gov.uk', 'archive.today']

arorg = 'archive.org'
warorg = 'web.archive.org'


def archives_map(it):
    for ars in ar_sites:
        if ars in it:
            return ars
    if arorg in it and warorg not in it:
        return arorg
    else:
        return warorg


def tidy_uris(s_list):
    if not os.path.exists(tidied_uirs):
        t = []
        for w in s_list:
            original = os.path.basename(w)
            m = match_end.findall(original)
            if len(m) > 0:
                p_name = original.replace(m[0], '')
            else:
                p_name = original.replace('.html', '')
            t.append({
                'quoted': quote(p_name),
                'unquoted': p_name,
                'original': original
            })
        dump_pickle(t, tidied_uirs)
        return t
    else:
        return read_pickle(tidied_uirs)


def dl_pages(tidied_uris):
    if not os.path.exists(wsmall_statues):
        useragent = 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0) Gecko/20100101 Firefox/44.01'
        statuses = {}
        with requests.session() as session:
            session.headers.update({'User-Agent': useragent})
            for tidy in tidied_uris:
                uri = base % tidy['unquoted']
                r = session.get(uri)
                scode = r.status_code
                print(uri, scode)
                statuses[tidy['original']] = scode
                if scode == 200:
                    with open('wiki_small_latest/%s' % tidy['original'], 'w') as out:
                        out.write(r.text)
        dump_pickle(statuses, wsmall_statues)
        return statuses
    else:
        return read_pickle(wsmall_statues)
```

```
def a_filter(a):
    if a is None:
        return False
    else:
        return a['href'][0] != '#'


def extract_links_count_vocab(wstatues):
    if not os.path.exists(wsmall_latest_links):
        with open(wsmall_latest_links, 'w') as out:
            out.write('wsmall_file,href\n')
            for o, scode in filter(lambda x: x[1] == 200, wstatues.items()):
                with BeautifulSoupFromFile(wsmall_latest_path % o, 'lxml') as soup:
                    all_a = soup.find_all('a', href=True)
                    for a in filter(a_filter, all_a):
                        out.write('%s,%s\n' % (o, a['href']))
        vocab_small_new()


def clean_new_wiki_name(name):
    m = match_end.findall(name)
    if len(m) > 0:
        return name.replace(m[0], '')
    else:
        return name.replace('.html', '')


def extract_links_old(wl):
    if not os.path.exists(wsmall_old_links):
        with open(wsmall_old_links, 'w') as out:
            out.write('wsmall_file,href\n')
            for wp in wl:
                with BeautifulSoupFromFile(wp, 'lxml') as soup:
                    all_a = soup.find_all('a', href=True)
                    for a in filter(a_filter, all_a):
                        href = a['href']
                        if href.startswith('../../../../articles') or href.startswith('../../../../'):
                            href = href[href.rfind('/') + 1:]
                            m = match_end.findall(href)
                            if len(m) > 0:
                                href = href.replace(m[0], '')
                            else:
                                href = href.replace('.html', '')
                        elif href.startswith('http') and '../../../../articles' in href:
                            # print(href[href.rfind('/')+1:])
                            href = href[href.rfind('/') + 1:]
                            m = match_end.findall(href)
                            if len(m) > 0:
                                href = href.replace(m[0], '')
                            else:
                                href = href.replace('.html', '')
                        original = os.path.basename(wp)
                        m = match_end.findall(original)
                        # print(m)
                        if len(m) > 0:
                            p_name = original.replace(m[0], '')
                        else:
                            p_name = original.replace('.html', '')
                        out.write('%s,%s\n' % (p_name, href))


def get_oldl_df():
    if not os.path.exists(wsmall_old_linkdf):
        names = ['wsmall_file', 'href']
        ldf = pd.read_csv(wsmall_old_links, sep=',', names=names)
        dump_pickle(ldf, wsmall_old_linkdf)
        return ldf
    else:
```

```python
        return read_pickle(wsmall_old_linkdf)


def get_link_df():
    if not os.path.exists(wsmall_latest_linkdf):
        names = ['wsmall_file', 'href']
        ldf = pd.read_csv(wsmall_latest_links, sep=',', names=names)
        dump_pickle(ldf, wsmall_latest_linkdf)
        return ldf
    else:
        return read_pickle(wsmall_latest_linkdf)


def check_webarchive(ldf):
    if os.path.exists(wsmall_latest_webarchive):
        archive_sites = []
        for archive_site in archiving_sites:
            archive_sites.append(ldf[ldf.href.str.startswith(archive_site)])
        about_wba = pd.concat(archive_sites)
        dump_pickle(about_wba, wsmall_latest_webarchive)
        return about_wba
    else:
        return read_pickle(wsmall_latest_webarchive)


def front_slash_nuke(r):
    if r.startswith('//'):
        return r.lstrip('//')
    return r


def domain_getter(href):
    d = tldextract.extract(href).registered_domain
    if d == '':
        return np.nan
    return d


def domain_info(ldf=None):
    if not os.path.exists(wsmall_latest_no_wlink):
        no_wlinks = ldf[ldf.href.str.contains(r'(www)|(http)')][~ldf.href.str.contains('wiki')]
        no_wlinks['href'] = no_wlinks['href'].apply(front_slash_nuke)
        dump_pickle(no_wlinks, wsmall_latest_no_wlink)
        if not os.path.exists(wsmall_latest_no_wlinkd):
            no_wlinks['domain'] = no_wlinks.href.map(domain_getter)
            dump_pickle(no_wlinks, wsmall_latest_no_wlinkd)
    else:
        no_wlinks = read_pickle(wsmall_latest_no_wlinkd)

    return no_wlinks


class LinkDict(dict):
    def __missing__(self, key):
        res = self[key] = 0
        return res


def compute_old_wsmall_links():
    if not os.path.exists('pickled/wsmallo_outl_temp.pickle'):
        old_link_df = get_oldl_df()
        out = defaultdict(LinkDict)
        for owfile in old_link_df.wsmall_file.unique():
            owdf = old_link_df[old_link_df.wsmall_file == owfile]
            links_to = old_link_df[old_link_df.wsmall_file.isin(owdf.href)]
            links_to_other = owdf[~owdf.href.isin(links_to.href)]
            out[owfile]['outlink_wsmall'] += links_to.wsmall_file.unique().size
            out[owfile]['outlink_other'] += links_to_other.wsmall_file.unique().size
            out[owfile]['total_outlinks'] += out[owfile]['outlink_wsmall'] + out[owfile]['outlink_other']
            for other in links_to.wsmall_file.unique():
                out[other]['inlink'] += 1
        dump_pickle(out, 'pickled/wsmallo_outl_temp.pickle')
        return out
    else:
        return read_pickle('pickled/wsmallo_outl_temp.pickle')


def old_wsmall_lstats_df():
    if not os.path.exists('pickled/wsmallo_outl_stats.pickle'):
        old_wsmall_links = compute_old_wsmall_links()
        it = {'wsmall': [], 'outlink_wsmall': [], 'outlink_other': [], 'total_outlinks': [], 'inlinks': []}
        for name in old_wsmall_links.keys():
            it['wsmall'].append(name)
            it['outlink_wsmall'].append(old_wsmall_links[name]['outlink_wsmall'])
            it['outlink_other'].append(old_wsmall_links[name]['outlink_other'])
            it['total_outlinks'].append(old_wsmall_links[name]['total_outlinks'])
            it['inlinks'].append(old_wsmall_links[name]['inlink'])
        lstats_df = pd.DataFrame(it)
        dump_pickle(lstats_df, 'pickled/wsmallo_outl_stats.pickle')
    else:
        lstats_df = read_pickle('pickled/wsmallo_outl_stats.pickle')

    return lstats_df


def check_ar_outlinks(ars):
    if not os.path.exists(wsmall_statues_ar):
        result = {}
        temp = []
        processed = 0
        c = 0
        with FuturesSession(session=requests.Session(), executor=ProcessPoolExecutor(max_workers=10)) as session:
            for href in ars.href:
                temp.append(href)
                if len(temp) >= 100:
                    pending = []
                    for url in temp:
                        result[url] = -1
                        pending.append(session.head(url, headers={'User-Agent': useragents[c]}, timeout=5.0))
                        c += 1
                        if c == 3:
                            c = 0
                    for future in pending:
                        try:
                            response = future.result()
                            url = response.url
                            scode = response.status_code
                            result[url] = scode
                        except Exception as e:
                            one = 1
                        processed += 1
                        if processed % 100 == 0:
                            print(processed)
                    temp.clear()
        print('outise the with')
        had_status = {'archive': [], 'status': []}
        timed_out = {'archive': [], 'status': []}
        for k, v in result.items():
            ar = archives_map(k)
            if v == -1:
                timed_out['archive'].append(ar)
                timed_out['status'].append(v)
                continue
            had_status['archive'].append(ar)
            had_status['status'].append(v)
        hs = pd.DataFrame(had_status)
        to = pd.DataFrame(timed_out)
        dump_pickle((hs, to), wsmall_statues_ar)
        return hs, to
    else:
        return read_pickle(wsmall_statues_ar)


def check_wiki_live_web_links():
    if not os.path.exists(wsmall_latest_nwl_status):
        link_df = get_link_df()
        di = domain_info(link_df)
        result = {}
        temp = []
        num = re.compile('^(?:(?:www\.)|(?:http://)|(?:https://))?(?:[0-9]{1,3}\.){3}.+$')
        c = 0
        processed = 0
        with FuturesSession(session=requests.Session(), executor=ProcessPoolExecutor(max_workers=10)) as session:
            for href in di.href.unique():
                if not href.startswith('http') and not href.startswith('https'):
                    href = '%s%s' % ('http://', href)
                if num.match(href) is not None:
                    result[href] = -2
                    continue
                temp.append(href)
                if len(temp) >= 1000:
                    pending = []
                    for url in temp:
                        result[url] = -1
                        pending.append(session.head(url, headers={'User-Agent': useragents[c]}, timeout=5.0))
                        c += 1
                        if c == 3:
                            c = 0
                    for future in pending:
                        try:
                            response = future.result()
                            url = response.url
                            scode = response.status_code
                            result[url] = scode
                        except Exception as e:
                            one = 1
                        processed += 1
                        if processed % 1000 == 0:
                            print(processed)
                    temp.clear()

        print('outise the with')
        dump_pickle(result, wsmall_latest_nwl_status)


def humansize(nbytes):
    if nbytes == 0: return '0 B'
    i = 0
    while nbytes >= 1024 and i < len(suffixes) - 1:
        nbytes /= 1024.
        i += 1
    t = ('%.2f' % nbytes).rstrip('0').rstrip('.')
    return '%s %s' % (t, suffixes[i])


def get_sizes(sl, t):
    if os.path.exists(size_compare):
        out = defaultdict(dict)
        for f in sl:
            size = os.path.getsize(f)
            out[os.path.basename(f)]['oldf_sizeh'] = humansize(size)
            out[os.path.basename(f)]['oldf_size'] = size
            out[os.path.basename(f)]['newf_size'] = -1
            out[os.path.basename(f)]['newf_sizeh'] = -1
        for it in t:
            try:
                size = os.path.getsize(wsmall_latest_p % it['original'])
                out[it['original']]['newf_size'] = size
                out[it['original']]['newf_sizeh'] = humansize(size)
            except:
                continue
        with open(size_compare, 'w') as sout:
            sout.write('old_size,old_sizeh,new_size,new_sizeh,dif,idx\n')
            c = 0
            difm = 0
            for wfile, sizes in out.items():
                print(wfile, sizes)
                if sizes['newf_size'] == -1:
                    dif = -sizes['oldf_size']
                else:
                    dif = sizes['newf_size'] - sizes['oldf_size']
                difm += dif
                sout.write('%d,%s,%d,%s,%d,%d\n' % (
                    sizes['oldf_size'], sizes['oldf_sizeh'], sizes['newf_size'], sizes['newf_sizeh'], dif, c))
                c += 1


def statuses_to_csv():
    if os.path.exists(status_csv):
        with open(status_csv, 'w') as sout, open(status2_csv, 'w') as sout2:
            sout.write('link,domain,suffix,idx,status\n')
            sout2.write('link,domain,suffix,idx,status\n')
            statuses = read_pickle(wsmall_latest_nwl_status)
            c = 0
            for link, status in statuses.items():
                tld = tldextract.extract(link)
                if status == -1:
                    sout2.write('%s,%s,%s,%d,%s\n' % (link, tld.domain, tld.suffix, c, 'timedout'))
                elif status == -2:
                    sout2.write('%s,%s,%s,%d,%s\n' % (link, tld.domain, tld.suffix, c, 'ip'))
                else:
                    sout.write('%s,%s,%s,%d,%d\n' % (link, tld.domain, tld.suffix, c, status))
                c += 1
    statuses = read_pickle(wsmall_latest_nwl_status)
    with open('output_files/status_count.table', 'w') as out:
        c = Counter()
        for link, status in statuses.items():
            if status == -1:
                c['timedout'] += 1
            elif status == -2:
                c['ip'] += 1
            else:
                c[status] += 1
        headers = ['status', 'count']
        data = []

        for k, v in c.items():
            data.append([k, v])
        out.write(tabulate(data, headers=headers, tablefmt='latex'))


def combine_vocab():
    with open('output_files/wsmall-vocabn.csv', 'r') as wnv, open('output_files/wsmall-vocab.csv', 'r') as wov, \
            open('output_files/wsmall-vocab-combines.csv', 'w') as wvc:
        wvc.write('wc,vc,which\n')
        for row in csv.DictReader(wnv):
            wvc.write('%s,%s,%s\n' % (row['wc'], row['vc'], 'Live Web'))
        for row in csv.DictReader(wov):
            wvc.write('%s,%s,%s\n' % (row['wc'], row['vc'], 'Data Set'))


def compute_wnew_link_stats(ldf):
    if not os.path.exists('pickled/wsmall_nlinkstats.pickle'):
        only_wiki = pd.concat(
            [ldf[ldf.href.str.contains('/wiki')], ldf[ldf.href.str.contains('en.wikipedia.org/wiki')]])
        only_wiki.href = only_wiki.href.apply(lambda h: os.path.basename(h))
        only_wiki.wsmall_file = only_wiki.wsmall_file.apply(clean_new_wiki_name)
        wn_lstats = defaultdict(LinkDict)
        for owfile in only_wiki.wsmall_file.unique():
            owdf = only_wiki[only_wiki.wsmall_file == owfile]
            links_to = only_wiki[only_wiki.wsmall_file.isin(owdf.href)]
            links_to_other = owdf[~owdf.href.isin(links_to.href)]
            wn_lstats[owfile]['outlink_wsmall'] += links_to.wsmall_file.unique().size
            wn_lstats[owfile]['outlink_other'] += links_to_other.wsmall_file.unique().size
            wn_lstats[owfile]['total_outlinks'] += wn_lstats[owfile]['outlink_wsmall'] + wn_lstats[owfile][
                'outlink_other']
            for other in links_to.wsmall_file.unique():
                wn_lstats[other]['inlink'] += 1
        it = {'wsmall': [], 'outlink_wsmall': [], 'outlink_other': [], 'total_outlinks': [], 'inlinks': []}
        for name in wn_lstats.keys():
            it['wsmall'].append(name)
            it['outlink_wsmall'].append(wn_lstats[name]['outlink_wsmall'])
            it['outlink_other'].append(wn_lstats[name]['outlink_other'])
            it['total_outlinks'].append(wn_lstats[name]['total_outlinks'])
            it['inlinks'].append(wn_lstats[name]['inlink'])
        lstats_df = pd.DataFrame(it)
        dump_pickle(lstats_df, 'pickled/wsmall_nlinkstats.pickle')
        return lstats_df
    else:
        return read_pickle('pickled/wsmall_nlinkstats.pickle')
```

Source Code 9: Wiki Small Vocab

```
import re
import random
import string
from util import wsmall, wsmalln, read_pickle, dump_pickle
from bs4 import BeautifulSoup
from nltk.tokenize import WordPunctTokenizer

no_wspace_punk = re.compile('(?:\s+)|[%s]' % string.punctuation)


def vocab(wfile, outfile):
    w_list, w_set = read_pickle(wfile)
    toke = WordPunctTokenizer()
    vocab = set()
    vocab_count = 0
    word_count = 0
    with open(outfile, 'w+') as vout:
        vout.write('wc,vc\n')
        for wf in w_list:
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        word_count += 1
                        if token not in vocab:
                            vocab.add(token)
                            vocab_count += 1
                            out = '%d,%d\n' % (word_count, vocab_count)
                            vout.write(out)


def vocab_backwards(wfile, outfile):
    w_list, w_set = read_pickle(wfile)
    toke = WordPunctTokenizer()
    vocab = set()
    vocab_count = 0
    word_count = 0
    with open(outfile, 'w+') as vout:
        vout.write('wc,vc\n')
        for wf in reversed(w_list):
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        word_count += 1
                        if token not in vocab:
                            vocab.add(token)
                            vocab_count += 1
                            out = '%d,%d\n' % (word_count, vocab_count)
                            vout.write(out)


def vocab_random(wfile, outfile):
    w_list, w_set = read_pickle(wfile)
    random.shuffle(w_list)
    toke = WordPunctTokenizer()
    vocab = set()
    vocab_count = 0
    word_count = 0
    with open(outfile, 'w+') as vout:
        vout.write('wc,vc\n')
        for wf in w_list:
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        word_count += 1
                        if token not in vocab:
                            vocab.add(token)
                            vocab_count += 1
                            out = '%d,%d\n' % (word_count, vocab_count)
                            vout.write(out)


def vocab_small():
    vocab(wsmall, 'output_files/wsmall-vocab.csv')
    vocab_backwards(wsmall, 'output_files/wsmall-vocabB.csv')
    vocab_random(wsmall, 'output_files/wsmall-vocabR.csv')


def vocab_small_new():
    vocab(wsmalln, 'output_files/wsmall-vocabn.csv')
    vocab_backwards(wsmalln, 'output_files/wsmall-vocabBn.csv')
    vocab_random(wsmalln, 'output_files/wsmall-vocabRn.csv')
```

Source Code 10: Wiki Small Plotter

```
import re
import random
from util import wsmall,wsmalln, read_pickle, dump_pickle
from bs4 import BeautifulSoup
from nltk.tokenize import WordPunctTokenizer

no_wspace_punk = re.compile('(?:\s+)|[%s]' % re.escape("""!"#$%&()*+,./:;<=>?@[\]^_{|}~"""))


def vocab(wfile, outfile):
    w_list, w_set = read_pickle(wfile)
    toke = WordPunctTokenizer()
    vocab = set()
    vocab_count = 0
    word_count = 0
    with open(outfile, 'w+') as vout:
        vout.write('wc,vc\n')
        for wf in w_list:
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        word_count += 1
                        if token not in vocab:
                            vocab.add(token)
                            vocab_count += 1
                            out = '%d,%d\n' % (word_count, vocab_count)
                            vout.write(out)


def vocab_backwards(wfile, outfile):
    w_list, w_set = read_pickle(wfile)
    toke = WordPunctTokenizer()
    vocab = set()
    vocab_count = 0
    word_count = 0
    with open(outfile, 'w+') as vout:
        vout.write('wc,vc\n')
        for wf in reversed(w_list):
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        word_count += 1
                        if token not in vocab:
                            vocab.add(token)
                            vocab_count += 1
                            out = '%d,%d\n' % (word_count, vocab_count)
                            vout.write(out)


def vocab_random(wfile, outfile):
    w_list, w_set = read_pickle(wfile)
    random.shuffle(w_list)
    toke = WordPunctTokenizer()
    vocab = set()
    vocab_count = 0
    word_count = 0
    with open(outfile, 'w+') as vout:
        vout.write('wc,vc\n')
        for wf in w_list:
            with open(wf, 'r') as wIn:
                wSoup = BeautifulSoup(wIn.read(), 'lxml')
                for token in toke.tokenize(no_wspace_punk.sub(' ', wSoup.text)):
                    if len(token) > 1:
                        word_count += 1
                        if token not in vocab:
                            vocab.add(token)
                            vocab_count += 1
                            out = '%d,%d\n' % (word_count, vocab_count)
                            vout.write(out)


def vocab_small():
    vocab(wsmall, 'output_files/wsmall-vocab.csv')
    vocab_backwards(wsmall, 'output_files/wsmall-vocabB.csv')
    vocab_random(wsmall, 'output_files/wsmall-vocabR.csv')


def vocab_small_new():
    vocab(wsmalln, 'output_files/wsmall-vocabn.csv')
    vocab_backwards(wsmalln , 'output_files/wsmall-vocabBn.csv')
    vocab_random(wsmalln , 'output_files/wsmall-vocabRn.csv')
```

Source Code 11: General Utility Functions

```
import pickle
from fs.osfs import OSFS

wsmall = 'pickled/wsmall.pickle'
wsmalln = 'pickled/wsmall_new.pickle'


def dump_pickle(obj, file):
    with open(file, 'wb') as out:
        pickle.dump(obj, out)


def read_pickle(name):
    with open(name, "rb") as input_file:
        return pickle.load(input_file)


def pick_file_list():
    ws = OSFS('wiki-small')
    small_list = []
    small_set = set()
    for file in ws.walkfiles():
        small_list.append('wiki-small%s' % file)
        small_set.add(file[file.rfind('/') + 1:])
    dump_pickle((small_list, small_set), 'pickled/wsmall.pickle')
    ws.close()


if __name__ == '__main__':
    pick_file_list()
```

Source Code 12: Modified NX Hits and Pagerank Algos

```
import networkx as nx
import numpy as np
import scipy


def hits(g, max_iter=5, normalized=True, tol=1.0e-6, nstart=None, ):
    if nstart is None:
        h = dict.fromkeys(g, 1.0 / g.number_of_nodes())
    else:
        h = nstart
        # normalize starting vector
        s = 1.0 / sum(h.values())
        for k in h:
            h[k] *= s
    i = 0
    while True:  # power iteration: make up to max_iter iterations
        hlast = h
        h = dict.fromkeys(hlast.keys(), 0)
        a = dict.fromkeys(hlast.keys(), 0)
        # this "matrix multiply" looks odd because it is
        # doing a left multiply a^T=hlast^T*g
        for n in h:
            for nbr in g[n]:
                a[nbr] += hlast[n] * g[n][nbr].get('weight', 1)
        # now multiply h=ga
        for n in h:
            for nbr in g[n]:
                h[n] += a[nbr] * g[n][nbr].get('weight', 1)
        # normalize vector
        s = 1.0 / max(h.values())
        for n in h: h[n] *= s
        # normalize vector
        s = 1.0 / max(a.values())
        for n in a: a[n] *= s
        # check convergence, l1 norm
        err = sum([abs(h[n] - hlast[n]) for n in h])
        if err < tol:
            break
        if i > max_iter:
            break
        i += 1
    if normalized:
        s = 1.0 / sum(a.values())
        for n in a:
            a[n] *= s
        s = 1.0 / sum(h.values())
        for n in h:
            h[n] *= s
    return h, a


def hits_scipy(G, max_iter=100, tol=1.0e-6, normalized=True):
    M = nx.to_scipy_sparse_matrix(G, nodelist=G.nodes())
    (n, m) = M.shape  # should be square
    A = M.T * M  # authority matrix
    x = scipy.ones((n, 1)) / n  # initial guess
    # power iteration on authority matrix
    i = 0
    while True:
        xlast = x
        x = A * x
        x = x / x.max()
        # check convergence, l1 norm
        err = scipy.absolute(x - xlast).sum()
        if err < tol:
            break
        if i > max_iter:
            break
        i += 1

    a = np.asarray(x).flatten()
    # h=M*a
    h = np.asarray(M * a).flatten()
    if normalized:
        h = h / h.sum()
        a = a / a.sum()
    h = dict(zip(G.nodes(), map(lambda x: '%.2f' % float(x), h)))
    a = dict(zip(G.nodes(), map(lambda x: '%.2f' % float(x), a)))
    return h, a


def pagerank_scipy(G, alpha=0.85, personalization=None,
                   max_iter=100, tol=1.0e-6, weight='weight',
                   dangling=None):
    N = len(G)
    if N == 0:
        return {}

    nodelist = G.nodes()
    M = nx.to_scipy_sparse_matrix(G, nodelist=nodelist, weight=weight,
                                  dtype=float)
    S = scipy.array(M.sum(axis=1)).flatten()
    S[S != 0] = 1.0 / S[S != 0]
    Q = scipy.sparse.spdiags(S.T, 0, *M.shape, format='csr')
    M = Q * M

    # initial vector
    x = scipy.repeat(1.0 / N, N)

    # Personalization vector
    if personalization is None:
        p = scipy.repeat(1.0 / N, N)
    else:
        missing = set(nodelist) - set(personalization)
        if missing:
            raise nx.NetworkXError('Personalization vector dictionary '
                                   'must have a value for every node. '
                                   'Missing nodes %s' % missing)
        p = scipy.array([personalization[n] for n in nodelist],
                        dtype=float)
        p = p / p.sum()

    # Dangling nodes
    if dangling is None:
        dangling_weights = p
    else:
        missing = set(nodelist) - set(dangling)
        if missing:
            raise nx.NetworkXError('Dangling node dictionary '
                                   'must have a value for every node. '
                                   'Missing nodes %s' % missing)
        # Convert the dangling dictionary into an array in nodelist order
        dangling_weights = scipy.array([dangling[n] for n in nodelist],
                                       dtype=float)
        dangling_weights /= dangling_weights.sum()
    is_dangling = scipy.where(S == 0)[0]

    # power iteration: make up to max_iter iterations
    for _ in range(max_iter):
        xlast = x
        x = alpha * (x * M + sum(x[is_dangling]) * dangling_weights) + \
            (1 - alpha) * p
        return dict(zip(nodelist, map(lambda x: '%.2f' % float(x), x)))
```

Source Code 13: Updated Context Classes

```
import csv
import pickle
import networkx as nx
from bs4 import BeautifulSoup
from functional import seq
from tabulate import tabulate
from random import sample


def build_lstransformer(func, seq_to='list'):
    if seq_to == 'list':
        return lambda ilist: seq(ilist).group_by(func).to_list()
    else:
        return lambda ilist: seq(ilist).group_by(func).to_dict()


def ltransformer_group_by(func):
    return lambda ilist: seq(ilist).group_by(func).to_list()


def default_ltransformer():
    return lambda ilist: ilist


def default_random_selector():
    return lambda ilist: sample(ilist, 1)[0]


def selected_list_joiner(selected, out):
    out.extend(selected)
```

```
def build_selected_joiner(func):
    def joiner(selected, out):
        return func((selected, out))

    return joiner


def default_formatter(item):
    return '%s\n' % item


def int_formatter(i):
    return '%d\n' % i


def build_formatter(format_s):
    return lambda item: format_s % item


class BeautifulSoupFromFile(object):
    def __init__(self, file_p, mode):
        self.file_p = file_p
        self.file_obj = open(file_p, 'r')
        self.mode = mode
        self.soup = None

    def __enter__(self):
        self.soup = BeautifulSoup(self.file_obj.read(), self.mode)
        return self.soup

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.file_obj.close()


class RunCommandSaveOut(object):
    def __init__(self, cline, file_p, print_file=False, command_fun=None):
        self.cline = cline
        self.file_p = file_p
        self.file_obj = open(file_p, 'w')
        self.print_file = print_file
        if command_fun is None:
            raise Exception('RunCommandSaveOut the command_fun was None')
        self.command_fun = command_fun

    def __enter__(self):
        spawn_process = self.command_fun(self.cline, self.file_obj)
        return spawn_process

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.file_obj.close()
        if self.print_file:
            with FileLinePrinter(self.file_p):
                one = 1


class AutoSaver(object):
    def __init__(self, save_type, file_name, formatter=default_formatter, selector=None):
        self.file_name = file_name
        self.save_type = save_type()
        self.formatter = formatter
        self.selector = selector

    def __enter__(self):
        return self.save_type

    def __exit__(self, exc_type, exc_val, exc_tb):
        if self.selector is None:
            outl = self.save_type
        else:
            outl = self.selector(self.save_type)
        with open(self.file_name, 'w') as out:
            for it in outl:
                out.write(self.formatter(it))


class AutoSaveCsv(object):
    def __init__(self, save_type, file_name, field_names):
        self.file_name = file_name
        self.save_type = save_type()
        self.field_names = field_names

    def __enter__(self):
        return self.save_type

    def __exit__(self, exc_type, exc_val, exc_tb):
        with open(self.file_name, 'w') as out:
            writer = csv.DictWriter(out, fieldnames=self.field_names)
            writer.writeheader()
            for it in self.save_type:
                writer.writerow(it)


class SelectFromFile(object):
    def __init__(self, q_file, selector, transformer=default_ltransformer()):
        self.q_file = open(q_file, 'r')
        self.line_transformer = transformer
        self.selector = selector

    def __enter__(self):
        lines = self.line_transformer(self.q_file.readlines())
        return self.selector(lines)

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.q_file.close()


class RandFinderFile(object):
    def __init__(self, q_file, transformer=default_ltransformer(), selector=default_random_selector()):
        self.q_file = open(q_file, 'r')
        self.line_transformer = transformer
        self.random_selector = selector

    def __enter__(self):
        lines = self.line_transformer(self.q_file.readlines())
        return self.random_selector(lines)

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.q_file.close()


class RandFinderSaver(RandFinderFile):
    def __init__(self, q_file, save_p, save_type, out_adder, formatter=default_formatter, sselector=None,
                 transformer=default_ltransformer(), selector=default_random_selector()):
        super(RandFinderSaver, self).__init__(q_file=q_file, transformer=transformer, selector=selector)
        self.selected = None
        self.save_p = save_p
        self.save_type = save_type
        self.formatter = formatter
        self.sselector = sselector
        self.out_adder = out_adder

    def __enter__(self):
        self.selected = super(RandFinderSaver, self).__enter__()
        return self.selected

    def __exit__(self, exc_type, exc_val, exc_tb):
        super(RandFinderSaver, self).__exit__(exc_tb, exc_val, exc_tb)
        with AutoSaver(self.save_type, self.save_p, formatter=self.formatter, selector=self.sselector) as out:
            self.out_adder(self.selected, out)
```

```
class FileLinePrinter(object):
    def __init__(self, file_p, line_transformer=lambda line: line.rstrip().lstrip()):
        self.file_p = open(file_p, 'r')
        self.line_transformer = line_transformer

    def __enter__(self):
        for line in map(self.line_transformer, self.file_p):
            print(line)
        return None

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.file_p.close()


class CleanFileLines(object):
    def __init__(self, file_p, remove_fun, save_back=False, sort_output=lambda x: x):
        self.file_p = file_p
        self.file_obj = open(file_p, 'r')
        self.remove_fun = remove_fun
        self.clean_lines = []
        self.save_back = save_back
        self.sort_output = sort_output

    def __enter__(self):
        for line in self.sort_output(self.file_obj):
            self.clean_lines.append(self.remove_fun(line))
        return self.clean_lines

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.file_obj.close()
        if self.save_back:
            with AutoSaver(list, self.file_p) as out:
                out.extend(self.clean_lines)


class GraphBuilder(object):
    def __init__(self, node_filep, edge_filep, nodes_config, edges_config, graph_type=None):
        self.node_filep = node_filep
        self.node_file_obj = open(node_filep, 'r')
        self.edge_filep = node_filep
        self.edges_config = edges_config
        self.nodes_config = nodes_config
        self.edge_file_obj = open(edge_filep, 'r')
        self.graph_type = graph_type

    def _make_empty_graph(self):
        if self.graph_type is None:
            g = nx.Graph()
        else:
            if self.graph_type in ['d', 'directed']:
                g = nx.DiGraph()
            elif self.graph_type in ['md', 'multid', 'multidirected']:
                g = nx.MultiDiGraph()
            else:
                g = nx.MultiGraph()
        return g

    def _read_edges(self, g):
        read_edges = self.edges_config['how_read']
        if read_edges == 'csv':
            csv_mapping = self.edges_config['csv_mapping']
            for row in csv.DictReader(self.edge_file_obj):
                g.add_edge(row[csv_mapping['node']], row[csv_mapping['edge']])
        elif read_edges == 'one_edge_per_line':
            for edge in self.edge_file_obj:
                n, e = edge.rstrip().lstrip().split(self.edges_config['sep'])
                g.add_edge(n, e)
        else:
            for edge in self.edge_file_obj:
                first = True
                n = None
                for it in edge.rstrip().lstrip().split(self.edges_config['sep']):
                    if first:
                        n = it
                        first = False
                    else:
                        g.add_edge(n, it)

    def _read_nodes(self, g):
        read_nodes = self.nodes_config['how_read']
        if read_nodes == 'one_line':
            for n in self.node_file_obj.read().rstrip().lstrip().split(self.nodes_config['sep']):
                g.add_node(n)
        else:
            for n in self.node_file_obj:
                n = n.rstrip().lstrip()
                g.add_node(n)

    def __enter__(self):
        g = self._make_empty_graph()
        self._read_nodes(g)
        self._read_edges(g)
        return g

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.node_file_obj.close()
        self.edge_file_obj.close()


class AutoSLatexTable(AutoSaveCsv):
    def __init__(self, save_type, file_name, field_names):
        super(AutoSLatexTable, self).__init__(save_type, file_name, field_names)

    def __exit__(self, exc_type, exc_val, exc_tb):
        with open(self.file_name, 'w') as out:
            out.write(tabulate(self.save_type, headers=self.field_names, tablefmt='latex'))


class AutoSaveTwoFileTypes(object):
    def __init__(self, f_clazz, f_filep, f_type, f_fn, s_clazz, s_filep, s_fn, s_type):
        self.f = f_clazz(f_type, f_filep, f_fn)
        self.s = s_clazz(s_type, s_filep, s_fn)

    def __enter__(self):
        return self.f.__enter__(), self.s.__enter__()

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.f.__exit__(exc_type, exc_val, exc_tb)
        self.s.__exit__(exc_type, exc_val, exc_tb)


def class_for_name(name):
    return globals().copy().get(name, None)
```