CS168 Spring Assignment 2

SUNet ID(s): johngold jsalloum
Name(s): Jonathan Gold Justin Salloum
Collaborators: None

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Part 1

(b)

Figure 1: Cosine Distance

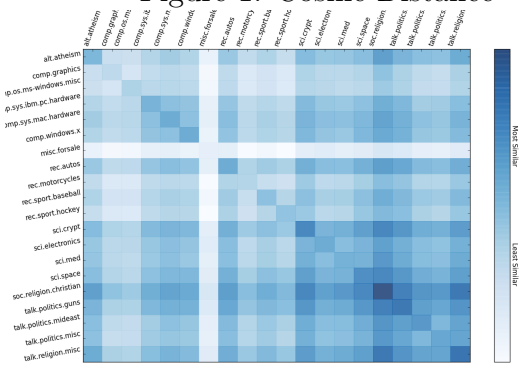


Figure 2: Jaccard Distance

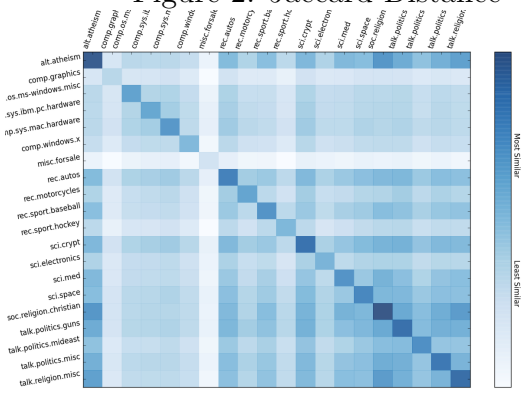
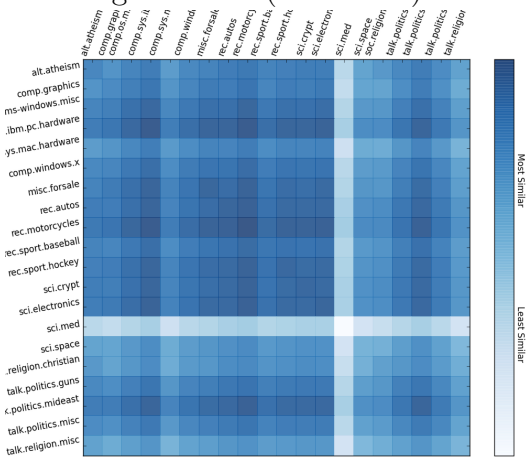


Figure 3: L2 (Euclidean) Distance



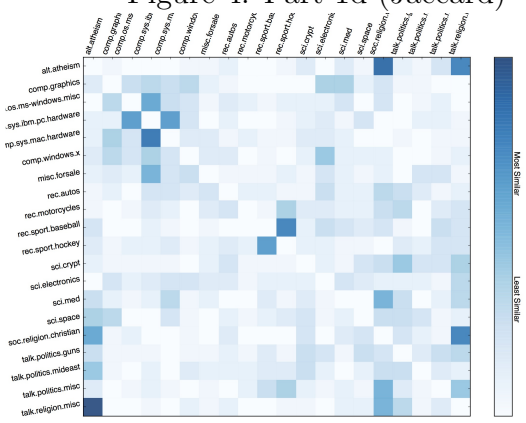
(c)

Jaccard distance and Cosine distance result in similar heatmaps. The diagonals are dark, which is to be expected, because articles from the same group should be similar to each other. They both also show misc.forsale as different from pretty much everything else. This is reasonable as that is a miscellaneous category and articles could be quite different from eachother.

The Euclidean distance (L2) is much darker than the other plots, which means it considers more articles similar with each other. However, the sci.med category is most dissimilar from everything else. Therefore we conclude both Cosine and Jaccard are most reasonable.

In terms of articles from different groups that are similar, miscellaneous religion, christian religion, and atheism all have high similarity scores. Additionally, the midest group is similar to the politics group. Both of these comparisons make sense intuitively

Figure 4: Part 1d (Jaccard)



(e)

The plots from (b) were all symmetric but (d) was not. Average similarity between A and B is the same as average similarity between B and A, that explains the symmertry in (b). However, when we are calculating the values for (b), the most similar article to one in group A might be in group B, but that article from B could be most similar to another article in group C, not necessarily the same one from A.

(f)

In (d) the most similar plots appear to be sports.hockey and sports.baseball. Misc.religion and atheism are similar. mac.hardware, ibm-pc.hardware, and windows.hardware are all similar as well. These results are to be expected, but only the religion similarity is as obvious in part (b). The hockey/baseball and mac/pc/hardware similarites are not as noticable.

Part 2

(a)

d=10 took 12 seconds
d=25 took 16 seconds
d=50 took 19 seconds
d=100 took 20 seconds
(original comparison took 4 min)

The computations are really fast for all of these dimensions, especially compared to the original non-reduced matrix. However, it seems that 25 dimensions is the sweet spot, due to similarity between the original part 1d. Surprisingly, d=100 seems to be less similar than our original non reduced part 1d, though we are using a different similarity metric.

Figure 5: 10 Dimensions

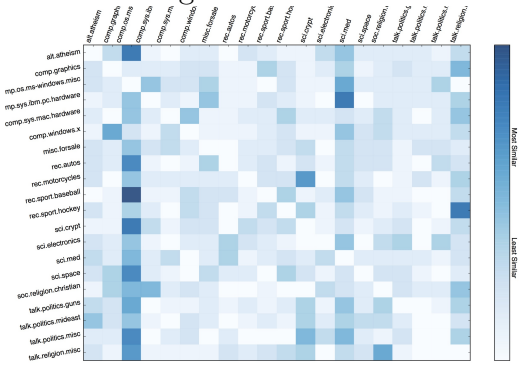


Figure 6: 25 Dimensions

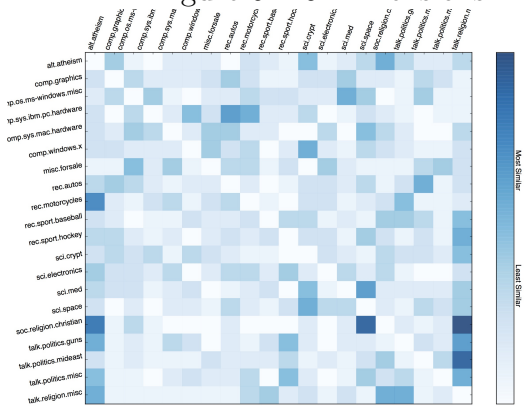
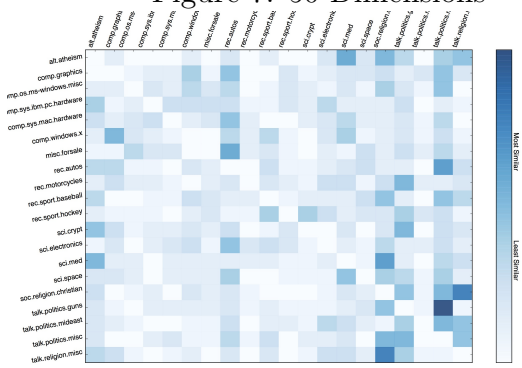


Figure 7: 50 Dimensions



(b)

The first plot (10 dimensions) is close to approximating a vertical line, while as we increase dimensions the plot becomes closer to a diagonal (45 degree) line. The diagonal line is desired because that means the distance is approximately equal for both the original and reduced dimension matrix. So for the 100 dimension case, there is not nearly as much error as there is for the 10 dimension case.

The error in this case is multiplicative. The most important step in reducing the error is when we go from 10 to 25, a multiplicative factor of 2.5, while only adding 15. When we go from 25 to 50, there isn't much of a change, although the additive difference is greater than going from 10 to 25 dimensions.

Figure 8: 100 Dimensions

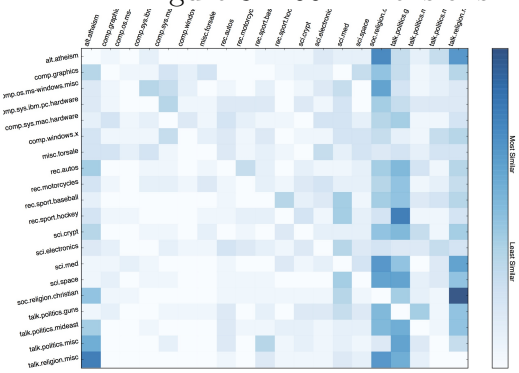


Figure 9: 10 Dimensions

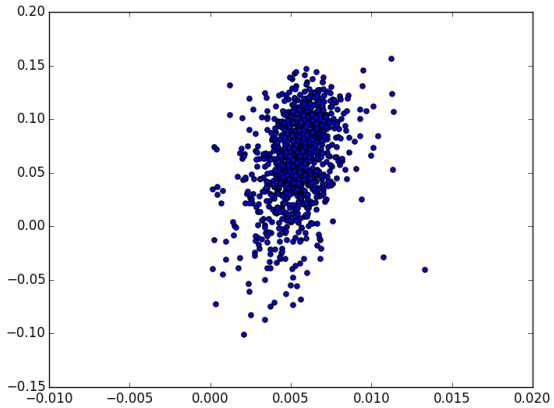


Figure 10: 25 Dimensions

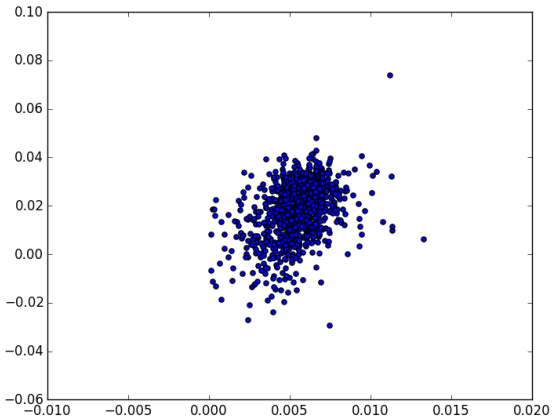
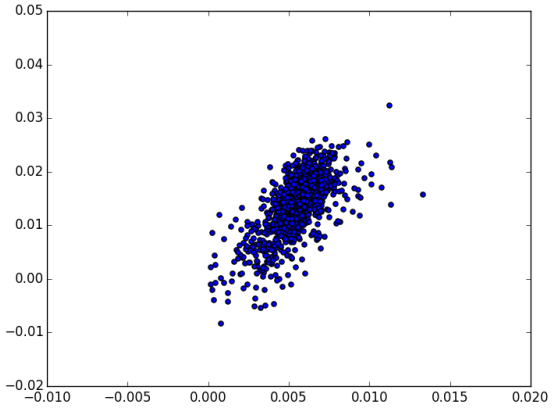


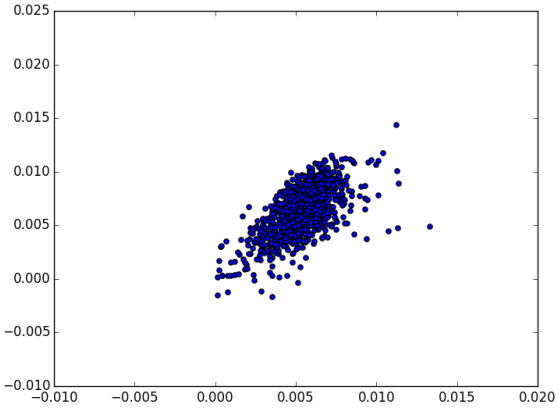
Figure 11: 50 Dimensions




```
#p2.py
from datetime import datetime
import numpy as np
from numpy import genfromtxt
import csv
from matplotlib import pyplot as plt
from scipy.sparse import csc_matrix
from scipy.sparse import csr_matrix
import scipy.spatial.distance


def main():
    #similarity_metrics3(l2)
```

Figure 12: 100 Dimensions




```
    article_matrix = read_csvs3()
    reduced_matrix = dimensionality_reduction(d=25, mat=article_matrix)
    #part_1d2(article_matrix)
    part_1d2(reduced_matrix)

    #part_2b(reduced_matrix, article_matrix)



#Makes a scatter plot between article 3 and all other articles in both matric
def part_2b(reduced_matrix, original_matrix):
    x_coords = []
    y_coords = []
    for i in range(1000):
        #Compute distance from article_3 to each of the matricies
        x_coords.append(cosine(original_matrix[2, :], original_matrix[i, :]))
        y_coords.append(cosine(reduced_matrix[2, :], reduced_matrix[i, :]))

    #Plot scatter of all the
    plt.scatter(x_coords,y_coords)
    plt.show()


#takes our N x K matrix, and converts it to a (N x D) matrix by using a D x K
def dimensionality_reduction(d=10, mat=None):
    n, k = mat.shape
```

```
    random_matrix = np.random.normal(0, 1, (d, k))
    new_matrix = np.zeros((n, d))
    for i in range(n):
        new_matrix[i , :] = (random_matrix.dot(mat[i , :]))
    new_matrix *= 1.0/(np.sqrt(d)) #From the lecture notes, not necessarily t
    return new_matrix


#Trying again with the giant matrix
def part_1d2(article_matrix):
    similarity_data = np.zeros((20,20))
    start = datetime.now()

    for i in range(20):
        for x in range(50):
            article_a = i*50 + x
            most_similar = -1
            winning_group = -1
            #print article_a
            start2 = datetime.now()

            for j in range(20):
                if(j != i):
                    for y in range(50):
                        article_b = j*50 + y

                        similarity = cosine(article_matrix[article_a , :], art

                        if(similarity > most_similar):
                            most_similar = similarity
                            winning_group = j
            #At the end of the article, add similarity info
            similarity_data[i, winning_group] += 1
    print "This took : ", datetime.now()-start, " seconds"

    groups = get_group_names()
    plot_heatmat(similarity_data, groups, groups)


#Uses the giant matrix form and some hard coding, sorryyy
def similarity_metrics3(sim_metric):
    article_matrix = read_csvs3()
```

```
    similarity_data = np.zeros((20,20))
    start = datetime.now()
    for i in range(20):
        for x in range(50):
            article_a = i*50 + x
            print article_a
            for j in range(20):
                for y in range(50):
                    article_b = j*50 + y
                    similarity_data[i, j] += sim_metric(article_matrix[articl
    print "This took ", datetime.now() - start, " seconds"
    groups = get_group_names()
    #print datetime.now().time()
    similarity_data /= 1000
    plot_heatmat(similarity_data, groups, groups)
    #print similarity_data
    #print similarity_data / 1000



#Plots a heatmap given a numpy array of data and labels
#http://stackoverflow.com/questions/14391959/heatmap-in-matplotlib-with-pcolo
def plot_heatmat(data, row_labels, col_labels):
    fig, ax = plt.subplots()
    heatmap = ax.pcolor(data, cmap=plt.cm.Blues, alpha=0.8)
    # put the major ticks at the middle of each cell
    ax.set_xticks(np.arange(data.shape[0])+0.5, minor=False)
    ax.set_yticks(np.arange(data.shape[1])+0.5, minor=False)
    # want a more natural, table-like display
    ax.invert_yaxis()
    ax.xaxis.tick_top()
    #More stuff from the stackoverflow...
    ax.set_xticklabels(row_labels, minor=False, rotation = 70)
    ax.set_yticklabels(col_labels, minor=False, rotation = 10)

    cbar = plt.colorbar(heatmap)

    cbar.ax.get_yaxis().set_ticks([])
    cbar.ax.get_yaxis().labelpad = 15
    cbar.ax.set_ylabel('Most Similar
Least Similar', rotation=270)
```

```
    plt.show()



def get_avg_similarity(a_articles, b_articles, words_in_articles, sim_metric)
    total_similarity = 0.0
    count = 0.0
    for article_a in a_articles:
        for article_b in b_articles:
            similarity = sim_metric(words_in_articles[article_a], words_in_a
            count += 1.0
            total_similarity += similarity
            #print similarity
            #raw_input("")
    return total_similarity/count


#Simple function that reads CSV of group names
def get_group_names():
    groups = []
    group_file = open('p2_dataset/groups.csv', 'rb')
    for line in group_file.readlines():
        groups.append(line.strip())
    return groups


#Use the numpy stuff, vectorized code, no loops
#into each of these functions, pass 2 vectors (np arrays)
def jaccard(x,y):
    return np.sum(np.minimum(x,y))/np.sum(np.maximum(x,y))

def l2(x,y):
    return -np.sqrt(np.sum((x - y)**2))

def cosine(x,y):
    return np.sum(x*y)/(np.sum(np.abs(x)) * np.sum(np.abs(y)))




#numpy 1000 x 61100 matrix
def read_csvs3():
```

```
    data = genfromtxt('p2_dataset/data50.csv', delimiter=' ')
    matrix = np.zeros((1000,61100))
    counter = 0
    prev_article_id = 1
    for article_id, word_id, count in data:
        if(article_id != prev_article_id):
            counter += 1
        matrix[counter, word_id] = count
        prev_article_id = article_id
    return matrix



if __name__ == "__main__":
    main()
```