CS5785 Fall 2018: Homework 3

Matheus Clemente Bafutto (mc2722)

Sungseo Park (sp2528)

Q1. Sentiment analysis of online reviews.

Q1(a). Parse each file with the specifications in readme.txt. Are the labels balanced? If not, what's the ratio between the two labels? Explain how you process these files.

```
amazon_data = [i.split('\t')[0] for i in amazon_lines]
amazon_label = [i.split('\t')[1] for i in amazon_lines]
imdb_data = [i.split('\t', 1)[0] for i in imdb_lines]
imdb_label = [i.split('\t', 1)[1] for i in imdb_lines]
yelp_data = [i.split('\t', 1)[0] for i in yelp_lines]
yelp_label = [i.split('\t', 1)[1] for i in yelp_lines]

# Count label0 and 1 from Amazon, Imdb and Yelp
print("Amazon - Label0 count: "+str(amazon_label.count('0'))+", Label1 count: "+str(amazon_label.count('1')))
print("Imdb - Label0 count: "+str(imdb_label.count('0'))+", Label1 count: "+str(imdb_label.count('1')))
print("Yelp - Label0 count: "+str(yelp_label.count('0'))+", Label1 count: "+str(yelp_label.count('1')))
```

```
Amazon - Label0 count: 500, Label1 count: 500
Imdb - Label0 count: 500, Label1 count: 500
Yelp - Label0 count: 500, Label1 count: 500
```

I read data from amazon, imdb and yelp and split into data set and label set. The I counted positive and negative review based on the labels. The result shows that the labels are balanced.

Q1(b). Pick your preprocessing strategy.
Explain the reasons for each of your decision (why or why not).

- Lowercase all of the words: We converted all of the words into the lowercase because we didn't want to count the duplicate words written in different cases.

```
# Q1-b: Remove noise and garbage of the data
# Lowercase all of the words
def lowercase(data):
    return [x.lower() for x in data]
```

- Strip punctuation: We removed all the punctuations because, for example, we didn't want to consider 'hello,' and 'hello' as different words.

```
def strip_punctuation(data):
    for i in range(len(data)):
        data[i]=''.join([letter for letter in data[i] if letter not in punctuation])
    return data
```

- Strip the stop words, e.g., "the", "and", "or": We removed all the stop words because these words don't convey any sentiment.

```
# Q1-b: Remove noise and garbage of the data
# Strip the stop words
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

stop_words = set(stopwords.words('english'))
```

```
-   def strip_stop_words(data):
-       for i in range(len(data)):
-           word_tokens = word_tokenize(data[i])
-           filtered_sentence = []
-           for word in word_tokens:
-               if word not in stop_words:
-                   filtered_sentence.append(word)
-           data[i] = " ".join(filtered_sentence)
-       return data
```

- Lemmatization of all the words: we processed Lemmatization to group together the inflected forms of a word so they can be analyzed as a single item.

```
-   # Q1-b: Remove noise and garbage of the data
-   # Lemmatization of all the words, ex) good, well, better, and best -> good
-   from nltk.stem import WordNetLemmatizer
-   def lemmatizer(data):
-       lemmatizer = WordNetLemmatizer()
-       for i in range(len(data)):
-           word_list = data[i].split(' ')
-           temp = []
-           for word in word_list:
-               temp.append(lemmatizer.lemmatize(word, 'v'))
-           data[i] = " ".join(temp)
-       return data
```

Q1(c). Split training and testing set.

```
1.  # Q1-c: Split training and testing set
2.  data_training_set = amazon_data[:400] + amazon_data[500:900] + imdb_data[:400] + imdb_data[500:900] +
     yelp_data[:400] + yelp_data[500:900]
3.  label_training_set = amazon_label[:400] + amazon_label[500:900] + imdb_label[:400] + imdb_label[500:9
    00] + yelp_label[:400] + yelp_label[500:900]
4.  data_testing_set = amazon_data[400:500] + amazon_data[900:1000] + imdb_data[400:500] + imdb_data[900:
    1000] + yelp_data[400:500] + yelp_data[900:1000]
5.  label_testing_set = amazon_label[400:500] + amazon_label[900:1000] + imdb_label[400:500] + imdb_label
    [900:1000] + yelp_label[400:500] + yelp_label[900:1000]
```

Q1(d). Bag of Words model.

We built a dictionary of unique words and generated feature vectors using the dictionary for training and testing set.

```
1.  # Q1-d: Bag of Words model
2.  data_training_list = ' '.join(data_training_set).split(" ")
3.  unique_words_dict = list(set(data_training_list))[1:]
4.
5.  # Q1-d: Generate feature vector for training and testing set
6.  def generate_feature_vector(data):
7.      feature_vectors = []
8.      for x in data:
9.          word_list = x.split(" ")
10.         count_list = []
11.         for word in unique_words_dict:
12.             count_list.append(word_list.count(word))
13.         feature_vectors.append(count_list)
14.     return feature_vectors
15.
16. training_feature_vector = generate_feature_vector(data_training_set)
17. testing_feature_vector = generate_feature_vector(data_testing_set)
18.
19. print(training_feature_vector[0])
20. print(training_feature_vector[1])
```

```
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
```

Q1(e). Pick your postprocessing strategy.
Whatever choices you make, explain why you made the decision.

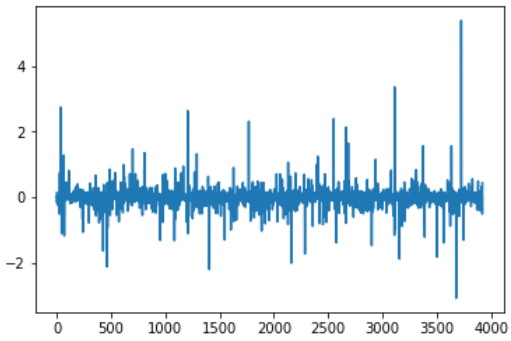
Since the data is sparsely distributed, we applied L1 normalization because L1 is suitable for sparseness of data.

```
# Q1-e: Pick your postprocessing strategy
from sklearn import preprocessing
def l2_normalization(x):
    return preprocessing.normalize(x, norm="l1")
```

Q1(f). Sentiment prediction.
Report the classification accuracy and confusion matrix. Inspecting the weight vector of the logistic regression, what are the words that play the most important roles in deciding the sentiment of the reviews? Repeat this with a Naive Bayes classifier and compare performance.

Classification accuracy for a logistic regression: 0.78
Confusion matrix for a logistic regression: [[288 47] [83 182]]
Words that play the most important roles for a logistic regression:
Positive words - ['good', 'excellent', 'best', 'nice', 'amaze', 'love', 'great']
Negative words - ['terrible', 'worst', 'poor', 'bad']

```
# Q1-
f: Sentiment prediction using logistic regression and report the classification accuracy and confusio
n matrix.
from sklearn import linear_model
from sklearn.metrics import confusion_matrix

logistic_reg = linear_model.LogisticRegression()
logistic_reg.fit(training_feature_vector, label_training_set)
classification_accuracy = logistic_reg.score(testing_feature_vector, label_testing_set)
print("Classification accuracy using a Logistic Regression:\n" + str(classification_accuracy) + '\n')

y_pred = logistic_reg.predict(testing_feature_vector)
matrix = confusion_matrix(label_testing_set, y_pred)
print("Confusion matrix:\n" + str(matrix))

Classification accuracy using a Logistic Regression:
0.7833333333333333

Confusion matrix using a Logistic Regression:
[[288  47]
 [ 83 182]]

# Q1-f: Weight vector of the logistic regression
import matplotlib.pyplot as plt
plt.plot(logistic_reg.coef_[0])
```



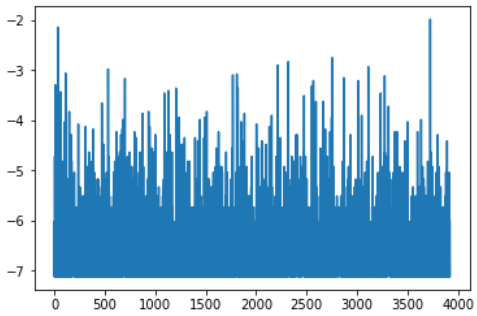
Given the graph above, we decided that the threshold values for finding positive and negative words would be +2 and -2 respectively.

```
# Q1-f: Positive/Negative words that play the most important roles in deciding the sentiment of the reviews
positive_words = []
negative_words = []

for i in range(len(logistic_reg.coef_[0])):
    if logistic_reg.coef_[0][i] >= 2:
        positive_words.append(unique_words_dict[i])
print('Positive words using a Logistic Regression:\n' + str(positive_words) + '\n')

for i in range(len(logistic_reg.coef_[0])):
    if logistic_reg.coef_[0][i] <= -2:
        negative_words.append(unique_words_dict[i])
print('Negative words using a Logistic Regression:\n' + str(negative_words))

Positive words using a Logistic Regression:
['good', 'excellent', 'best', 'nice', 'amaze', 'love', 'great']

Negative words using a Logistic Regression:
['terrible', 'worst', 'poor', 'bad']
```

Classification accuracy for a naive bayes: 0.8
Confusion matrix for a naive bayes: [[281 54] [66 199]]
Words that play the most important roles for a naive bayes: ['good', 'one', 'work', 'phone', 'film', 'love', 'great']

```
# Q1-f: Repeat this with a Naive Bayes classifier and compare performance
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
bnb.fit(training_feature_vector, label_training_set)
classification_accuracy = bnb.score(testing_feature_vector, label_testing_set)
print("Classification accuracy using a Naive Bayes:\n" + str(classification_accuracy) + '\n')

y_pred = bnb.predict(testing_feature_vector)

matrix = confusion_matrix(label_testing_set, y_pred)
print("Confusion matrix:\n" + str(matrix))

Classification accuracy using a Naive Bayes:
0.8

Confusion matrix:
[[281  54]
 [ 66 199]]

# Q1-f: Weight vector of a Naive Bayes classifier
import matplotlib.pyplot as plt
```

```
23. plt.plot(bnb.coef_[0])
```



Given the graph above, we decided that the threshold values for finding the words would be -3.

```
1.  # Q1-f: Words that play the most important roles in a Naive Bayes classifier
2.  words = []
3.
4.  for i in range(len(bnb.coef_[0])):
5.      if bnb.coef_[0][i] >= -3:
6.          words.append(unique_words_dict[i])
7.  print('Words using a Naive Bayes:\n' + str(words))
8.
9.  Words using a Naive Bayes:
10. ['good', 'one', 'work', 'phone', 'film', 'love', 'great']
```

Q1(g). N-gram model.
Try n = 2, repeat (d)-(g) and report your results.

N-gram classification accuracy using a Logistic Regression: 0.56
N-gram classification accuracy using a Naive Bayes: 0.56

```
1.  # Q1-g: N-gram Model (N=2)
2.  def generate_unique_n_gram_dict(data):
3.      n_gram_dict = []
4.
5.      for x in data:
6.          word_list = x.split(" ")
7.          for i in range(len(word_list)-1):
8.              if word_list[i+1] != None:
9.                  n_gram_dict.append(word_list[i] + " " + word_list[i+1])
10.
11.     return list(set(n_gram_dict))
12.
13. # print(generate_unique_n_gram_dict(['Alice fell down rabbit hole','Hello world']))
14. unique_n_gram_dict = generate_unique_n_gram_dict(data_training_set)
15.
16. # Q1-g: Generate feature vector for N-gram Model
17. def generate_n_gram_feature_vector(data):
18.     n_gram_feature_vector = []
19.
20.     for x in data:
21.         word_list = x.split(" ")
22.
23.         n_gram_word_list=[]
24.         for i in range(len(word_list) - 1):
25.             if word_list[i+1] != None:
26.                 n_gram_word_list.append(word_list[i] + " " + word_list[i+1])
27.
28.         feature_vector=[]
29.         for j in unique_n_gram_dict:
30.             feature_vector.append(n_gram_word_list.count(j))
```

```
31.
32.         n_gram_feature_vector.append(feature_vector)
33.
34.     return n_gram_feature_vector
35.
36. training_n_gram_feature_vector = generate_n_gram_feature_vector(data_training_set)
37. testing_n_gram_feature_vector = generate_n_gram_feature_vector(data_testing_set)
38.
39. # Q1-g: Pick your postprocessing strategy (normalization)
40. training_n_gram_feature_vector = l2_normalization(training_n_gram_feature_vector)
41. testing_n_gram_feature_vector = l2_normalization(testing_n_gram_feature_vector)
42.
43. # Q1-
    g: Sentiment prediction using logistic regression and report the classification accuracy and confusio
    n matrix.
44. n_gram_logistic_reg = linear_model.LogisticRegression()
45. n_gram_logistic_reg.fit(training_n_gram_feature_vector, label_training_set)
46. classification_accuracy = n_gram_logistic_reg.score(testing_n_gram_feature_vector, label_testing_set)

47. print("N-
    gram classification accuracy using a Logistic Regression:\n" + str(classification_accuracy) + '\n')
48.
49. y_pred = n_gram_logistic_reg.predict(testing_n_gram_feature_vector)
50. n_gram_matrix = confusion_matrix(label_testing_set, y_pred)
51. print("N-gram confusion matrix:\n" + str(n_gram_matrix))
52.
53. N-gram classification accuracy using a Logistic Regression:
54. 0.56
55.
56. N-gram confusion matrix:
57. [[ 95 240]
58.  [ 24 241]]


1.  # Q1-
    g: Positive/Negative words that play the most important roles in deciding the sentiment of the review
    s
2.  n_gram_positive_words = []
3.  n_gram_negative_words = []
4.
5.  for i in range(len(n_gram_logistic_reg.coef_[0])):
6.      if n_gram_logistic_reg.coef_[0][i] >= 1:
7.          n_gram_positive_words.append(unique_n_gram_dict[i])
8.  print('Positive words using N-gram and Logistic Regression:\n' + str(n_gram_positive_words) + '\n')
9.
10. for i in range(len(n_gram_logistic_reg.coef_[0])):
11.     if n_gram_logistic_reg.coef_[0][i] <= -1:
12.         n_gram_negative_words.append(unique_n_gram_dict[i])
13. print('Negative words using N-gram and Logistic Regression:\n' + str(n_gram_negative_words))
14.
15. Positive words using N-gram and Logistic Regression:
16. ['work great', 'highly recommend', 'great phone', 'great product']
17.
18. Negative words using N-gram and Logistic Regression:
19. ['dont buy', 'waste time', 'avoid cost', 'waste money']


1.  # Q1-g: Repeat this with a Naive Bayes classifier and compare performance
2.  from sklearn.naive_bayes import BernoulliNB
3.
4.  bnb = BernoulliNB()
5.  bnb.fit(training_n_gram_feature_vector, label_training_set)
6.  classification_accuracy = bnb.score(testing_n_gram_feature_vector, label_testing_set)
7.  print("N-
    gram classification accuracy using a Naive Bayes:\n" + str(classification_accuracy) + '\n')
8.
9.  y_pred = bnb.predict(testing_n_gram_feature_vector)
10. matrix = confusion_matrix(label_testing_set, y_pred)
11. print("N-gram Confusion matrix:\n" + str(matrix))
```

```

N-gram classification accuracy using a Naive Bayes:
0.56

N-gram Confusion matrix:
[[ 97 238]
 [ 26 239]]

# Q1-g: Words that play the most important roles in a Naive Bayes classifier
words = []

for i in range(len(bnb.coef_[0])):
    if bnb.coef_[0][i] >= -5:
        words.append(unique_n_gram_dict[i])
print('Words using N-gram and Naive Bayes:\n' + str(words))

Words using N-gram and Naive Bayes:
['work well', 'one best', 'work great', 'great food', 'highly recommend', 'sound quality', 'great pho
ne']
```

Q1(h). PCA for bag of words model.
Report corresponding clustering and classification results.

[PCA for bag of words model, d=10]
Classification accuracy using a Logistic Regression: 0.5266666666666666
Confusion matrix using a Logistic Regression: [[264 71] [213 52]]
Classification accuracy using a Naive Bayes: 0.5466666666666666
Confusion matrix using a Naive Bayes: [[252 83] [189 76]]

[PCA for bag of words model, d=50]
Classification accuracy using a Logistic Regression: 0.55
Confusion matrix using a Logistic Regression: [[235 100] [170 95]]
Classification accuracy using a Naive Bayes: 0.4816666666666667
Confusion matrix using a Naive Bayes: [[221 114] [197 68]]

[PCA for bag of words model, d=100]
Classification accuracy using a Logistic Regression: 0.5483333333333333
Confusion matrix using a Logistic Regression: [[228 107] [164 101]]
Classification accuracy using a Naive Bayes: 0.5383333333333333
Confusion matrix using a Naive Bayes: [[252 83] [194 71]]

```
# Q1-h: PCA for bag of words model
import numpy as np

def pca(feature_vector, dimension):
    mean = np.mean(feature_vector, axis=0)
    new_feature_vector = feature_vector - mean
    U, s, Vh = np.linalg.svd(new_feature_vector)
    eigen_value, eigen_vector = s, Vh
    new_eigen_vector = eigen_vector[:dimension]
    new_feature_vector = np.asmatrix(feature_vector).dot(new_eigen_vector.T)

    return new_feature_vector

def generate_pca_logreg_NB(training_feature_vector, testing_feature_vector, dimension):
    new_training_feature_vector = pca(training_feature_vector, dimension)
    new_testing_feature_vector = pca(testing_feature_vector, dimension)

    pca_logistic_reg = linear_model.LogisticRegression()
```

```
    pca_logistic_reg.fit(new_training_feature_vector, label_training_set)
    classification_accuracy = pca_logistic_reg.score(new_testing_feature_vector, label_testing_set)
    print("Classification accuracy using a Logistic Regression:\n" + str(classification_accuracy))

    y_pred = pca_logistic_reg.predict(new_testing_feature_vector)
    matrix = confusion_matrix(label_testing_set, y_pred)
    print("Confusion matrix using a Logistic Regression:\n" + str(matrix))

    bnb = BernoulliNB()
    bnb.fit(new_training_feature_vector, label_training_set)
    classification_accuracy = bnb.score(new_testing_feature_vector, label_testing_set)
    print("Classification accuracy using a Naive Bayes:\n" + str(classification_accuracy))

    y_pred = bnb.predict(new_testing_feature_vector)
    matrix = confusion_matrix(label_testing_set, y_pred)
    print("Confusion matrix using a Naive Bayes:\n" + str(matrix) + '\n')

print("[PCA for bag of words model, d=10]")
generate_pca_logreg_NB(training_feature_vector, testing_feature_vector, 10)
print("[PCA for bag of words model, d=50]")
generate_pca_logreg_NB(training_feature_vector, testing_feature_vector, 50)
print("[PCA for bag of words model, d=100]")
generate_pca_logreg_NB(training_feature_vector, testing_feature_vector, 100)
```

Q1(i). Algorithms comparison and analysis.
According to the above results, compare the performances of bag of words, 2-gram and PCA for bag of words. Which method performs best in the prediction task and why? What do you learn about the language that people use in online reviews (e.g., expressions that will make the posts positive/negative)?

- bag of words:
  - Classification accuracy for a Logistic Regression: 0.78
  - Classification accuracy for a Naive Bayes: 0.8
- 2-gram
  - Classification accuracy using a Logistic Regression: 0.56
  - Classification accuracy using a Naive Bayes: 0.56
- PCA for bag of words, d=10
  - Classification accuracy using a Logistic Regression: 0.5266666666666666
  - Classification accuracy using a Naive Bayes: 0.5466666666666666

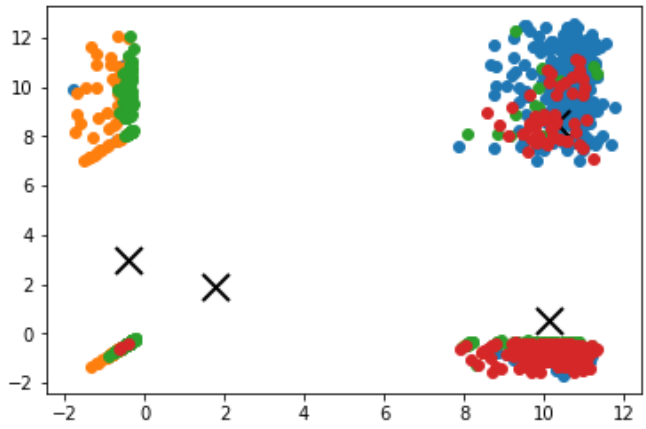
Bag of words performs best because every word in the dictionary is weighted. On the contrary, the probability of repeating two words is small in 2-gram and reducing dimensions could result in reducing some crucial information in PCA for bag of words. Thus, these two methods have less accuracy than the bag of words.

Results from the logistic regression shows that the words people used affected the sentiment of a sentence. For example, ‘good', 'excellent', 'best', 'nice', 'amaze', 'love', 'great' make a sentence positive. On the other hand, 'terrible', 'worst', 'poor', 'bad' make a sentence negative.

Q2. Clustering for text analysis.

Q2(a). Cluster the documents using k-means and various values of k (go up to at least k = 20). Select a value of k. For that value, report the top 10 words of each cluster in order of the largest positive distance from the average value across all data. Report the top ten documents that fall closest to each cluster center.

We kept plotting the clusters and centroid until the members of the clusters no longer change. Based on the information and the number of the clusters in the graph below, we set k to 4.



This algorithm has captured the related groups according to the document topic. For example, first cluster has 6 titles related to biology. Second cluster has 5 titles related to science. Third cluster has 5 titles related to earth science and general science. Fourth cluster has 5 titles related to “Corrections and Clarifications”. The algorithm can be useful to readers in terms of search. For instance, articles could be tagged based on the algorithm to improve search speed.

[Top ten documents that fall closest to each cluster center.]
Top 10 of cluster #1:
"Requirement of NAD and SIR2 for Life-Span Extension by Calorie Restriction in Saccharomyces Cerevisiae"
"Suppression of Mutations in Mitochondrial DNA by tRNAs Imported from the Cytoplasm"
"Thermal, Catalytic, Regiospecific Functionalization of Alkanes"
"Algorithmic Gladiators Vie for Digital Glory"
"Similar Requirements of a Plant Symbiont and a Mammalian Pathogen for Prolonged Intracellular Survival"
"Mothers Setting Boundaries"
"Reopening the Darkest Chapter in German Science"
"Distinct Classes of Yeast Promoters Revealed by Differential TAF Recruitment"
"Turning up the Heat on Histoplasma capsulatum"
"An Arresting Start for MAPK"

Top 10 of cluster #2:
"Information Technology Takes a Different Tack"
"Algorithmic Gladiators Vie for Digital Glory"
"Reopening the Darkest Chapter in German Science"
"Archaeology in the Holy Land"
"Heretical Idea Faces Its Sternest Test"
"National Academy of Sciences Elects New Members"
"Baedeker's Guide, or Just Plain 'Trouble'?"
"Divining Diet and Disease from DNA"

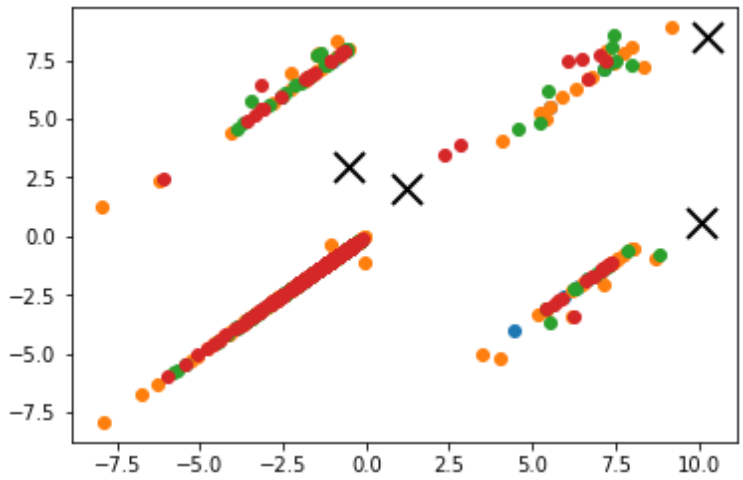
"Science Survives in Breakthrough States"
"Vaccine Studies Stymied by Shortage of Animals"

Top 10 of cluster #3:
"The Formation of Chondrules at High Gas Pressures in the Solar Nebula"
"Algorithmic Gladiators Vie for Digital Glory"
"Reopening the Darkest Chapter in German Science"
"Heretical Idea Faces Its Sternest Test"
"Thermal, Catalytic, Regiospecific Functionalization of Alkanes"
"Corrections and Clarifications: A Short Fe-Fe Distance in Peroxodiferric Ferritin: Control of Fe Substrate versus Cofactor Decay?"
"Corrections and Clarifications: Charon's First Detailed Spectra Hold Many Surprises"
"Corrections and Clarifications: Unearthing Monuments of the Yarmukians"
"Population Dynamical Consequences of Climate Change for a Small Temperate Songbird"
"Nitric Acid Trihydrate (NAT) in Polar Stratospheric Clouds"

Top 10 of cluster #4:
"Algorithmic Gladiators Vie for Digital Glory"
"Reopening the Darkest Chapter in German Science"
"National Academy of Sciences Elects New Members"
"Corrections and Clarifications: A Short Fe-Fe Distance in Peroxodiferric Ferritin: Control of Fe Substrate versus Cofactor Decay?"
"Corrections and Clarifications: Charon's First Detailed Spectra Hold Many Surprises"
"Corrections and Clarifications: Unearthing Monuments of the Yarmukians"
"Heretical Idea Faces Its Sternest Test"
"Archaeology in the Holy Land"
"Corrections and Clarifications: One Hundred Years of Quantum Physics"
"Corrections and Clarifications: Biotech Research Proves a Draw in Canada"

Q2(b). Repeat the analysis above, but cluster terms instead of documents. The terms are listed in science2k-vocab.txt Comment on these results. How might such an algorithm be useful? What is different about clustering terms from clustering documents?

We kept plotting the clusters and centroid until the members of the clusters no longer change. Based on the information and the number of the clusters in the graph below, we set k to 4.



This algorithm has captured the related groups according to the frequency of the terms in the documents. For this reason, it is difficult to identify a topic on each cluster. Thus, it is more useful to use the document cluster to identify a topic on each cluster than the term cluster.

Top 10 of cluster #1:
release
expressing
volume
fig
width
cells
obtained
right
mediated
land

Top 10 of cluster #2:
expressing
volume
mediated
states
described
receptor
right
release
fig
land

Top 10 of cluster #3:
hand
exhibit
cross
expressing
class
volume
able
clock
transgenic
detect

Top 10 of cluster #4:
tail
expressing
volume
right
cross
described
receptor

states
second
trend

Q3. EM algorithm and implementation.

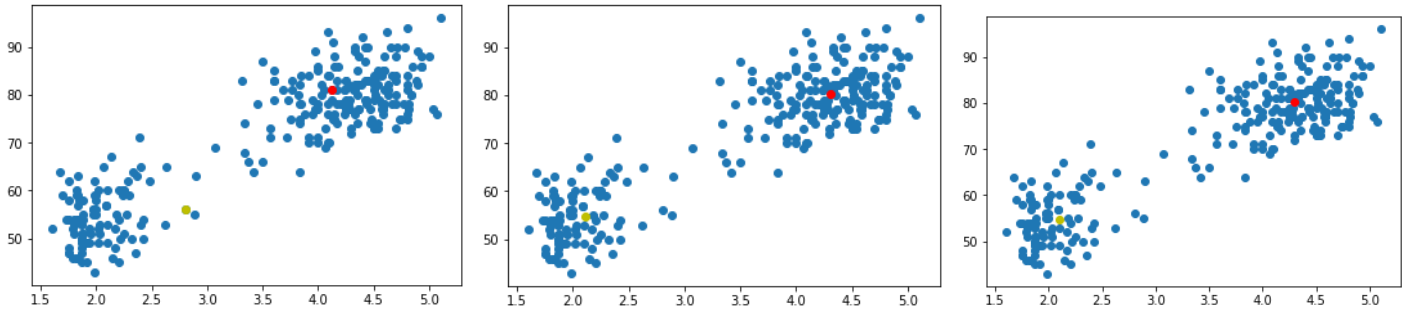
Q3(a): Show that the alternating algorithmfor k-means (in Lec. 11) is a special case of the EMalgorithmand show the corresponding objective functions for E-step and M-step.

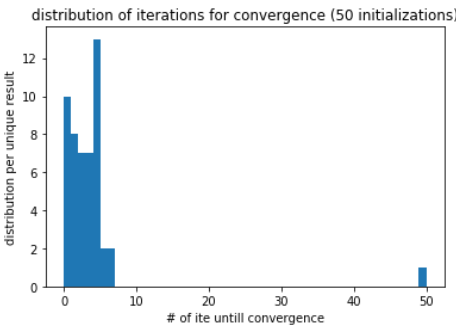
kmeans is a subset of gmm algorithms where we disregard variance and only focus on the mean part of the algorithm kmeans tackle the absence of variance in it by assigning the closest point to each mean to each cluster.It is as if we did EM with only the mean part of the gmm implementation both in the e-step as in the m-step.

Q3(c): Implement a bimodal GMMmodel to fit all data points using EMalgorithm. Explain the reasoning behind your termination criteria. For this problem, we assume the covariance matrix is spherical (i.e., it has the form of ¾2I for scalar ¾) and you can randomly initialize Gaussian parameters. For evaluation purposes, please submit the following figures:

- **Plot the trajectories of two mean vectors in 2 dimensions (i.e., coordinates vs. iteration).**
- **Run your program for 50 times with different initial parameter guesses. Show the distribution of the total number of iterations needed for algorithm to converge.**

Termination criteria for gmm algorithm: run for a maximum of 50 iteration. If both means change by a distance less than 0.01, assume convergence and stop.





Q3(d): Repeat the task in (c) but with the initial guesses of the parameters generated from the following process:

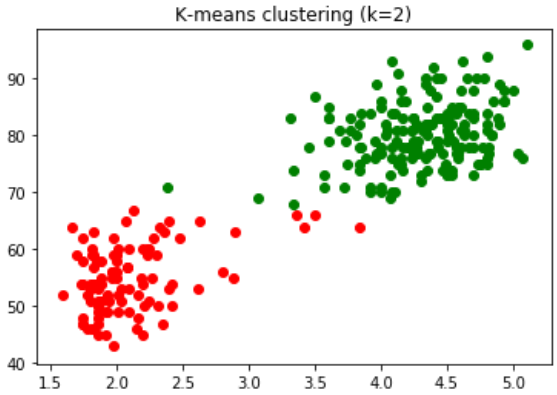
- **Run a k-means algorithm over all the data points with K = 2 and label each point with one of the two clusters.**
- **Estimate the first guess of the mean and covariance matrices using maximum likelihood over the labeled data points.**

Compare the algorithm performances of (c) and (d).

Calculate mean and covariance of cluster calculated with kmeans as initialization for gmm after kmeans, Calculate mean and cov for each cluster and use them in another gmm run.

In their very essence, both gmm and kmeans are very similar to each other. kmeans would be an instance of gmm where the gaussians would approximate from delta functions.

All in all, when it comes to mean estimation, both kmeans and gmm have converged to very similar final results. kmeans is overall faster than gmm since it essentially skips covariance calculations
this is why kmeans is usually run before gmm to estimate its best initialization values.The gmm estimated with kmean initialization has a faster convergence than the random initialized gmm because the kmeans algorithm provides a reasonable guess for initial values. This is true for both mean and covariance matrix.



Q4. Multidimensional scaling for genetic population differences.

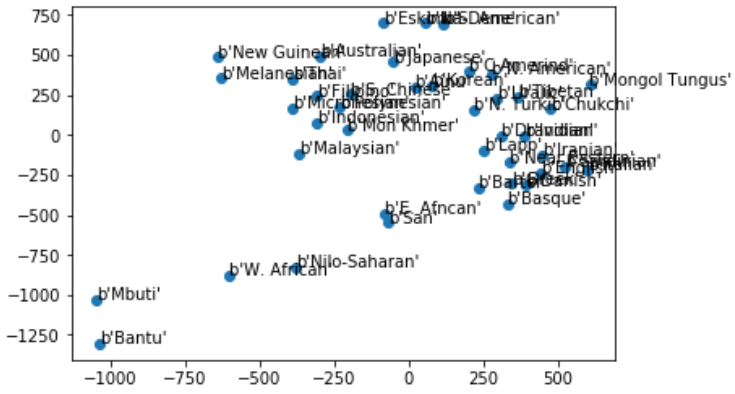
Q4(a)(i): What assumptions are being made? Under what circumstances could this fail? How could we measure how much information is being lost? Please explain.

2D MDS on the distance matrix D implies we can fit all distances perfectly in a 2D plane. It also implies there is no way of knowing the data's original rotation, mirror and translation.

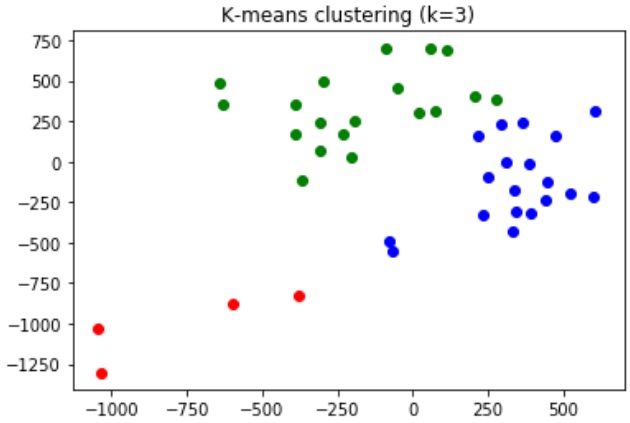
Q4(a)(ii): How many dimensions are necessary to capture most of the variation in the data? Briefly explain your method and justify why it makes sense.

A good quality measure for each number of dimensions being considered in an MDS can be inferred from the principal components of the original distance matrix D. Choose a number of dimensions for MDS that essentially allows it to capture the most amount of variance from the original matrix D. In our implementation, the number of dimensions was probably 5 or 6.

Q4(a)(iii): UseMDS to embed the distancematrix into only two dimensions and show the resulting scatterplot. Label each point with the name of its population.



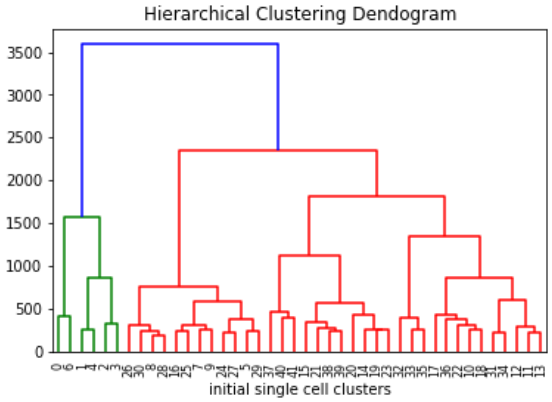
Q4(b): k-means on 2D embedding. Select an appropriate k and run k-means on the scatterplot. Show the resulting clusters. Since we are working with only two dimensions, this clustering is likely to lose a lot of high dimensional structure. Do you agree with the resulting clustering? What information seems to be lost?

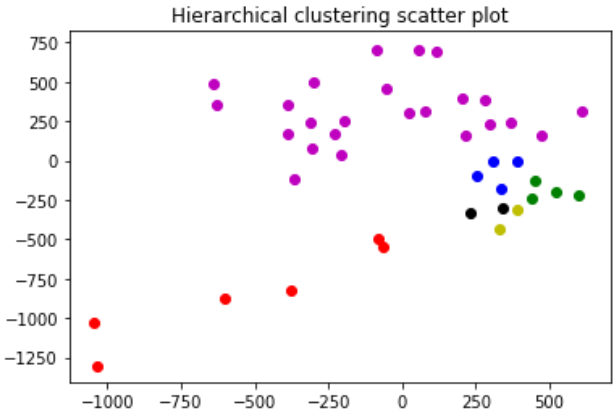


The resulting clusters from kmeans seem to make sense for the 2D plane approximation of this dataset. Since we know that the optimal number of dimensions for this data is around 5 to 6 dimensions, there are about 4 dimensions of information that could contribute to the clustering in kmeans which are being ignored currently. However, kmeans does a decent job given the approximation on its input.

Q4(c): Comparing hierarchical clustering with K-Means. Show the resulting tree as a dendrogram, labeling the x axis with the categorical population names and the y axis with the Nei's distance between clusters. Select a distance cutoff that roughly corresponds with your chosen k earlier, balancing the number of points in each cluster with the number of resulting clusters. Visualize the resulting clustering by coloring the corresponding points on your 2D MDS embedding. How does this clustering compare with the k-means clustering you computed earlier?

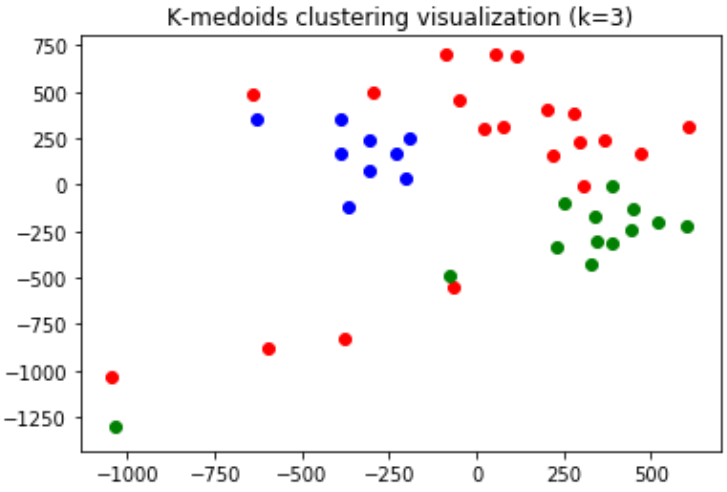
Hierarchical clustering algorithm has restricted ourselves with k=6 on its clustering process. This is understandable since the number of dimensions that best represent the data is around 6 and different from kmeans, hierarchical clustering takes this into consideration.





Q4(d): Compare k-medoids with k-means. Repeat the above experiment, but applying k-medoid clustering on the original distance matrix. Show the resulting clusters on a 2D scatterplot. Are there any significant differences between the clustering chosen by k-medoids compared to k-means?

While kmeans clearly cluster clumps of data from vectorized samples, kmedoids is a way to cluster data together in a non vectorial format. Some differences where noticed on the clustering structure of both approaches we attribute this to the capacity of the kmedoids algorithm to cluster data taking its full dimensionality into consideration whereas kmeans is restricted to the vectorized representation given by MDS.



References:

- https://scikit-learn.org/stable/
- https://www.reddit.com/r/learnpython/comments/6zqe70/trying_to_remove_punctuation_from_a_list_of
- https://stackoverflow.com/questions/4371231/removing-punctuation-from-python-list-items
- https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
- http://www.nltk.org/howto/stem.html
- https://stackoverflow.com/questions/32276391/feature-normalization-advantage-of-l2-normalization
- https://stackoverflow.com/questions/47243202/how-to-get-the-weight-vector-in-logistic-regression-using-sklearn
- https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.svd.html
- https://wikidocs.net/4691
- https://stackabuse.com/k-means-clustering-with-scikit-learn/
- http://fromdatawithlove.thegovans.us/2013/05/clustering-using-scikit-learn.html
- https://stackoverflow.com/questions/26795535/output-50-samples-closest-to-each-cluster-center-using-scikit-learn-k-means-libr
- https://mubaris.com/posts/kmeans-clustering/
- https://www.quora.com/Why-is-L1-regularization-supposed-to-lead-to-sparsity-than-L2
- https://stats.stackexchange.com/questions/45643/why-l1-norm-for-sparse-models
- http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/

- https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html
- https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.multivariate_normal.html
- https://www.youtube.com/watch?v=qMTuMa86NzU
- https://docs.python.org/3/library/random.html

- https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html#module-scipy.cluster.hierarchy
- Bauckhage C. Numpy/scipy Recipes for Data Science: k-Medoids Clustering[R]. Technical Report, University of Bonn, 2015.
- https://github.com/letiantian/kmedoids
- https://stackoverflow.com/questions/20484195/typeerror-range-object-does-not-support-item-assignment
- https://github.com/letiantian/kmedoids/issues/4
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html#scipy.cluster.hierarchy.linkage