CHANDRIMA BHATTACHARYA

&

ZEYU WANG

In [1]:

```
"""Question 1"""
```

Out[1]:

```
'Question 1'
```

In [2]:

```
import pandas as pd
import numpy as np
import operator
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction import DictVectorizer
from sklearn.preprocessing import normalize
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score
import string
import nltk
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Chandrima\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Chandrima\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Chandrima\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]:

```
True
```

In [3]:

```
#a:  Parse the following data
amazon_data = pd.read_csv(r'F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/sen
                          sep = ".  ", header = None, names = ["review", "value"])
amazon_data.head()
```

```
C:\Users\Chandrima\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: Pars
erWarning: Falling back to the 'python' engine because the 'c' engine does n
ot support regex separators (separators > 1 char and different from '\s+' ar
e interpreted as regex); you can avoid this warning by specifying engine='py
thon'.
  This is separate from the ipykernel package so we can avoid doing imports
until
```

Out[3]:

	review	value
0	So there is no way for me to plug it in here i...	0
1	Good case, Excellent value	1
2	Great for the jawbone	1
3	Tied to charger for conversations lasting more...	0
4	The mic is great	1

In [4]:

```
yelp_data = pd.read_csv(r'F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/senti
                          sep = ".  ", header = None, names = ["review", "value"])
yelp_data.head()
```

```
C:\Users\Chandrima\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: Pars
erWarning: Falling back to the 'python' engine because the 'c' engine does n
ot support regex separators (separators > 1 char and different from '\s+' ar
e interpreted as regex); you can avoid this warning by specifying engine='py
thon'.
```

Out[4]:

	review	value
0	Wow... Loved this place	1
1	Crust is not good	0
2	Not tasty and the texture was just nasty	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

In [5]:

```
imdb_data = pd.read_csv(r'F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/senti
                        sep = ".  ", header = None, names = ["review", "value"])
imdb_data.head()
```

```
C:\Users\Chandrima\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: Pars
erWarning: Falling back to the 'python' engine because the 'c' engine does n
ot support regex separators (separators > 1 char and different from '\s+' ar
e interpreted as regex); you can avoid this warning by specifying engine='py
thon'.
```

Out[5]:

	review	value
0	A very, very, very slow-moving, aimless movie ...	0
1	Not sure who was more lost - the flat characte...	0
2	Attempting artiness with black & white and cle...	0
3	Very little music or anything to speak of.	0
4	The best scene in the movie was when Gerardo i...	1

In [6]:

```
print ("Amazon    |   Yelp    |   IMDB")
print (amazon_data.shape, "|", yelp_data.shape, "|", imdb_data.shape)
```

```
Amazon    |   Yelp    |   IMDB
(1000, 2) | (1000, 2) | (1000, 2)
```

In [7]:

```
# Label Balance
"""Given that the shape of all three text files includes 100 reviews with either a 0 or 1 v
```

Out[7]:

```
'Given that the shape of all three text files includes 100 reviews with eith
er a 0 or 1 value and we know that sum of the values are 500, we know that t
he labels must be balanced.'
```

In [8]:

```
#b: Preprocessing of the data

## Remove stopwords
def strip_punctuation(line):
    result = ""
    for c in line:
        if c not in string.punctuation:
            result += c
    return result

## Lemmatization of all the words
def lemmatize_line(line):
    lemma = WordNetLemmatizer()
    word_list = nltk.word_tokenize(line)
    lemmatized_output = ' '.join([lemma.lemmatize(w, pos='a') for w in word_list])
    word_list = nltk.word_tokenize(lemmatized_output)
    lemmatized_output = ' '.join([lemma.lemmatize(w, pos='v') for w in word_list])
    return lemmatized_output

## Strip stop words
def strip_stop_words(line):
    word_list = nltk.word_tokenize(line)
    filtered_words = ' '.join([word for word in word_list if word not in stopwords.words('e
    return filtered_words
```

In [9]:

```
def preprocess_data(raw_data):
    data = raw_data.apply(lambda x: x.astype(str).str.lower())
    pd.to_numeric(data['value'])
    data['review'].astype('str')
    data['review'] = data['review'].apply(lambda x: lemmatize_line(x))
    data['review'] = data['review'].apply(lambda x: strip_stop_words(x))
    data['review'] = data['review'].apply(lambda x: strip_punctuation(x))
    return data
```

In [10]:

```
amazon_data_preprocess = preprocess_data(amazon_data)
amazon_data_preprocess.head()
```

Out[10]:

	review	value
0	way plug us unless go converter	0
1	good case excellent value	1
2	great jawbone	1
3	tie charger conversations last 45 minutesmajor...	0
4	mic great	1

In [11]:

```
yelp_data_preprocess = preprocess_data(yelp_data)
yelp_data_preprocess.head()
```

Out[11]:

	review	value
0	wow love place	1
1	crust good	0
2	tasty texture nasty	0
3	stop late may bank holiday rick steve recommen...	1
4	selection menu great price	1

In [12]:

```
imdb_data_preprocess = preprocess_data(imdb_data)
imdb_data_preprocess.head()
```

Out[12]:

	review	value
0	slowmoving aimless movie distress drift yo...	0
1	sure lose flat character audience nearly hal...	0
2	attempt artiness black white clever camera an...	0
3	little music anything speak	0
4	best scene movie gerardo try find song keep ru...	1

In [13]:

```
#c: Split train-test as 80:20
def split_data(data):
    tr, te = train_test_split(data, test_size=0.2)
    return tr, te
```

In [14]:

```
amazon_train, amazon_test = split_data(amazon_data_preprocess)
yelp_train, yelp_test = split_data(yelp_data_preprocess)
imdb_train, imdb_test = split_data(imdb_data_preprocess)
```

In [15]:

```
train_data = pd.DataFrame()
train_data = pd.concat([amazon_train, imdb_train], ignore_index = True)
train_data = pd.concat([train_data, yelp_train], ignore_index = True)
train_data.shape
```

Out[15]:

```
(2400, 2)
```

In [16]:

```
test_data = pd.DataFrame()
test_data = pd.concat([amazon_test, imdb_test], ignore_index = True)
test_data = pd.concat([test_data, yelp_test], ignore_index = True)
test_data.shape
```

Out[16]:

```
(600, 2)
```

In [17]:

```
x_train, y_train = train_data['review'], train_data['value']
x_test, y_test = test_data['review'], test_data['value']
print (x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(2400,) (2400,) (600,) (600,)
```

In [18]:

```
#d: build a dictionary of unique words for training set
train_vectorizer = CountVectorizer()
train_x_bow = train_vectorizer.fit_transform(x_train).todense()
test_vectorizer = CountVectorizer(vocabulary=train_vectorizer.get_feature_names())
test_x_bow = test_vectorizer.fit_transform(x_test).todense()
```

In [19]:

```
"""Above we have vectorized training set first, followed by limiting the vocubulary for tes
```

Out[19]:

```
'Above we have vectorized training set first, followed by limiting the vocub
ulary for test set as otherwise the test-set might contains words which will
be absent in training set, and we will get incorrect feature vectors'
```

In [20]:

```
# Report feature vectors of any two reviews in the training set
print(x_train[4])
print(train_x_bow[4])
print(x_train[21])
print(train_x_bow[21])
```

```
echo problem  very unsatisfactor
[[0 0 0 ... 0 0 0]]
 warranty  problems reoccurebottom line  put money somewhere else  cingular
support
[[0 0 0 ... 0 0 0]]
```

In [21]:

```
#e: Post-processing
train_x_bow_normal = normalize(train_x_bow, norm='l1')
test_x_bow_normal = normalize(test_x_bow, norm='l1')
```

In [22]:

```
"""Here, I have used L1 normalization because the reviews are qualitative so the rendundanc
```

Out[22]:

```
"Here, I have used L1 normalization because the reviews are qualitative so t
he rendundancy is unavoidable and there's no absolute measurement."
```

In [23]:

```
#f: Sentiment Analysis using logistic regression and Naive-based (Gaussian, Bernoulli and M
def sentiment_prediction(x_tr, y_tr, x_te, y_te):

    lr_clf = LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial').
    lr_score = lr_clf.score(x_te, y_te)
    print("Logistic regression accuracy: {}".format(lr_score))
    print("Confussion Matrix:", "\n", confusion_matrix(y_te, lr_clf.predict(x_te)))

    b_nb = BernoulliNB()
    b_fit = b_nb.fit(x_tr, y_tr)
    b_nb_score = b_fit.score(x_te, y_te)
    print("Accuracy of Naive Bayes Classifier with Bernoulli prior: {}".format(b_nb_score))
    print("Confussion Matrix:", "\n", confusion_matrix(y_te, b_fit.predict(x_te)))

    gaussian_nb = GaussianNB()
    gaussian_nb.fit(x_tr, y_tr)
    gaussian_nb_score = gaussian_nb.score(x_te, y_te)
    print("Accuracy of Naive Bayes Classifier with Gaussian prior: {}".format(gaussian_nb_s
    print("Confussion Matrix:", "\n", confusion_matrix(y_te, gaussian_nb.predict(x_te)))

    return lr_score, gaussian_nb_score, b_nb_score, lr_clf.coef_
```

In [24]:

```
lr_score, gaussian_nb_score, b_nb_score, lr_coeff = sentiment_prediction(train_x_bow_normal
                                                                         test_x
```

```
Logistic regression accuracy: 0.7883333333333333
Confussion Matrix:
 [[255  66]
 [ 61 218]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.8133333333333334
Confussion Matrix:
 [[262  59]
 [ 53 226]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.6883333333333334
Confussion Matrix:
 [[260  61]
 [126 153]]
```

In [25]:

```
vocabulary_bow = train_vectorizer.get_feature_names()
lr_coef_val = lr_coeff.tolist()[0]
weight_vector = dict(zip(vocabulary_bow, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
weight_vector
```

Out[25]:

```
[('great', 3.342498970911084),
 ('love', 2.297285972463214),
 ('nice', 1.902211561012528),
 ('excellent', 1.7382161891503407),
 ('amaze', 1.5418266737078257),
 ('good', 1.5301828050738429),
 ('best', 1.484047481851821),
 ('delicious', 1.3159122566579229),
 ('fantastic', 1.230057723356934),
 ('well', 1.219354117628204),
 ('awesome', 1.1642286385405223),
 ('wonderful', 1.087780198567724),
 ('perfect', 1.0828387595913571),
 ('beautiful', 1.04841777935078),
 ('friendly', 1.0207149155839712),
 ('price', 0.9733547293305825),
 ('enjoy', 0.9668645452318055),
 ('comfortable', 0.9379343418781981),
```

In [26]:

```
"""I see Burnoulli for NBC gives me the best result, followed by Logistic Regression and th
```

Out[26]:

```
'I see Burnoulli for NBC gives me the best result, followed by Logistic Regr
ession and the worst is Gaussian NBC'
```

In [27]:

```
#g: N-gram model
train_vectorizer_2gram = CountVectorizer(ngram_range=(2, 2))
train_x_2gram = train_vectorizer_2gram.fit_transform(x_train).todense()
test_vectorizer_2gram = CountVectorizer(ngram_range=(2, 2), vocabulary=train_vectorizer_2gr
test_x_2gram = test_vectorizer_2gram.fit_transform(x_test).todense()
print(x_train[4])
print(train_x_2gram[4])
print(x_train[21])
print(train_x_2gram[21])
```

```
echo problem  very unsatisfactor
[[0 0 0 ... 0 0 0]]
 warranty  problems reoccurebottom line  put money somewhere else  cingular
support
[[0 0 0 ... 0 0 0]]
```

In [28]:

```
train_x_2gram_normal = normalize(train_x_2gram, norm='l1')
test_x_2gram_normal = normalize(test_x_2gram, norm='l1')
lr_score, gaussian_nb_score, b_nb_score, lr_coeff = sentiment_prediction(train_x_2gram_norm
                                                                         y_trai
```

```
Logistic regression accuracy: 0.5966666666666667
Confussion Matrix:
 [[114 207]
 [ 35 244]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.5966666666666667
Confussion Matrix:
 [[109 212]
 [ 30 249]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.6683333333333333
Confussion Matrix:
 [[282  39]
 [160 119]]
```

In [29]:

```
vocabulary_ng = train_vectorizer_2gram.get_feature_names()
lr_coef_val = lr_coeff.tolist()[0]
weight_vector = dict(zip(vocabulary_ng, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
weight_vector
```

Out[29]:

```
[('work great', 1.2007221477664503),
 ('highly recommend', 1.061456145145739),
 ('great phone', 0.836510713908666),
 ('food good', 0.7456916059201351),
 ('one best', 0.7403189151851477),
 ('love place', 0.7399525083544034),
 ('good price', 0.6681850208031861),
 ('great product', 0.6499576092806894),
 ('excellent product', 0.6330740446141688),
 ('easy use', 0.6326868650730371),
 ('really good', 0.5986319213213821),
 ('good product', 0.5864949812650904),
 ('love phone', 0.5723222224750062),
 ('great price', 0.5681441125462072),
 ('pretty good', 0.5567358109681317),
 ('food delicious', 0.540999000359394),
 ('great deal', 0.533278800774241),
 ('nt disappoint', 0.5203255508206126),
```

In [30]:

```
"""For this case, I see the output of all the process is similar, but wierdly Gaussian perf
```

Out[30]:

```
'For this case, I see the output of all the process is similar, but wierdly
Gaussian performs slightly better'
```

In [31]:

```
#h : PCA Analysis using SVD for Bag of Word Model
p,n = np.shape(train_x_bow_normal)
cov_matrix = np.dot(train_x_bow_normal.T, train_x_bow_normal)/(p-1)
u, s, vh = np.linalg.svd(cov_matrix, full_matrices=True)
```

In [32]:

```
# PCA Analysis using SVD for N-gram model
p2,n2 = np.shape(train_x_2gram_normal)
cov_matrix2 = np.dot(train_x_2gram_normal.T, train_x_2gram_normal)/(p-1)
u2, s2, vh2 = np.linalg.svd(cov_matrix2, full_matrices=True)
```

In [33]:

```
# Dimension=10
train_x_10 = np.dot(train_x_bow_normal, u[:,:10])
test_x_10 = np.dot(test_x_bow_normal, u[:,:10])
print ("PCA dim 10 for Bag of Word Model")
lr_score, gaussian_nb_score, b_nb_score, lr_coeff = sentiment_prediction(train_x_10, y_train

train_x_10_ng = np.dot(train_x_2gram_normal, u2[:,:10])
test_x_10_ng = np.dot(test_x_2gram_normal, u2[:,:10])
print ("PCA dim 10 for Ngram Model")
lr_score_ng, gaussian_nb_score_ng, b_nb_score_ng, lr_coeff_ng = sentiment_prediction(train_x
```

```
PCA dim 10 for Bag of Word Model
Logistic regression accuracy: 0.635
Confussion Matrix:
 [[234  87]
 [132 147]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.6066666666666667
Confussion Matrix:
 [[219 102]
 [134 145]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.6233333333333333
Confussion Matrix:
 [[268  53]
 [173 106]]
PCA dim 10 for Ngram Model
Logistic regression accuracy: 0.48333333333333334
Confussion Matrix:
 [[ 12 309]
 [  1 278]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.5216666666666666
Confussion Matrix:
 [[198 123]
 [164 115]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.55
Confussion Matrix:
 [[315   6]
 [264  15]]
```

In [34]:

```
print ("Bag of Words")
lr_coef_val = lr_coeff.tolist()[0]
weight_vector = dict(zip(vocabulary_bow, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)

print ("N-gram model")
lr_coef_val = lr_coeff_ng.tolist()[0]
weight_vector = dict(zip(vocabulary_ng, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)
```

```
Bag of Words
[('010', 2.8577505755728434), ('1010', 1.7759771532555801), ('110', 1.423135
02364502), ('15', -0.6693773261571053), ('13', -0.9502073859538739), ('12',
-1.2898382254225256), ('11', -1.6560349037860573), ('10', -1.672427221606093
6), ('1199', -2.353145926039954), ('100', -3.1641965961671294)]
N-gram model
[('010 grade', 1.301424693125726), ('10 movie', 1.1854864740915152), ('10 1
0', 0.7551715058608989), ('10 feet', 0.750863293325864), ('10 oyvey', 0.3577
0008633371153), ('10 minutes', 0.32908254476652155), ('10 plus', 0.262841373
44038633), ('10 simply', -0.37613923757685475), ('10 save', -1.0087971191978
81), ('10 110', -1.6805937767361494)]
```

In [37]:

```
# Dimension=50
train_x_50 = np.dot(train_x_bow_normal, u[:,:50])
test_x_50 = np.dot(test_x_bow_normal, u[:,:50])
print ("PCA dim 50 for Bag of Word Model")
lr_score, gaussian_nb_score, b_nb_score, lr_coeff = sentiment_prediction(train_x_50, y_train

train_x_50_ng = np.dot(train_x_2gram_normal, u2[:,:50])
test_x_50_ng = np.dot(test_x_2gram_normal, u2[:,:50])
print ("PCA dim 50 for Ngram Model")
lr_score_ng, gaussian_nb_score_ng, b_nb_score_ng, lr_coeff_ng = sentiment_prediction(train_x
```

```
PCA dim 50 for Bag of Word Model
Logistic regression accuracy: 0.6883333333333334
Confussion Matrix:
 [[231  90]
 [ 97 182]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.6366666666666667
Confussion Matrix:
 [[242  79]
 [139 140]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.605
Confussion Matrix:
 [[153 168]
 [ 69 210]]
PCA dim 50 for Ngram Model
Logistic regression accuracy: 0.5016666666666667
Confussion Matrix:
 [[ 26 295]
 [  4 275]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.5466666666666666
Confussion Matrix:
 [[212 109]
 [163 116]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.5566666666666666
Confussion Matrix:
 [[309  12]
 [254  25]]
```

In [38]:

```
print ("Bag of Words")
lr_coef_val = lr_coeff.tolist()[0]
weight_vector = dict(zip(vocabulary_bow, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)

print ("N-gram model")
lr_coef_val = lr_coeff_ng.tolist()[0]
weight_vector = dict(zip(vocabulary_ng, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)
```

```
Bag of Words
[('010', 3.0175075444588986), ('2007', 2.1616212233553), ('20th', 1.78239967
6560226), ('1010', 1.7084880952585775), ('15lb', 1.5640357645893908), ('11
0', 1.5339136787718948), ('40min', 1.3775368316401804), ('2000', 1.175990972
34076), ('34ths', 0.9874919338561612), ('350', 0.9604229230422221), ('45',
0.9537271698196184), ('5320', 0.9509793983742273), ('42', 0.907159607272713
8), ('70s', 0.8205158174925946), ('1979', 0.7618382864541814), ('5020', 0.74
94372122304447), ('2mp', 0.6052035005921853), ('24', 0.5303940609376162),
('2006', 0.4958316457191242), ('3o', 0.49216090494015213), ('70000', 0.38949
97871463807), ('325', 0.33827299203740013), ('1995', 0.28781694644633266),
('2005', 0.22129709031873795), ('1947', 0.18220312674656752), ('20the', 0.16
901724279099792), ('70', 0.11887847749019537), ('25', 0.05392323288165849),
('1948', -0.0710876072905581), ('1998', -0.1098688603137161), ('23', -0.1105
2739653559647), ('30', -0.1359252357881056), ('1980', -0.37910378530586036),
('2160', -0.4467284834279109), ('17', -0.6590020200483581), ('18th', -0.7318
015194425378), ('5year', -0.7518320114474671), ('18', -0.766736016361032),
('15', -0.7750780479749669), ('13', -0.8796997258714572), ('35', -0.88160591
07661472), ('40', -0.8896548360329873), ('12', -1.3426266178223056), ('192
8', -1.3870624619430134), ('10', -1.6238857678880338), ('11', -1.64565506189
41097), ('20', -1.7275498339443063), ('510', -1.895987251183887), ('1199', -
2.5014903483341366), ('100', -3.0434084506771555)]
N-gram model
[('010 grade', 1.3013849907268018), ('10 movie', 1.1794210903601388), ('13 b
uck', 0.77690648844648), ('13 megapixels', 0.7675158561189311), ('10 10', 0.
7572655495461976), ('10 feet', 0.7503144716139528), ('18 months', 0.66143187
8127891), ('10 star', 0.6505509661249359), ('15lb piece', 0.614520343653597
3), ('10 years', 0.582825359744074), ('2005 buy', 0.5660864676106206), ('194
7 masterpiece', 0.559542703734381), ('100 time', 0.5579954205973328), ('100
recommend', 0.5491969764510868), ('20 leave', 0.5290284385235178), ('12 meg
a', 0.5268455578653997), ('110 set', 0.5268455578653983), ('1199 sandwich',
0.5160537264320827), ('2007 every', 0.5071702287164133), ('1998 deep', 0.492
3223429890355), ('1995 monster', 0.4816989756845576), ('17 burger', 0.414910
452871482), ('2005 begin', 0.3923112604773328), ('10 oyvey', 0.3589625186873
818), ('12 minutes', 0.3554630701590764), ('10 minutes', 0.333238969443557
1), ('12 mile', 0.3298700880487148), ('10 plus', 0.26129384920178117), ('20
feet', 0.1496809144052962), ('20 minutes', 0.10578232783119595), ('1979 firs
t', 0.0881900847306004), ('15 minutes', 0.07178901948497245), ('12 ridiculou
s', 0.01947313249602486), ('10 simply', -0.3763591019550324), ('20th centur
y', -0.4475803549167901), ('20th 2005', -0.4475803549168037), ('20the cove
r', -0.4576153367190696), ('20 30', -0.4791193812310562), ('20 ca', -0.48169
897568455783), ('1980 experience', -0.49483311440377414), ('20 years', -0.51
42150622026452), ('12 hours', -0.5160537264320831), ('11 months', -0.5268455
578653983), ('110 scale', -0.526845557865399), ('18th century', -0.540764780
8736893), ('15 second', -0.6152671778127168), ('1948 quite', -0.892006270050
6257), ('10 save', -1.0109232834180493), ('10 time', -1.0195726668350913),
('10 110', -1.6763894351624153)]
```

In [39]:

```
# Dimension=100
train_x_100 = np.dot(train_x_bow_normal, u[:,:100])
test_x_100 = np.dot(test_x_bow_normal, u[:,:100])
print ("PCA dim 100 for Bag of Word Model")
lr_score, gaussian_nb_score, b_nb_score, lr_coeff = sentiment_prediction(train_x_100, y_tra

train_x_100_ng = np.dot(train_x_2gram_normal, u2[:,:100])
test_x_100_ng = np.dot(test_x_2gram_normal, u2[:,:100])
print ("PCA dim 100 for Ngram Model")
lr_score_ng, gaussian_nb_score_ng, b_nb_score_ng, lr_coeff_ng = sentiment_prediction(train_
```

```
PCA dim 100 for Bag of Word Model
Logistic regression accuracy: 0.7283333333333334
Confussion Matrix:
 [[243  78]
 [ 85 194]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.6316666666666667
Confussion Matrix:
 [[239  82]
 [139 140]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.6266666666666667
Confussion Matrix:
 [[152 169]
 [ 55 224]]
PCA dim 100 for Ngram Model
Logistic regression accuracy: 0.515
Confussion Matrix:
 [[ 35 286]
 [  5 274]]
Accuracy of Naive Bayes Classifier with Bernoulli prior: 0.55
Confussion Matrix:
 [[215 106]
 [164 115]]
Accuracy of Naive Bayes Classifier with Gaussian prior: 0.5733333333333334
Confussion Matrix:
 [[307  14]
 [242  37]]
```

In [40]:

```
print ("Bag of Words")
lr_coef_val = lr_coeff.tolist()[0]
weight_vector = dict(zip(vocabulary_bow, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)

print ("N-gram model")
lr_coef_val = lr_coeff_ng.tolist()[0]
weight_vector = dict(zip(vocabulary_ng, lr_coef_val))
weight_vector = sorted(weight_vector.items(), key = lambda x: x[1], reverse = True)
print (weight_vector)
```

```
Bag of Words
[('010', 2.9834418998057752), ('2007', 2.167599341128274), ('absolutely', 1.
8732291568286217), ('20th', 1.7393013703611717), ('1010', 1.717023232621198
3), ('15lb', 1.5479557435823645), ('110', 1.518133239512729), ('40min', 1.39
58184690452062), ('accessoryone', 1.2059869310839324), ('accident', 1.154519
796613082), ('2000', 1.1145959270282133), ('5320', 0.973337337236965), ('34t
hs', 0.9682212674411931), ('45', 0.9598660273249721), ('350', 0.950987404851
4296), ('42', 0.8718156847168762), ('70s', 0.8440325971574535), ('5020', 0.7
878417263875597), ('910', 0.7749260965204591), ('1979', 0.7439842558805747),
('accuse', 0.7074871823765152), ('accurate', 0.6380581179634662), ('2mp', 0.
5744384637057823), ('24', 0.519366714077008), ('3o', 0.4990231724183387),
('accolades', 0.49696301364022627), ('2006', 0.4856098465279725), ('accessib
le', 0.426715095157759), ('absolute', 0.4187971577467303), ('70000', 0.40546
563909125805), ('ache', 0.3940249310070008), ('accessable', 0.36584498715124
7), ('325', 0.3538826301230212), ('accent', 0.3404664863588703), ('80', 0.33
01218012394722), ('achievement', 0.3111013745755125), ('accessory', 0.306686
2012135552), ('1995', 0.2948191899869568), ('accomodate', 0.278264312576600
7), ('accountant', 0.25995555710947144), ('2005', 0.23346842352134967), ('19
47', 0.20170302984551328), ('20the', 0.17865901123008066), ('abroad', 0.0895
7494959354406), ('70', 0.07247535794685991), ('25', 0.05865042082135328),
('accordingly', 0.03872375531613293), ('accept', 0.028423210878726794), ('ab
ysmal', 0.010857051387616976), ('785', -0.052647237185990306), ('1948', -0.0
6293852692257758), ('23', -0.10115573003758613), ('absolutel', -0.1019929503
077234), ('1998', -0.10674042439710427), ('ability', -0.12458132911227182),
('8pm', -0.13272346298730736), ('30', -0.13821373748533808), ('accidentall
y', -0.152183526039955), ('abstruse', -0.19576147504367075), ('810', -0.2059
736112394467), ('abhor', -0.21497755516137548), ('absolutley', -0.2183582634
113116), ('acclaim', -0.227573664919918), ('ackerman', -0.2298946839894156
8), ('access', -0.2729484985376612), ('academy', -0.2784279675264577), ('198
0', -0.3711787878545898), ('accommodations', -0.39988892163910034), ('2160',
-0.4075436307782216), ('8530', -0.5027651291972507), ('815pm', -0.5997631306
850566), ('acknowledge', -0.6301803310291424), ('17', -0.6490294654717448),
('ac', -0.6626448532191959), ('acceptable', -0.6675920052525571), ('18th', -
0.710867249644412), ('8125', -0.7444811165702646), ('5year', -0.745646011206
6449), ('15', -0.7462470994891878), ('18', -0.768565034177833), ('act', -0.7
909565784925332), ('abandon', -0.7976889606773196), ('90', -0.81370219872854
89), ('35', -0.8675332453307769), ('accord', -0.880269347565283), ('13', -0.
8994893794620034), ('40', -0.9053392169134851), ('750', -1.017177392371124
7), ('aailiyah', -1.1623520635328606), ('744', -1.3084119713255833), ('12',
-1.318211496006284), ('1928', -1.3704361062738248), ('10', -1.623572945244
9), ('11', -1.6479620383505782), ('20', -1.7168706234820474), ('95', -1.7921
449723196794), ('510', -1.8471745105937192), ('able', -1.9606004979449747),
('1199', -2.4797341911931294), ('100', -3.0651981132352843)]
N-gram model
[('010 grade', 1.304923397790976), ('10 movie', 1.1810600765349242), ('13 bu
ck', 0.7750674182541492), ('13 megapixels', 0.7659648554782152), ('10 10',
0.756031603169735), ('10 feet', 0.7522102359727986), ('18 months', 0.6605010
```

```
950238082), ('10 star', 0.6513963396767805), ('785 hot', 0.630981385385321
2), ('15lb piece', 0.6134691278575708), ('10 years', 0.5835707467865591),
('2005 buy', 0.5681846042516381), ('1947 masterpiece', 0.55864747910991),
('100 time', 0.556784146756771), ('100 recommend', 0.5500111410333852), ('20
leave', 0.5305742250289868), ('40min pass', 0.5300334502474656), ('12 mega',
0.5257310942520028), ('110 set', 0.5257310942520018), ('70 claim', 0.5185444
282460603), ('1199 sandwich', 0.5167673095347537), ('2007 every', 0.50887779
11080635), ('1998 deep', 0.4914969267478018), ('1995 monster', 0.48271150104
471383), ('24 hours', 0.46122345894767336), ('2160 tracfone', 0.449001783717
4799), ('30 40', 0.43217829024610677), ('ability pull', 0.420323667737197),
('17 burger', 0.41806552315392126), ('35 big', 0.4120664950007633), ('40 yea
rs', 0.39515153103769673), ('2005 begin', 0.39362587727639303), ('45 minute
s', 0.3927843302360884), ('95 garbage', 0.37416785906103317), ('510 maintai
n', 0.3705206690437831), ('10 oyvey', 0.36098642609070597), ('12 minutes',
0.3571136381441496), ('10 minutes', 0.3321442823056211), ('12 mile', 0.32872
32734824405), ('5020 pretty', 0.2736509425309856), ('34ths gristle', 0.26893
46886302015), ('10 plus', 0.2575512705786513), ('80 wonderful', 0.2175717355
009501), ('23 decent', 0.15404135611751438), ('20 feet', 0.1499833000610818
5), ('42 optimal', 0.13806460102768317), ('8pm start', 0.12452868965390543),
('20 minutes', 0.10319597866512915), ('1979 first', 0.08918116038106631),
('aailiyah pretty', 0.08872164014963929), ('15 minutes', 0.0697035690393703
5), ('ability phone', 0.0549134746956784), ('8530 blackberry', 0.01962649706
731371), ('12 ridiculous', 0.019459376044383733), ('80 flick', -0.0264516430
85599344), ('90 minutes', -0.10523431916619405), ('5year work', -0.115190728
19568894), ('70s grainy', -0.12017680355345685), ('ability dwight', -0.16601
250558286548), ('810 score', -0.20688021510086174), ('25 years', -0.21804223
257655578), ('abandon factory', -0.23070394431372948), ('30 min', -0.2350551
6506822274), ('5year old', -0.2470357162867931), ('90 food', -0.263234410024
7313), ('ability meld', -0.30619159803811885), ('40 handle', -0.315548525631
3202), ('45 minutesmajor', -0.32935718450913853), ('90 child', -0.3348375250
5852965), ('510 right', -0.3652732725003369), ('325 cellphone', -0.374558263
00582157), ('10 simply', -0.376382503011259), ('510 complaints', -0.38148731
59467417), ('350 headset', -0.4072081553838106), ('2mp pics', -0.42429974594
93747), ('40 minutes', -0.4259350665092096), ('20th century', -0.44900178371
74656), ('20th 2005', -0.44900178371747873), ('90 mainly', -0.45091217822719
76), ('20the cover', -0.4572941441226327), ('70000 time', -0.457785392149405
45), ('3o minutes', -0.45785427577875154), ('20 30', -0.47826689250644705),
('20 ca', -0.48271150104471405), ('30 minutes', -0.4863926632997367), ('1980
experience', -0.4958229540628439), ('20 years', -0.5158034144657175), ('12 h
ours', -0.5167673095347538), ('11 months', -0.5257310942520017), ('110 scal
e', -0.5257310942520024), ('18th century', -0.5418896032011234), ('15 secon
d', -0.6163893659656858), ('815pm fast', -0.6202990146821818), ('750 see', -
0.6369543208134447), ('744 plug', -0.8251923564996154), ('1948 quite', -0.88
89339310507927), ('10 save', -1.0076313975904052), ('10 time', -1.0209959187
181086), ('23 bar', -1.115423865765869), ('10 110', -1.6726809518236487)]
```

In [41]:

```
#i : Algorithm review
"""1) Bag of words using NBC with Burnoulli performs best. It might be because bag of words
```

Out[41]:

```
'1) Bag of words using NBC with Burnoulli performs best. It might be because
bag of words reserved all features of words and single word could represent
features better.'
```

In [42]:

```
"""2) The reason that PCA did not work well for this dataset might be because:
The word features are relatively evenly distributed in all features, which means the direct
Originally the dataset has around 2000 features. Reducing them to ~100 features reduced too
```

Out[42]:

```
'2) The reason that PCA did not work well for this dataset might be becaus
e:\nThe word features are relatively evenly distributed in all features, whi
ch means the directions with highest variance cannot represent most informat
ion of the original dataset. So reducing dimensions will lose considerable p
art of original information.\nOriginally the dataset has around 2000 feature
s. Reducing them to ~100 features reduced too much information.'
```

In [43]:

```
"""3) We see that Bag of Words always perform better than N-gram model, for all cases, both
```

Out[43]:

```
'3) We see that Bag of Words always perform better than N-gram model, for al
l cases, both with and without PCA.'
```

In [ ]:

In [ ]:

```
"""Question 2"""
```

In [5]:

```
import numpy as np
import pandas as pd
import random
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import seaborn as sns
import os
```

In [19]:

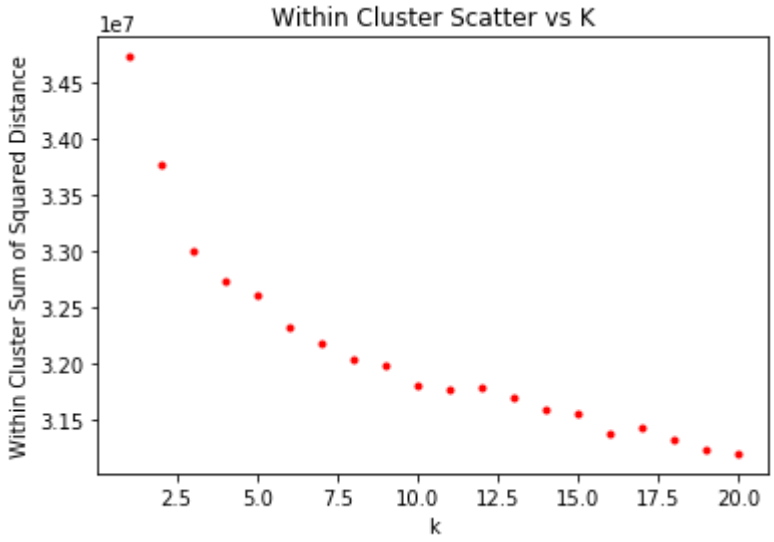
```
# Download all documents
document = np.load("F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/Clustering/
vocabulary = pd.read_table('F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/Clu
titles = pd.read_table('F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/Cluster
document.shape
```

Out[19]:

```
(1373, 5476)
```

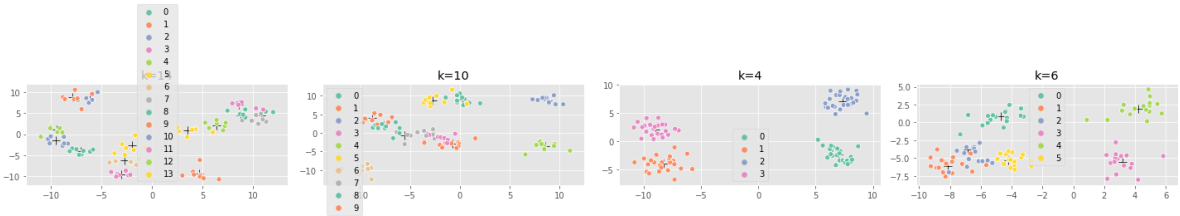
In [3]:

```
#a : Cluster the documents using k-means and various values of k
def cluster_k(data,max_k):
    sum_of_sq_dist = []
    for k in range(1,max_k+1):
        kmeans = KMeans(n_clusters=k, random_state=0).fit(data)
        sum_of_sq_dist.append(kmeans.inertia_)
    plt.plot(range(1,max_k+1), sum_of_sq_dist, 'r.')
    plt.xlabel('k')
    plt.ylabel('Within Cluster Sum of Squared Distance')
    plt.title('Within Cluster Scatter vs K')
    plt.show()

cluster_k(document,20)
```



In [6]:

```
# plot for various values of k
def mean_plots(data):
    plt.style.use('ggplot')
    x=data
    plt.figure(figsize=(20, 3))
    for k in range(4):
        K = random.randint(1,20)
        x, y = make_blobs(centers=K)
        kmeans = KMeans(n_clusters=K, random_state=0).fit(x)
        plt.subplot(1,4,k+1)
        sns.scatterplot(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], marker=
        sns.scatterplot(x[:,0], x[:,1], hue=y, palette=sns.color_palette("Set2", n_colors=K
        plt.title('k={}'.format(K))
    plt.tight_layout()
    plt.show()

mean_plots(document)
```



In [7]:

```
"""We see k=4, gives very reasonable output, with distinct seperation. Hence, the k we sele
```

Out[7]:

```
'We see k=4, gives very reasonable output, with distinct seperation. Hence,
the k we select is 4'
```

In [9]:

```
# calculate how many documnents fall in each bins
kmeans_bin = KMeans(n_clusters=4, random_state=0).fit(document)
np.bincount(kmeans_bin.labels_)
```

Out[9]:

```
array([371, 230, 559, 213], dtype=int64)
```

In [16]:

```
# Return the top 10 features
center = np.mean(document, axis=0)
for i in range(4):
    largest = np.argsort(kmeans_bin.cluster_centers_[i] - center)[::-1][:10]
    print( i+1,vocabulary[largest].reshape(1,-1), "\n")
```

```
1 [['protein' 'cell' 'cells' 'expression' 'proteins' 'fig' 'gene'
  'expressed' 'binding' 'specific']]

2 [['fig' 'values' 'period' 'mean' 'lower' 'average' 'estimates' 'range'
  'estimate' 'indicate']]

3 [['says' 'mail' 'researchers' 'scientists' 'years' 'news' 'people'
  'issue' 'year' 'world']]

4 [['energy' 'electron' 'fig' 'density' 'temperature' 'shows' 'structure'
  'measured' 'constant' 'measurements']]
```

In [17]:

```
representing Biology, the forth representing Atomic Chemistry or Physics. The clustering is
```

Out[17]:

```
'We see a distinctive pattern, the first group representing Biology, the for
th representing Atomic Chemistry or Physics. The clustering is bringing out
some unique connection.'
```

In [20]:

```
# Return the top 10 closest documents to each centroid
for i in range(4):
    d = kmeans_bin.transform(document)[:, i]
    ind = np.argsort(d)[:10]
    print(i+1, titles[ind], "\n" )
```

```
1 [['Requirement of NAD and SIR2 for Life-Span Extension by Calorie Restrict
ion 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']
 ['Reopening the Darkest Chapter in German Science']
 ['Similar Requirements of a Plant Symbiont and a Mammalian Pathogen for Pro
longed Intracellular Survival']
 ['Mothers Setting Boundaries']
 ['Turning up the Heat on Histoplasma capsulatum']
 ['Distinct Classes of Yeast Promoters Revealed by Differential TAF Recruitm
ent']
 ['An Arresting Start for MAPK']]

2 [['Algorithmic Gladiators Vie for Digital Glory']
 ['Reopening the Darkest Chapter in German Science']
 ['Population Dynamical Consequences of Climate Change for a Small Temperate
Songbird']
 ['The Formation of Chondrules at High Gas Pressures in the Solar Nebula']
 ['Subducted Seamount Imaged in the Rupture Zone of the 1946 Nankaido Earthq
uake']
 ['Homogenization of Fish Faunas across the United States']
 ['Tectonic Implications of U-Pb Zircon Ages of the Himalayan Orogenic Belt
in Nepal']
 ['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']]

3 [['Algorithmic Gladiators Vie for Digital Glory']
 ['Reopening the Darkest Chapter in German Science']
 ['Information Technology Takes a Different Tack']
 ['National Academy of Sciences Elects New Members']
 ['Archaeology in the Holy Land']
 ['Heretical Idea Faces Its Sternest Test']
 ['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']
 ['Divining Diet and Disease from DNA']]

4 [['The Formation of Chondrules at High Gas Pressures in the Solar Nebula']
 ['Algorithmic Gladiators Vie for Digital Glory']
 ['Thermal, Catalytic, Regiospecific Functionalization of Alkanes']
 ['Reopening the Darkest Chapter in German Science']
 ['Heretical Idea Faces Its Sternest Test']
 ['Information Storage and Retrieval through Quantum Phase']
 ['Synthesis and Characterization of Helical Multi-Shell Gold Nanowires']
 ['A Monoclinic Post-Stishovite Polymorph of Silica in the Shergotty Meteori
te']
```

```
['Quantum Dots as Tunable Kondo Impurities']
['Ambipolar Pentacene Field-Effect Transistors and Inverters']]
```

In [21]:

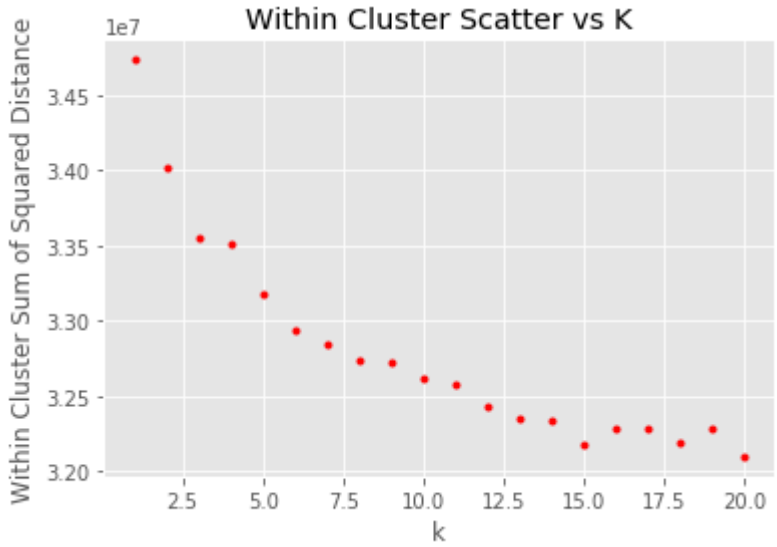
```
here is a pattern but not very distinct as features. We still see a little Biology clusterin
```

Out[21]:

```
'The clustering shown by documents are not very good! It shows multiple repi
tition for same title. There is a pattern but not very distinct as features.
We still see a little Biology clustering in 1 and Physics clustering in 4.'
```

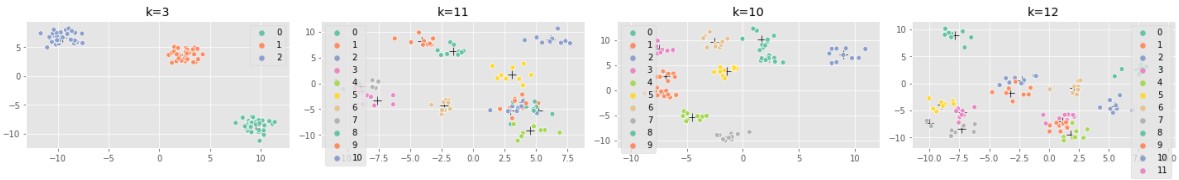
In [23]:

```
#b : Repeat analysis for term-wise matrix
term = np.load("F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/Clustering/scie
cluster_k(term, 20)
```



In [26]:

```
# Separation of clusters
mean_plots(term)
```



In [28]:

```
"""Plot with k=3 looks the best for me. k=10 also looks good but intercluster distance is n
```

Out[28]:

```
'Plot with k=3 looks the best for me. k=10 also looks good but intercluster
distance is not so good. Hence, we go with k=3'
```

In [34]:

```
# Dividing into bins
kmeans_bin2 = KMeans(n_clusters=3, random_state=0).fit(term_mat)
np.bincount(kmeans_bin2.labels_)
```

Out[34]:

```
array([ 523, 4275,  678], dtype=int64)
```

In [35]:

```
# Top features
center2 = np.mean(term, axis=0)
for i in range(3):
    largest = np.argsort(kmeans_bin2.cluster_centers_[i] - center2)[::-1][:10]
    print( i+1,vocabulary[largest], "\n")
```

```
1 [['adult']
 ['spectrum']
 ['national']
 ['suggested']
 ['mars']
 ['ratios']
 ['provide']
 ['century']
 ['mass']
 ['extended']]

2 [['mediated']
 ['conserved']
 ['width']
 ['strongly']
 ['past']
 ['environmental']
 ['interactions']
 ['separated']
 ['atom']
 ['contact']]

3 [['question']
 ['frequency']
 ['people']
 ['high']
 ['coding']
 ['new']
 ['bar']
 ['self']
 ['knowledge']
 ['quantum']]
```

In [36]:

```
, but with only top 10 words it is difficult to predict the groups. Hence, we need more word
```

Out[36]:

```
'The term seems to be clustering, but with only top 10 words it is difficult
to predict the groups. Hence, we need more words to deduce the exact field o
f study.'
```

In [32]:

```
# Top titles
center2 = np.mean(term, axis=0)
for i in range(3):
    largest = np.argsort(kmeans_bin2.cluster_centers_[i] - center2)[::-1][:10]
    print( i+1,titles[largest], "\n")
```

```
1 [['Noxa, a BH3-Only Member of the Bcl-2 Family and Candidate Mediator of p
53-Induced Apoptosis']
 ['Positional Syntenic Cloning and Functional Characterization of the Mammal
ian Circadian Mutation tau']
 ['Central Role for G Protein-Coupled Phosphoinositide 3-Kinase g in Inflamm
ation']
 ['Kinesin Superfamily Motor Protein KIF17 and mLin-10 in NMDA Receptor-Cont
aining Vesicle Transport']
 ['Regulated Cleavage of a Contact-Mediated Axon Repellent']
 ['Role of the Mouse ank Gene in Control of Tissue Calcification and Arthrit
is']
 ['An Oral Vaccine against NMDAR1 with Efficacy in Experimental Stroke and E
pilepsy']
 ['Requirement of JNK for Stress-Induced Activation of the Cytochrome c-Medi
ated Death Pathway']
 ['Function of PI3Kg in Thymocyte Development, T Cell Activation, and Neutro
phil Migration']
 ['Regulation of STAT3 by Direct Binding to the Rac1 GTPase']]

2 [['National Academy of Sciences Elects New Members']
 ['NIH, under Pressure, Boosts Minority Health Research']
 ['Science Survives in Breakthrough States']
 ['Ground Zero: AIDS Research in Africa']
 ['Sharp Jump in Teaching Fellows Draws Fire from Educators']
 ['Africa Boosts AIDS Vaccine R&D']
 ['A New Breed of Scientist-Advocate Emerges']
 ['Flushing out Nasty Viruses in the Balkans']
 ["Stephen Straus's Impossible Job"]
 ['Bastions of Tradition Adapt to Alternative Medicine']]

3 [['NEAR at Eros: Imaging and Spectral Results']
 ['The Atom-Cavity Microscope: Single Atoms Bound in Orbit by Single Photon
s']
 ['Advances in the Physics of High-Temperature Superconductivity']
 ['The Formation and Early Evolution of the Milky Way Galaxy']
 ['Subduction and Slab Detachment in the Mediterranean-Carpathian Region']
 ['The Galactic Center: An Interacting System of Unusual Sources']
 ["Earth's Core and the Geodynamo"]
 ['Quantum Criticality: Competing Ground States in Low Dimensions']
 ["Sediments at the Top of Earth's Core"]
 ['Internal Structure and Early Thermal Evolution of Mars from Mars Global S
urveyor Topography and Gravity']]
```

In [37]:

```
s to be Biological Pathway, the second seems to be major break-throughs and third seems a fi
```

Out[37]:

```
'The grouping of the title seems better than the previous. We can understand
the groups and fields better. For example, the first one seems to be Biologi
cal Pathway, the second seems to be major break-throughs and third seems a f
ield with Geography and Physics in it.'
```

In [ ]:

In [17]:

```
"""Question 3"""
```

Out[17]:

```
'Question 3'
```

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.mixture import GaussianMixture
from sklearn import cluster
import collections
import operator
import matplotlib.pylab as plt
import os
import sys
import warnings
```

In [55]:

```
#a: Kmeans is a special case of EM Algorithm
```

In [56]:

```
%%latex
K-mean is a special case of EM algorithm.
It sets the probability of hidden clusters given input $y_i$ and model parameter $\theta$,
```

K-mean is a special case of EM algorithm. It sets the probability of hidden clusters given input y_i and model parameter θ, i.e., responsibilities γ_i^z as binary 0 and 1.

In [54]:

```
%%latex
1. E-step: Compute responsibilities $\gamma_{i}^{z}$ to determnine whether data $i$ belong
$$
\gamma_{i}^{z} =
\begin{cases}
    1, & \quad \text{when } z=\underset{z}{argmin} \parallel y_i-\mu_z \parallel \\
    0, & \quad \text{other}
\end{cases}
$$
```

1. E-step: Compute responsibilities γ_i^z to determnine whether data i belong to cluster z or not.

$$\gamma_i^z = \begin{cases} 1, & \text{when } z = \underset{z}{argmin} \parallel y_i - \mu_z \parallel \\ 0, & \text{other} \end{cases}$$

In [53]:

```
%%latex
2. M-step: Update new cluster centres based on responsibilities.
$$
\mu^{z}=\frac{\sum_{i=1}^{N}{\gamma_{i}^{z}y_i}}{\sum_{i=1}^{N}{\gamma_{i}^{z}}}
$$

3. Iterate E-step and M-step until $\gamma_{i}^{z}$ does not change or reaches maximum iter
```

2. M-step: Update new cluster centres based on responsibilities.

$$\mu^z = \frac{\sum_{i=1}^{N} \gamma_i^z y_i}{\sum_{i=1}^{N} \gamma_i^z}$$

3. Iterate E-step and M-step until γ_i^z does not change or reaches maximum iteration.

In [3]:

```
#b: Import data
data = pd.read_csv(r'F:/Annie/CornellMS/Semester 4/Machine Learning/Homework/HW3/EMM/faithf
data = data.drop(['id'], axis =1)
print (data.shape, "\n", data.head())
```

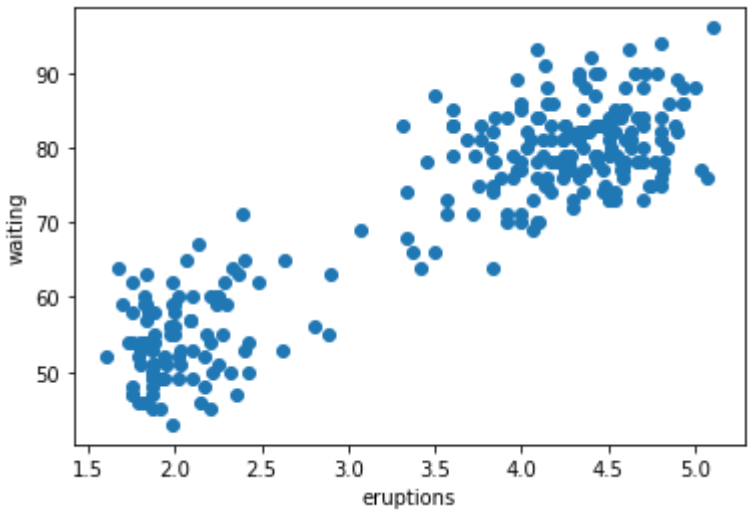
```
(272, 2)
    eruptions  waiting
0       3.600       79
1       1.800       54
2       3.333       74
3       2.283       62
4       4.533       85
```

In [4]:

```
# Parse and plot all the data in 2D plane
plt.scatter(data['eruptions'], data['waiting'])
plt.xlabel('eruptions')
plt.ylabel('waiting')
```

Out[4]:

```
Text(0, 0.5, 'waiting')
```



In [5]:

```
#c: Fitting Gaussian-bimodal distribution
def Gaussian(data):
    clf = GaussianMixture(n_components=2, covariance_type='spherical', init_params='random'
    clf.fit(data)
    print ("No of steps needed for convergence is ", clf.n_iter_)
    return clf.n_iter_
```

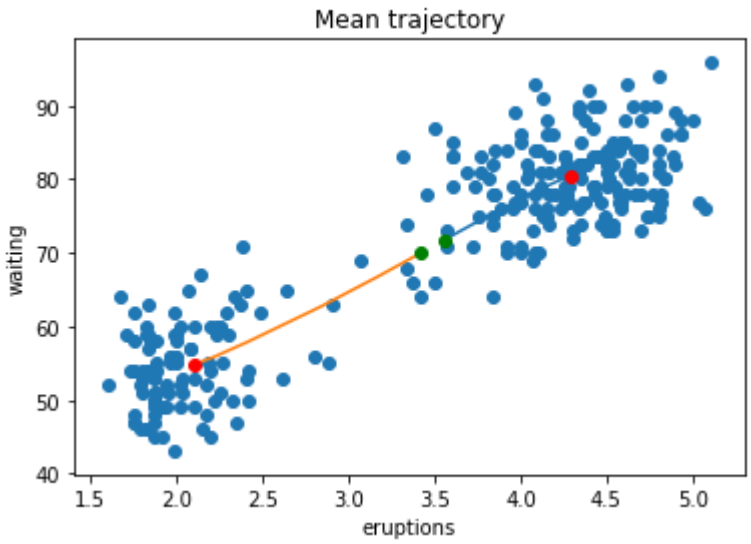
In [13]:

```
# Plot trajectories of two mean vectors in 2-dimensions
if not sys.warnoptions:
    warnings.simplefilter("ignore")

clf = GaussianMixture(n_components=2, covariance_type='spherical', init_params='random', wa
clf.fit(data)

means1 = clf.means_[0]
means2 = clf.means_[1]

while (not clf.converged_):
    means1 = np.append(means1, clf.means_[0])
    means2 = np.append(means2, clf.means_[1])
    clf.fit(data)
length = len(means2)
print (length)
print (means1, "\n", means2)
```

```
22
[ 3.553681   71.75474281  3.553681   71.75474281  3.63371201 72.78955978
  3.80122152 74.93280261  4.07148465 78.23883498  4.27797922 80.42992638
  4.3215886  80.75738068  4.31255202 80.53724865  4.30357977 80.38438983
  4.29825768 80.31355451  4.29569169 80.2841132 ]
 [ 3.42051106 70.02149023  3.42051106 70.02149023  3.33868125 68.96340945
  3.16427617 66.73167602  2.8390941  62.73787319  2.41323538 57.93379497
  2.22549049 55.9695833   2.15427524 55.31054945  2.11836474 54.97136858
  2.1048929  54.82995479  2.10031448 54.77612238]
```

In [14]:

```
# Plot the means
means1 = means1.reshape((-1,2))
means2 = means2.reshape((-1,2))
means1 = pd.DataFrame({'x': means1[:,0], 'y': means1[:,1]})
means2 = pd.DataFrame({'x': means2[:,0], 'y': means2[:,1]})

plt.plot(means1['x'], means1['y'], label = 'mean1')
plt.plot(means2['x'], means2['y'], label = 'mean2')

plt.plot(means1['x'][:1], means1['y'][:1], 'o-',c='green', label="start")
plt.plot(means2['x'][:1], means2['y'][:1], 'o-',c='green')

plt.plot(means1['x'][-1:], means1['y'][-1:], 'o-',c='red', label="end")
plt.plot(means2['x'][-1:], means2['y'][-1:], 'o-',c='red')

plt.scatter(data['eruptions'], data['waiting'])
plt.xlabel('eruptions')
plt.ylabel('waiting')

plt.title('Mean trajectory')
plt.show()
```



In [15]:

```
ermination Criteria
The termination criteria was to continue iterating until the function converges (using conve
```

Out[15]:

```
'The termination criteria was to continue iterating until the function conve
rges (using converge_ function defined in scikit learn). I believed this wou
ld produce the greatest accuracy with a reasonable number of iterations. '
```

In [8]:

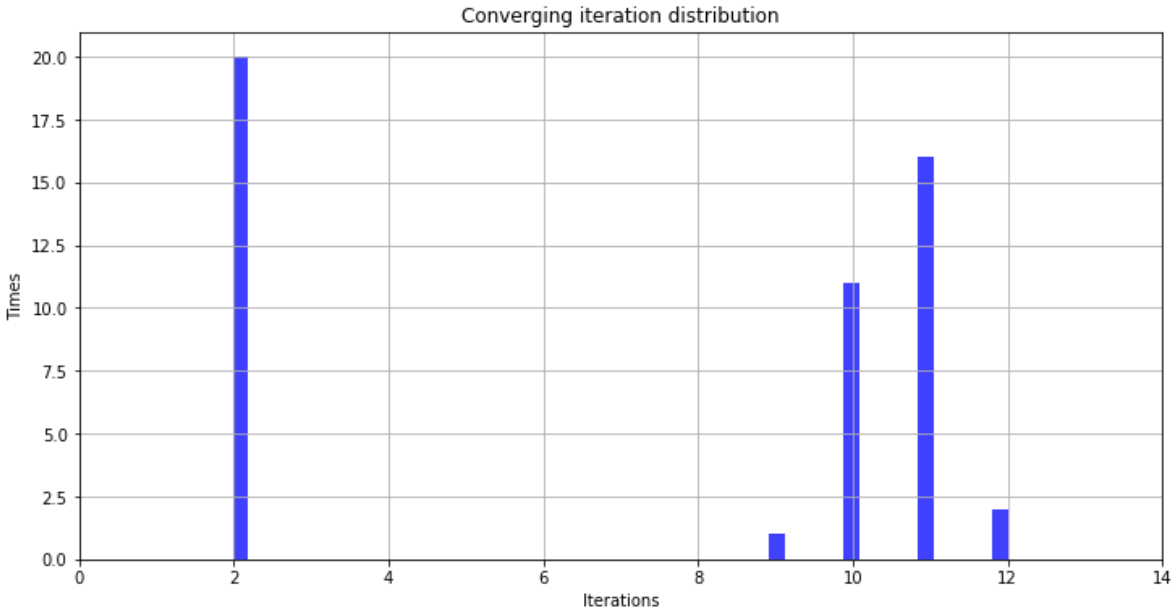
```
# Run GMM model for 50 times and plot the distribution
iterations = []
for i in range(50):
    step = Gaussian(data)
    iterations.append(step)
counts = collections.Counter(iterations)
max_count = sorted(counts.items(), key=operator.itemgetter(1))[-1][1]
max_step = sorted(counts.items(), key=operator.itemgetter(0))[-1][0]
```

```
No of steps needed for convergence is  2
No of steps needed for convergence is  10
No of steps needed for convergence is  11
No of steps needed for convergence is  11
No of steps needed for convergence is  10
No of steps needed for convergence is  2
No of steps needed for convergence is  2
No of steps needed for convergence is  2
No of steps needed for convergence is  2
No of steps needed for convergence is  11
No of steps needed for convergence is  12
No of steps needed for convergence is  2
No of steps needed for convergence is  10
No of steps needed for convergence is  9
No of steps needed for convergence is  11
No of steps needed for convergence is  2
No of steps needed for convergence is  10
No of steps needed for convergence is  11
No of steps needed for convergence is  11
No of steps needed for convergence is  11
No of steps needed for convergence is  11
No of steps needed for convergence is  11
No of steps needed for convergence is  11
No of steps needed for convergence is  2
No of steps needed for convergence is  2
No of steps needed for convergence is  10
No of steps needed for convergence is  10
No of steps needed for convergence is  2
No of steps needed for convergence is  2
No of steps needed for convergence is  10
No of steps needed for convergence is  2
No of steps needed for convergence is  2
No of steps needed for convergence is  2
No of steps needed for convergence is  12
No of steps needed for convergence is  11
No of steps needed for convergence is  10
No of steps needed for convergence is  2
No of steps needed for convergence is  2
No of steps needed for convergence is  11
No of steps needed for convergence is  10
No of steps needed for convergence is  2
No of steps needed for convergence is  11
No of steps needed for convergence is  2
No of steps needed for convergence is  11
No of steps needed for convergence is  2
No of steps needed for convergence is  11
No of steps needed for convergence is  2
No of steps needed for convergence is  10
```

```
No of steps needed for convergence is  11
No of steps needed for convergence is  10
```

In [9]:

```
# Plot bar graph for 50 run
fig, ax = plt.subplots()
fig.set_size_inches(12, 6)
n, bins, patches = plt.hist(iterations, len(iterations)+2, facecolor='b', alpha=0.75)
plt.xlabel('Iterations')
plt.ylabel('Times')
plt.title('Converging iteration distribution')
plt.axis([0, max_step+2, 0, max_count+1])
plt.grid(True)
plt.show()
```



In [10]:

```
# d: K-mean neighbor distribution
def Gaussian_kmean(data, data_mean, data_covar, data_label):
    clf = GaussianMixture(n_components=2, covariance_type='spherical', precisions_init=1/da
                          init_params='kmeans', weights_init=pd.Series(data_label).value_co
    clf.fit(data)
    print ("No of steps needed for convergence is ", clf.n_iter_)
    return clf.n_iter_
```

In [11]:

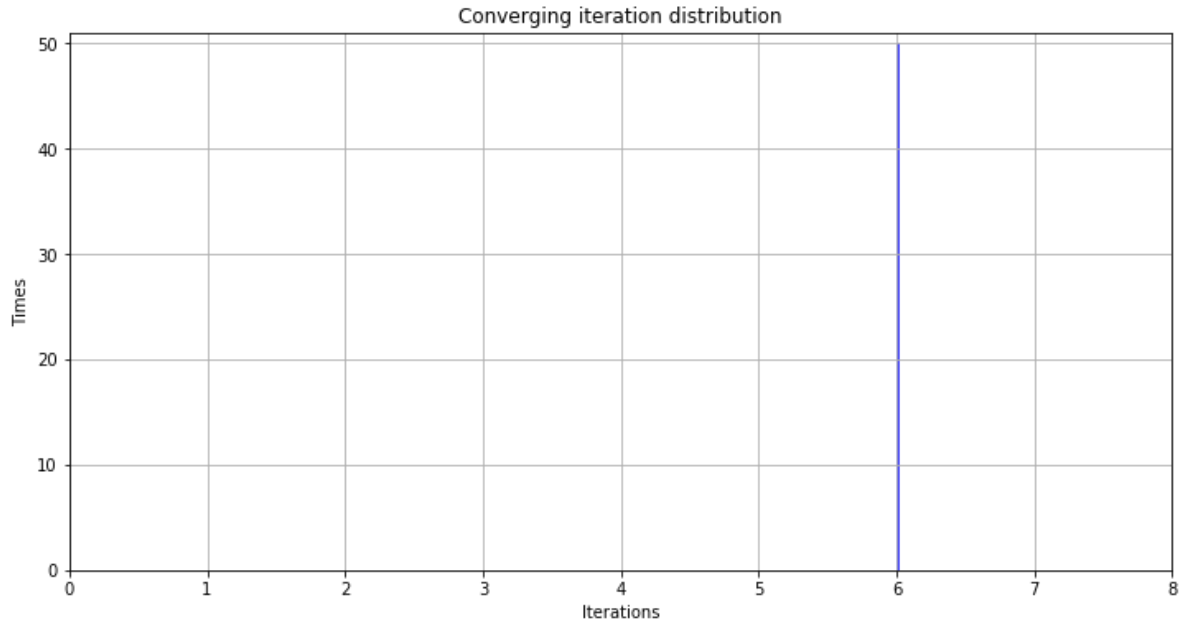
```
# Estimate the first guess of the mean and covariance matrices using maximum likelihood ove
clf = cluster.KMeans(n_clusters=2, init = 'random')
clf.fit(data)

clusters = clf.labels_
data1 = data[clusters == 0]
data2 = data[clusters == 1]
data1_mean = np.array(data1.mean(axis = 0))
data2_mean = np.array(data2.mean(axis = 0))
data_means = np.concatenate((data1_mean, data2_mean)).reshape(2,2)

data1_var = data1.var().mean()
data2_var = data2.var().mean()

data_covar = np.array([data1_var, data2_var])

gmm_kmeans_init = GaussianMixture(n_components=2, covariance_type = 'spherical', warm_start
                                  weights_init=pd.Series(clf.labels_).value_counts(normaliz
                                  precisions_init=1/data_covar).fit(data)
```

In [12]:

```
# Run GMM model for 50 times and plot the distribution
iterations = []
for i in range(50):
    step = Gaussian_kmean(data, data_means, data_covar, clf.labels_)
    iterations.append(step)
counts = collections.Counter(iterations)
max_count = sorted(counts.items(), key=operator.itemgetter(1))[-1][1]
max_step = sorted(counts.items(), key=operator.itemgetter(0))[-1][0]

fig, ax = plt.subplots()
fig.set_size_inches(12, 6)
n, bins, patches = plt.hist(iterations, len(iterations)+2, facecolor='b', alpha=0.75)
plt.xlabel('Iterations')
plt.ylabel('Times')
plt.title('Converging iteration distribution')
plt.axis([0, max_step+2, 0, max_count+1])
plt.grid(True)
plt.show()
```

```
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
```

```
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
No of steps needed for convergence is  6
```



In [16]:

```
(a) and (b)
ogram above, when k means is utilized to cluster, convergence is reached quicker. This is be
```

Out[16]:

```
'As shown by the histogram above, when k means is utilized to cluster, conve
rgence is reached quicker. This is because we are using kmeans to allocate c
luster centre as opposed to randomly assigning, which makes it better as the
dataset is seperable to begin with and that is identified by the algorithm.'
```

In [ ]:

In [4]:
```
from sklearn.manifold import MDS
from sklearn.cluster import KMeans
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import euclidean_distances
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from pyclustering.cluster.kmedoids import kmedoids
import numpy.linalg as la
import pylab
import scipy.cluster.hierarchy as shc
import warnings
import sys
if not sys.warnoptions:
    warnings.simplefilter("ignore")
import os
os.chdir("C:/Users/WZY77/Downloads")
```

In [6]:
```
data = np.load("./mds-population.npz")

data['D'].shape
```

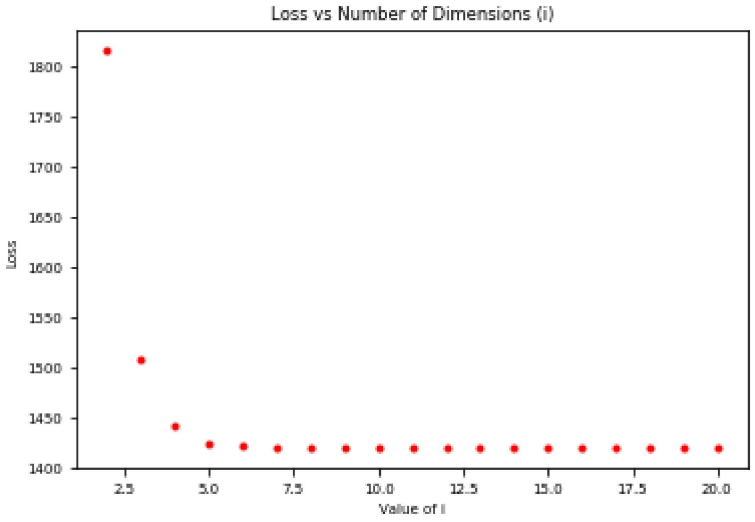
Out[6]:
```
(42, 42)
```

(a)

In [43]:
```
mds_data = MDS(n_components=2,dissimilarity='precomputed')
datanew = mds_data.fit_transform(data['D'])
datanew.shape
```

Out[43]:
```
(42, 2)
```

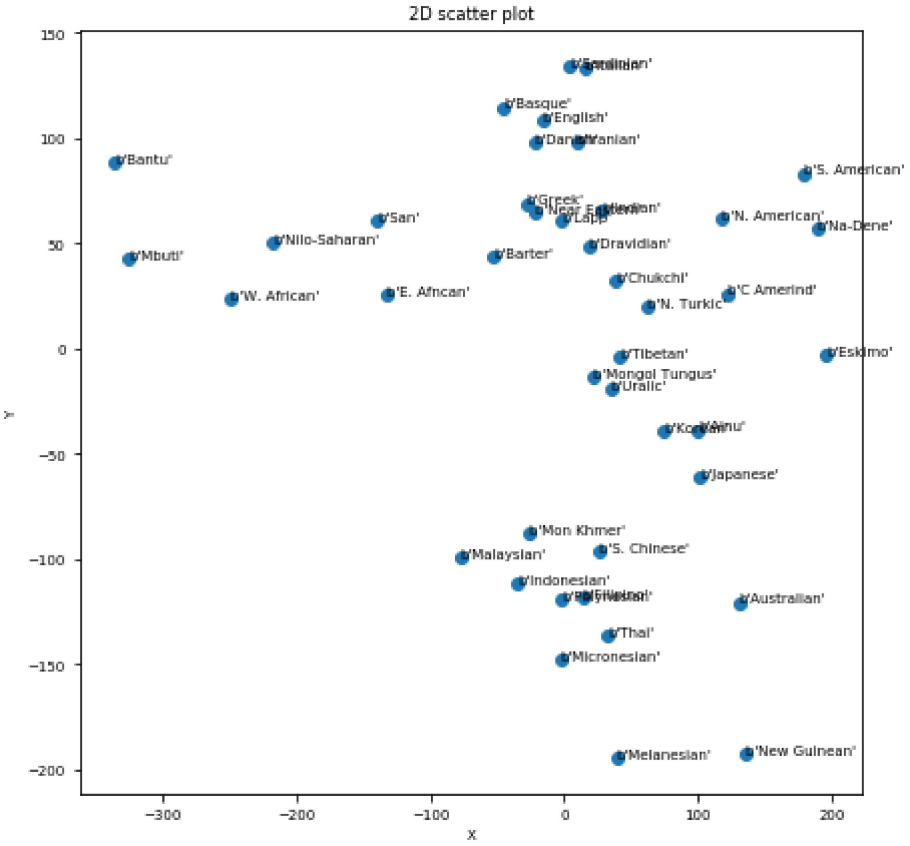
In [44]:
```
loss = []
for i in range (2, 21):
    mdsd = MDS(n_components=i,dissimilarity='precomputed')
    xnew = mdsd.fit_transform(data['D'])
    xeuc = euclidean_distances(xnew,xnew)
    xla = la.norm(data['D']-xeuc)
    loss.append(xla)

fig, ax = plt.subplots()

plt.xlabel('Value of i')
plt.ylabel('Loss')
plt.title('Loss vs Number of Dimensions (i)')
plt.plot(range(2,21),loss,'r.')
plt.show()
```



i. MDS is used for showing the dissimilarity in data by representing them as distances between points in a low dimensional space. It assumes that euclidean distance between pairs of points approximates the Nei's distance between populations. And it also assumes that each point of data is equally important. However, if the multidimensional data does not translate well into smaller dimensions, this method may fail. In order to determine the amount of information that is loss, we can use stress.

ii. According to the Loss-Number of i plot above, 5 dimensions seems to be enough to capture most of variation of data.

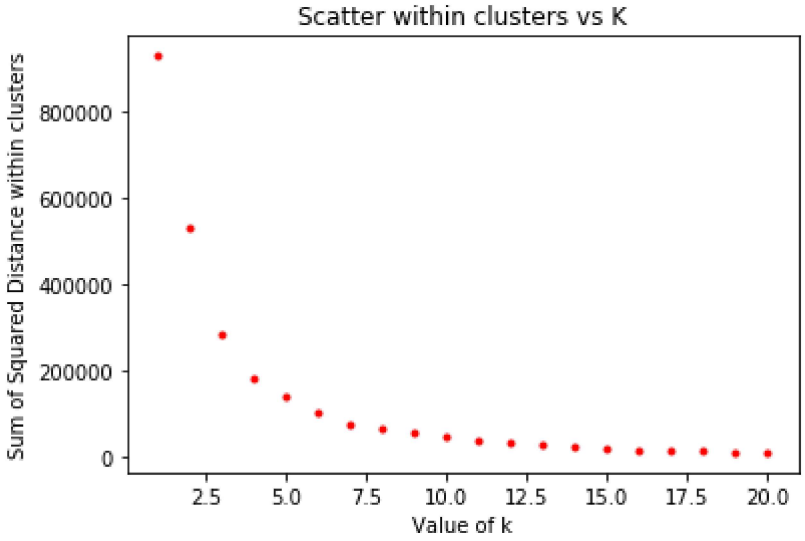
In [46]:
```
x = []
y = []
for t in datanew:
    x.append(t[0])
    y.append(t[1])

plt.rcParams.update({'font.size': 7})
plt.scatter(x, y)
for i, text in enumerate(data['population_list']):
    plt.annotate(text, (x[i], y[i]))
plt.xlabel('X')
plt.ylabel('Y')
plt.title('2D scatter plot')
plt.gcf().set_size_inches((7, 7))
plt.show()
```



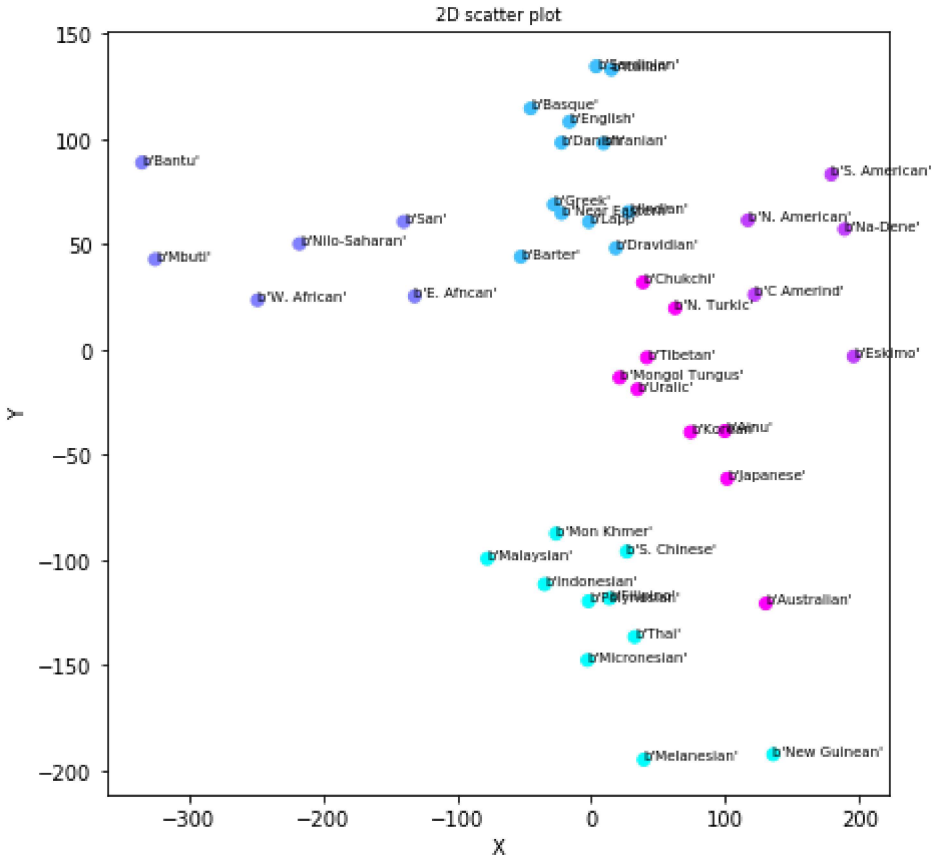
iii. Scatterplot shows above.

(b)

In [47]:
```
plt.rcParams.update({'font.size': 10})
def kmeans_test(data,k):
    ssd = []
    for k in range(1,k+1):
        kmeans = KMeans(n_clusters=k, random_state=0).fit(data)
        ssd.append(kmeans.inertia_)
    plt.plot(range(1,k+1), ssd, 'r.')
    plt.xlabel('Value of k')
    plt.ylabel('Sum of Squared Distance within clusters')
    plt.title('Scatter within clusters vs K')
    plt.show()

kmeans_test(datanew,20)
```



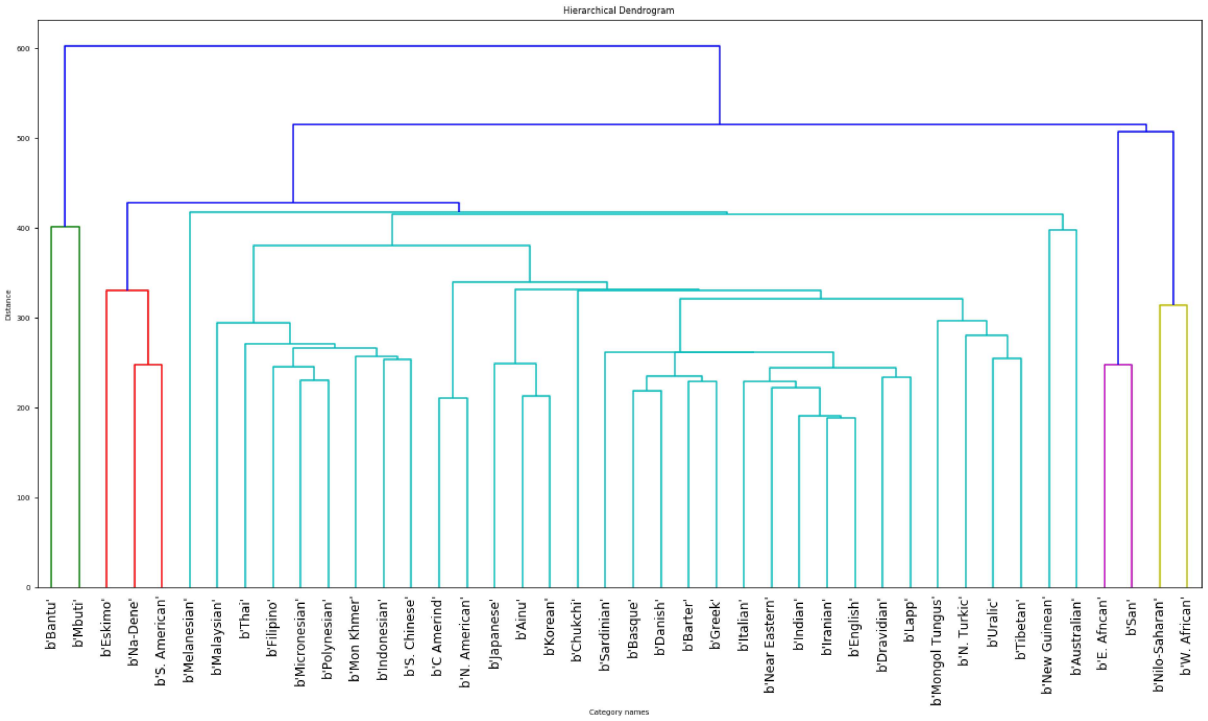
In [48]:
```
km = KMeans(n_clusters=5, random_state=0).fit(datanew)
plt.scatter(x,y, c=km.labels_, cmap=pylab.cm.cool)
plt.rcParams.update({'font.size': 7})
for i, text in enumerate(data['population_list']):
    plt.annotate(text, (x[i], y[i]))
plt.xlabel('X')
plt.ylabel('Y')
plt.title('2D scatter plot')
plt.gcf().set_size_inches((7, 7))
plt.show()
```



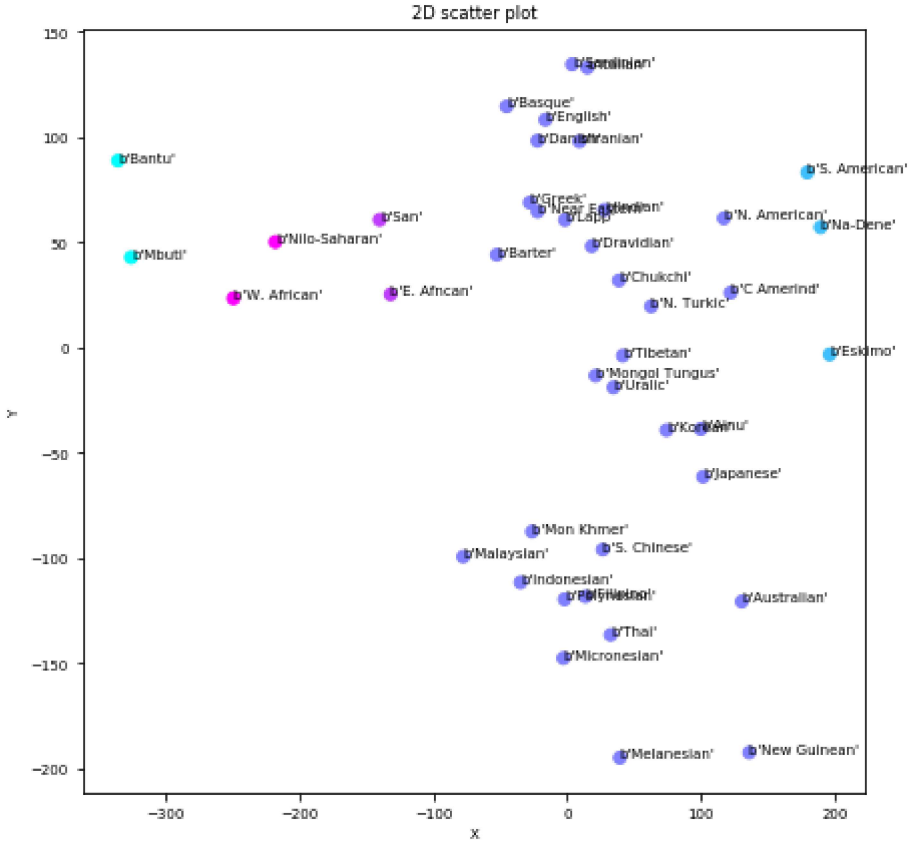
This result seems to be acceptable since it shows some reasonable clusters. But the features, which are distances in this case, inside each cluster have been lost during K-means.

(c)

In [49]:
```
dist = linkage(data['D'], 'single')
plt.figure(figsize=(20, 10))
plt.title('Hierarchical Dendrogram')
plt.xlabel('Category names')
plt.ylabel('Distance')
pop = data['population_list']

dendrogram(dist,leaf_rotation=90., leaf_font_size=12.,leaf_label_func=lambda v: st
plt.show()
```



In [50]:
```
cutoff = 420
clust = fcluster(dist, cutoff, criterion='distance')
plt.scatter(x,y, c=clust, cmap=pylab.cm.cool)
plt.rcParams.update({'font.size': 7})
for i, text in enumerate(data['population_list']):
    plt.annotate(text, (x[i], y[i]))
plt.xlabel('X')
plt.ylabel('Y')
plt.title('2D scatter plot')
plt.gcf().set_size_inches((7, 7))
plt.show()
```



We choose a cut-off that would make the same number of clusters as k-means, but the result of hierarchical clustering is not really acceptable since it makes a large cluster with 4 little clusters, which do not make intuitive sense.

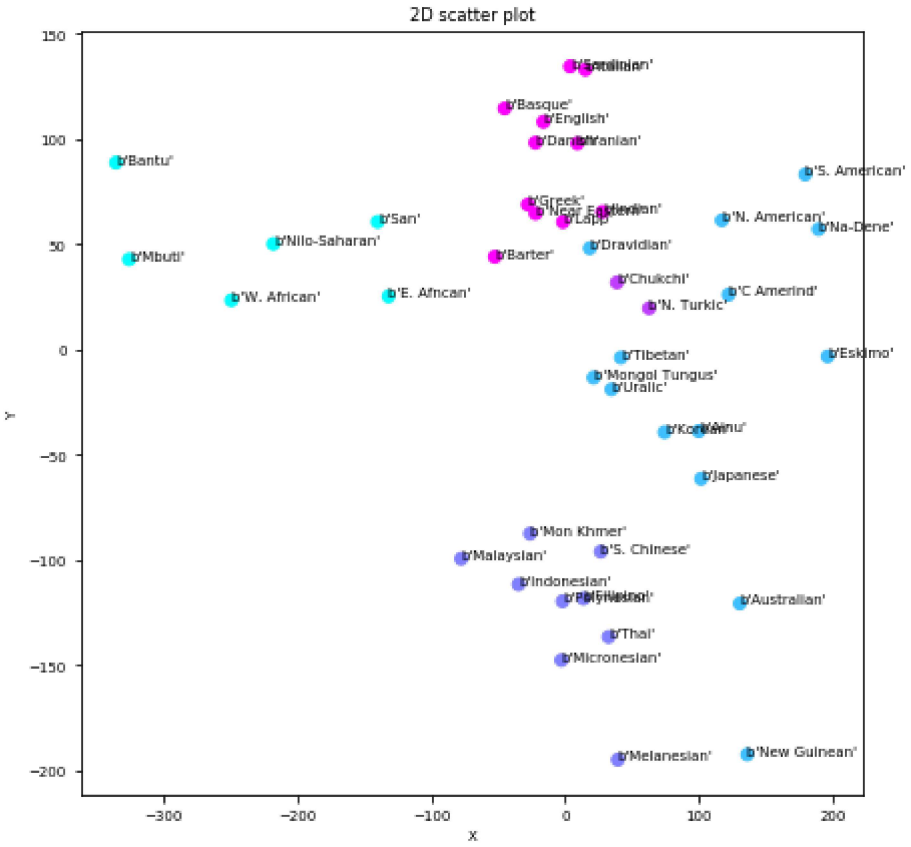
(d)

In [54]:
```
initial_medoids = [1,10,15,25,30]
kmd = kmedoids(data['D'], initial_medoids, data_type='distance_matrix')
kmd.process()
clusters = kmd.get_clusters()
medoids = kmd.get_medoids()
print(clusters)
```

```
[[2, 0, 1, 3, 4, 6], [17, 10, 11, 12, 13, 16, 18, 31, 32, 33, 34, 35, 40, 41],
[23, 14, 15, 19, 20, 21, 37, 38, 39], [22, 36], [29, 5, 7, 8, 9, 24, 25, 26, 2
7, 28, 30]]
```

In [55]:
```
df = pd.DataFrame(list(range(42)))
df['Clusters'] = np.NaN

for i in range(len(clusters)):
    for j in range(0,42):
            if [k in clusters[i] for k in df[0]][j]:
                df.iloc[j,1] = i+1
```

In [56]:
```
clst=list(df['Clusters'])
plt.scatter(x,y, c=clst, cmap=pylab.cm.cool)
plt.rcParams.update({'font.size': 7})
for i, text in enumerate(data['population_list']):
    plt.annotate(text, (x[i], y[i]))
plt.xlabel('X')
plt.ylabel('Y')
plt.title('2D scatter plot')
plt.gcf().set_size_inches((7, 7))
plt.show()
```



As the plot showing above, k-medoids also gives us a reasonable result, but one of the clusters are still too little to be convincing. Comparing with k-means, we would say k-means still has a better result than k-medoids.

In [ ]:
```
```

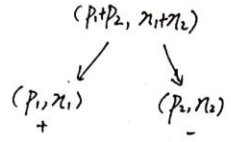
Written Answers:

Question 1

WRITTEN EXERCISES.

1. Decision trees.

(a) Suppose $p_1 > n_1$, $p_2 < n_2$, then we have a tree like this:



for training mistake we have: n_1+p_2

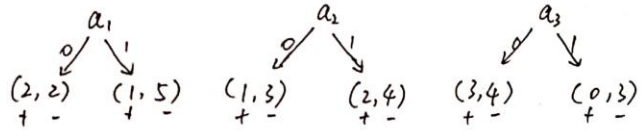
for weighted impurity, we have:

$$(p_1+n_1)\cdot I\left(\frac{p_1}{p_1+n_1}\right)+(p_2+n_2)\cdot I\left(\frac{p_2}{p_2+n_2}\right)$$

$$=(p_1+n_1)\cdot\frac{n_1}{p_1+n_1}+(p_2+n_2)\frac{p_2}{p_2+n_2}=n_1+p_2$$, the same as training mistake.

thus the min-error impurity is equivalent to grow the tree greedily to minimize training error.

(b) We have trees:



Gini index for a_1: $\frac{1}{2}\cdot\frac{1}{2}+\frac{1}{6}\cdot\frac{5}{6}=\frac{7}{18}\approx 0.389$

a_2: $\frac{1}{4}\cdot\frac{3}{4}+\frac{1}{3}\cdot\frac{2}{3}=\frac{59}{144}\approx 0.410$

a_3: $\frac{3}{7}\cdot\frac{4}{7}+0\cdot 1=\frac{12}{49}\approx 0.245$ (smallest)

min-error impurity for a_1: $2+1=3$

a_2: $1+2=3$

a_3: $3+0=3$

thus a_3 will be chosen at root for Gini index, while either of a_1, a_2, a_3 could be chosen by min-error impurity

(c). Same tree as that in (a).

for min-error, before making the split, is either p_1+p_2 or n_1+n_2.

for weighted impurity of the split:

if $p_1>n_1$, $p_2>n_2$ or $p_1<n_1$, $p_2<n_2$, it is the same as min-error (p_1+p_2 or n_1+n_2)

if $p_1>n_1$, $p_2<n_2$, it will be n_1+p_2 which is smaller than p_1+p_2 or n_1+n_2.

if $p_1<n_1$, $p_2>n_2$, it will be n_2+p_1, still smaller than p_1+p_2 or n_1+n_2.

thus the general condition will be ($p_1>n_1$, $p_2<n_2$) or ($p_1<n_1$, $p_2>n_2$).

(d) The answer of (b) and (c) suggest that min-error is suitable for growing a tree under some special case, but it should not be the best way to grow a tree (as a greedy method that always find local optimum).

Question 2

2. Bootstrap aggregation:

Since for N samples, we are drawing N samples with replacement $\Rightarrow$ each draw is independent

for one sample, the probability of not been selected in one draw is:

$$\frac{N-1}{N} = 1 - \frac{1}{N}$$

$\Rightarrow$ for N times, the probability of one sample not been selected is:

$$(1-\frac{1}{N})^N$$

$\Rightarrow$ the fraction of samples does not appear at all is:

$$\frac{N\cdot(1-\frac{1}{N})^N}{N} = (1-\frac{1}{N})^N$$

The limit of this expectation as $N \to \infty$ is:

$$\lim_{n\to\infty}(1-\frac{1}{N})^N = \lim_{x\to\infty}\left[(1+\frac{1}{(-N)})^{(-N)}\right]^{(-1)}$$

$$= e^{-1}$$

$$= \frac{1}{e}$$