CS5785 Homework 3: Due by 12:59 PM EST Thursday November 14, 2019

Castillo, Eduardo (ec833)
ec833@cornell.edu

Jayaraman, Shobhna (sj747)
sj747@cornell.edu

1 Executive Summary

1.1 Key Parameters

This assignment builds upon fundamental concepts such as Multidimensional Analysis and Hierarchical Clustering and expands the foregoing discussion to relevant topics in neighboring domains such as Linear Discriminant Analysis (LDA) and Singular Value Decomposition (SVD).

A set of two written exercises along with four programming exercises were leveraged to build a balanced connection between the theory behind these analysis methods and real¬–word scenarios where their value and usefulness are realized.

1.2 Programming Exercise 1

1. The following code shows the implementation of Programming Exercise 1.

Question 1 Part A

In [424]:
```
import pandas as pd
```

In [425]:
```
yelp = []
for line in open('sentiment labelled sentences/yelp_labelled.txt'):
    line = line.strip('\n')
    label = line[-1]
    review = line[:-2]
    yelp.append([review, int(label)])
yelp = pd.DataFrame(yelp, columns = ['Review', 'Sent'])
```

In [426]:
```
amazon = []
for line in open('sentiment labelled sentences/amazon_cells_labelled.txt'
    line = line.strip('\n')
    review = line[:-2]
    label = line[-1]
    amazon.append([review, int(label)])
amazon = pd.DataFrame(amazon, columns = ['Review', 'Sent'])
```

In [427]:
```
imdb = []
for line in open('sentiment labelled sentences/imdb_labelled.txt'):
    line = line.strip('\n')
    review = line[:-2]
    label = line[-1]
    imdb.append([review, int(label)])
imdb = pd.DataFrame(imdb, columns = ['Review', 'Sent'])
```

In [428]:
```
#review ratios
r_yelp = (yelp.Sent == 1).sum()/ (len(yelp) - (yelp.Sent == 1).sum())
print('ratio for yelp are ', r_yelp)

r_amazon= (amazon.Sent == 1).sum()/ (len(amazon) - (amazon.Sent == 1).sum
print('ratio for amazon are ', r_amazon)

r_imdb = (imdb.Sent == 1).sum()/ (len(imdb) - (imdb.Sent == 1).sum())
print('ratio for imdb are ', r_imdb)
```

```
ratio for yelp are  1.0
ratio for amazon are  1.0
ratio for imdb are  1.0
```

The labels are all balanced.

PART B

In [429]:
```
import string
import re
from nltk import *
from nltk.stem.snowball import SnowballStemmer as SBS
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import WordNetLemmatizer
```

In [430]:
```
#here we do lemmetization
def lemmetize(s):
    ps = SBS('english')
    ls_stem = [ps.stem(word) for word in s.split()]
    return ' '.join(ls_stem)
```

In [431]:
```
def rmv(s, ls):
    return ' '.join(filter(lambda s: s not in ls, s.split()))
```

In [432]:
```
# pre procssing data
def preprocess(df):
    data = df

    # remove punctuation
    punc = '[' + string.punctuation + ']'
    data.Review = data.Review.apply(lambda x: re.sub(punc, '', x))

    # lowercase
    data.Review = data.Review.apply(lambda x: x.lower())

    # remove stopwords
    sw = stopwords.words('english')
    data.Review = data.Review.apply(lambda x: rmv(x, sw))

    # Lemmatization
    data.Review = data.Review.apply(lambda x: lemmetize(x))
    return data
yelp = preprocess(yelp)
amazon = preprocess(amazon)
imdb = preprocess(imdb)

```

In [433]:
```
yelp.head()
```

Out[433]:

	Review	Sent
0	wow love place	1
1	crust good	0
2	tasti textur nasti	0
3	stop late may bank holiday rick steve recommen...	1
4	select menu great price	1

Reason:

Lowercase: Uppercase words and lowercase words are equivalent in a sentence. Lemmatization: The words such as sit, sitting and sat have similar meanings. It is easier to train a model if all words are lemmatized. Punctuations are removed because they are not that essential while training. I, we, them are stop words and they are redundant in sentences.

PART C

In [434]:
```

def customsplit(df):
    df_1 = df[df.Sent == 1]
    df_0 = df[df.Sent == 0]
    train1 = df_1[:400]
    test1 = df_1[400:]
    train0 = df_0[:400]
    test0 = df_0[400:]
    return train1, test1, train0, test0
```

In [435]:
```
a_train1, a_test1, a_train0, a_test0 = customsplit(amazon)
y_train1, y_test1, y_train0, y_test0 = customsplit(yelp)
i_train1, i_test1, i_train0, i_test0 = customsplit(imdb)
```

In [436]:
```
trains = [a_train1, a_train0, y_train1, y_train0, i_train1, i_train0]
tests = [a_test1, a_test0, y_test1, y_test0, i_test1, i_test1]
train = pd.concat(trains)
test = pd.concat(tests)

train.reset_index(inplace = True)
train.drop('index',axis = 1, inplace = True)
test.reset_index(inplace = True)
test.drop('index',axis = 1, inplace = True)
```

In [437]:
```
train.shape, test.shape
```

Out[437]:
```
((2400, 2), (600, 2))
```

PART D

In [438]:
```
my_dict = np.array(list(set(' '.join(train.Review).split())))
print('There are ', len(my_dict), ' unique words training set')
```

```
There are  3629  unique words training set
```

We can't use test data at this point for creating dictionary because we need to use the training data to train the model and training data may not have all the words that test data has.

In [439]:
```
def bag(s):
    a = s.split()
    v = np.sum([dic == w for w in a],axis=0)
    return v
```

In [440]:
```
train['vector'] = train.Review.apply(bag)
test['vector'] = test.Review.apply(bag)
```

In [441]:
```
# picking any two reviews to report feature vectors
for z in [20,25]:
    print(train.Review[z])
    print(train.vector[z])
```

```
excel bluetooth headset
[0 0 0 ... 0 0 0]
phone pretti sturdi ive never larg problem
[0 0 0 ... 0 0 0]
```

In [442]:
```
train['log vector'] = train.vector.apply(lambda x: np.log(x+1))
test['log vector'] = test.vector.apply(lambda x: np.log(x+1))

```

Log-Normalization to reduce the variance.

PART F

In [443]:
```
from sklearn.linear_model import LogisticRegression as lr
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
```

In [444]:
```
x_train = np.vstack(train['log vector'])
y_train = train.Sent
x_test = np.vstack(test['log vector'])
y_test = test.Sent
```

In [445]:
```
paras = {'C': [0.1, 0.5, 1, 5, 10, 50]}
clf = GridSearchCV(lr(penalty = 'l1'), paras, cv = 5, n_jobs = -1, scorir
clf.fit(x_train, y_train)
```

```
/Applications/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.
22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[445]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fal
se,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=Non
e,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l1',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=-1, param_grid={'C': [0.1, 0.5, 1, 5, 10, 5
0]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [446]:
```
lr_prediction = clf.predict(x_test)
print('Accuracy: ', clf.best_estimator_.score(x_test, y_test))
```

```
Accuracy:  0.815
```

In [447]:
```
print('Confusion Matrix:')
confusion_matrix(y_test, lr_prediction)
```

```
Confusion Matrix:
```

Out[447]:
```
array([[179,  21],
       [ 90, 310]])
```

In [448]:
```
l = lr(penalty = 'l2', C = 5)
l.fit(x_train, y_train)
ind = abs(l.coef_)
z = dic[np.argsort(ind)]
z[0][-10:]
```

```
/Applications/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.
22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[448]:
```
array(['beauti', 'fantast', 'worst', 'amaz', 'excel', 'bad', 'delici',
       'poor', 'love', 'great'], dtype='<U32')
```

In [449]:
```
z = dic[np.argsort(ind)]
z[0][-10:]
```

Out[449]:
```
array(['beauti', 'fantast', 'worst', 'amaz', 'excel', 'bad', 'delici',
       'poor', 'love', 'great'], dtype='<U32')
```

In [450]:
```
from sklearn.naive_bayes import BernoulliNB
```

In [451]:
```
BNB = BernoulliNB()
BNB.fit(x_train, y_train)
```

Out[451]:
```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

In [452]:
```
BNB_prediction = BNB.predict(x_test)
print('Accuracy: ', BNB.score(x_test, y_test))
```

```
Accuracy:  0.785
```

In [453]:
```
print('Confusion Matrix:')
confusion_matrix(y_test, BNB_prediction)
```

```
Confusion Matrix:
```

Out[453]:
```
array([[168,  32],
       [ 97, 303]])
```

In [454]:
```
def two_gram(s):
    s_s = s.split()
    return [s_s[i] + ' ' + s_s[i+1] for i in range(len(s_s)-1)]
```

In [455]:
```
train['two_gram'] = train.Review.apply(two_gram)
test['two_gram'] = test.Review.apply(two_gram)
```

In [456]:
```
import itertools
dic_w = np.array(list(set(itertools.chain(*train.two_gram))))
```

In [457]:
```
# two-gram feature
def two_g(s):
    if s == []:
        return np.array([0]*len(dic_w))
    else:
        v = np.sum([dic_w == w for w in s],axis=0)
        return v
```

In [458]:
```
train['vector_2g'] = train.two_gram.apply(two_g)
test['vector_2g'] = test.two_gram.apply(two_g)
```

In [459]:
```
train['vector_2g_log'] = train.vector_2g.apply(lambda x: np.log(x+1))
test['vector_2g_log'] = test.vector_2g.apply(lambda x: np.log(x+1))
```

In [460]:
```
x_train_l = np.vstack(train['vector_2g_log'])
y_train_l = train.Sent
x_test_l = np.vstack(test['vector_2g_log'])
y_test_l = test.Sent
```

In [461]:
```

paras = {'C': [0.1, 0.5, 1, 2, 5, 10, 50]}
clf = GridSearchCV(lr(penalty='l2'), paras, cv = 5, n_jobs = -1)
clf.fit(x_train_l, y_train_l)
```

```
/Applications/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.
22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[461]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=Fal
se,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=Non
e,
                                          max_iter=100, multi_class='warn',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='warn',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'C': [0.1, 0.5, 1, 2, 5, 10, 50]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [462]:
```

lr_prediction = clf.predict(x_test_l)
print('Accuracy: ', clf.best_estimator_.score(x_test_l, y_test_l))
```

```
Accuracy:  0.55
```

In [463]:
```
print('Confusion Matrix:')
confusion_matrix(y_test_l, lr_prediction)
```

```
Confusion Matrix:
```

Out[463]:
```
array([[176,  24],
       [246, 154]])
```

In [464]:
```
BNB = BernoulliNB()
BNB.fit(x_train_l, y_train_l)

BNB_prediction = BNB.predict(x_test_l)
print('Accuracy: ', BNB.score(x_test_l, y_test_l))

print('Confusion Matrix:')
confusion_matrix(y_test_l, BNB_prediction)
```

```
Accuracy:  0.5483333333333333
Confusion Matrix:
```

Out[464]:
```
array([[179,  21],
       [250, 150]])
```

In [465]:
```
# feature matrix
fm = np.vstack(train['log vector'])
print(fm.shape)
```

```
(2400, 3629)
```

In [466]:
```
def sub_m(df):
    mean_vector = np.mean(df,axis=1).reshape(-1,1)

# subtract mean
    return df - mean_vector
fm_sub_m = sub_m(fm)
```

In [467]:
```
def get_cm(c):
    m = c.T

    num_f, num_o = m.shape

#finding the mean vector
    mean_vector = np.mean(m,axis=1).reshape(-1,1)
#subtract mean : vector
    fm_sub_m = m - mean_vector

    # find covariance
    covariance_m = fm_sub_m.dot(fm_sub_m.T)/(num_o - 1)

    return covariance_m

cm = get_cm(fm)
```

In [468]:
```
# diagnonalization
def diagonalize(m):

    evalues, evectors = np.linalg.eigh(m)
    idx = evalues.argsort()[::-1]
    evalues = np.diag(evalues[idx])
    evectors = evectors[:,idx]

    return evalues, evectors
```

In [469]:
```
E, P = diagonalize(cm)
```

In [470]:
```
P.shape
```

Out[470]: (3629, 3629)

In [471]:
```
#principal component analysis

# dimensionality reduction
x_train10 = fm_sub_m.dot(P[:,:10])
x_train50 = fm_sub_m.dot(P[:,:50])
x_train100 = fm_sub_m.dot(P[:,:100])
```

In [472]:
```
#  dimensionality reduction for test data
fm_test = np.vstack(test['log vector'])
test_mean = np.mean(fm_test,axis=1).reshape(-1,1)

# subtract mean
fm_test_sub_m = fm_test - test_mean

x_test10 = fm_test_sub_m.dot(P[:,:10])
x_test50 = fm_test_sub_m.dot(P[:,:50])
x_test100 = fm_test_sub_m.dot(P[:,:100])
```

In [473]:

```
for i in [10, 50,100]:

    exec('x_train_pc = x_train' + str(i))
    exec('x_test_pc = x_test' + str(i))

    paras = {'C': [0.1, 0.5, 1, 5, 10, 50]
             }
    clf = GridSearchCV(lr(penalty = 'l2'), paras, cv = 5, n_jobs = -1, sc
    clf.fit(x_train_pc, y_train)

    lr_prediction = clf.predict(x_test_pc)

    print('Performance with PC ' + str(i) +':')
    print('Accuracy: ', clf.best_estimator_.score(x_test_pc, y_test_l))
    print('Confusion Matrix: \n', confusion_matrix(y_test_l, lr_predictic
    print('\n')
```

```
/Applications/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.
22. Specify a solver to silence this warning.
  FutureWarning)

Performance with PC 10:
Accuracy:  0.5483333333333333
Confusion Matrix:
 [[174  26]
 [245 155]]



/Applications/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.
22. Specify a solver to silence this warning.
  FutureWarning)

Performance with PC 50:
Accuracy:  0.675
Confusion Matrix:
 [[170  30]
 [165 235]]


Performance with PC 100:
Accuracy:  0.6966666666666667
Confusion Matrix:
 [[158  42]
 [140 260]]



/Applications/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.
22. Specify a solver to silence this warning.
  FutureWarning)
```

In [474]:

```
for i in [10, 50, 100]:

    exec('x_train_pc = x_train' + str(i))
    exec('x_test_pc = x_test' + str(i))

    BNB = BernoulliNB()
    BNB.fit(x_train_pc, y_train)
    BNB_prediction = BNB.predict(x_test_pc)

    print('Performance with PC ' + str(i) +':')
    print('Accuracy: ', BNB.score(x_test_pc, y_test))
    print('Confusion Matrix: \n', confusion_matrix(y_test, BNB_predictior
    print('\n')
```

```
Performance with PC 10:
Accuracy:  0.585
Confusion Matrix:
 [[120  80]
 [169 231]]


Performance with PC 50:
Accuracy:  0.6083333333333333
Confusion Matrix:
 [[129  71]
 [164 236]]


Performance with PC 100:
Accuracy:  0.64
Confusion Matrix:
 [[129  71]
 [145 255]]
```

In [475]:

```
# feature matrix
fm2 = np.vstack(train.vector_2g_log)

fm2 = sub_m(fm2)

# covariance matrix
def get_cm(c):
    m = c.T

    num_f, num_o = m.shape

    # find mean
    mean_vector = np.mean(m,axis=1).reshape(-1,1)

    # subtract mean
    fm_sub_m = m - mean_vector

    # find covariance
    covariance_m = fm_sub_m.dot(fm_sub_m.T)/(num_o - 1)

    return covariance_m

cm2 = get_cm(fm2)
```

In [476]:

```
# diagnonalization
def diagonalize(m):

    evalues, evectors = np.linalg.eigh(m)
    idx = evalues.argsort()[::-1]
    evalues = np.diag(evalues[idx])
    evectors = evectors[:,idx]

    return evalues, evectors

E2, P2 = diagonalize(cm2)


#  dimensionality reduction
x_2_train10 = fm2.dot(P2[:,:10])
x_2_train50 = fm2.dot(P2[:,:50])
x_2_train100 = fm2.dot(P2[:,:100])


#  dimensionality reduction for test data
fm2_test = np.vstack(test.vector_2g_log)
fm2_test = sub_m(fm2_test)

x_2_test10 = fm2_test.dot(P2[:,:10])
x_2_test50 = fm2_test.dot(P2[:,:50])
x_2_test100 = fm2_test.dot(P2[:,:100])
```

```
for i in [10, 50,100]:

    exec('x_2_train_pc = x_2_train' + str(i))
    exec('x_2_test_pc = x_2_test' + str(i))

```

```
    paras = {'C': [0.1, 0.5, 1, 5, 10, 50]}
    clf = GridSearchCV(lr(penalty = 'l2'), paras, cv = 5, n_jobs = -1, scoring = 'accuracy' )
    clf.fit(x_2_train_pc, y_train)

    lr_prediction = clf.predict(x_2_test_pc)

    print('Performance with PC ' + str(i) +':')
    print('Accuracy: ', clf.best_estimator_.score(x_2_test_pc, y_test_l))
    print('Confusion Matrix: \n', confusion_matrix(y_test, lr_prediction))

```

In [477]:

```
for i in [10, 50, 100]:

    exec('x_2_train_pc = x_2_train' + str(i))
    exec('x_2_test_pc = x_2_test' + str(i))

    BNB = BernoulliNB()
    BNB.fit(x_2_train_pc, y_train)
    BNB_prediction = BNB.predict(x_2_test_pc)

    print('Performance with PC ' + str(i) +':')
    print('Accuracy: ', BNB.score(x_2_test_pc, y_test))
    print('Confusion Matrix: \n', confusion_matrix(y_test, BNB_predictior

```

```
Performance with PC 10:
Accuracy:  0.37166666666666665
Confusion Matrix:
 [[194   6]
 [371  29]]


Performance with PC 50:
Accuracy:  0.4266666666666667
Confusion Matrix:
 [[173  27]
 [317  83]]


Performance with PC 100:
Accuracy:  0.5716666666666667
Confusion Matrix:
 [[ 44 156]
 [101 299]]
```

In [478]:
```
l.fit(x_train, y_train)
ind = abs(l.coef_)
z = dic[np.argsort(ind)]
z[0][-10:]
```

```
/Applications/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.
22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[478]:
```
array(['beauti', 'fantast', 'worst', 'amaz', 'excel', 'bad', 'delici',
       'poor', 'love', 'great'], dtype='<U32')
```

In [479]:
```
l1 = lr(penalty = 'l2', C = 5)
l1.fit(x_train_l, y_train_l)
ind = abs(l1.coef_)
z1 = dic_w[np.argsort(ind)]
z1[0][-10:]
```

```
/Applications/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/log
istic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.
22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[479]:
```
array(['good price', 'realli good', 'easi use', 'food good',
       'great product', 'one best', 'wast time', 'great phone',
       'high recommend', 'work great'], dtype='<U39')
```

1.3 Programming Exercise 2

1. The following code shows the implementation of Programming Exercise 2.

In [79]:

```
import numpy as np
import pandas as pd
from scipy.spatial.distance import cdist
```

In [80]:

```
sdoc = np.load("data/science2k-doc-word.npy")
sdoc.shape
```

Out[80]:

```
(1373, 5476)
```

In [81]:

```
sdoc = pd.DataFrame(sdoc)
```

In [83]:

```
# calculate total distance
def km_error(df1):
    total_dis = 0
    X = df1.groupby('label').mean()
    Z = np.array(X)
    for i, x in enumerate(Z):
        df = df1.copy()
        d = df[df.label == i]
        dis = cdist(d.drop('label', axis = 1), x.reshape(1,-1), 'euclidean')
        total_dis += sum(dis)
    return total_dis[0]
```

In [84]:

```
from sklearn.cluster import KMeans
import numpy as np
```

In [85]:

```
dis_ls = []
for k in range(2,21):
    train = sdoc.copy()
    km = KMeans(n_clusters = k, random_state = 0).fit(train)
    train['label'] = km.labels_
    td = km_error(train)
    print('cluster', str(k), 'error', td)
    dis_ls.append(td)
```

```
cluster 2 error 211789.79089102792
cluster 3 error 209323.87071948452
cluster 4 error 208526.90691437514
cluster 5 error 208134.82206325192
cluster 6 error 207296.03018906165
cluster 7 error 206873.24896295991
cluster 8 error 206360.8173431936
cluster 9 error 206332.757901656
cluster 10 error 205704.26211145474
cluster 11 error 205594.25561705336
cluster 12 error 205583.84919467373
cluster 13 error 205433.84831757174
cluster 14 error 204494.90556770342
cluster 15 error 204947.5537942867
cluster 16 error 204241.0950741877
cluster 17 error 204478.93545369722
cluster 18 error 204151.48365093337
cluster 19 error 203708.37685529623
cluster 20 error 203500.9295086367
```

In [86]:

```
train = sdoc.copy()
km = KMeans(n_clusters = 3, random_state = 0).fit(train)
train['label'] = km.labels_
td = km_error(train)
print('cluster ', str(3), 'error', td)
```

```
cluster  3 error 209323.87071948452
```

In [ ]:

```
def top10(df):
    z = df.drop('label',axis = 1)
    m = z.mean(axis = 0)
    dis_m = cdist(z, m.reshape(1,-1), 'euclidean')
    dis = np.array([a[0] for a in dis_m])
    ind = np.argsort(dis)[::-1]

    if len(dis) >= 10:
        return z.iloc[ind[-10:]]
    else:
        return z.iloc[ind]

top_10 = train.groupby('label').apply(top10)
```

In [ ]:

```
title = []
for line in open("data/science2k-titles.txt"):
    line = line.strip('\n')
    title.append(line)
```

In [ ]:

```
title = pd.DataFrame(title)
title = title.iloc[list(top_10.index.get_level_values(1))]
title.set_index(top_10.index)
```

question 2 part B

In [89]:

```
term = np.load("data/science2k-word-doc.npy")
term.shape
```

Out[89]:

```
(5476, 1373)
```

In [90]:

```
term = pd.DataFrame(term)
```

In [91]:

```
def km_error(df1):
    total_dis = 0
    X = df1.groupby('label').mean()
    Z = np.array(X)
    for i, x in enumerate(Z):
        df = df1.copy()
        d = df[df.label == i]
        dis = cdist(d.drop('label', axis = 1), x.reshape(1,-1), 'euclidean')
        total_dis += sum(dis)
    return total_dis[0]
```

In [92]:

```
dis_term = []
for k in range(2,21):
    train1 = term.copy()
    km = KMeans(n_clusters = k, random_state = 0).fit(train1)
    train1['label'] = km.labels_
    td = km_error(train1)
    print('cluster', str(k), 'error', td)
    dis_term.append(td)
```

```
cluster 2 error 396156.01716650825
cluster 3 error 393795.80582557374
cluster 4 error 393679.8972904857
cluster 5 error 391664.62995578436
cluster 6 error 390454.5102117503
cluster 7 error 389902.52240952256
cluster 8 error 389299.1223250805
cluster 9 error 389403.53928272764
cluster 10 error 388792.15276354755
cluster 11 error 388511.03558790457
cluster 12 error 387554.5831533615
cluster 13 error 387300.65011879255
cluster 14 error 387125.67095517163
cluster 15 error 386149.5941851195
cluster 16 error 386905.93672083144
cluster 17 error 386735.38724336924
cluster 18 error 386332.7584477437
cluster 19 error 386662.7846062708
cluster 20 error 385724.6568586844
```

In [93]:

```
train1 = term.copy()
km = KMeans(n_clusters = 3, random_state = 0).fit(train1)
train1['label'] = km.labels_
td = km_error(train1)
print('cluster ', str(3), 'error ', td)
```

```
cluster  3 error  393795.80582557374
```

In [ ]:

```
s2v = []
for line in open("data/science2k-vocab.txt"):
    line = line.strip('\n')
    s2v.append(line)
```

In [ ]:

```
def top10(df):
    z = df.drop('label', axis =1 )
    m = z.mean(axis = 0)
    dis_m = cdist(z, m.reshape(1,-1), 'euclidean')
    dis = [a[0] for a in dis_m]
    ind = np.argsort(dis)[::-1]
    if len(dis) >= 10:
        return z.iloc[ind[-10:]]
    else:
        return z.iloc[ind]
top_10_term = train1.groupby('label').apply(top10)
```

In [ ]:

```
s2v = pd.DataFrame(s2v)
s2v.set_index(top_10_term.index)
```

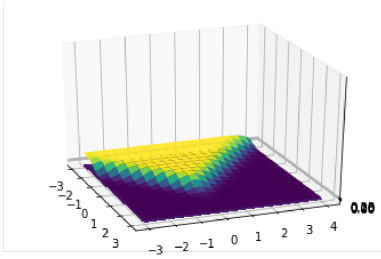
1.4 Programming Exercise 3

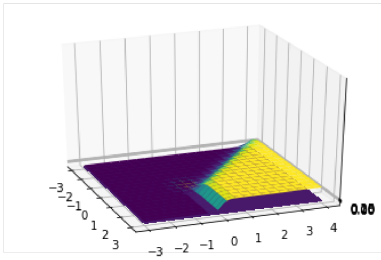
1. The following code shows the implementation of Programming Exercise 3.

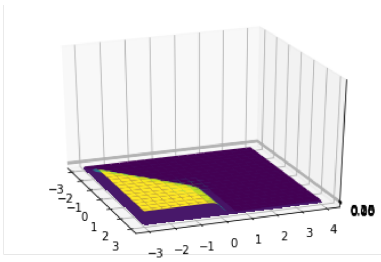
In [2]:
```
import numpy as np
np.set_printoptions(threshold=np.inf)

import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

# Our 2-dimensional distribution will be over variables X and Y
N = 60
X1 = np.linspace(-3, 3, N)
Y1 = np.linspace(-3, 4, N)

X2 = np.linspace(-3, 3, N)
Y2 = np.linspace(-3, 4, N)

X3 = np.linspace(-3, 3, N)
Y3 = np.linspace(-3, 4, N)


X1, Y1 = np.meshgrid(X1, Y1)
X2, Y2 = np.meshgrid(X2, Y2)
X3, Y3 = np.meshgrid(X3, Y3)

a = .2

# Mean vector and covariance matrix
mu1 = np.array([0.,0.])
Sigma1 = np.array([[ a, 0], [0,  a]])

mu2 = np.array([1.,1.])
Sigma2 = np.array([[ a , 0], [0,  a]])

mu3 = np.array([1.,-1.])
Sigma3 = np.array([[ a , 0], [0,  a]])


# Pack X and Y into a single 3-dimensional array
pos1 = np.empty(X1.shape + (2,))
pos1[:, :, 0] = X1
pos1[:, :, 1] = Y1

pos2 = np.empty(X2.shape + (2,))
pos2[:, :, 0] = X2
pos2[:, :, 1] = Y2

pos3 = np.empty(X3.shape + (2,))
pos3[:, :, 0] = X3
pos3[:, :, 1] = Y3


def multivariate_gaussian(pos1, mu1, Sigma1):
    """Return the multivariate Gaussian distribution on array pos.

    pos is an array constructed by packing the meshed arrays of variables
    x_1, x_2, x_3, ..., x_k into its _last_ dimension.
```

```
    """

    n1 = mu1.shape[0]
#     print(n1)
    Sigma_det1 = np.linalg.det(Sigma1)
    Sigma_inv1 = np.linalg.inv(Sigma1)
    N1 = np.sqrt((2*np.pi)**n1 * Sigma_det1)
    # This einsum call calculates (x-mu)T.Sigma-1.(x-mu) in a vectorized
    # way across all the input variables.
    fac1 = np.einsum('...k,kl,...l->...', pos1-mu1, Sigma_inv1, pos1-mu1)

    return np.exp(-fac1 / 2) / N1

# The distribution on the variables X, Y packed into pos.
Z1= multivariate_gaussian(pos1, mu1, Sigma1)
Z2= multivariate_gaussian(pos2, mu2, Sigma2)
Z3= multivariate_gaussian(pos3, mu3, Sigma3)

# print(Z1[30]);
# print(Z1.shape);
# Create a surface plot and projected filled contour plot under it.
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X1, Y1, Z1/(Z1+Z2+Z3), rstride=3, cstride=3, linewidth=1, anti
aliased=True,
                cmap=cm.viridis)

cset = ax.contourf(X1, Y1, Z1, zdir='z', offset=-0.15, cmap=cm.viridis)

# Adjust the limits, ticks and view angle
ax.set_zlim(-0.15,10)
ax.set_zticks(np.linspace(0,0.2,5))
ax.view_init(27, -21)

plt.show()

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X1, Y1, Z2/(Z1+Z2+Z3), rstride=3, cstride=3, linewidth=1, anti
aliased=True,
                cmap=cm.viridis)

cset = ax.contourf(X1, Y1, Z1, zdir='z', offset=-0.15, cmap=cm.viridis)

# Adjust the limits, ticks and view angle
ax.set_zlim(-0.15,10)
ax.set_zticks(np.linspace(0,0.2,5))
ax.view_init(27, -21)

plt.show()

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X1, Y1, Z3/(Z1+Z2+Z3), rstride=3, cstride=3, linewidth=1, anti
aliased=True,
                cmap=cm.viridis)
```

```
cset = ax.contourf(X1, Y1, Z1, zdir='z', offset=-0.15, cmap=cm.viridis)

# Adjust the limits, ticks and view angle
ax.set_zlim(-0.15,10)
ax.set_zticks(np.linspace(0,0.2,5))
ax.view_init(27, -21)

plt.show()
```



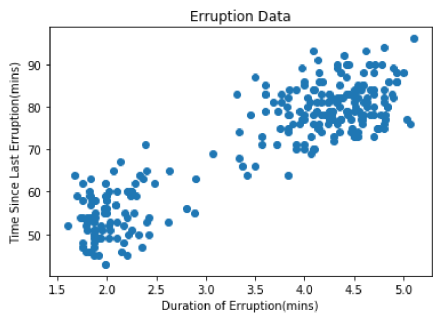




In [3]:

```
import numpy as np
import matplotlib.pyplot as plt

with open("Problem 3/P3.txt", 'r') as erruptions:
     features = np.array([line.split(' ') for line in erruptions][1:], dtype=f
loat)[:,1:]
print(features.shape)
# print(features)

plt.scatter(features[:,0],features[:,1])
plt.title("Erruption Data")
plt.xlabel("Duration of Erruption(mins)")
plt.ylabel("Time Since Last Erruption(mins)")
plt.show()
```

```
(272, 2)
```

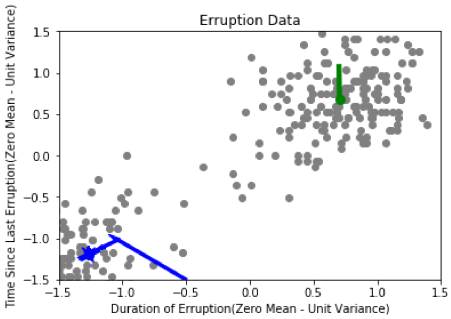


In [14]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# Calculate mean erruption duration
mu_erruption_duration=features[:,0].mean()

# Calculate mean time between erruptions
mu_time_between_erruptions=features[:,1].mean()

# Put the two numbers above on [2,1] array
mu_features = np.array([mu_erruption_duration,mu_time_between_erruptions])

# Create new feature vector with mean subtracted
features_centered = features-mu_features

# Compute the variance across feature 0 (all rows, collumn 0)
var1 = (np.var(features[:,0]))

# Compute the variance across feature 1 (all rows, collumn 1)
var2 = (np.var(features[:,1]))

# Compute standarized feature 0 = Ratio of (i) signed distance from mean and
 (ii) STANDARD DEVIATION, NOT VARIANCE
erruption_duration_std = features_centered[:,0]/np.sqrt(var1)

##################################################
# Verify variance of standarized feature 0 = 1
# print(np.var(erruption_duration_std))
##################################################

# Compute standarized feature 1 = Ratio of (i) signed distance from mean and
 (ii) STANDARD DEVIATION, NOT VARIANCE
time_between_erruptions_std = features_centered[:,1]/np.sqrt(var2)

features_std =np.column_stack((erruption_duration_std,time_between_erruptions_
std))
# print (features_std)

##################################################
#  Verify variance of standarized feature 1 = 1  ##
#  print(np.var(erruption_duration_std))         ##
##################################################

# Map individual standard vectors to single DISTANCE vector
dist_std = np.zeros(len(erruption_duration_std))
for i,j in enumerate(dist_std):
    dist_std[i]= np.linalg.norm([erruption_duration_std[i],time_between_errupt
ions_std[i]])

covar_mat_array = [np.array([[1,0]  ,[0,1]]),np.array([[1,0]  ,[0,1]])]
means_array = np.array([[-.5,-1.5],[.7,1.1]])
means_array_update = means_array
pi_hat_case = 0.5

c_case = np.zeros([2,len(features_std)])
```

```
d = np.zeros([2,len(features_std),len(features_std)])

def get_responsibilities(features,means, covars,pi_hat,c):
    rv0 = multivariate_normal(means[0],covars[0])
    rv1 = multivariate_normal(means[1],covars[0])
    for i,j in enumerate(erruption_duration_std):
#         print(i)
        c[0,i]= (1-pi_hat) * rv0.pdf(features_std[i])/((1-pi_hat) * rv0.pdf(fe
atures_std[i])+(pi_hat) * rv1.pdf(features_std[i]))
        c[1,i]=   (pi_hat) * rv1.pdf(features_std[i])/((1-pi_hat) * rv0.pdf(fe
atures_std[i])+(pi_hat) * rv1.pdf(features_std[i]))

#         PRINTS COVARANCES IF ENABLED
#         print(c[0,i])
#         print(c[1,i])

        d[0,i,i]=c[0,i]
        d[1,i,i]=c[1,i]

    means[0] = np.dot(np.transpose(c[0,:]),features_std)/(np.dot(np.transpose(
c[0,:]),np.ones(len(features_std))))
#     print('mean 1 is ', means[0])

    means[1] = np.dot(np.transpose(c[1,:]),features_std)/(np.dot(np.transpose(
c[1,:]),np.ones(len(features_std))))
#     print('mean 2 is ', means[1])

    covars[0] = np.linalg.multi_dot([np.transpose(features_std-means[0]),d[0],
features_std-means[0]])/(np.dot(np.transpose(c[0,:]),np.ones(len(features_std
))) )
    covars[1] = np.linalg.multi_dot([np.transpose(features_std-means[1]),d[1],
features_std-means[1]])/(np.dot(np.transpose(c[1,:]),np.ones(len(features_std
))) )

    return(np.array(means))




iterations = 500
v_mean = np.empty([iterations,2,2])

v_mean[0]=means_array_update

ax = plt.axes()
for _,i in enumerate(range(iterations-1)):
    v_mean[i+1]=get_responsibilities(features_std,means_array,covar_mat_array,
pi_hat_case,c_case)

    dx1=v_mean[i+1,0,0]-v_mean[i,0,0]
    dy1=v_mean[i+1,0,1]-v_mean[i,0,1]

    ax.arrow(v_mean[i,0,0], v_mean[i,0,1], dx1, dy1, head_width=.05, head_leng
th=.1, fc='b', ec='b',width=.03)

    dx2=v_mean[i+1,1,0]-v_mean[i,1,0]
    dy2=v_mean[i+1,1,1]-v_mean[i,1,1]
```

```
    ax.arrow(v_mean[i,1,0], v_mean[i,1,1], dx2, dy2, head_width=.05, head_leng
th=.01, fc='g', ec='g',width=.03)

plt.xlim(-1.5,1.5)
plt.ylim(-1.5,1.5)
plt.plot(v_mean[iterations-1,0,0], v_mean[iterations-1,0,1], marker='o', marke
rsize=8, color="b")
plt.plot(v_mean[iterations-1,1,0], v_mean[iterations-1,1,1], marker='o', marke
rsize=8, color="g")
plt.scatter(erruption_duration_std,time_between_erruptions_std, color='gray')
plt.title("Erruption Data")
plt.xlabel("Duration of Erruption(Zero Mean - Unit Variance)")
plt.ylabel("Time Since Last Erruption(Zero Mean - Unit Variance)")
plt.show()

print("Centers this is iteration are:")
print("(Data Centered at Zero - Normalized to Unit Variance)")
print(v_mean[iterations-1,0])
print(v_mean[iterations-1,1])
print('\n\n')
```

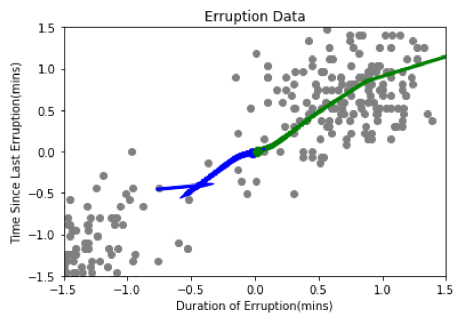



```
Centers this is iteration are:
(Data Centered at Zero - Normalized to Unit Variance)
[-1.26372416 -1.19853173]
[0.71102951 0.67434924]
```

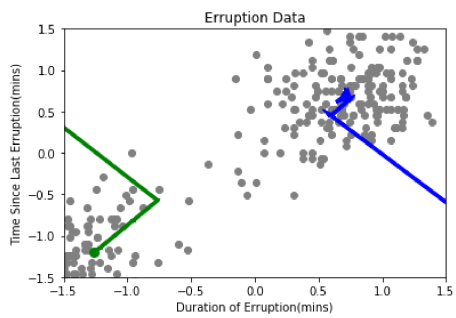
In [5]:
```
for j1 in range(50):

    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.stats import multivariate_normal



    # x, y = np.mgrid[1:6:.06, 0:150:1.5]
    # pos = np.empty(x.shape + (2,))
    # pos[:, :, 0] = x; pos[:, :, 1] = y


    # Calculate mean erruption duration
    mu_erruption_duration=features[:,0].mean()

    # Calculate mean time between erruptions
    mu_time_between_erruptions=features[:,1].mean()

    # Put the two numbers above on [2,1] array
    mu_features = np.array([mu_erruption_duration,mu_time_between_erruptions])

    # Create new feature vector with mean subtracted
    features_centered = features-mu_features

    # Compute the variance across feature 0 (all rows, collumn 0)
    var1 = (np.var(features[:,0]))

    # Compute the variance across feature 1 (all rows, collumn 1)
    var2 = (np.var(features[:,1]))

    # Compute standarized feature 0 = Ratio of (i) signed distance from mean a
nd (ii) STANDARD DEVIATION, NOT VARIANCE
    erruption_duration_std = features_centered[:,0]/np.sqrt(var1)

    ##################################################
    # Verify varianve of standarized feature 0 = 1
    # print(np.var(erruption_duration_std))
    ##################################################

    # Compute standarized feature 1 = Ratio of (i) signed distance from mean a
nd (ii) STANDARD DEVIATION, NOT VARIANCE
    time_between_erruptions_std = features_centered[:,1]/np.sqrt(var2)

    features_std =np.column_stack((erruption_duration_std,time_between_errupti
ons_std))
    # print (features_std)

    ##################################################
    #  Verify varianve of standarized feature 1 = 1  ##
    #  print(np.var(erruption_duration_std))         ##
    ##################################################

    # Map individual standard vectors to single distance vector
    dist_std = np.zeros(len(erruption_duration_std))
    for i,j in enumerate(dist_std):
```

```
        dist_std[i]= np.linalg.norm([erruption_duration_std[i],time_between_er
ruptions_std[i]])
    # print(dist_std)

    # plt.scatter(erruption_duration_std,time_between_erruptions_std)
    # plt.title("Erruption Data")
    # plt.xlabel("Duration of Erruption(mins)")
    # plt.ylabel("Time Since Last Erruption(mins)")
    # plt.show()

    covar_mat_array = [np.array([[1,0]  ,[0,1]]),np.array([[1,0]  ,[0,1]])]
    means_array = np.array([(np.random.random(2)-.5)*4,(np.random.random(2)-.5
)*6])
    means_array_update = means_array
    pi_hat_case = 0.5

    c_case = np.zeros([2,len(features_std)])
    d = np.zeros([2,len(features_std),len(features_std)])

    def get_responsibilities(features,means, covars,pi_hat,c):
        rv0 = multivariate_normal(means[0],covars[0])
        rv1 = multivariate_normal(means[1],covars[0])
        for i,j in enumerate(erruption_duration_std):
    #         print(i)
            c[0,i]= (1-pi_hat) * rv0.pdf(features_std[i])/((1-pi_hat) * rv0.pd
f(features_std[i])+(pi_hat) * rv1.pdf(features_std[i]))
            c[1,i]=   (pi_hat) * rv1.pdf(features_std[i])/((1-pi_hat) * rv0.pd
f(features_std[i])+(pi_hat) * rv1.pdf(features_std[i]))

    #         PRINTS COVARANCES IF ENABLED
    #         print(c[0,i])
    #         print(c[1,i])

            d[0,i,i]=c[0,i]
            d[1,i,i]=c[1,i]

        means[0] = np.dot(np.transpose(c[0,:]),features_std)/(np.dot(np.transp
ose(c[0,:]),np.ones(len(features_std))))
    #     print('mean 1 is ', means[0])

        means[1] = np.dot(np.transpose(c[1,:]),features_std)/(np.dot(np.transp
ose(c[1,:]),np.ones(len(features_std))))
    #     print('mean 2 is ', means[1])

        covars[0] = np.linalg.multi_dot([np.transpose(features_std-means[0]),d
[0],features_std-means[0]])/(np.dot(np.transpose(c[0,:]),np.ones(len(features_
std))) )
        covars[1] = np.linalg.multi_dot([np.transpose(features_std-means[1]),d
[1],features_std-means[1]])/(np.dot(np.transpose(c[1,:]),np.ones(len(features_
std))) )

        return(np.array(means))




    iterations = 500
```

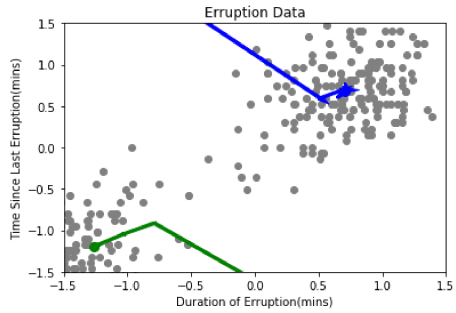
```
    v_mean = np.empty([iterations,2,2])

    v_mean[0]=means_array_update

    ax = plt.axes()
    for _,i in enumerate(range(iterations-1)):
        v_mean[i+1]=get_responsibilities(features_std,means_array,covar_mat_ar
ray,pi_hat_case,c_case)

        dx1=v_mean[i+1,0,0]-v_mean[i,0,0]
        dy1=v_mean[i+1,0,1]-v_mean[i,0,1]

        ax.arrow(v_mean[i,0,0], v_mean[i,0,1], dx1, dy1, head_width=.05, head_
length=.1, fc='b', ec='b',width=.03)

        dx2=v_mean[i+1,1,0]-v_mean[i,1,0]
        dy2=v_mean[i+1,1,1]-v_mean[i,1,1]

        ax.arrow(v_mean[i,1,0], v_mean[i,1,1], dx2, dy2, head_width=.05, head_
length=.01, fc='g', ec='g',width=.03)

    plt.xlim(-1.5,1.5)
    plt.ylim(-1.5,1.5)
    plt.plot(v_mean[iterations-1,0,0], v_mean[iterations-1,0,1], marker='o', m
arkersize=8, color="b")
    plt.plot(v_mean[iterations-1,1,0], v_mean[iterations-1,1,1], marker='o', m
arkersize=8, color="g")
    plt.scatter(erruption_duration_std,time_between_erruptions_std, color='gra
y')
    plt.title("Erruption Data")
    plt.xlabel("Duration of Erruption(Zero Mean - Unit Variance)")
    plt.ylabel("Time Since Last Erruption(Zero Mean - Unit Variance)")
    plt.show()

    print("Centers this is iteration are:")
    print("(Data Centered at Zero - Normalized to Unit Variance)")
    print(v_mean[iterations-1,0])
    print(v_mean[iterations-1,1])
    print('\n\n')
```



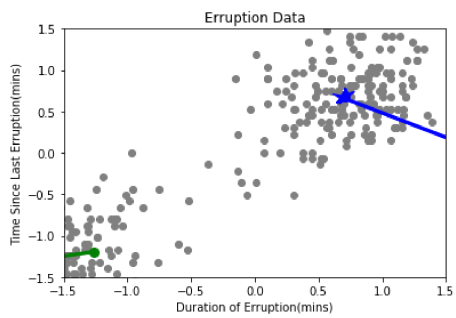

```
Centers this is iteration are:
[-0.01639321 -0.0070657 ]
[0.0163932  0.00706569]
```



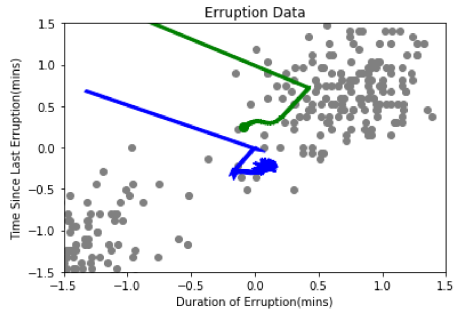

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



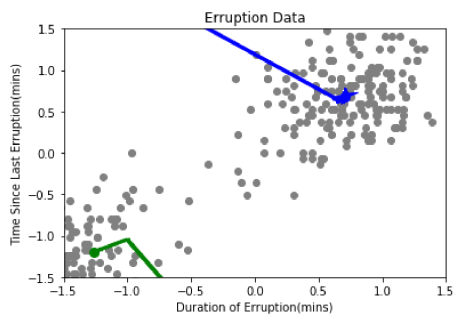

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



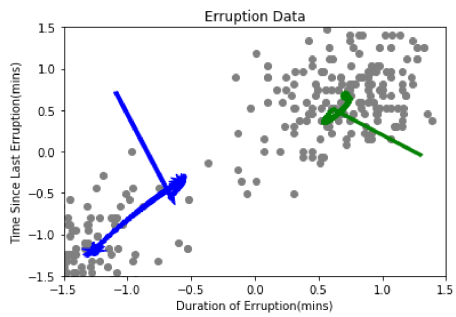

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



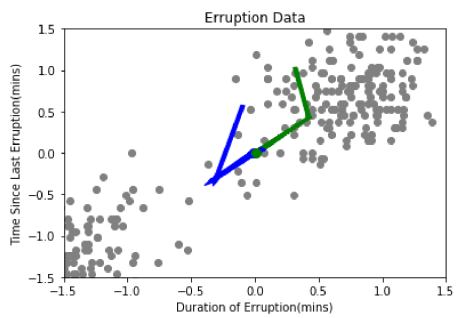

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



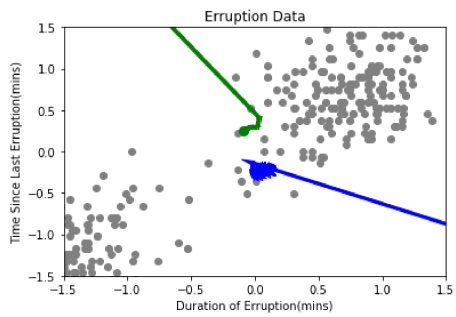

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



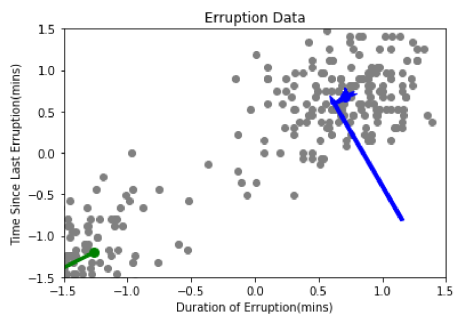

```
Centers this is iteration are:
[-1.26372416 -1.19853173]
[0.71102951 0.67434924]
```



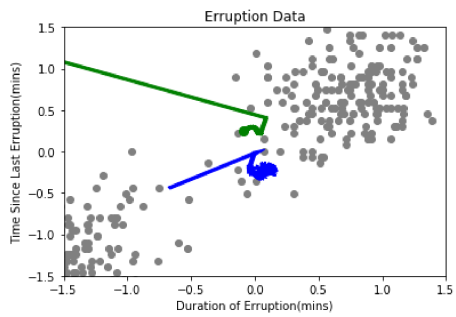

```
Centers this is iteration are:
[-0.00996324 -0.0043547 ]
[0.00996324 0.0043547 ]
```



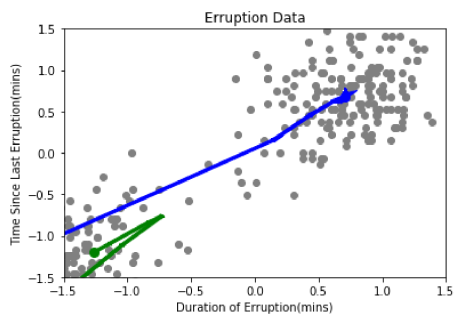

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



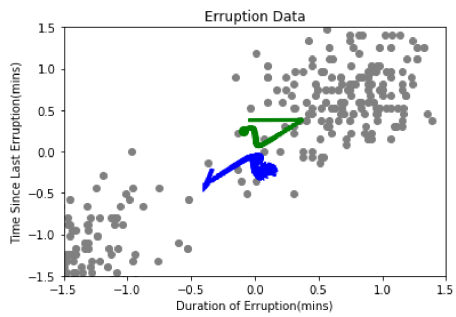

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



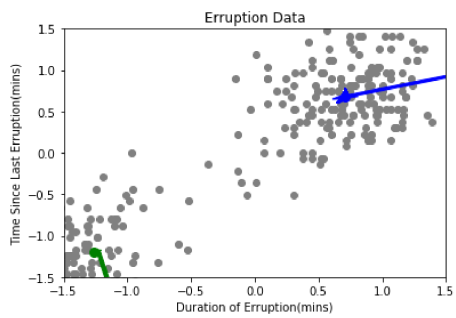

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



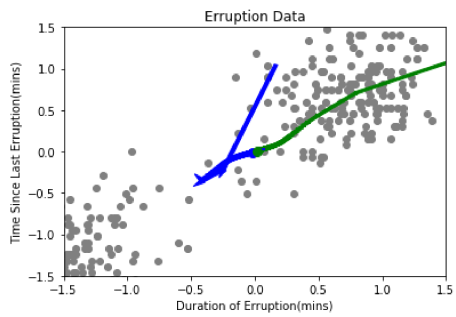

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



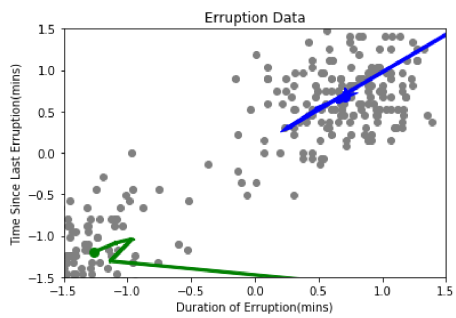

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



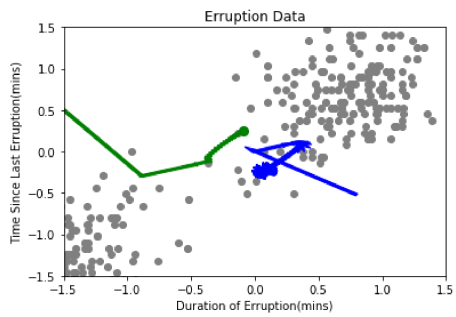

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



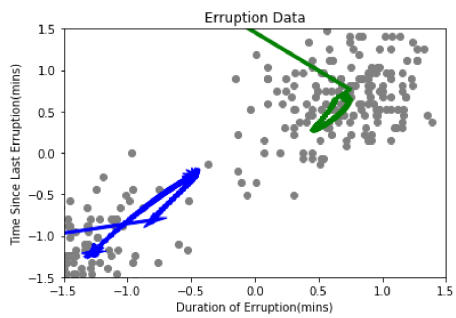

```
Centers this is iteration are:
[-0.01739207 -0.00750044]
[0.01739205 0.00750043]
```



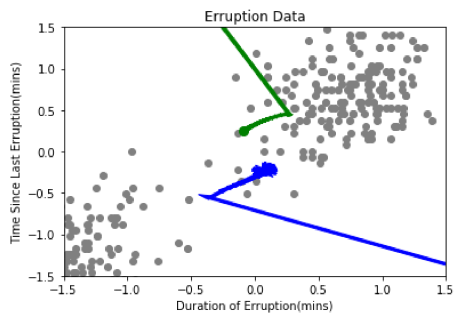

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



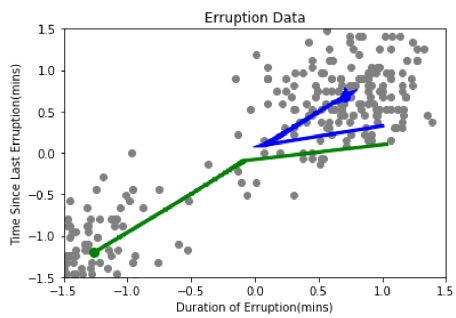

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



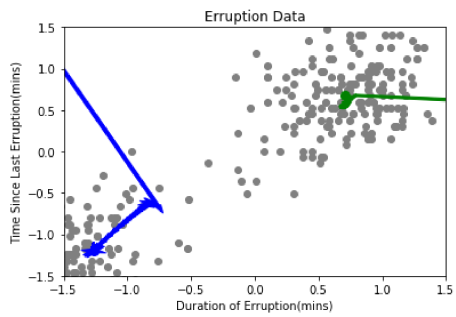

```
Centers this is iteration are:
[-1.26372416 -1.19853173]
[0.71102951 0.67434924]
```



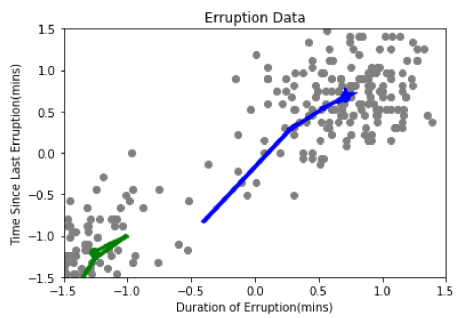

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



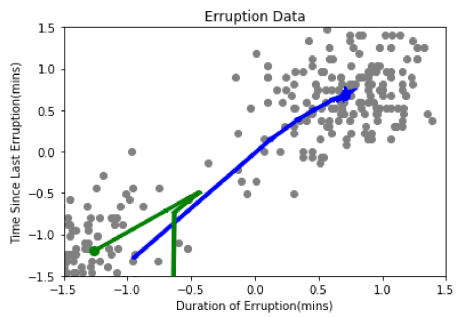

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



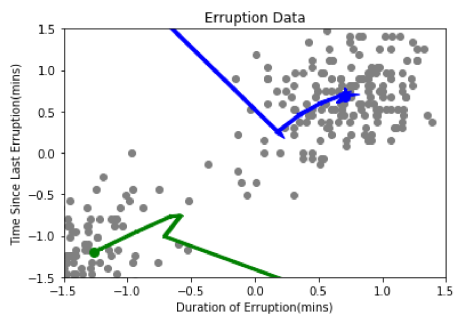

```
Centers this is iteration are:
[-1.26372416 -1.19853173]
[0.71102951 0.67434924]
```



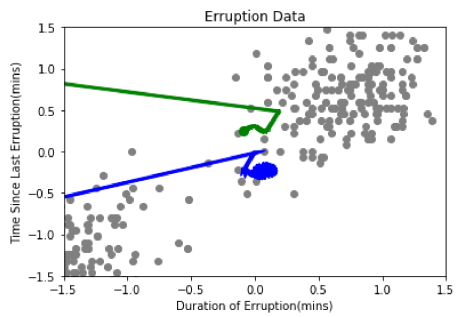

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



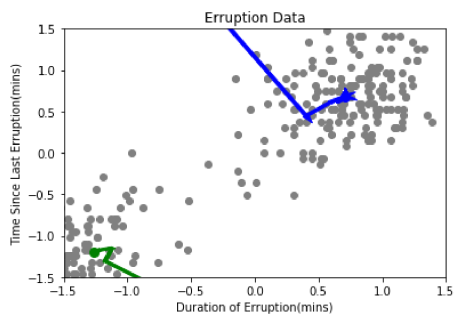

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



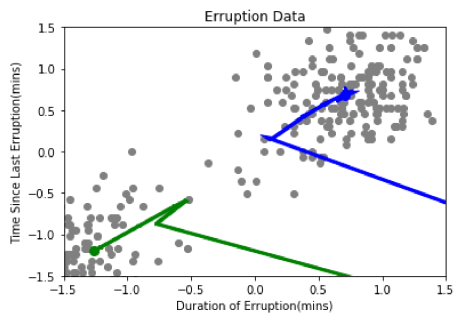

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



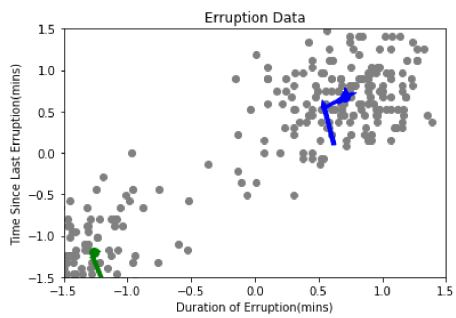

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



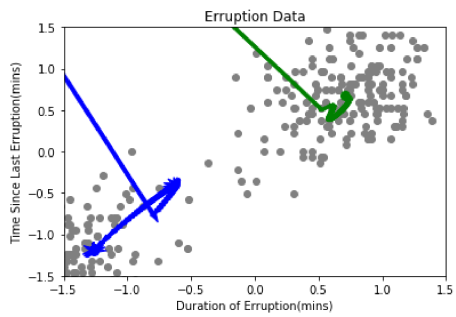

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



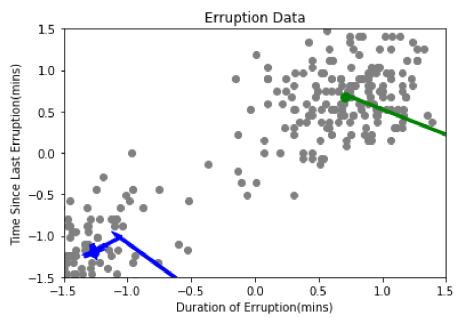

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



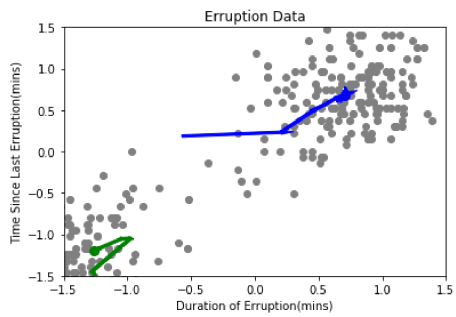

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



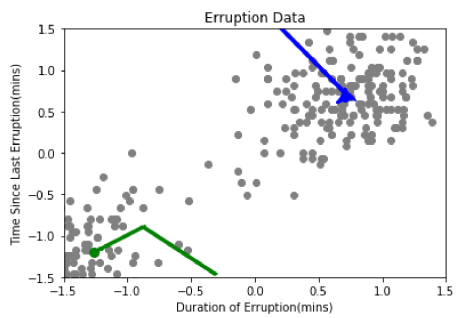

```
Centers this is iteration are:
[-1.26372416 -1.19853173]
[0.71102951 0.67434924]
```



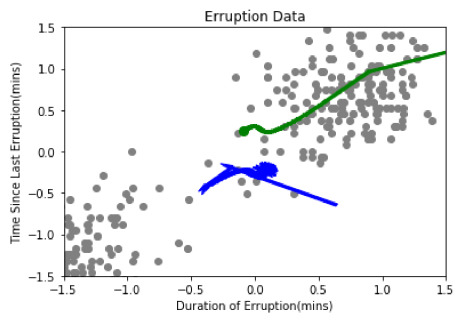

```
Centers this is iteration are:
[-1.26372416 -1.19853173]
[0.71102951 0.67434924]
```



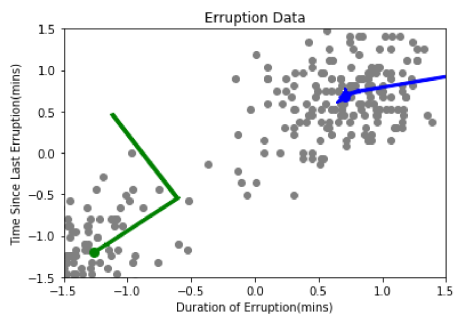

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



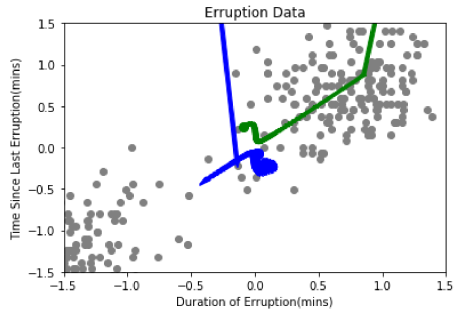

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



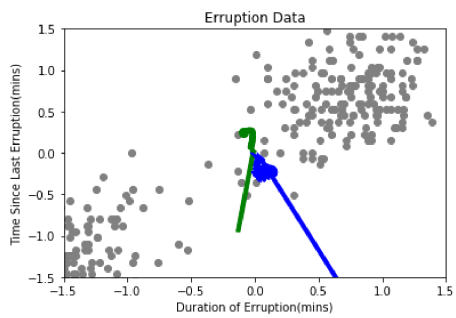

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



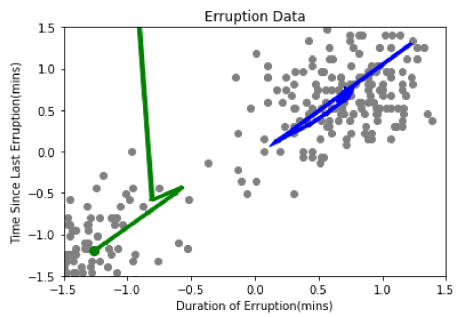

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



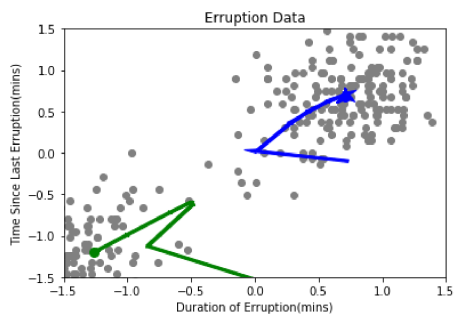

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



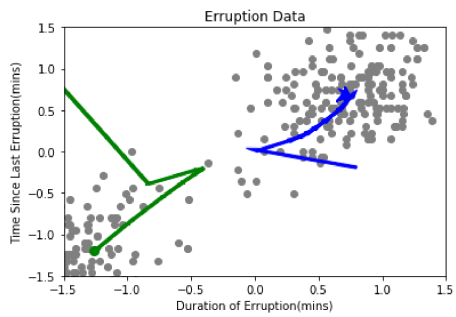

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



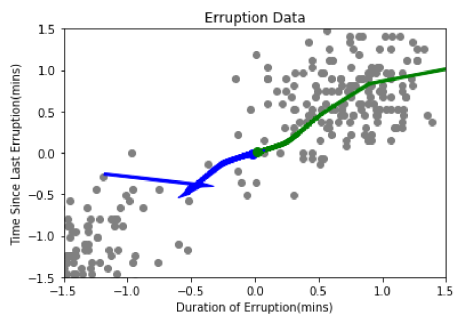

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



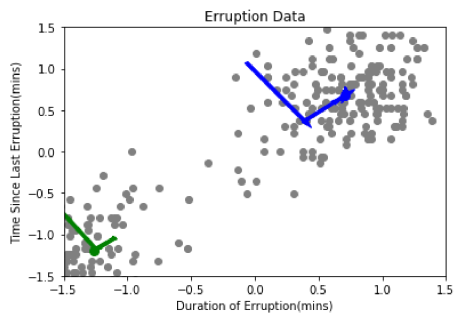

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



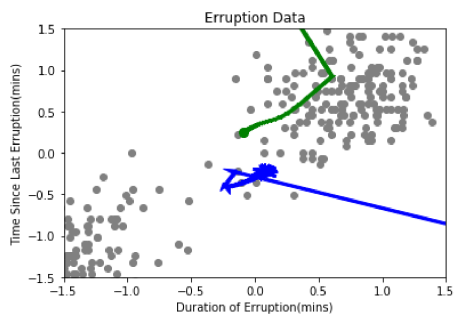

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



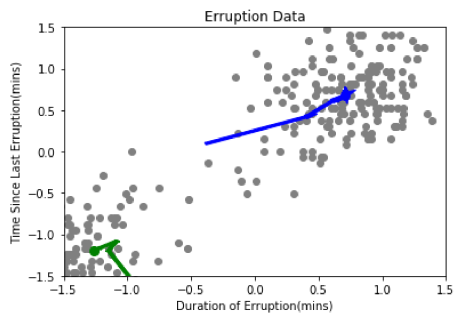

```
Centers this is iteration are:
[-0.01863942 -0.00804402]
[0.01863939 0.00804401]
```



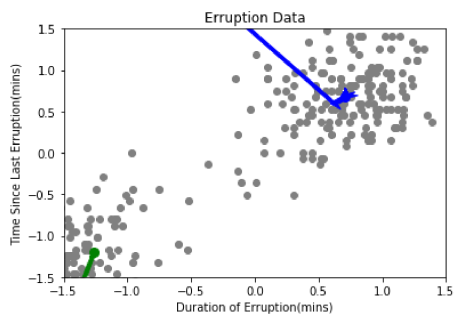

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



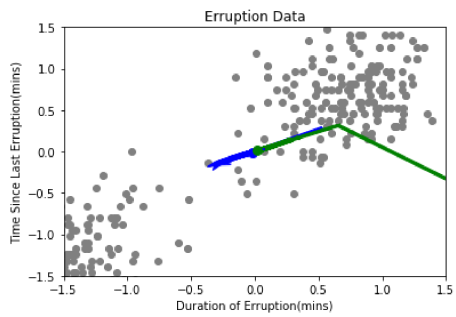

```
Centers this is iteration are:
[ 0.07912339 -0.23410459]
[-0.0854831   0.25292126]
```



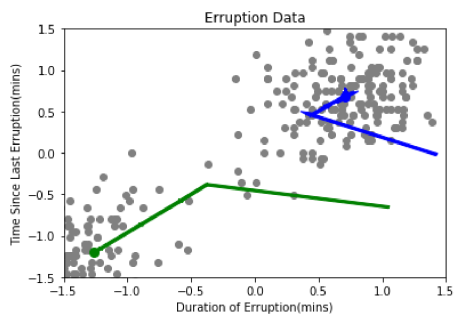

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



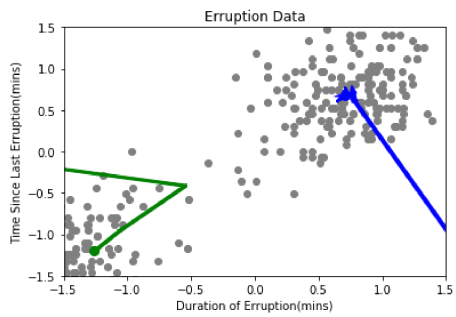

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



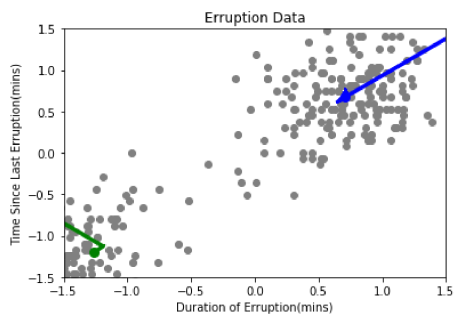

```
Centers this is iteration are:
[-0.01908372 -0.00823782]
[0.01908369 0.00823781]
```



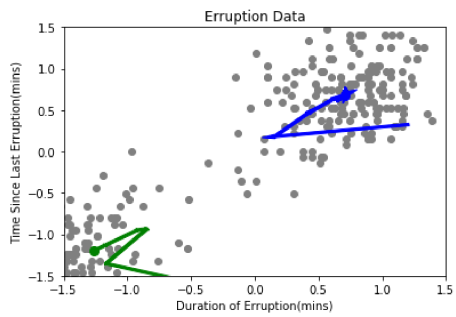

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```



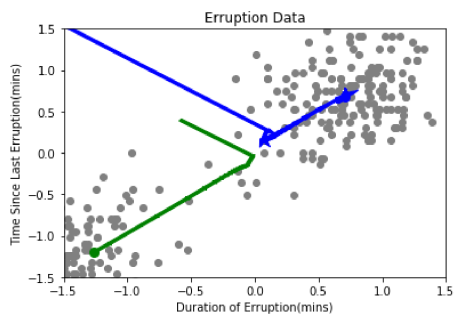

```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```




```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```




```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```

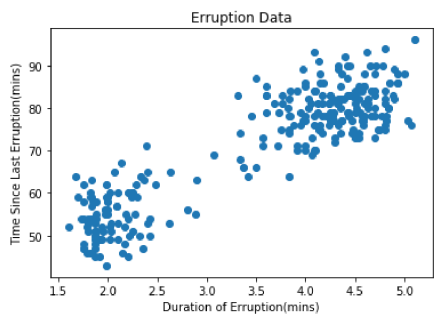



```
Centers this is iteration are:
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```

In [20]:

```
import numpy as np
import matplotlib.pyplot as plt

with open("Problem 3/P3.txt", 'r') as erruptions:
     features = np.array([line.split(' ') for line in erruptions][1:], dtype=f
loat)[:,1:]
print(features.shape)
#print(features)

plt.scatter(features[:,0],features[:,1])
plt.title("Erruption Data")
plt.xlabel("Duration of Erruption(mins)")
plt.ylabel("Time Since Last Erruption(mins)")
plt.show()
```

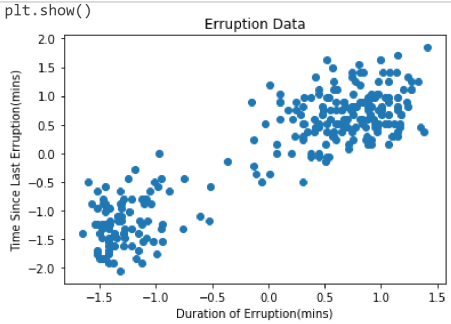
```
(272, 2)
```



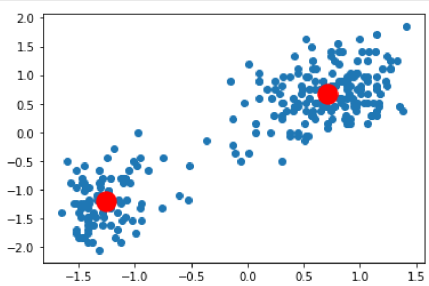
In [3]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal


# Calculate mean erruption duration
mu_erruption_duration=features[:,0].mean()

# Calculate mean time between erruptions
mu_time_between_erruptions=features[:,1].mean()

# Put the two numbers above on [2,1] array
mu_features = np.array([mu_erruption_duration,mu_time_between_erruptions])

# Create new feature vector with mean subtracted
features_centered = features-mu_features

# Compute the variance across feature 0 (all rows, collumn 0)
var1 = (np.var(features[:,0]))

# Compute the variance across feature 1 (all rows, collumn 1)
var2 = (np.var(features[:,1]))

# Compute standarized feature 0 = Ratio of (i) signed distance from mean and
 (ii) STANDARD DEVIATION, NOT VARIANCE
erruption_duration_std = features_centered[:,0]/np.sqrt(var1)

#####################################################
# Verify variance of standarized feature 0 = 1
# print(np.var(erruption_duration_std))
#####################################################

# Compute standarized feature 1 = Ratio of (i) signed distance from mean and
 (ii) STANDARD DEVIATION, NOT VARIANCE
time_between_erruptions_std = features_centered[:,1]/np.sqrt(var2)

features_std =np.column_stack((erruption_duration_std,time_between_erruptions_
std))
# print (features_std)

#####################################################
#  Verify variance of standarized feature 1 = 1  ##
#  print(np.var(erruption_duration_std))         ##
#####################################################

# Map individual standard vectors to single DISTANCE vector
dist_std = np.zeros(len(erruption_duration_std))
for i,j in enumerate(dist_std):
    dist_std[i]= np.linalg.norm([erruption_duration_std[i],time_between_errupt
ions_std[i]])


plt.scatter(erruption_duration_std,time_between_erruptions_std)
plt.title("Erruption Data")
plt.xlabel("Duration of Erruption(mins)")
```

```
plt.ylabel("Time Since Last Erruption(mins)")
plt.show()
```



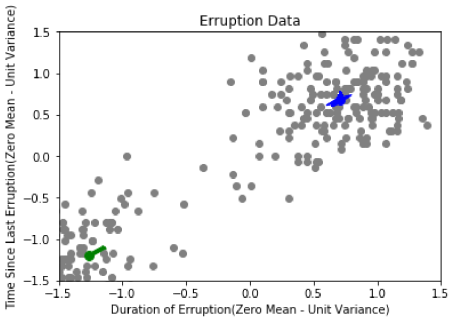
In [19]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, init='k-means++', max_iter=300, n_init=10, rando
m_state=0)
pred_y = kmeans.fit_predict(features_std)
plt.scatter(features_std[:,0], features_std[:,1])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=30
0, c='red')
plt.show()
```



In [18]:
```
for j1 in range(1):

    import numpy as np
    import matplotlib.pyplot as plt
    from scipy.stats import multivariate_normal



    # x, y = np.mgrid[1:6:.06, 0:150:1.5]
    # pos = np.empty(x.shape + (2,))
    # pos[:, :, 0] = x; pos[:, :, 1] = y


    # Calculate mean erruption duration
    mu_erruption_duration=features[:,0].mean()

    # Calculate mean time between erruptions
    mu_time_between_erruptions=features[:,1].mean()

    # Put the two numbers above on [2,1] array
    mu_features = np.array([mu_erruption_duration,mu_time_between_erruptions])

    # Create new feature vector with mean subtracted
    features_centered = features-mu_features

    # Compute the variance across feature 0 (all rows, collumn 0)
    var1 = (np.var(features[:,0]))

    # Compute the variance across feature 1 (all rows, collumn 1)
    var2 = (np.var(features[:,1]))

    # Compute standarized feature 0 = Ratio of (i) signed distance from mean a
nd (ii) STANDARD DEVIATION, NOT VARIANCE
    erruption_duration_std = features_centered[:,0]/np.sqrt(var1)

    ##################################################
    # Verify varianve of standarized feature 0 = 1
    # print(np.var(erruption_duration_std))
    ##################################################

    # Compute standarized feature 1 = Ratio of (i) signed distance from mean a
nd (ii) STANDARD DEVIATION, NOT VARIANCE
    time_between_erruptions_std = features_centered[:,1]/np.sqrt(var2)

    features_std =np.column_stack((erruption_duration_std,time_between_errupti
ons_std))
    # print (features_std)

    ##################################################
    #  Verify varianve of standarized feature 1 = 1  ##
    #  print(np.var(erruption_duration_std))         ##
    ##################################################

    # Map individual standard vectors to single distance vector
    dist_std = np.zeros(len(erruption_duration_std))
    for i,j in enumerate(dist_std):
```

```
        dist_std[i]= np.linalg.norm([erruption_duration_std[i],time_between_er
ruptions_std[i]])
    # print(dist_std)

    # plt.scatter(erruption_duration_std,time_between_erruptions_std)
    # plt.title("Erruption Data")
    # plt.xlabel("Duration of Erruption(mins)")
    # plt.ylabel("Time Since Last Erruption(mins)")
    # plt.show()

    covar_mat_array = [np.array([[1,0]  ,[0,1]]),np.array([[1,0]  ,[0,1]])]
    means_array = np.array([kmeans.cluster_centers_[0,:],kmeans.cluster_center
s_[1,:]])
    means_array_update = means_array
    pi_hat_case = 0.5

    c_case = np.zeros([2,len(features_std)])
    d = np.zeros([2,len(features_std),len(features_std)])

    def get_responsibilities(features,means, covars,pi_hat,c):
        rv0 = multivariate_normal(means[0],covars[0])
        rv1 = multivariate_normal(means[1],covars[0])
        for i,j in enumerate(erruption_duration_std):
    #         print(i)
            c[0,i]= (1-pi_hat) * rv0.pdf(features_std[i])/((1-pi_hat) * rv0.pd
f(features_std[i])+(pi_hat) * rv1.pdf(features_std[i]))
            c[1,i]=   (pi_hat) * rv1.pdf(features_std[i])/((1-pi_hat) * rv0.pd
f(features_std[i])+(pi_hat) * rv1.pdf(features_std[i]))

    #         PRINTS COVARANCES IF ENABLED
    #         print(c[0,i])
    #         print(c[1,i])

            d[0,i,i]=c[0,i]
            d[1,i,i]=c[1,i]

        means[0] = np.dot(np.transpose(c[0,:]),features_std)/(np.dot(np.transp
ose(c[0,:]),np.ones(len(features_std))))
    #     print('mean 1 is ', means[0])

        means[1] = np.dot(np.transpose(c[1,:]),features_std)/(np.dot(np.transp
ose(c[1,:]),np.ones(len(features_std))))
    #     print('mean 2 is ', means[1])

        covars[0] = np.linalg.multi_dot([np.transpose(features_std-means[0]),d
[0],features_std-means[0]])/(np.dot(np.transpose(c[0,:]),np.ones(len(features_
std))) )
        covars[1] = np.linalg.multi_dot([np.transpose(features_std-means[1]),d
[1],features_std-means[1]])/(np.dot(np.transpose(c[1,:]),np.ones(len(features_
std))) )

        return(np.array(means))




    iterations = 500
```

```
    v_mean = np.empty([iterations,2,2])

    v_mean[0]=means_array_update

    ax = plt.axes()
    for _,i in enumerate(range(iterations-1)):
        v_mean[i+1]=get_responsibilities(features_std,means_array,covar_mat_ar
ray,pi_hat_case,c_case)

        dx1=v_mean[i+1,0,0]-v_mean[i,0,0]
        dy1=v_mean[i+1,0,1]-v_mean[i,0,1]

        ax.arrow(v_mean[i,0,0], v_mean[i,0,1], dx1, dy1, head_width=.05, head_
length=.1, fc='b', ec='b',width=.03)

        dx2=v_mean[i+1,1,0]-v_mean[i,1,0]
        dy2=v_mean[i+1,1,1]-v_mean[i,1,1]

        ax.arrow(v_mean[i,1,0], v_mean[i,1,1], dx2, dy2, head_width=.05, head_
length=.01, fc='g', ec='g',width=.03)

    plt.xlim(-1.5,1.5)
    plt.ylim(-1.5,1.5)
    plt.plot(v_mean[iterations-1,0,0], v_mean[iterations-1,0,1], marker='o', m
arkersize=8, color="b")
    plt.plot(v_mean[iterations-1,1,0], v_mean[iterations-1,1,1], marker='o', m
arkersize=8, color="g")
    plt.scatter(erruption_duration_std,time_between_erruptions_std, color='gra
y')
    plt.title("Erruption Data")
    plt.xlabel("Duration of Erruption(Zero Mean - Unit Variance)")
    plt.ylabel("Time Since Last Erruption(Zero Mean - Unit Variance)")
    plt.show()

    print("Centers this is iteration are:")
    print("(Data Centered at Zero - Normalized to Unit Variance)")
    print(v_mean[iterations-1,0])
    print(v_mean[iterations-1,1])
    print('\n\n')
```




```
Centers this is iteration are:
(Data Centered at Zero - Normalized to Unit Variance)
[0.71158326 0.67612817]
[-1.2601228  -1.19733638]
```

In [ ]:

1.5 Programming Exercise 4

1. The following code shows the implementation of Programming Exercise 4.

(a) First, use multidimensional scaling (MDS) to coerce D into a 2-dimensional vector representation. You can use the functions in sklearn.manifold for this.

i. MDS will attempt to output a set of points x 2 R42,m such that the Euclidean distance between every pair of points approximates the Nei's distance between these populations. What assumptions are being made? Under what circumstances could this fail? How could we measure how much information is being lost? Please explain.

Answer: It is being assumed that the distances are a result of linearly independent (or just slightly lineraly dependent) features in m dimensional space. This further implies that all distances will follow vector distance relationships (i.e.: triangle inequality, distance relationships).

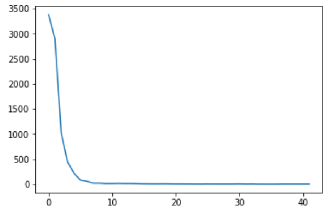
This could fail if the dissimilarity data is, for instance, a quantitative encoding of a qualitative variable, where the above limitations would not necessarily hold.

In [190]:
```
import sys
import numpy as np
import sklearn.manifold
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import single, fcluster
from scipy.spatial.distance import pdist

np.set_printoptions(threshold=sys.maxsize)

with np.load('mds-population.npz') as data:
#     print(data['D'])
#     print(data['population_list'])
    differences = data['D']
    labels = data['population_list']

MDS_differences = np.zeros([len(labels),len(labels)])
MDS_result = sklearn.manifold.MDS(n_components=2,dissimilarity='precomputed',random_state=0)
features_MDS_2D = MDS_result.fit_transform(differences)


# for i in range(len(differences)):
#     for j in range(len(differences)):
#         MDS_differences[i,j] = differences[i,j] - np.linalg.norm(features_MDS_2D[i,:]-features_MDS_2D[j,:])
#         print("The Actual Difference from the dataset is", differences[i,j])
#         print("The 2D Difference Approximation is", np.linalg.norm(features_MDS_2D[i,:]-features_MDS_2D
[j,:]))
#         print("\n")
```

ii. One way of increasing the quality of the output is by increasing the dimensionality of the MDS result. How many dimensions are necessary to capture most of the variation in the data? There are several ways of making this judgment: for instance, you might sample some quality measure between x and D while varyingm, or you might count the nonzero singular values of D, or inspect the singular values of x at some high dimension like m= 20. Briefly explain your method and justify why it makes sense.

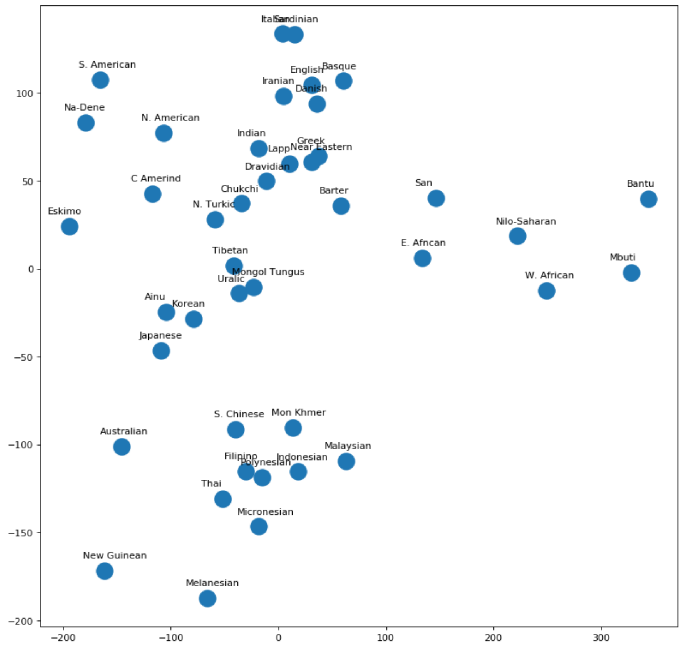
Answer: One effective method, which is implemented above, is to look at the reduction in the approximation error, as m in increased. As shown in the plot below, beyond m = 8, diminishing returns are obtained by increasing the dimensionality of the approximation.

In [191]:
```
MDS_differences = np.zeros([len(labels),len(labels),len(labels)])
MDS_step_delta = np.zeros([len(labels),2])

# print(MDS_differences.shape)

for i1 in range(len(labels)):

    MDS_result = sklearn.manifold.MDS(n_components=i1+1,dissimilarity='precomputed',random_state=0)
    features_MDS = MDS_result.fit_transform(differences)
    # print(features_MDS)

    for i in range(len(differences)):
        for j in range(len(differences)):
            MDS_differences[i1,i,j] = differences[i,j] - np.linalg.norm(features_MDS[i,:]-features_MDS[j,:])
    MDS_step_delta[i1,0] = i1
    MDS_step_delta[i1,1] = np.linalg.norm(MDS_differences[i1] - MDS_differences[i1-1])

plt.plot(MDS_step_delta[:,0],MDS_step_delta[:,1])
plt.show()
```



iii. Use MDS to embed the distancematrix into only two dimensions and show the resulting scatterplot. Label each point with the name of its population.

In [192]:
```
from matplotlib.pyplot import figure

fig, ax = plt.subplots(figsize=(12,12),dpi=80)

for i, txt in enumerate(labels):
    ax.annotate(str(txt)[2:-1], (features_MDS_2D[i,0]-20, features_MDS_2D[i,1]+7))

scatter = ax.scatter(features_MDS_2D[:,0], features_MDS_2D[:,1],s=300)
plt.show()
```



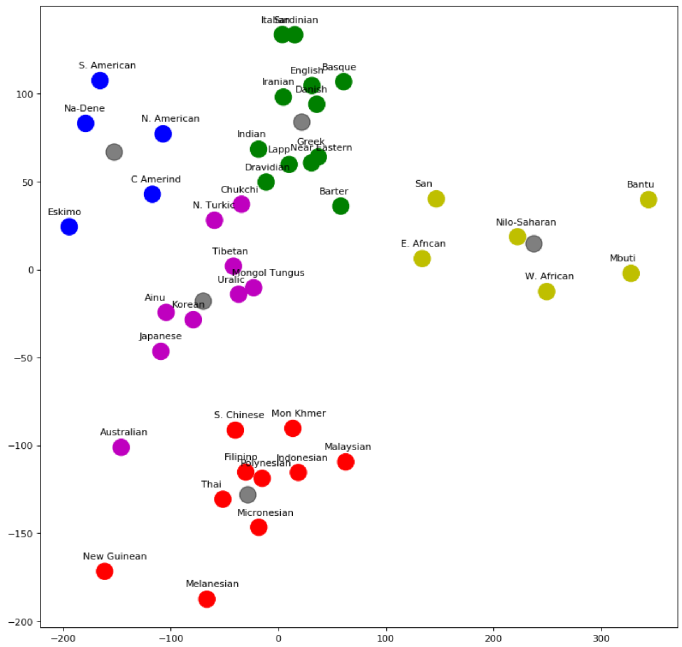
b) k-means on 2D embedding. Select an appropriate k and run k-means on the scatterplot. Show the resulting clusters.

Since we are working with only two dimensions, this clustering is likely to lose a lot of highdimensional structure. Do you agree with the resulting clustering? What information seems to be lost?

Answer: The clustering is fairly reasonable. On the other hand, some genomic pairs are inherently more similar; this information is not captured in an elementary 2D k-means representation of the data.

In [193]:

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10, random_state=0)
pred_y = kmeans.fit_predict(features_MDS_2D)

c_dic = {0:'r',1:'g',2:"y",3:"b",4:"m"}
c_dic[0]
c_list = ([c_dic[i] for i in kmeans.labels_])


fig, ax = plt.subplots(figsize=(12,12),dpi=80)
for i, txt in enumerate(labels):
    ax.annotate(str(txt)[2:-1], (features_MDS_2D[i,0]-20, features_MDS_2D[i,1]+7))

scatter = ax.scatter(features_MDS_2D[:,0], features_MDS_2D[:,1],c=c_list,s=300)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='k', alpha=.50)
plt.show()
```

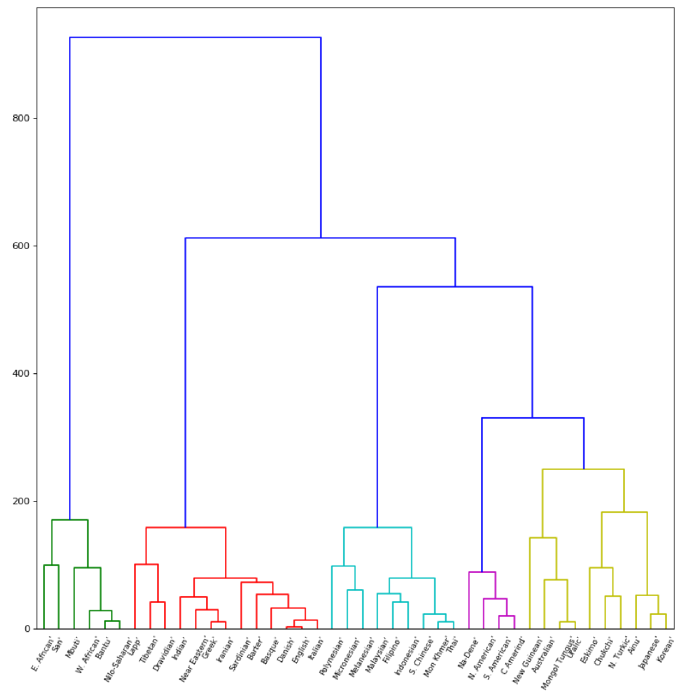


Comparing hierarchical clustering with K-Means. Use hierarchical clustering to cluster the original distance matrix. We suggest using the functions in scipy.cluster.hierarchy for this, as this library can plot the resulting graph structure. Show the resulting tree as a dendrogram, labeling the x axis with the categorical population names and the y axis with the Nei's distance between clusters. (It's also possible to do this with sklearn, but traversing the resulting structure is much harder). To turn the resulting tree into a flat clustering of points, cut off the dendrogram at a certain distance by merging all subclusters within this distance together. This can be done with the scipy.cluster.hierarchy.fcluster function. 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?

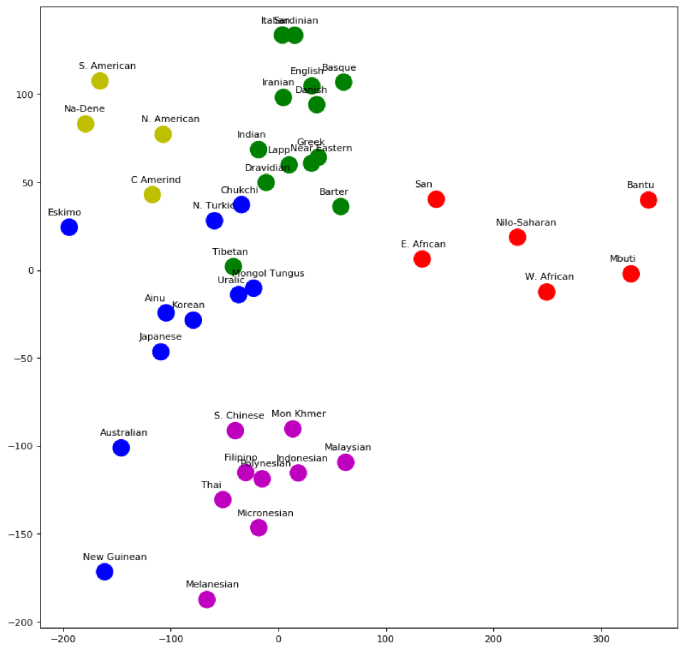
Answer: Reasonable agreement is obtained. However, hierarchical clustering differs from the k-means result in that the latter is inherently rigid with respect to classifications with respect to cluster centers.

In [194]:

```python
from scipy.cluster import hierarchy
import matplotlib.pyplot as plt

# First define the leaf label function.
def llf(id):
    return str(labels[id])[2:]
# The text for the leaf nodes is going to be big so force
# a rotation of 90 degrees.

scipy.cluster.hierarchy.set_link_color_palette(None)

# print(differences)
unique_differences = scipy.spatial.distance.squareform(differences,force='tovector')
# print(unique_differences)
Z = hierarchy.linkage(unique_differences, 'ward',optimal_ordering=True)
# print(Z)
G = scipy.cluster.hierarchy.fcluster(Z,300,criterion='distance')
# print("G Unique:",np.unique(G))
# print("G ",(G))


plt.figure(figsize=(12,12),dpi=80)
dn = hierarchy.dendrogram(Z,count_sort='descending',leaf_label_func=llf, leaf_rotation=60, color_threshold=300
)
plt.show()
```



In [195]:
```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10, random_state=0)
pred_y = kmeans.fit_predict(features_MDS_2D)

c_dic = {0:'r',1:'g',2:"y",3:"b",4:"m"}
c_list_kmeans = ([c_dic[i] for i in kmeans.labels_])
c_list_hierarchical = ([c_dic[i-1] for i in G])


fig, ax = plt.subplots(figsize=(12,12),dpi=80)
for i, txt in enumerate(labels):
    ax.annotate(str(txt)[2:-1], (features_MDS_2D[i,0]-20, features_MDS_2D[i,1]+7))

scatter = ax.scatter(features_MDS_2D[:,0], features_MDS_2D[:,1],c=c_list_hierarchical,s=300)
plt.show()
```



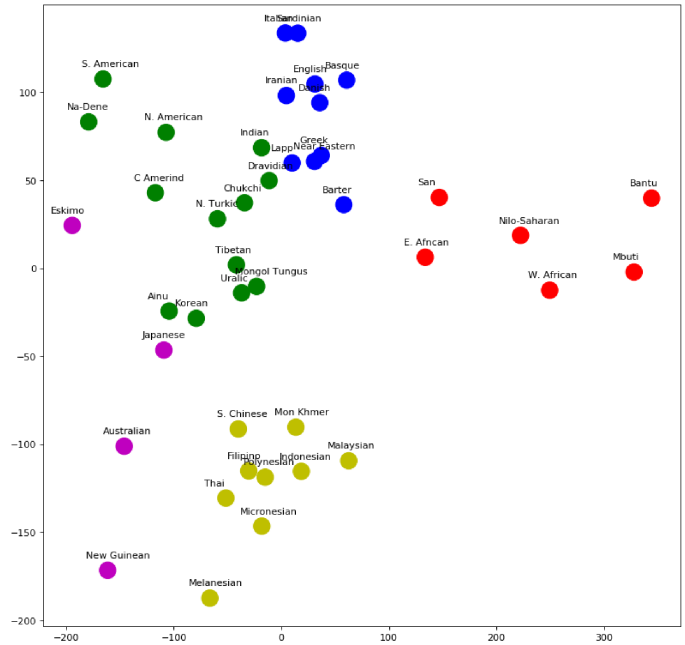
Compare k-medoids with k-means. Repeat the above experiment, but applying k-medoid clustering on the original distance matrix. Show the resulting clusters on a 2D scatterplot. Are there any significant differences between the clustering chosen by k-medoids compared to k-means?

Answer: The k-medoids clustering is substantially different from the result obtained with k-means, allowing significantly higher more assymetry in cluster shape (manifoldy covariance).

In [202]:

```
from pyclustering.cluster.kmedoids import kmedoids
from pyclustering.cluster import cluster_visualizer
from pyclustering.utils import read_sample
from pyclustering.samples.definitions import FCPS_SAMPLES

# Load list of points for cluster analysis.
sample = features_MDS_2D

# Set random initial medoids.
initial_medoids = [1,10,20,30,41]
# Create instance of K-Medoids algorithm.
kmedoids_instance = kmedoids(sample, initial_medoids)
# Run cluster analysis and obtain results.
kmedoids_instance.process()
clusters = kmedoids_instance.get_clusters()
# Show allocated clusters.
# print(clusters)
# Display clusters.

a=np.zeros(len(features_MDS_2D))

for i1,j1 in enumerate(clusters):
    for i2,j2 in enumerate(j1):
        a[j2]=i1

print(a)

c_dic = {0:'r',1:'g',2:"y",3:"b",4:"m"}
c_list_kmedoids = ([c_dic[i] for i in a])


fig, ax = plt.subplots(figsize=(12,12),dpi=80)
for i, txt in enumerate(labels):
    ax.annotate(str(txt)[2:-1], (features_MDS_2D[i,0]-20, features_MDS_2D[i,1]+7))


scatter = ax.scatter(features_MDS_2D[:,0], features_MDS_2D[:,1],c=c_list_kmedoids,s=300)
plt.show()
```

```
[0. 0. 0. 0. 0. 3. 0. 1. 3. 3. 1. 1. 4. 1. 2. 2. 1. 1. 1. 2. 2. 2. 1. 2.
 3. 3. 3. 3. 3. 3. 3. 1. 4. 1. 1. 1. 1. 2. 2. 2. 4. 4.]
```



In [ ]:

2 Written Exercises

2.1 Written Exercise 1

2.2 Written Exercise 1

1. Bellow is our Solution for Written Exercise 1.

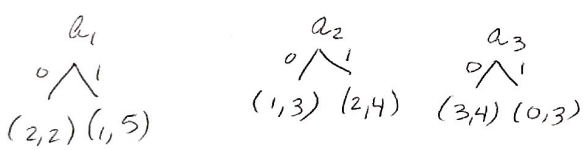
$$I(r) = \min(r, 1-r)$$

We know $\max(P1, N1) = p^1$, $\max(P2, N2) = p^2$

Plugging into weighted impurity of split:

$$= (P_1 + N_1) \cdot \frac{N_1}{P_1 + N_1} + (P_2 + N_2) \cdot \frac{N_2}{P_2 + N_2} = \boxed{N_1 + N_2}$$

(b) Trees of given data



a_1 : $\left[\frac{2}{4}\frac{2}{4} + \frac{1}{6}\frac{5}{6}\right] \cdot 2 \cong 0.78$

a_2 $\left[\frac{1}{4}\frac{3}{4} + \frac{2}{6}\frac{4}{6}\right] \cdot 2 \cong 0.82$

a_3 $\left[\frac{3}{7}\frac{4}{7} + \frac{0}{3} + 1\right] \cdot 2 \cong 0.49 \leftarrow$ lowest val

Min error:

a_1 : $2+1=3$
a_2 : $1+2=3$
a_3 : $0+3=3$

} Hence, split can be selected arbitrarily.

(d) In general, the min-error impurity is not generally optimal in optimizing branching for decision trees.

(c) Before splits, min error impurity function

$$P_1+P_2+\cancel{N_1}+N_2 \cdot I\left[\frac{P_1+P_2}{P_1+P_2+\cancel{N_1}+N_2}\right]$$

$$= \min\left\{(P_1+P_2) \text{ OR } (N_1+N_2)\right\}$$

If $(P_1 > N_1)$ AND $(P_2 < N_2)$ then

based on (a) error $= N_1 + P_2$

If $(P_1 < N_1)$ AND $(P_2 > N_2)$, then, based on (a)

error $= P_1 + N_2$

$P_1 + P_2 > N_1 + P_2$

$N_1 + N_2 > P_1 + N_2$

Test

2.3 Written Exercise 2

Bootstrap aggregation ("bagging") Suppose we have a training set of N examples, and we use bagging to create a bootstrap replicate by drawing N samples with replacement to form a new training set.

Because each sample is drawn with replacement, some examples may be included in this bootstrap replicate multiple times, and some examples will be omitted from it entirely.

As a function of N, compute the expected fraction of the training set that does not appear at all in the bootstrap replicate. What is the limit of this expectation as N approaches infinity?

Let P be the expected fraction of the training set that will be excluded. Below is the probability for a given object not to be drawn from n samples in n attempts, with replacements.

$\left(\frac{N-1}{N}\right)^N$ which applies to all N objects in the set. Therefore, $\lim_{x\to\infty}\left(\frac{N-1}{N}\right)^N = E(P) = \frac{1}{e}$ ■

References

[1] Pandas Python Data Analysis Library

[2] SKLearn Python Data Analysis Library

[3] Scipy Python Data Analysis Library

[4] MatLibPlot Python Library

[5] Pyclustring Data Mining Library