CS 5785 Applied Machine learning

Homework 3

Team Member: Xueqi Wei, Queenie Liu

Q1

November 14, 2019

1 1. Sentiment analysis of online reviews.

[206]:
```
import numpy as np
import pandas as pd
from matplotlib import pylab as plt
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import confusion_matrix
import matplotlib.cm as cm
from sklearn import metrics
from sklearn.decomposition import PCA
%matplotlib inline
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from sklearn import preprocessing
from sklearn.preprocessing import FunctionTransformer
from sklearn.linear_model import LogisticRegression
import re
import string
```

(a) Download Sentiment Labelled Sentences Data Set. There are three data files under the root folder. yelp_labelled.txt, amazon_cells_labelled.txt and imdb_labelled.txt. Parse each file with the specifications in readme.txt. Are the labels balanced? If not, what's the ratio between the two labels? Explain how you process these files.

[59]:
```
#load dataset
#import pandas as pd
yelp = pd.read_csv("yelp_labelled.txt", header = None, delimiter = "\t")
amazon = pd.read_csv("amazon_cells_labelled.txt", header = None, delimiter =␣
 ↪'\t')
imdb = pd.read_csv("imdb_labelled.txt", header = None, delimiter = '\t+')

print(yelp.shape)
print(amazon.shape)
print(imdb.shape)

#print(yelp)
```

```
#print(amazon)
#print(imdb)
```

```
(1000, 2)
(1000, 2)
(1000, 2)
```

```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5: ParserWarning:
Falling back to the 'python' engine because the 'c' engine does not support
regex separators (separators > 1 char and different from '\s+' are interpreted
as regex); you can avoid this warning by specifying engine='python'.
  """
```

[55]:
```
yelp_value = yelp.values
yelp_1 = np.sum(yelp_value[:,1] == 1)
print("The number of 1 in yelp is ", yelp_1)
yelp_0 = np.sum(yelp_value[:,1] == 0)
print("The number of 0 in yelp is ", yelp_0)

if (yelp_1 == yelp_0):
    print("Thus, the labels in yelp are balenced")
```

```
The number of 1 in yelp is  500
The number of 0 in yelp is  500
Thus, the labels in yelp are balenced
```

[18]:
```
amazon_value = amazon.values
amazon_1 = np.sum(amazon_value[:,1] == 1)
print("The number of 1 in amazon is ", amazon_1)
amazon_0 = np.sum(amazon_value[:,1] == 0)
print("The number of 0 in amazon is ", amazon_0)

if (amazon_1 == amazon_0):
    print("Thus, the labels in amazon are balenced")
```

```
The number of 1 in amazon is  500
The number of 0 in amazon is  500
Thus, the labels in amazon are balenced
```

[60]:
```
imdb_value = imdb.values
imdb_1 = np.sum(imdb_value[:,1] == 1)
print("The number of 1 in imdb is ", imdb_1)
imdb_0 = np.sum(imdb_value[:,1] == 0)
print("The number of 0 in imdb is ", imdb_0)

if (imdb_1 == imdb_0):
    print("Thus, the labels in imdb are balenced")
```

```
The number of 1 in imdb is  500
The number of 0 in imdb is  500
Thus, the labels in imdb are balenced
```

We use pd.read_csv to read the data file and use numpy.sum to count the positive and negative amount of labels in each file. The labels are balanced since each of them has 500 positive and 500 negative in the file.

(b) Pick your preprocessing strategy. Since these sentences are online reviews, they may contain significant amounts of noise and garbage. You may or may not want to do one or all of the following. Explain the reasons for each of your decision (why or why not). Lowercase all of the words. Lemmatization of all the words (i.e., convert every word to its root so that all of "running," "run," and "runs" are converted to "run" and and all of "good," "well," "better," and "best" are converted to "good"; this is easily done using nltk.stem). Strip punctuation. Strip the stop words, e.g., "the", "and", "or". Something else? Tell us about it.

[138]:
```
def load_data(data):
    positive_data, positive_label = [], []
    negative_data, negative_label = [], []
    f = open(data, 'r')
    for line in f:
        temp = line.strip().split('\t')
        if temp[1] == '0':
            positive_data.append(temp[0])
            positive_label.append(int(temp[1]))
        else:
            negative_data.append(temp[0])
            negative_label.append(int(temp[1]))
    f.close()
    return positive_data, positive_label, negative_data, negative_label

amazon_positive_data, amazon_positive_label, amazon_negative_data,␣
 ↪amazon_negative_label = read_data('amazon_cells_labelled.txt')
imdb_positive_data, imdb_positive_label, imdb_negative_data,␣
 ↪imdb_negative_label = read_data('imdb_labelled.txt')
yelp_positive_data, yelp_positive_label, yelp_negative_data,␣
 ↪yelp_negative_label = read_data('yelp_labelled.txt')
```

[139]:
```
def preprocess_data(data):
    # Lowercase all of the words
    data = np.char.lower(data)

    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))

    #Strip puncturation
    for i in range(len(data)):
        data[i] = re.sub(r'[^\w\s]','',data[i])

    #Lemmatization of all the words + Strip the stop words
    for i in range(len(data)):
        list_ = nltk.word_tokenize(data[i])
        sen = []
        for n in list_:
            if not n in stop_words and not n.isdigit():
                sen.append(lemmatizer.lemmatize(n))
        data[i] = ' '.join(sen)
    return data.tolist()

#processing the data
amazon_data_processed = preprocess_data(amazon_positive_data) +␣
 ↪preprocess_data(amazon_negative_data)
imdb_data_processed = preprocess_data(imdb_positive_data) +␣
 ↪preprocess_data(imdb_negative_data)
yelp_data_processed = preprocess_data(yelp_positive_data) +␣
 ↪preprocess_data(yelp_negative_data)

#double check the length of the data
print(len(amazon_data_processed))
print(len(yelp_data_processed))
print(len(imdb_data_processed))
```

```
1000
1000
1000
```

1.Lowercase all of the words: Yes, it will help with key word matching. 2.Lemmatization of all the words: Yes, it will help to increase the accuracy. 3.Strip punctuation: Yes, punctuation is not taken into account. 4.Strip the stop words: Yes, removing stop words can help extract more important information.

(c) Split training and testing set. In this assignment, for each file, please use the first 400 instances for each label as the training set and the remaining 100 instances as testing set. In total, there are 2400 reviews for training and 600 reviews for testing.

[140]:
```
#Split training and testing set
#use the first 400 instances for each label as the training set and the␣
 ↪remaining 100 instances as testing set

#split training and testing set for yelp
x_Train_yelp = yelp_data_processed[0:400] + yelp_data_processed[500:900]
x_Test_yelp = yelp_data_processed[400:500] + yelp_data_processed[900:1000]
y_Train_yelp = yelp_label_processed[0:400] + yelp_label_processed[500:900]
```

```
y_Test_yelp = yelp_label_processed[400:500] + yelp_label_processed[900:1000]

#split training and testing set for amazon
x_Train_amazon = amazon_data_processed[0:400] + amazon_data_processed[500:900]
x_Test_amazon = amazon_data_processed[400:500] + yelp_data_processed[900:1000]
y_Train_amazon = amazon_label_processed[0:400] + amazon_label_processed[500:900]
y_Test_amazon = amazon_label_processed[400:500] + yelp_label_processed[900:1000]

#split training and testing set for imdb
x_Train_imdb = imdb_data_processed[0:400] + imdb_data_processed[500:900]
x_Test_imdb = imdb_data_processed[400:500] + yelp_data_processed[900:1000]
y_Train_imdb = imdb_label_processed[0:400] + imdb_label_processed[500:900]
y_Test_imdb = imdb_label_processed[400:500] + yelp_label_processed[900:1000]

#In total, there are 2400 reviews for training and 600 reviews for testing
x_train = x_Train_yelp + x_Train_amazon + x_Train_imdb
x_test = x_Test_yelp + x_Test_amazon + x_Test_imdb
y_train = y_Train_yelp + y_Train_amazon + y_Train_imdb
y_test = y_Test_yelp + y_Test_amazon + y_Test_imdb

print(len(x_train))
print(len(x_test))
print(len(y_train))
print(len(y_test))
```

```
2400
600
2400
600
```

[178]:
```
x_train = amazon_data_positive[:400] + amazon_data_negative[:400]␣
 ↪+imdb_data_positive[:400]+imdb_data_negative[:400]+yelp_data_positive[:
 ↪400]+yelp_data_negative[:400]
x_test = amazon_data_positive[400:] + amazon_data_negative[400:]␣
 ↪+imdb_data_positive[400:]+imdb_data_negative[400:]+yelp_data_positive[400:
 ↪]+yelp_data_negative[400:]
y_train = amazon_label_positive[:400] + amazon_label_negative[:400]␣
 ↪+imdb_label_positive[:400]+imdb_label_negative[:400]+yelp_label_positive[:
 ↪400]+yelp_label_negative[:400]
y_test = amazon_label_positive[400:] + amazon_label_negative[400:]␣
 ↪+imdb_label_positive[400:]+imdb_label_negative[400:]+yelp_label_positive[400:
 ↪]+yelp_label_negative[400:]
print(len(y_train))
print(len(y_test))
```

```
2400
600
```

(d) Bag of Words model. Extract features and then represent each review using bag of words model, i.e., every word in the review becomes its own element in a feature vector. In order to do this, first, make one pass through all the reviews in the training set (Explain why we can't use testing set at this point) and build a dictionary of unique words. Then,make another pass through the review in both the training set and testing set and count up the occurrences of each word in your dictionary. The i th element of a review's feature vector is the number of occurrences of the i th dictionary word in the review. Implement the bag of words model and report feature vectors of any two reviews in the training set.

[148]:
```
#bag of words model.
dict_unique_words = dict()
count_unique_word = 0
for x in x_train:
    for word in x.split(' '):
        if word not in dict_unique_words:
            dict_unique_words[word] = count_unique_word
            count_unique_word +=1
train_featured = []
test_featured = []
for x in x_train:
    word_featured = [0]*count_unique_word
    for word in x.split(' '):
        word_featured[dict_unique_words[word]] += 1
    train_featured.append(word_featured)

for x in x_test:
    word_featured = [0]*count_unique_word
    for word in x.split(' '):
        if word in dict_unique_words:
            word_featured[dict_unique_words[word]] += 1
    test_featured.append(word_featured)

train_featured = np.array(train_featured)
test_featured = np.array(test_featured)


print(count_unique_word)
print(x_train[0])
print(sum(train_featured[0]))
print(x_train[1])
print(sum(train_featured[1]))
```

```
4194
crust good
2
tasty texture nasty
3
```

(e)Pick your postprocessing strategy. Since the vast majority of English words will not appear in most of the reviews, most of the feature vector elements will be 0. This suggests that we need a postprocessing or normalization strategy that combats the huge variance of the elements in the feature vector. You may want to use one of the following strategies. Whatever choices you make, explain why you made the decision. log-normalization. For each element of the feature vector x, transform it into f (x) Æ log (x Å1). l1 normalization. Normalize the l1 normof the feature vector, x Æ x j x j . l2 normalization. Normalize the l2 normof the feature vector, x Æ x kxk . Standardize the data by subtracting the mean and dividing by the variance.

Explanation: l2 can provide a higher accuracy of the normalization so we choose to postprocess the data through L2 strategy.

[156]:
```
#postprocessing
#from sklearn import preprocessing
#from sklearn.preprocessing import FunctionTransformer

#log-normalization. For each element of the feature vector x, transform it into␣
 ↪f (x) Æ log (x Å1)
transformer = FunctionTransformer(np.log1p, validate=True)
train_featured_log = transformer.transform(train_featured)
test_featured_log = transformer.transform(test_featured)
print(train_featured_log)
print(test_featured_log)

#I1 normalization. Normalize the l1 normof the feature vector, x Æ x j x j
train_featured_l1 = preprocessing.normalize(train_featured, norm='l1')
test_featured_l1 = preprocessing.normalize(test_featured, norm='l1')
print(train_featured_l1)
print(test_featured_l1)

#l2 normalization. Normalize the l2 normof the feature vector, x Æ x kxk
train_featured_l2 = preprocessing.normalize(train_featured, norm='l2')
test_featured_l2 = preprocessing.normalize(test_featured, norm='l2')
print(train_featured_l2)
print(test_featured_l2)

#Standardize the data by subtracting the mean and dividing by the variance.
train_featured_scale = preprocessing.scale(train_featured)
test_featured_scale = preprocessing.scale(test_featured)
print(train_featured_scale)
print(test_featured_scale)
```

```
[[0.69314718 0.69314718 0.         ... 0.         0.         0.        ]
 [0.         0.         0.69314718 ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.69314718 0.         ... 0.69314718 0.69314718 0.69314718]]
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[[0.5        0.5        0.         ... 0.         0.         0.        ]
 [0.         0.         0.33333333 ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.0625     0.         ... 0.0625     0.0625     0.0625    ]]
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[[0.70710678 0.70710678 0.         ... 0.         0.         0.        ]
 [0.         0.         0.57735027 ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.25       0.         ... 0.25       0.25       0.25      ]]
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[[ 3.46265794e+01  3.33590038e+00 -6.13523873e-02 ... -2.04166684e-02
  -2.04166684e-02 -2.04166684e-02]
 [-2.88795491e-02 -2.73730329e-01  1.62992842e+01 ... -2.04166684e-02
  -2.04166684e-02 -2.04166684e-02]
 [-2.88795491e-02 -2.73730329e-01 -6.13523873e-02 ... -2.04166684e-02
  -2.04166684e-02 -2.04166684e-02]
 ...
 [-2.88795491e-02 -2.73730329e-01 -6.13523873e-02 ... -2.04166684e-02
  -2.04166684e-02 -2.04166684e-02]
 [-2.88795491e-02 -2.73730329e-01 -6.13523873e-02 ... -2.04166684e-02
  -2.04166684e-02 -2.04166684e-02]
 [-2.88795491e-02  3.33590038e+00 -6.13523873e-02 ...  4.89795876e+01
   4.89795876e+01  4.89795876e+01]]
```

```
[[ 0.          -0.26231865 -0.10050378 ...  0.           0.
   0.        ]
 [ 0.          -0.26231865 -0.10050378 ...  0.           0.
   0.        ]
 [ 0.          -0.26231865 -0.10050378 ...  0.           0.
   0.        ]
 ...
 [ 0.          -0.26231865 -0.10050378 ...  0.           0.
   0.        ]
 [ 0.          -0.26231865 -0.10050378 ...  0.           0.
   0.        ]
 [ 0.          -0.26231865 -0.10050378 ...  0.           0.
   0.        ]]
```

(f) Sentiment prediction. Train a logistic regression model (you can use existing packages here) on the training set and test on the testing set. Report the classification accuracy and confusion matrix. Inspecting the weight vector of the logistic regression, what are the words that play the most important roles in deciding the sentiment of the reviews? Repeat this with a Naive Bayes classifier and compare performance.

[193]:
```python
#Sentiment prediction

#classfication accuracy
#Logistic Regression
L_R = LogisticRegression()
L_R.fit(train_featured_l2,y_train)
y_prediction_LR = L_R.predict(test_featured_l2)
print("Logistic Accuracy:",metrics.accuracy_score(y_test, y_prediction_LR))
#Confusion Matrix for Logistic Regression
cm_LR = confusion_matrix(y_test, y_prediction_LR)
print("Confusion matrix for logistic regression:\n", cm_LR)

#Naive Bayes classfier
Ber_NB = BernoulliNB()
Ber_NB.fit(train_featured_l2,y_train)
y_prediction_NB = Ber_NB.predict(test_featured_l2)
print("Bernoulli Accuracy:",metrics.accuracy_score(y_test, y_prediction_NB))
#Confusion Matrix for Naive Bayes
cm_NB = confusion_matrix(y_test, y_prediction_NB)
print("Confusion matrix for Naive Bayes:\n", cm_NB)

# the words that play the most important roles in deciding the sentiment of the␣
 ↪reviews
def Most_Important_Word(m, d):
    freq = abs(m.coef_[0])
    most_important_word_idx = np.argsort(freq)[::-1][:10]
    re = []
    for word,value in d.items():
        if value in most_important_word_idx:
            re.append(str(word))
    return re
print("Top 10 important word of Logistic Regression:", Most_Important_Word(L_R,␣
 ↪dict_unique_words))
print("Top 10 important word of Naive Bayes:", Most_Important_Word(Ber_NB,␣
 ↪dict_unique_words))
```

```
//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

```
Logistic Accuracy: 0.8066666666666666
Confusion matrix for logistic regression:
 [[253  47]
 [ 69 231]]
Bernoulli Accuracy: 0.8133333333333334
Confusion matrix for Naive Bayes:
 [[251  49]
 [ 63 237]]
Top 10 important word of logistic: ['good', 'worst', 'poor', 'bad', 'love',
'best', 'nice', 'great', 'delicious', 'excellent']
Top 10 important word of Naive Bayes: ['crust', 'chilly', 'unremarkable',
'author', 'livingworking', 'abstruse', 'culture', 'reenactment', 'emotionally',
'masculinity']
```

(g) N-gram model. Similar to the bag of words model, but now you build up a dictionary of ngrams, which are contiguous sequences of words. For example, “Alice fell down the rabbit hole” would then map to the 2-grams sequence: [“Alice fell”, “fell down”, “down the”, “the rabbit”, “rabbit hole”], and all five of those symbols would be members of the n-gram dictionary. Try n Æ 2, repeat (d)-(g) and report your results.

[200]:
```
#N-gram model

n_gram_unique_word_dict = dict()
n_gram_unique_word_count = 0
n_gram_train_featured = []
n_gram_test_featured = []

for n in x_train:
    words = n.split()
    for i in range(len(words)-2):
        sequence = ' '.join(words[i:i+2])
        if sequence not in n_gram_unique_word_dict:
            n_gram_unique_word_dict[sequence] = unique_word_count
            unique_word_count +=1

```

```
for n in x_train:
    word_featured = [0]*unique_word_count
    words = n.split()
    for i in range(len(words) - 2):
        sequence = ' '.join(words[i:i + 2])
        word_featured[n_gram_unique_word_dict[sequence]] += 1
    n_gram_train_featured.append(word_featured)
n_gram_train_featured_l2 = preprocessing.normalize(n_gram_train_featured,␣
 ↪norm='l2')

for n in x_test:
    word_featured = [0]*unique_word_count
    words = n.split()
    for i in range(len(words) - 2):
        sequence = ' '.join(words[i:i + 2])
        if sequence in n_gram_unique_word_dict:
            word_featured[n_gram_unique_word_dict[sequence]] += 1
    n_gram_test_featured.append(word_featured)

n_gram_test_featured_l2 = preprocessing.normalize(n_gram_test_featured,␣
 ↪norm='l2')

L_R = LogisticRegression()
L_R.fit(n_gram_train_featured_l2,y_train)
y_prediction_LR = L_R.predict(n_gram_test_featured_l2)
print("Logistic Accuracy:",metrics.accuracy_score(y_test, y_prediction_LR))
print("Top 10 important word of Logistic Regression:", find_important(L_R,␣
 ↪n_gram_unique_word_dict))
cm_lr = confusion_matrix(y_test, y_prediction_LR)
print("Confusion matrix for Logistic Regression:\n", cm_lr)
```

```
//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)

Logistic Accuracy: 0.64
Top 10 important word of Logistic Regression: ['is not', 'was not', 'waste
your', 'and the', 'was very', 'would not', 'the worst', 'the best', 'a great',
'I love']
Confusion matrix for Logistic Regression:
 [[229  71]
 [145 155]]
```

(h) PCA for bag of words model. The features in the bag of words model have large redundancy. Implement PCA to reduce the dimension of features calculated in (e) to 10, 50 and 100 respectively. Using these lower-dimensional feature vectors and repeat (f), (g). Report corresponding clustering and classification results. (Note: You should implement PCA yourself, but you can use numpy.svd or some other SVD package. Feel free to double-check your PCA implementation against an existing one)

[202]:
```
#PCA for bag of words model
#from sklearn.decomposition import PCA

#Implement PCA
def PCA(features,n):
    t = np.mean(features, 0) # mean of each column
    features = features - t
    eva, evs, eve = np.linalg.svd(features, full_matrices=False)
    e_ = eva[:, :n]*evs[:n]
    return e_

def lg(pca_x_train, pca_x_test, y_train, y_test):
    L_R = LogisticRegression()
    L_R.fit(pca_x_train,y_train)
    y_prediction_LR = L_R.predict(pca_x_test)
    return metrics.accuracy_score(y_test, y_prediction_LR)

def Implement_PCA(train_featured,test_featured, n):
    pca_x_train = PCA(train_featured,n)
    pca_x_test = PCA(test_featured,n)
    accuracy = lg(pca_x_train, pca_x_test, y_train, y_test)
    print("dimension of features:", n, "accuracy after PCA:", accuracy)

#to reduce the dimension of features calculated in (e) to 10, 50 and 100␣
 ↪respectively.
print("PCA for bag of words:")
Implement_PCA(train_featured, test_featured, 10)
Implement_PCA(train_featured, test_featured, 50)
Implement_PCA(train_featured, test_featured, 100)
print("2-gram:")
Implement_PCA(n_gram_train_featured, n_gram_test_featured, 10)
Implement_PCA(n_gram_train_featured, n_gram_test_featured, 50)
Implement_PCA(n_gram_train_featured, n_gram_test_featured, 100)
```

```
PCA for bag of words:

//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)

dimension of features: 10, accuracy after PCA: 0.35333333333333333

//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
```

```
solver to silence this warning.
  FutureWarning)

dimension of features: 50, accuracy after PCA: 0.3466666666666667

//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)

dimension of features: 100, accuracy after PCA: 0.395
2-gram:

//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)

dimension of features: 10, accuracy after PCA: 0.485

//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)

dimension of features: 50, accuracy after PCA: 0.465
dimension of features: 100, accuracy after PCA: 0.46166666666666667

//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

As we can see from the result, when dimensions of features is 100, after pca, its accuracy is the highest.

[205]:
```python
#solve the logistic regression for dimensions of features = 100
from sklearn.decomposition import PCA
pca = PCA(n_components = 100)
pca_1 = pca.fit_transform(train_featured)
pca_2 = pca.fit_transform(test_featured)
L_R = LogisticRegression()
L_R.fit(pca_1,y_train)
y_prediction_LR_100 = L_R.predict(pca_2)
print("Logistic Accuracy:",metrics.accuracy_score(y_test, y_prediction_LR_100))
print(pca_1)
```

```
[[ 5.42417182e-02  8.65271223e-01  1.44121633e-01 ... -1.35898162e-02
  -1.26040918e-02 -1.04450488e-02]
 [ 1.03991568e-02 -4.80540250e-03 -1.27947749e-01 ...  3.85017940e-02
```

```
   3.69153521e-02 -2.14583840e-02]
 [ 6.70847491e-05 -6.67484864e-03 -1.35120194e-01 ... -1.17845867e-01
   9.49185506e-02  2.66480539e-02]
 ...
 [-4.55276367e-02  2.96946723e-03 -8.80614976e-02 ... -1.31945176e-01
  -7.07224558e-02 -7.24686186e-02]
 [-1.24948553e-01 -2.58973259e-02 -4.21827373e-02 ... -1.78451911e-02
   5.11616847e-02 -1.09037247e-01]
 [ 6.47968718e-02  9.24611277e-01  1.96114322e-01 ... -7.74640416e-02
  -2.05394148e-01  1.13887436e-01]]
Logistic Accuracy: 0.58

//anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

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

From the above results, we can find that the bag of words method through Logistic model performs best in the prediction task. It might be becasue it will not reduce dimensions of features and preserve meaningful information such as words and sentences itself. From the results of the top 10, we can find that most important words are positive emotional expressions such as “good”, “delicious”.

```
[ ]:
```

Q2

November 14, 2019

1 2. Clustering for text analysis.

[7]:
```
import numpy as np
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
#from scipy.spatial import distance
#from scipy.spatial.distance import euclidean
import matplotlib.pyplot as plt
```

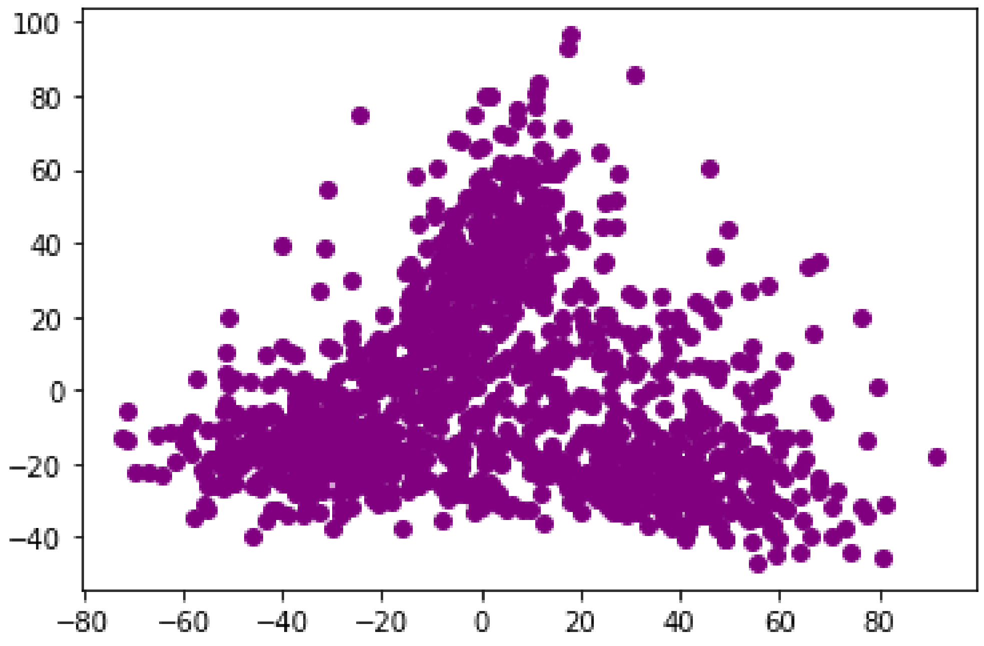
[58]:
```
#load the dataset
data_doc_word = np.load ("science2k-doc-word.npy")
data_word_doc = np.load ("science2k-word-doc.npy")
print(data_doc_word.shape)
print(data_doc_word)
print(data_word_doc.shape)
print(data_word_doc)
vocab = open("science2k-vocab.txt", "r").read().split()
titles = open("science2k-titles.txt", "r").read().split("\n")
```

```
(1373, 5476)
[[-0.2521619 -0.2521619  9.36371   ... -0.2521619 -0.2521619 -0.2521619]
 [-0.2875293 -0.2875293  8.229864  ... -0.2875293 -0.2875293 -0.2875293]
 [-0.3634041 -0.3634041  9.252468  ... -0.3634041 -0.3634041 -0.3634041]
 ...
 [10.04846   -0.7713402  9.132197  ... -0.7713402 -0.7713402 -0.7713402]
 [11.00702   -0.8423803 10.38288   ... -0.8423803 -0.8423803 -0.8423803]
 [ 9.710337  -0.7527946  9.556191  ... -0.7527946 -0.7527946 -0.7527946]]
(5476, 1373)
[[-6.755691   -6.755691   -6.755691   ...  4.064107    5.093713
   3.707441  ]
 [-4.028205   -4.028205   -4.028205   ... -4.028205   -4.028205
  -4.028205  ]
 [-0.03370464 -1.132184   -0.03370464 ...  0.2539608   1.57568
   0.6594092 ]
 ...
 [-0.1301101  -0.1301101  -0.1301101  ... -0.1301101  -0.1301101
  -0.1301101 ]
 [-0.05128021 -0.05128021 -0.05128021 ... -0.05128021 -0.05128021
  -0.05128021]
 [-0.06441435 -0.06441435 -0.06441435 ... -0.06441435 -0.06441435
  -0.06441435]]
```

(a) The extra data file science2k-doc-word.npy contains a 1373č5476 matrix, where each row is an article in Science described by 5476 word features. The articles and words are in the same order as in the vocabulary and titles files above. You can read this file using numpy.load("science2k-doc-word.npy") To obtain the features, we performed the following transformation. First, we computed perdocument smoothed word frequencies. Second, we took the log of those frequencies. Finally, we centered the per-document log frequencies to have zero mean. Cluster the documents using k-means and various values of k (go up to at least k = 20). Select a value of k. For that value, report the top 10 words of each cluster in order of the largest positive distance from the average value across all data. More specifically, if x is the 5476-vector of average values across documents and mi is the i th mean, report the words associated with the top components in mi ąx. Report the top ten documents that fall closest to each cluster center. You can find the titles in the science2k-titles.dat file. Comment on these results. What has the algorithm captured? How might such an algorithm be useful?

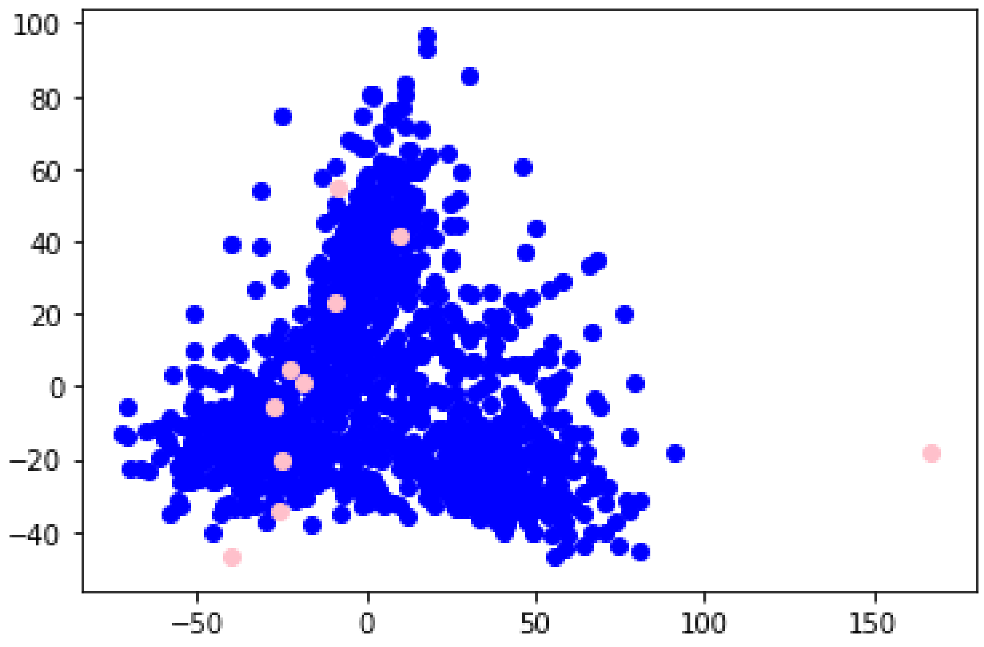
[55]:
```
#from sklearn.decomposition import PCA
#import matplotlib.pyplot as plt
pca = PCA(n_components = 2)
data_doc_word_2= pca.fit_transform(data_doc_word)
plt.scatter(data_doc_word_2[:, 0], data_doc_word_2[:, 1], color = 'purple')
```

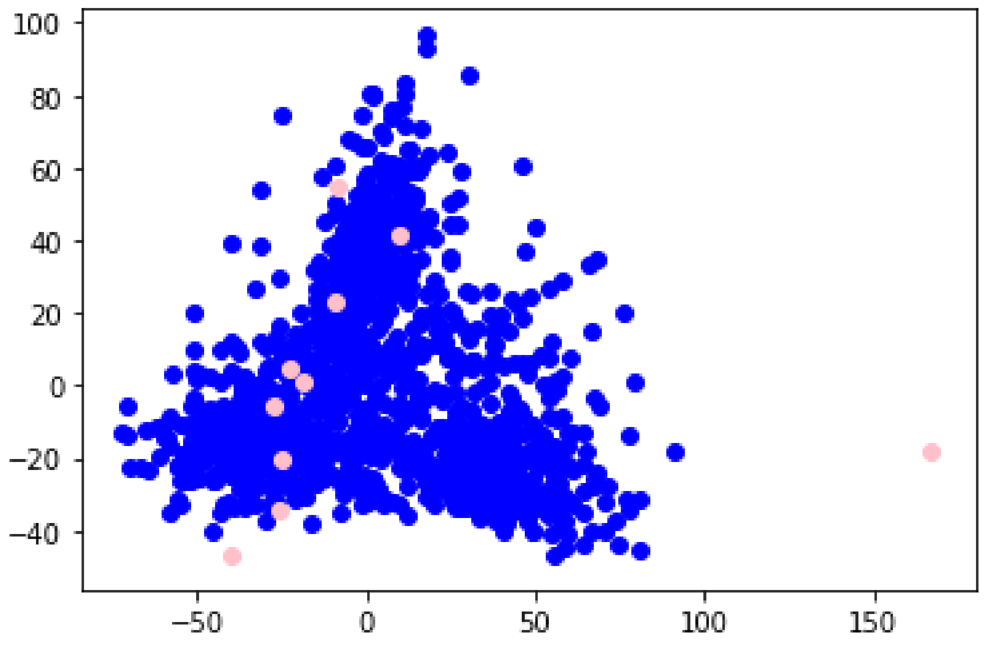
[55]:
```
<matplotlib.collections.PathCollection at 0x1a2f2b68d0>
```

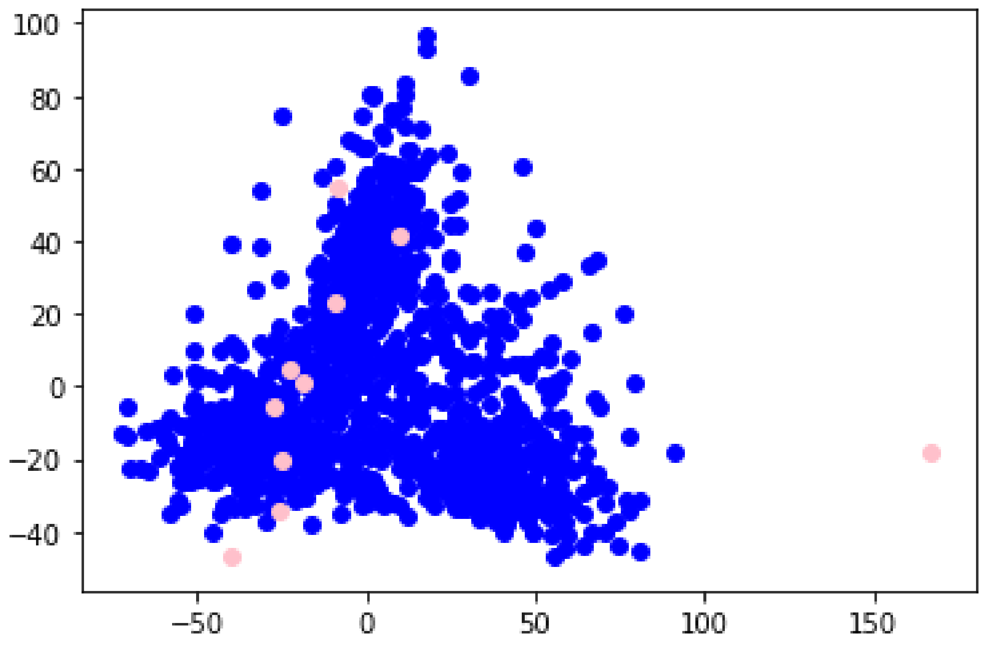


[54]:
```
# Analysis for k

#data_doc_word = np.load ("science2k-doc-word.npy")
#titles = open("science2k-titles.txt", "r").read().split("\n")

#select a value of k up to 20, I choose k = 10
k = 10

#from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = k, init='k-means++').fit(data_doc_word)
cluster = kmeans.predict(data_doc_word)
ctd = kmeans.cluster_centers_
results = []
for n in range(10):
    results.append([(data_doc_word[x], titles[x]) for x in range(len(cluster))␣
 ↪if cluster[x] == n])
print(results[0][0])

#Cluster the documents using k-means and various values of k (go up to at least␣
 ↪k = 20).
#I choose to cluser from k = 2 to k = 21
for k in range(2, 21):
    _ctd = pca.fit_transform(ctd)
    _data_doc_word = pca.fit_transform(data_doc_word)

    #Plot the figures of the clusters.
    plt.figure()
    plt.scatter(_data_doc_word[:, 0], _data_doc_word[:, 1], color = u'b')
    plt.scatter(_ctd[:, 0], _ctd[:, 1], color = 'pink')
    plt.show()
```

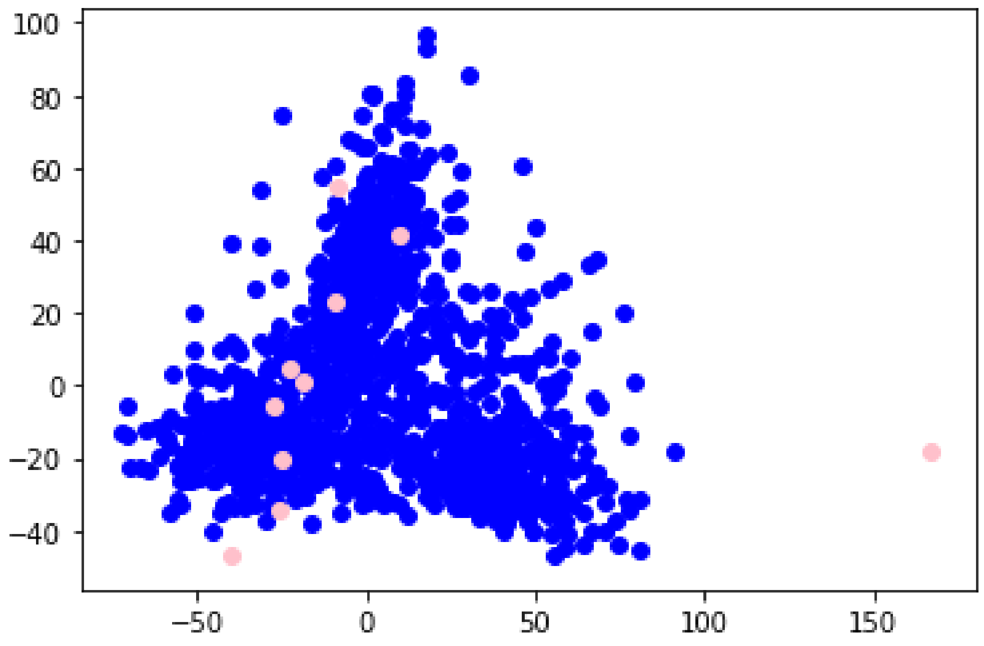
```
(array([-0.8826863, -0.8826863,  9.580446 , ..., -0.8826863, -0.8826863,
       -0.8826863]), '"Finally, the Book of Life and Instructions for Navigating
It"')
```


100
80
60
40
20
0
−20
−40
−50
0
50
100
150

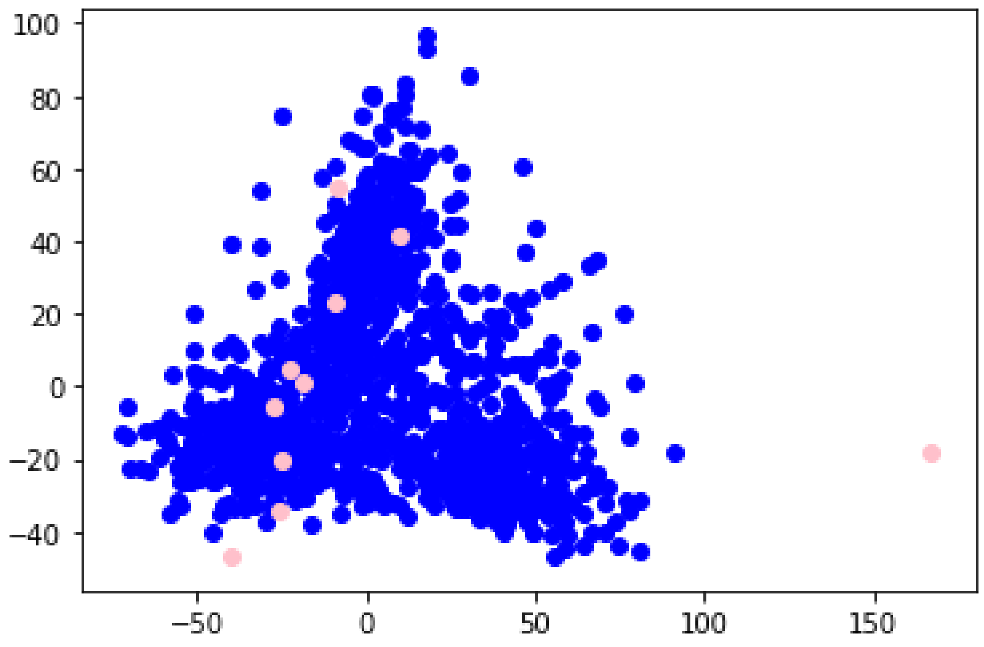

100
80
60
40
20
0
−20
−40
−50
0
50
100
150

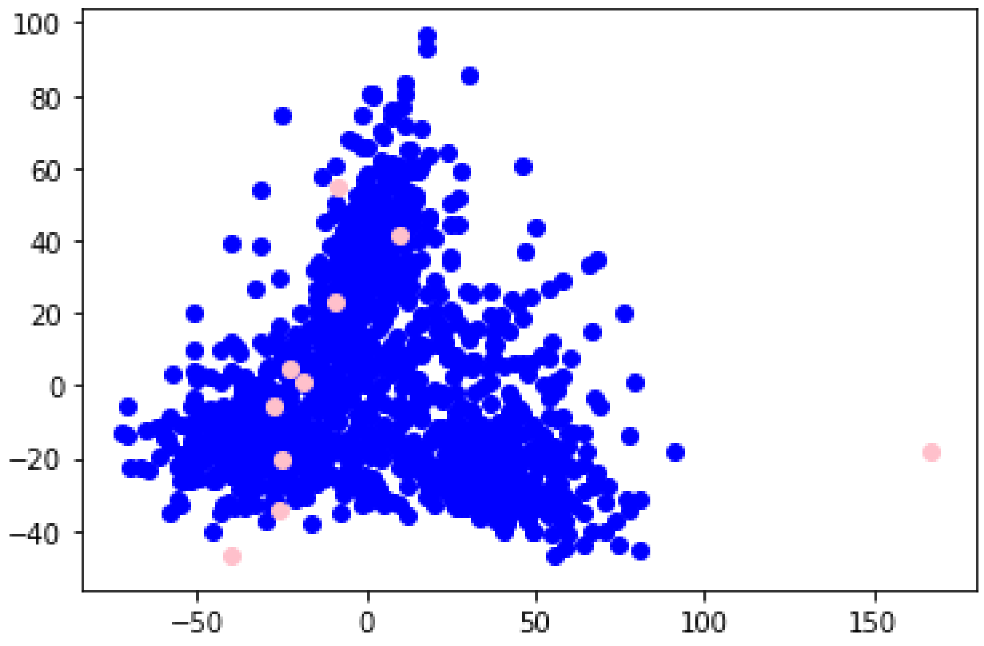


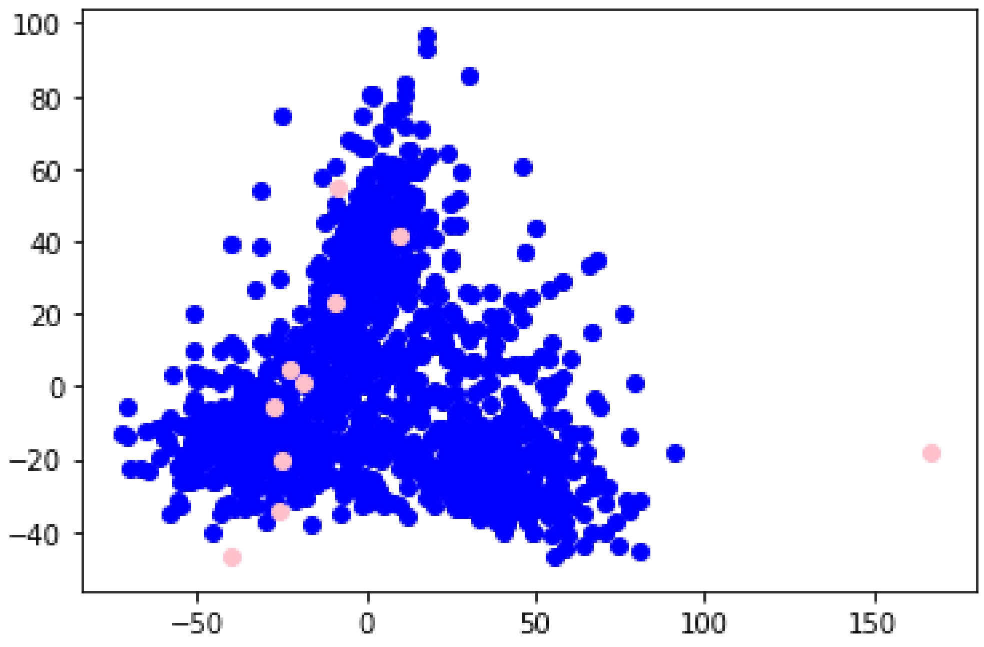
100
80
60
40
20
0
−20
−40
−50
0
50
100
150

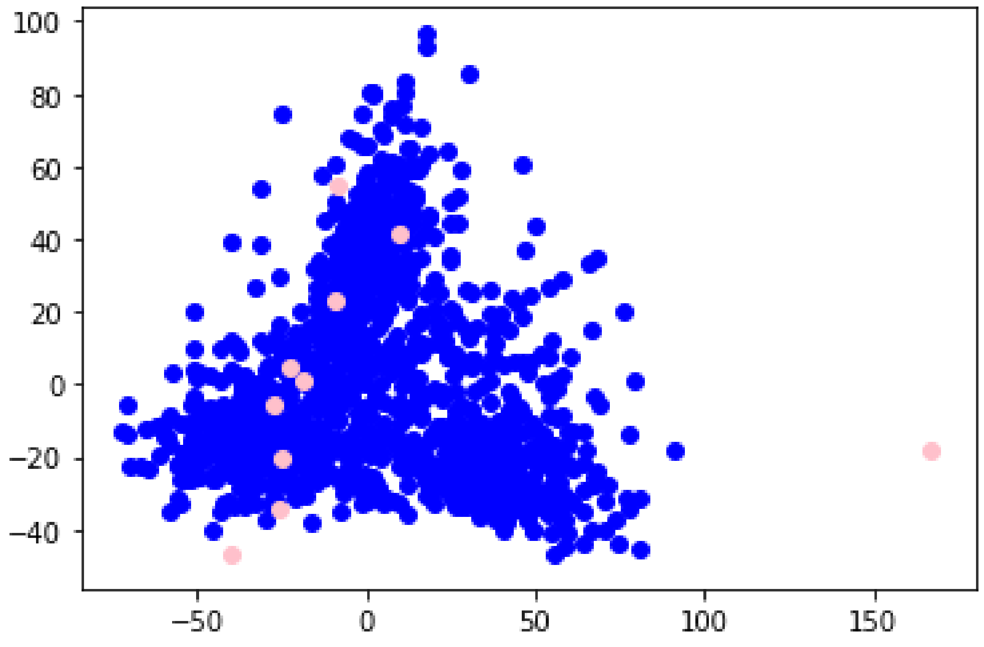


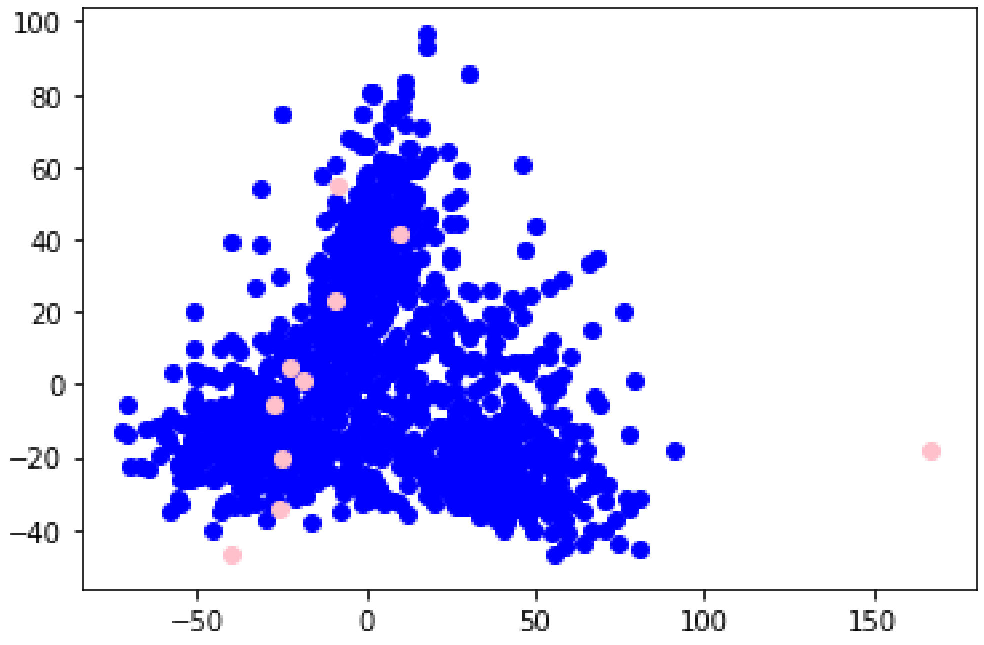
100
80
60
40
20
0
−20
−40
−50
0
50
100
150

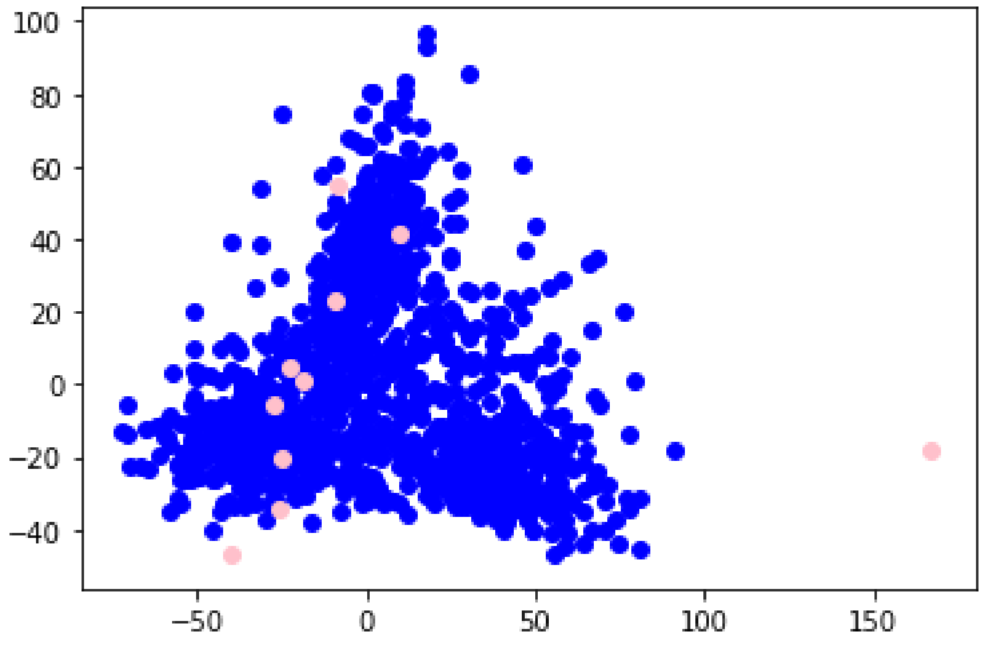

100
80
60
40
20
0
−20
−40
−50
0
50
100
150

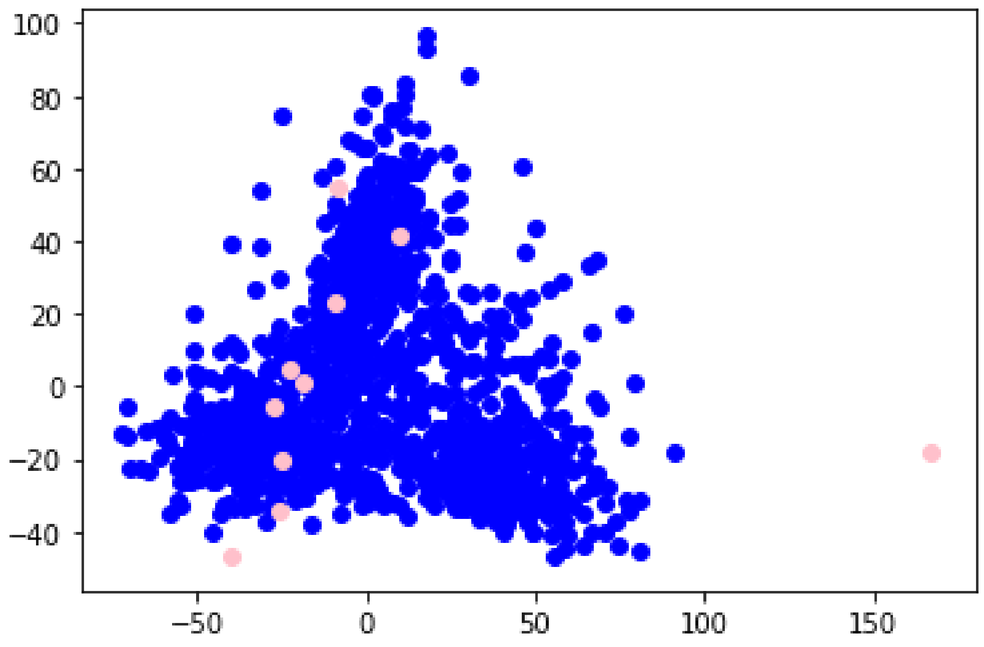

100
80
60
40
20
0
−20
−40
−50
0
50
100
150

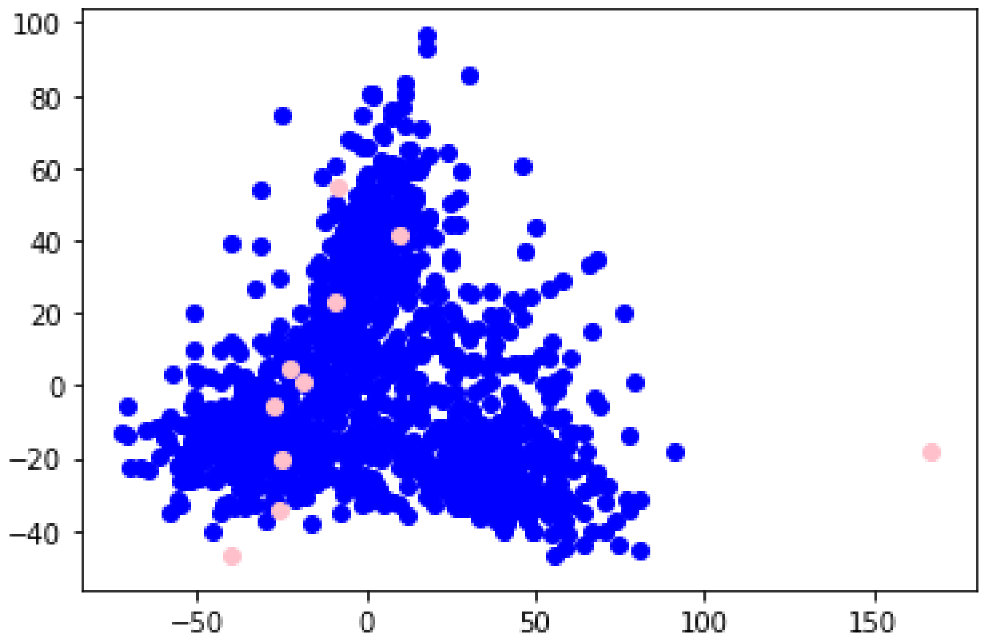

100
80
60
40
20
0
−20
−40
−50
0
50
100
150

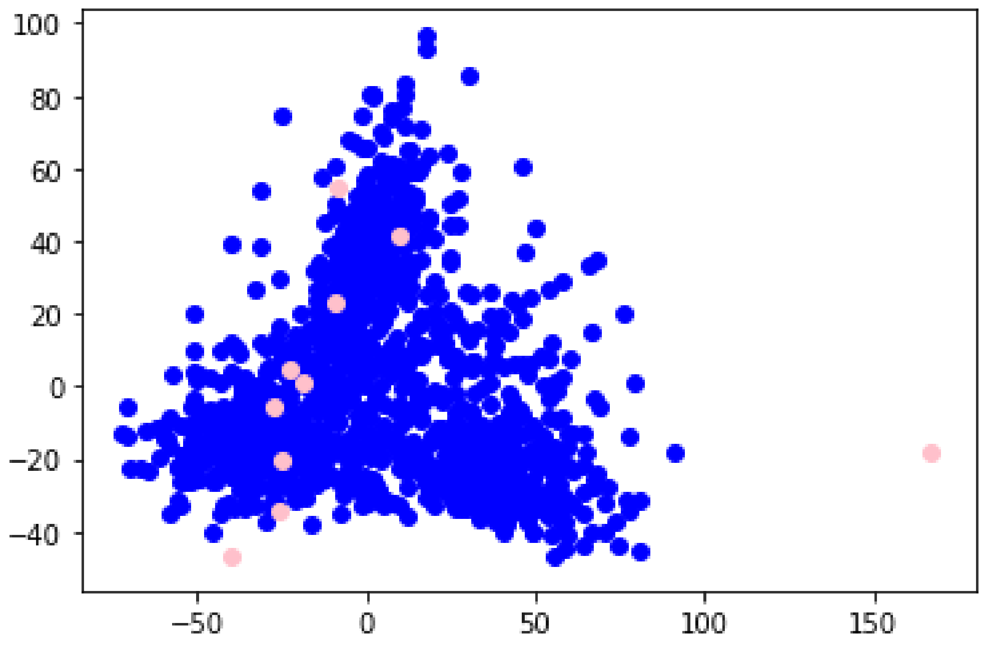

100
80
60
40
20
0
−20
−40
−50
0
50
100
150

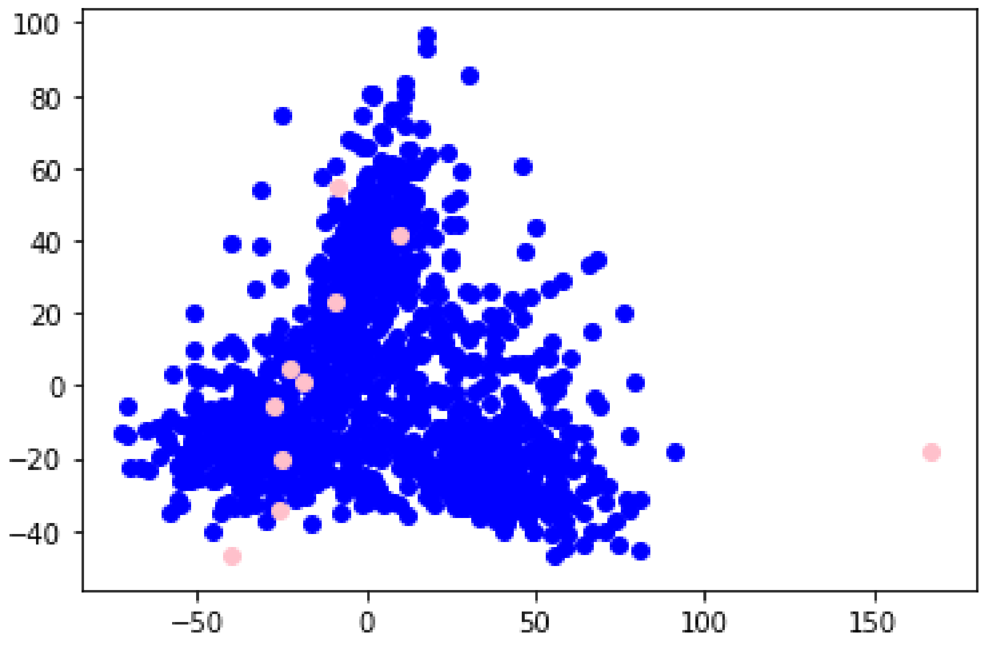

100
80
60
40
20
0
−20
−40
−50
0
50
100
150

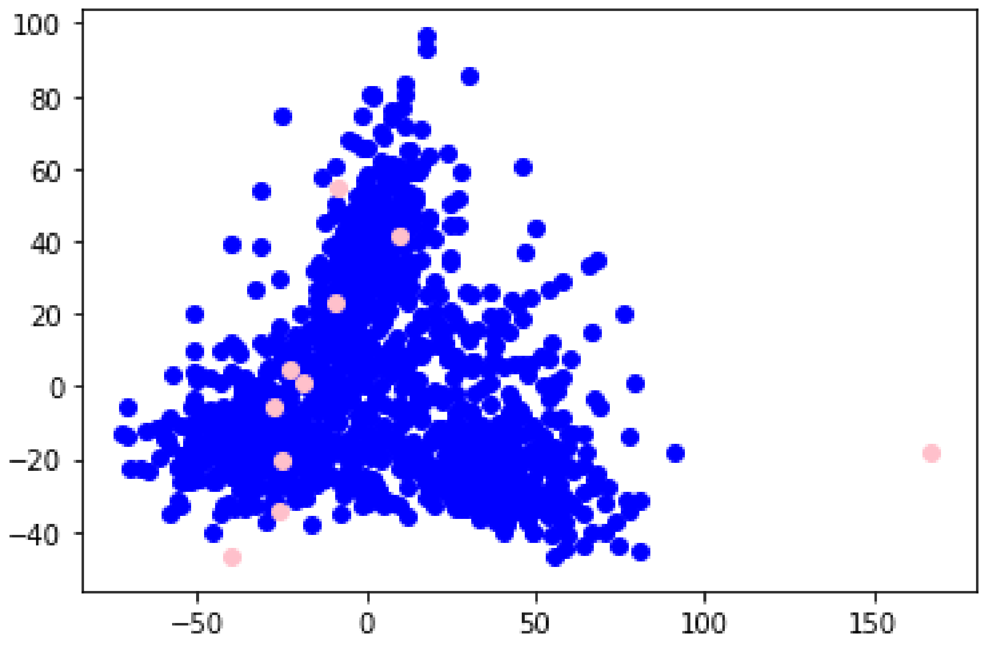

100
80
60
40
20
0
−20
−40
−50
0
50
100
150

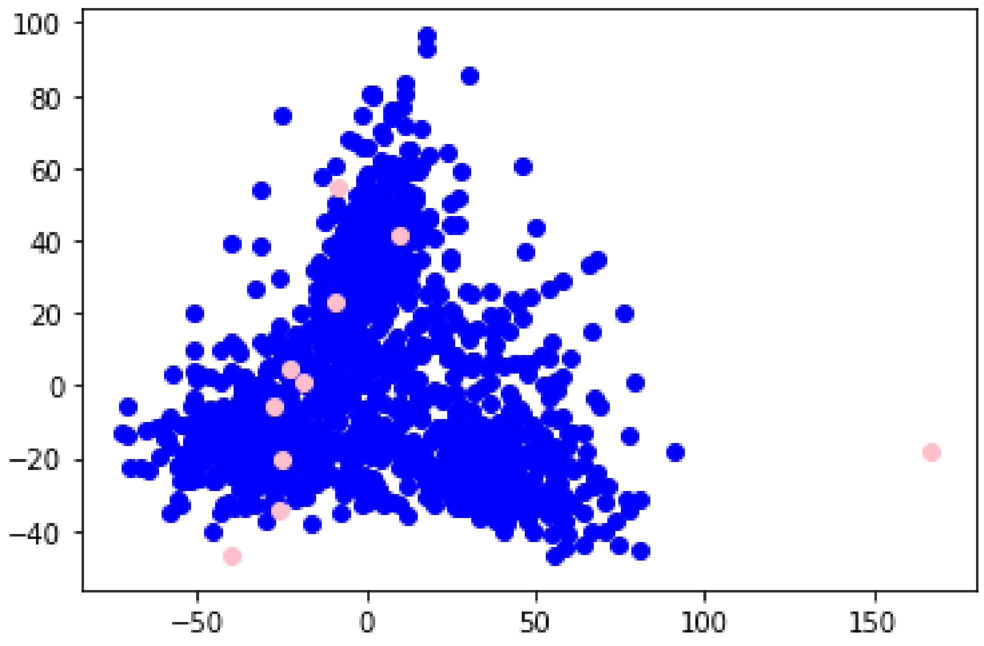

100
80
60
40
20
0
−20
−40
−50
0
50
100
150

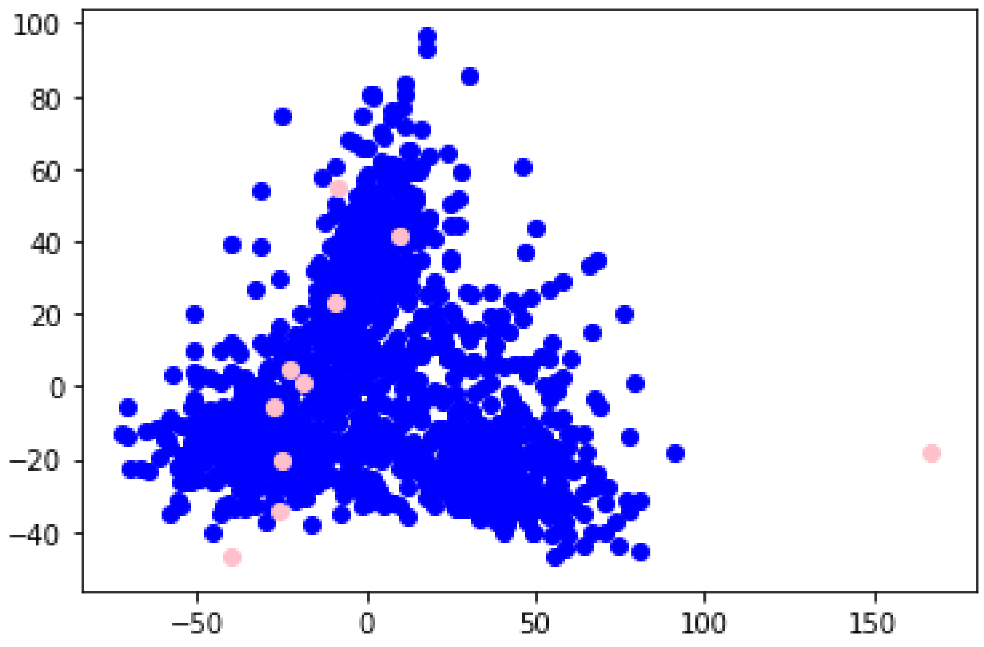

100
80
60
40
20
0
−20
−40
−50
0
50
100
150

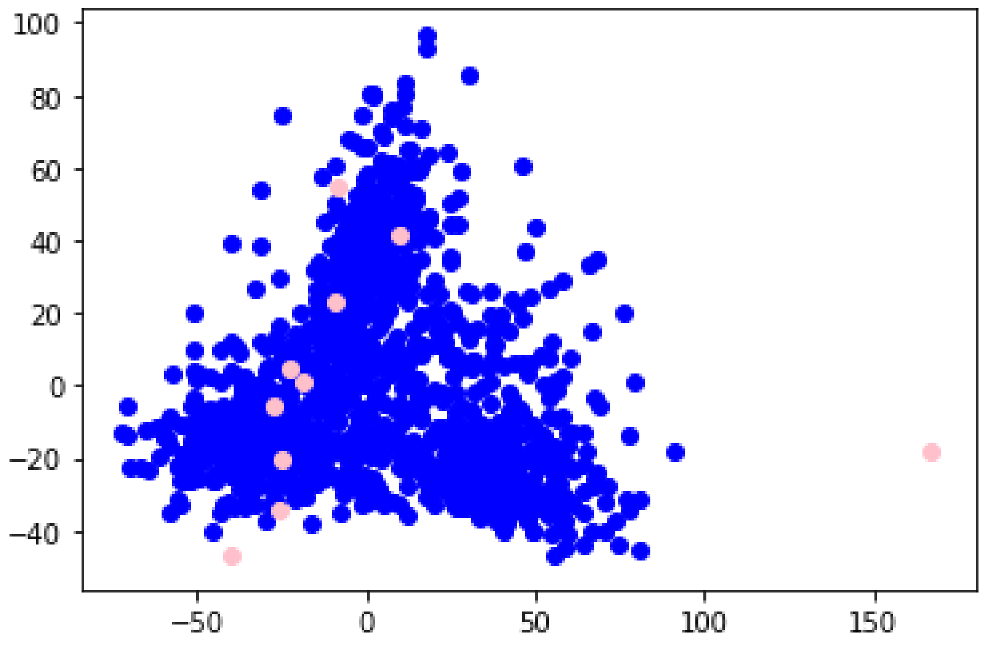

100
80
60
40
20
0
−20
−40
−50
0
50
100
150

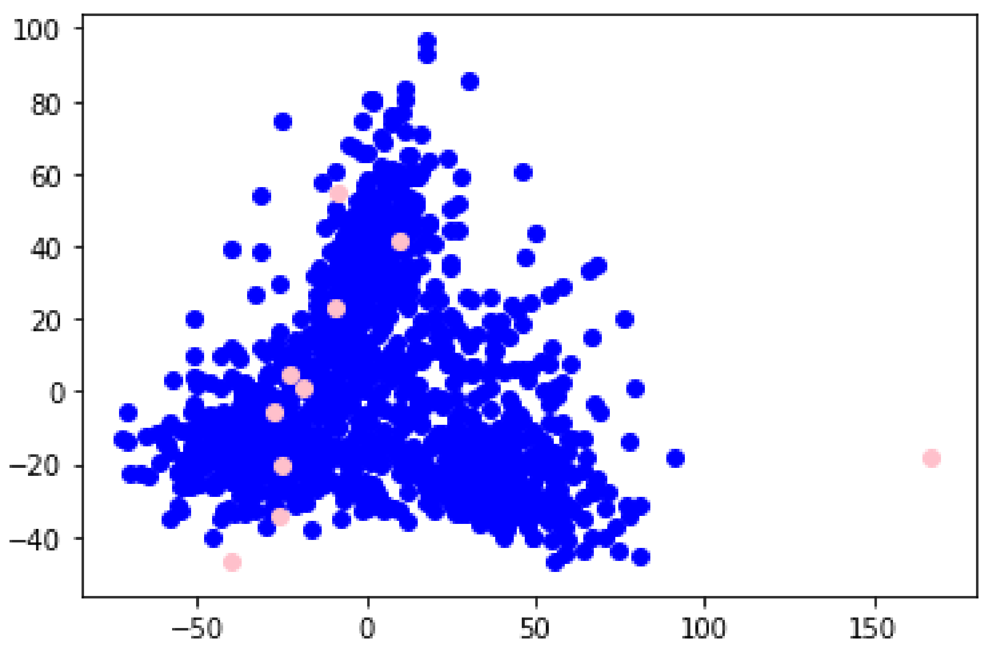

100
80
60
40
20
0
−20
−40
−50
0
50
100
150


100
80
60
40
20
0
−20
−40
−50
0
50
100
150


100
80
60
40
20
0
−20
−40
−50
0
50
100
150


100
80
60
40
20
0
−20
−40
−50
0
50
100
150


100
80
60
40
20
0
−20
−40
−50
0
50
100
150



[40]:
```python
#from sklearn.cluster import KMeans

#report the top 10 words of each cluster in order of the largest positive␣
↪distance from the average value across all data.
def Report_Top_Ten(data_doc_word, n, titles):
    kmeans = KMeans(n_clusters = k, init='k-means++').fit(data_doc_word)
    prediction = kmeans.predict(data_doc_word)

    TopTen = {}

    for clusters in range (n):
        ind_ = [i for i, x in enumerate (prediction) if x == clusters]
        distant = kmeans.transform(data_doc_word)
        dist_ = distant[:,clusters]
        frequency = np.argsort(dist_)[::]
        RankIndex = [index for index in frequency if index in ind_][:10]
        TopTen[clusters + 1] = [titles[ind] for ind in RankIndex]

    return TopTen

# Report documents closest to the center
Report_Top_Ten(data_doc_word, 10, titles)
```

[40]:
```
{1: ['"Closing in on a Deadly Parasite\'s Genome"',
  '"The Business of Stem Cells"',
  '"Malaria Researchers Wait for Industry to Join Fight"',
  '"Simple Hosts May Help Reveal How Bacteria Infect Cells"',
  '"Africa Boosts AIDS Vaccine R&D"',
  '"Research (Genomics) Is Crucial to Attacking Malaria"',
  '"A Renewed Assault on an Old and Deadly Foe"',
  '"DNA Arrays Reveal Cancer in Its Many Forms"',
  '"Stephen Straus\'s Impossible Job"',
  '"South Africa\'s New Enemy"'],
 2: ['"Synthesis and Characterization of Helical Multi-Shell Gold Nanowires"',
  '"Ambipolar Pentacene Field-Effect Transistors and Inverters"',
  '"Graphical Evolution of the Arnold Web: From Order to Chaos"',
  '"High-Gain Harmonic-Generation Free-Electron Laser"',
  '"Prospects for the Polymer Nanoengineer"',
  '"Mechanisms of Ordering in Striped Patterns"',
  '"Viscosity Mechanisms in Accretion Disks"',
  '"A Light-Emitting Field-Effect Transistor"',
  '"Signs of Extreme Gravity"',
  '"Anomalous Polarization Profiles in Sunspots: Possible Origin of Umbral
Flashes"'],
 3: ['"Structure of Yeast Poly(A) Polymerase Alone and in Complex with
3\'-dATP"',
  '"Structure of Murine CTLA-4 and Its Role in Modulating T Cell
Responsiveness"',
  '"Structure of the S15,S6,S18-rRNA Complex: Assembly of the 30S Ribosome
Central Domain"',
  '"Atomic Structure of PDE4: Insights into Phosphodiesterase Mechanism and
Specificity"',
  '"Twists in Catalysis: Alternating Conformations of Escherichia coli
Thioredoxin Reductase"',
  '"The Productive Conformation of Arachidonic Acid Bound to Prostaglandin
Synthase"',
  '"Redox Signaling in Chloroplasts: Cleavage of Disulfides by an Iron-Sulfur
Cluster"',
  '"Convergent Solutions to Binding at a Protein-Protein Interface"',
  '"Structural Basis of Smad2 Recognition by the Smad Anchor for Receptor
Activation"',
  '"Structure of the Protease Domain of Memapsin 2 (b-Secretase) Complexed with
Inhibitor"'],
 4: ['"Greenland Ice Sheet: High-Elevation Balance and Peripheral Thinning"',
  '"Mass Balance of the Greenland Ice Sheet at High Elevations"',
  '"Glacial Climate Instability"',
  '"The Role of the Southern Ocean in Uptake and Storage of Anthropogenic Carbon
Dioxide"',
  '"Temporal Trends in Deep Ocean Redfield Ratios"',
  '"Synchronous Radiocarbon and Climate Shifts during the Last Deglaciation"',
  '"Dynamics of the Pacific-North American Plate Boundary in the Western United
States"',
  '"A High-Resolution Millennial Record of the South Asian Monsoon from
Himalayan Ice Cores"',
  '"Tropical Climate at the Last Glacial Maximum Inferred from Glacier Mass-
Balance Modeling"',
  '"Upwelling Intensification as Part of the Pliocene-Pleistocene Climate
Transition"'],
 5: ['"Algorithmic Gladiators Vie for Digital Glory"',
  '"Reopening the Darkest Chapter in German Science"',
  '"National Academy of Sciences Elects New Members"',
  '"Corrections and Clarifications: Unearthing Monuments of the Yarmukians"',
  '"Corrections and Clarifications: Charon\'s First Detailed Spectra Hold Many
Surprises"',
  '"Corrections and Clarifications: A Short Fe-Fe Distance in Peroxodiferric
Ferritin: Control of Fe Substrate versus Cofactor Decay?"',
  '"Heretical Idea Faces Its Sternest Test"',
  '"Information Technology Takes a Different Tack"',
  '"Archaeology in the Holy Land"',
  '"Corrections and Clarifications: Marking Time for a Kingdom"'],
 6: ['"Suppression of Mutations in Mitochondrial DNA by tRNAs Imported from the
Cytoplasm"',
  '"Distinct Classes of Yeast Promoters Revealed by Differential TAF
Recruitment"',
  '"Efficient Initiation of HCV RNA Replication in Cell Culture"',
  '"Negative Regulation of the SHATTERPROOF Genes by FRUITFULL during
Arabidopsis Fruit Development"',
  '"T Cell-Independent Rescue of B Lymphocytes from Peripheral Immune
Tolerance"',
  '"Patterning of the Zebrafish Retina by a Wave of Sonic Hedgehog Activity"',
  '"Reduced Food Intake and Body Weight in Mice Treated with Fatty Acid Synthase
Inhibitors"',
  '"Coupling of Stress in the ER to Activation of JNK Protein Kinases by
Transmembrane Protein Kinase IRE1"',
  '"An Anti-Apoptotic Role for the p53 Family Member, p73, during Developmental
Neuron Death"',
  '"Disruption of Signaling by Yersinia Effector YopJ, a Ubiquitin-like Protein
Protease"'],
 7: ['"A Stable Bicyclic Compound with Two Si=Si Double Bonds"',
  '"Xenon as a Complex Ligand: The Tetra Xenono Gold(II) Cation in
<latex>$AuXe_4^{2+}(Sb_2F_{11}^-)_2$</latex>"',
  '"A Cyclic Carbanionic Valence Isomer of a Carbocation: Diphosphino Analogs of
Diaminocarbocations"',
  '"Efficient Activation of Aromatic C-H Bonds for Addition to C-C Multiple
Bonds"',
  '"Planar Hexacoordinate Carbon: A Viable Possibility"',
  '"Stable Versions of Transient Push-Pull Carbenes: Extending Lifetimes from
Nanoseconds to Weeks"',
  '"A Single-Molecule Study of RNA Catalysis and Folding"',
  '"Electron-Induced Inversion of Helical Chirality in Copper Complexes of
N,N-Dialkylmethionines"',
  '"Green, Catalytic Oxidation of Alcohols in Water"',
  '"Resonant Formation of DNA Strand Breaks by Low-Energy (3 to 20 eV)
Electrons"'],
 8: ['"Homogenization of Fish Faunas across the United States"',
  '"Influences of Dietary Uptake and Reactive Sulfides on Metal Bioavailability
from Aquatic Sediments"',
  '"Evidence for Ecological Causation of Sexual Dimorphism in a Hummingbird"',
  '"Requirement of NAD and SIR2 for Life-Span Extension by Calorie Restriction
in Saccharomyces Cerevisiae"',
  '"Population Dynamical Consequences of Climate Change for a Small Temperate
Songbird"',
  '"Species Diversity and Biological Invasions: Relating Local Process to
Community Pattern"',
  '"Pig Cloning by Microinjection of Fetal Fibroblast Nuclei"',
  '"Crossing the Hopf Bifurcation in a Live Predator-Prey System"',
  '"Cholinergic Synaptic Inhibition of Inner Hair Cells in the Neonatal
Mammalian Cochlea"',
  '"Selectivity for 3D Shape That Reveals Distinct Areas within Macaque Inferior
Temporal Cortex"'],
 9: ['"On a Slippery Slope to Mediocrity?"',
  '"Solar Missions Brighten NASA\'s Hopes for Space Science Research"',
  '"Support Grows for British Exercise to Allocate University Funds"',
  '"Bringing Science to the National Parks"',
  '"Clinton\'s Science Legacy: Ending on a High Note"',
  '"Graduate Educators Struggle to Grade Themselves"',
  '"CERN Link Breathes Life into Russian Physics"',
  '"Controversy Flares up over NASA Solar Project"',
  '"Packard Heir Signs up for National \'Math Wars\'"',
  '"Does Science Drive the Productivity Train?"'],
 10: ['"The Formation of Chondrules at High Gas Pressures in the Solar Nebula"',
  '"Reconstruction of the Amazon Basin Effective Moisture Availability over the
past 14,000 Years"',
  '"Rapid Kimberlite Ascent and the Significance of Ar-Ar Ages in Xenolith
Phlogopites"',
  '"Isotopic Evidence for Variations in the Marine Calcium Cycle over the
Cenozoic"',
  '"Remobilization in the Cratonic Lithosphere Recorded in Polycrystalline
Diamond"',
  '"Calcium-Aluminum-Rich Inclusions from Enstatite Chondrites: Indigenous or
Foreign?"',
  '"Support for the Lunar Cataclysm Hypothesis from Lunar Meteorite Impact Melt
Ages"',
  '"Accretion of Primitive Planetesimals: Hf-W Isotopic Evidence from Enstatite
Chondrites"',
  '"Extinct <latex>$^{129}I$</latex> in Halite from a Primitive Meteorite:
Evidence for Evaporite Formation in the Early Solar System"',
  '"Rapid Flooding of the Sunda Shelf: A Late-Glacial Sea-Level Record"']}
```

Clustering the documents by words provides as an insight into the complete themes of the documents. This algorithm might be helpful if we want to understand the exact content of these documents.

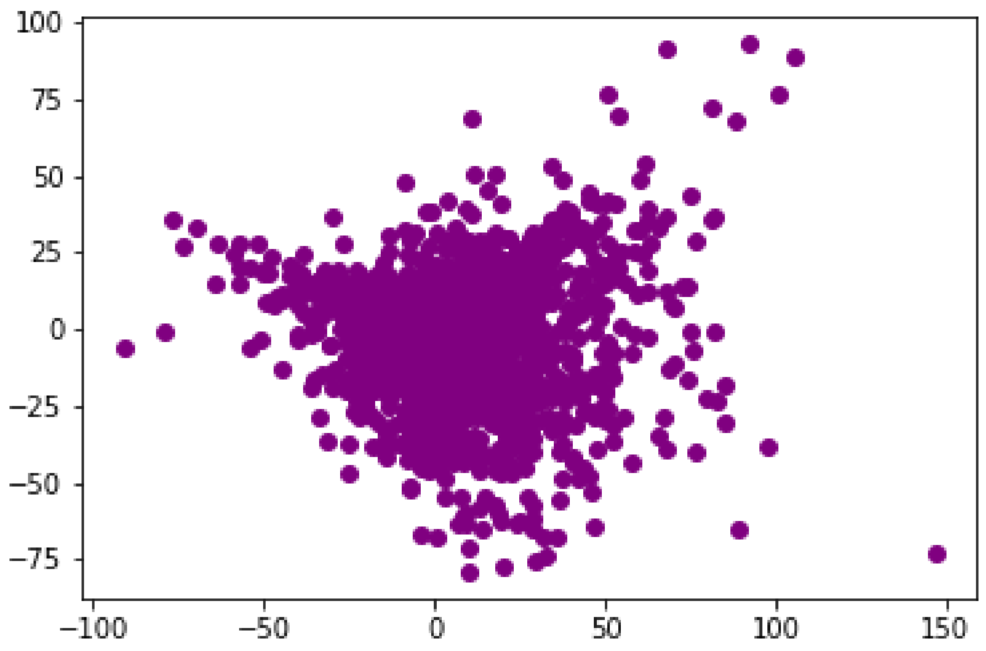
(b) The file science2k-word-doc.txt is similar, but capture term-wise rather than documentwise features. That is, for each term, we count the frequency as the number of documents that term appears in rather than the other way around. This allows us to characterize individual terms. This matrix is 5476č1373, where each row is a termin Science described by 1373 "document" features. These are transformed document frequencies (as above). Repeat the analysis above, but cluster terms instead of documents. The terms are listed in science2k-vocab.txt Comment on these results. How might such an algorithm be useful? What is different about clustering terms from clustering documents?

[42]:
```
#load the dataset
data_word_doc = np.load ("science2k-word-doc.npy")
vocab = open("science2k-vocab.txt", "r").read().split()
print(data_word_doc.shape)
print(data_word_doc)
print(vocab[0])
```

```
(5476, 1373)
[[-6.755691   -6.755691   -6.755691   ...  4.064107    5.093713
   3.707441  ]
 [-4.028205   -4.028205   -4.028205   ... -4.028205   -4.028205
  -4.028205  ]
 [-0.03370464 -1.132184   -0.03370464 ...  0.2539608   1.57568
   0.6594092 ]
 ...
 [-0.1301101  -0.1301101  -0.1301101  ... -0.1301101  -0.1301101
  -0.1301101 ]
 [-0.05128021 -0.05128021 -0.05128021 ... -0.05128021 -0.05128021
  -0.05128021]
 [-0.06441435 -0.06441435 -0.06441435 ... -0.06441435 -0.06441435
  -0.06441435]]
fig
```

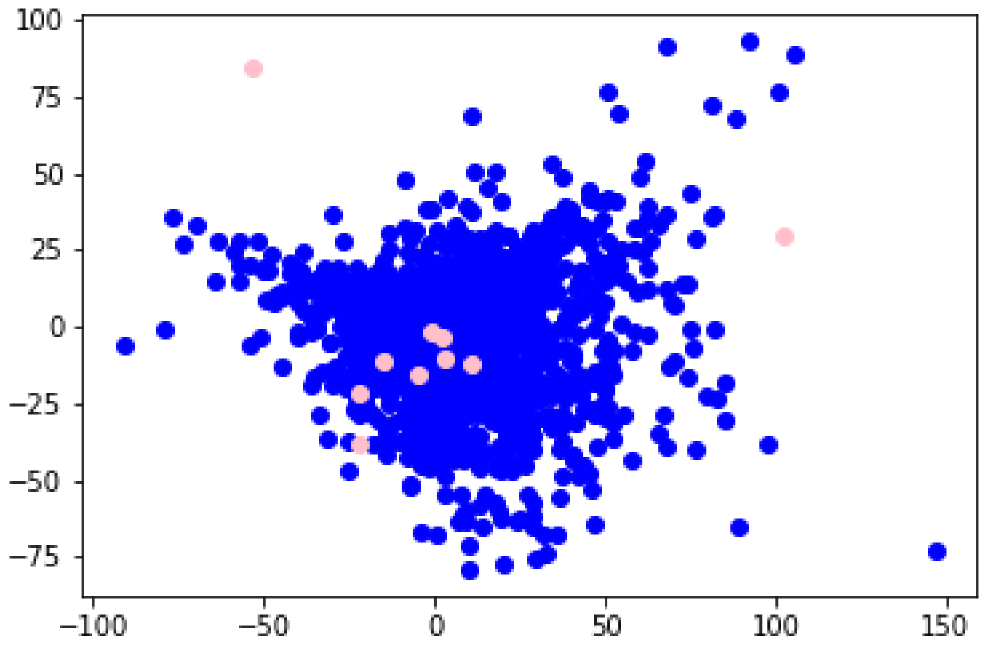
[59]:
```
#from sklearn.decomposition import PCA
#import matplotlib.pyplot as plt
pca = PCA(n_components = 2)
data_word_doc_2= pca.fit_transform(data_word_doc)
plt.scatter(data_word_doc_2[:, 0], data_word_doc_2[:, 1], color = 'purple')
```

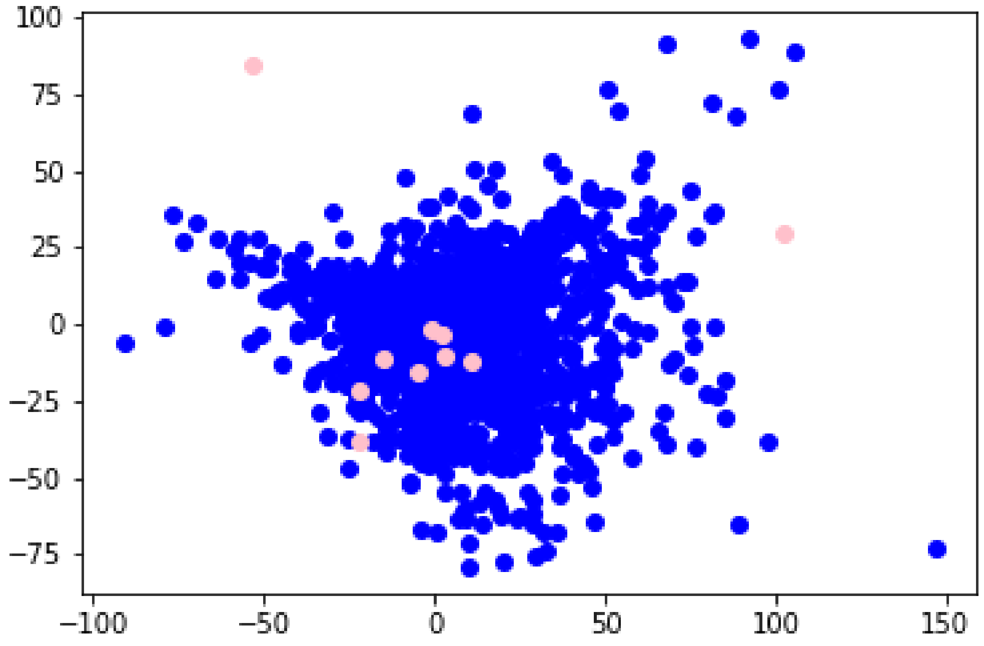
[59]:
```
<matplotlib.collections.PathCollection at 0x107ab2208>
```

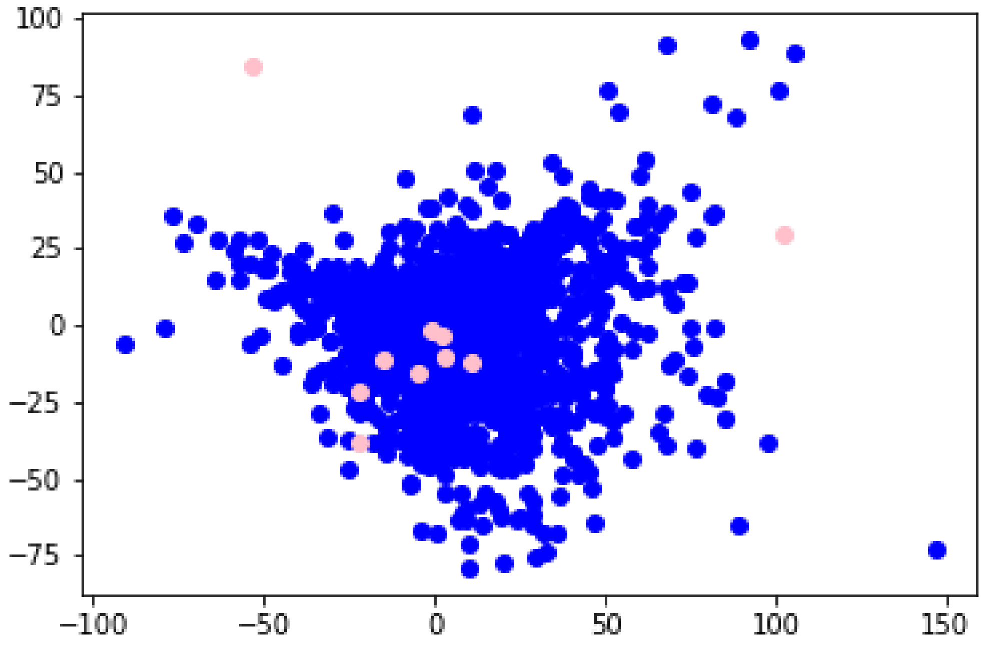


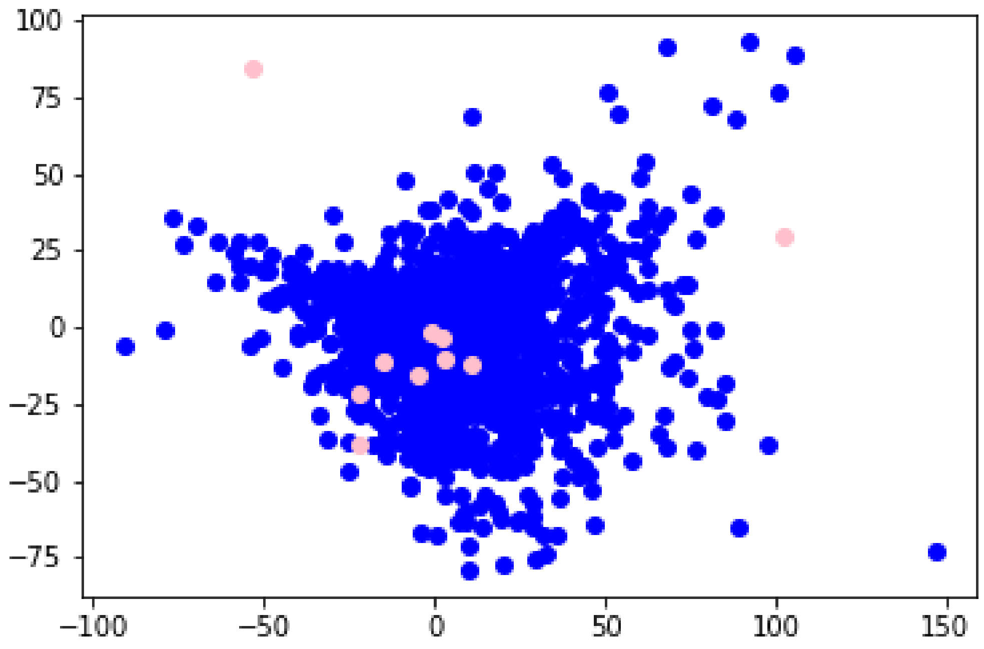
[57]:
```
# Analysis for k


#data_word_doc = np.load ("science2k-word-doc.npy")
#vocab = open("science2k-vocab.txt", "r").read().split()

#select a value of k up to 20, I choose k = 10
k = 10

#from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=k, random_state=0).fit(data_word_doc)
cluster = kmeans.predict(data_word_doc)
ctd = kmeans.cluster_centers_
results = []
for n in range(10):
    results.append([(data_word_doc[x], vocab[x]) for x in range(len(cluster))␣
 ↪if cluster[x] == n])
print(results[0][0])

#select a value of k up to 20, I choose k = 10
k = 10

#Cluster the documents using k-means and various values of k (go up to at least␣
 ↪k = 20).
#I choose to cluser from k = 2 to k = 21
for k in range(2, 21):
    _ctd = pca.fit_transform(ctd)
    _data_word_doc = pca.fit_transform(data_word_doc)

    #Plot the figures of the clusters.
    plt.figure()
    plt.scatter(_data_word_doc[:, 0], _data_word_doc[:, 1], color = u'b')
    plt.scatter(_ctd[:, 0], _ctd[:, 1], color = 'pink')
    plt.show()
```

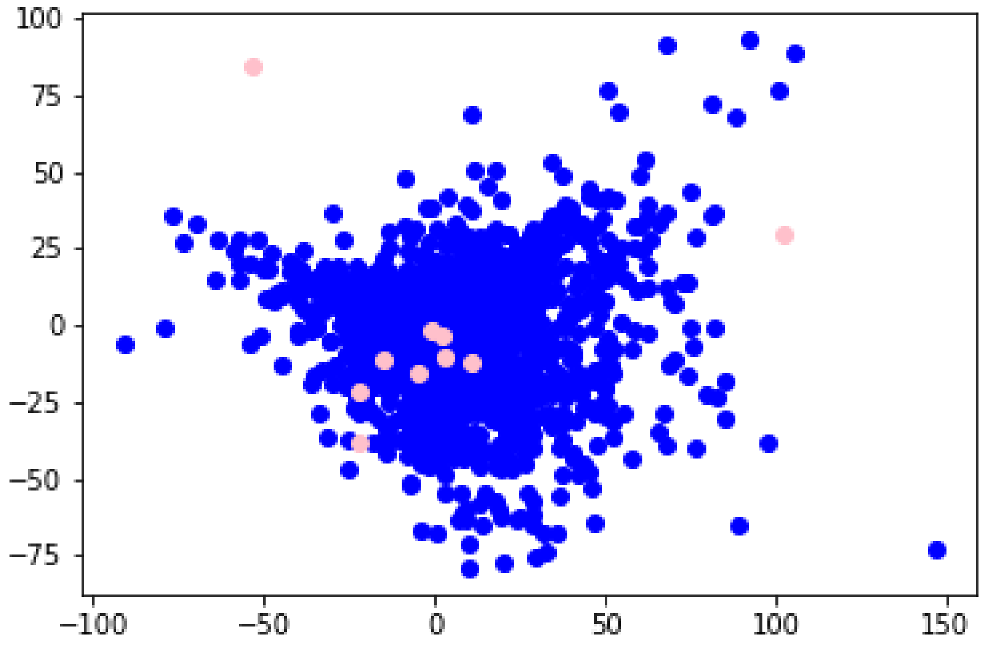
```
(array([-4.028205, -4.028205, -4.028205, ..., -4.028205, -4.028205,
       -4.028205]), 'cells')
```

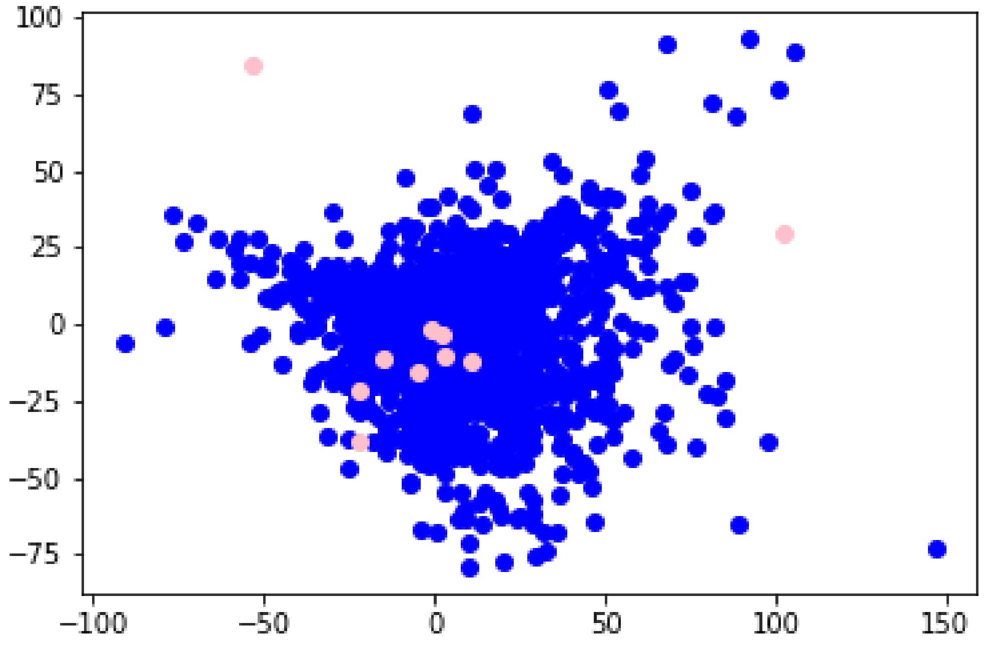


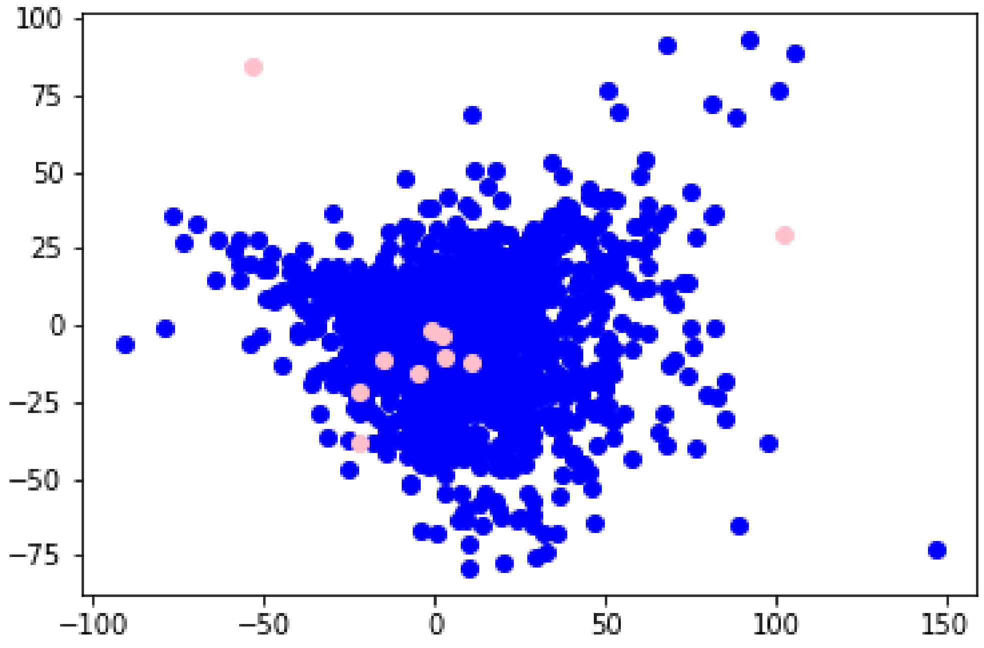

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

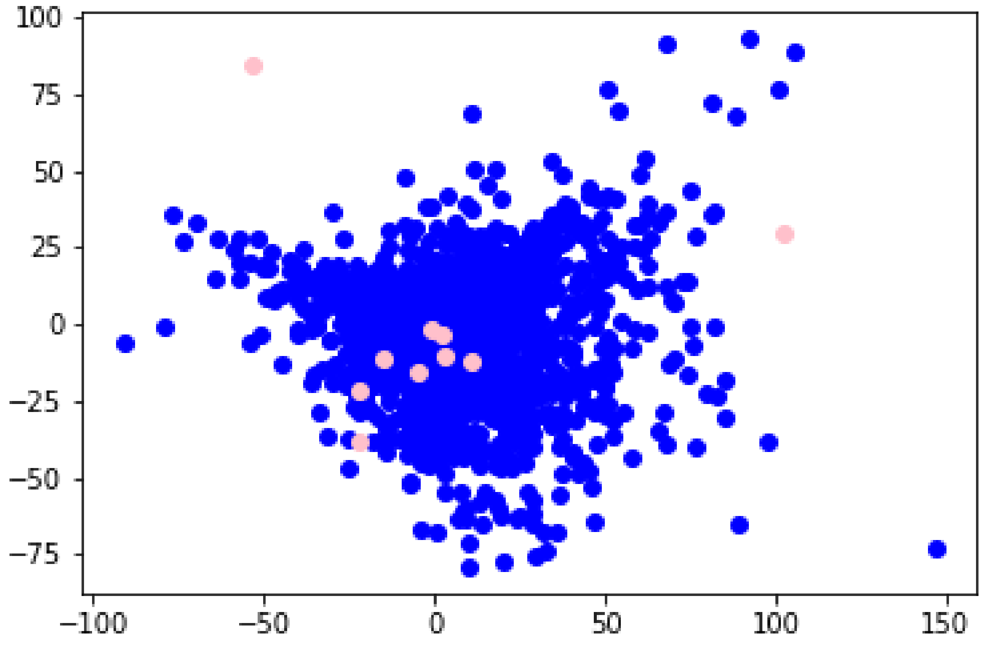

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

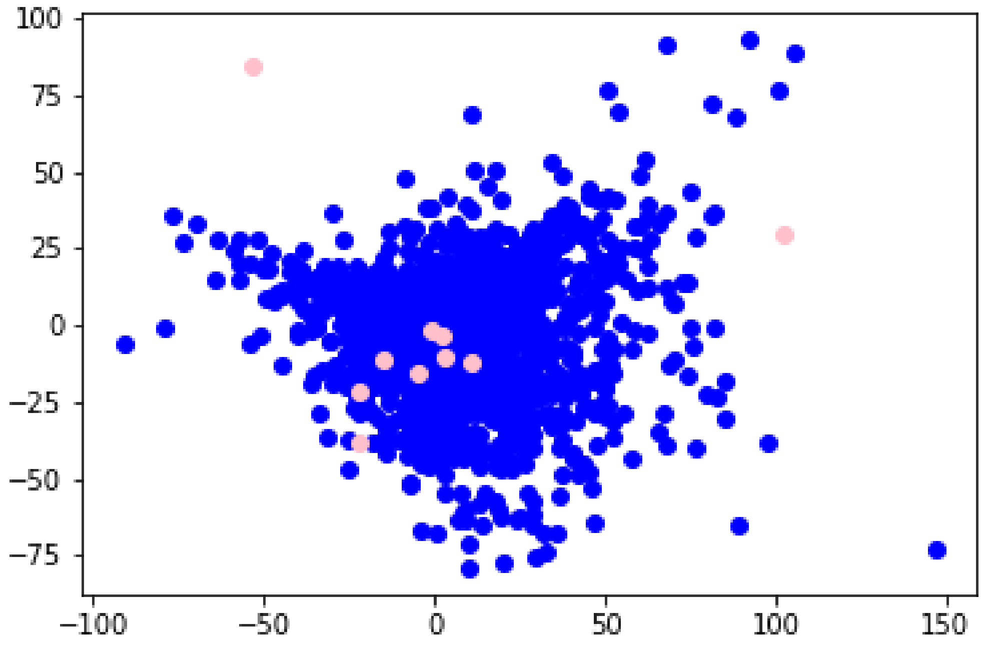

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

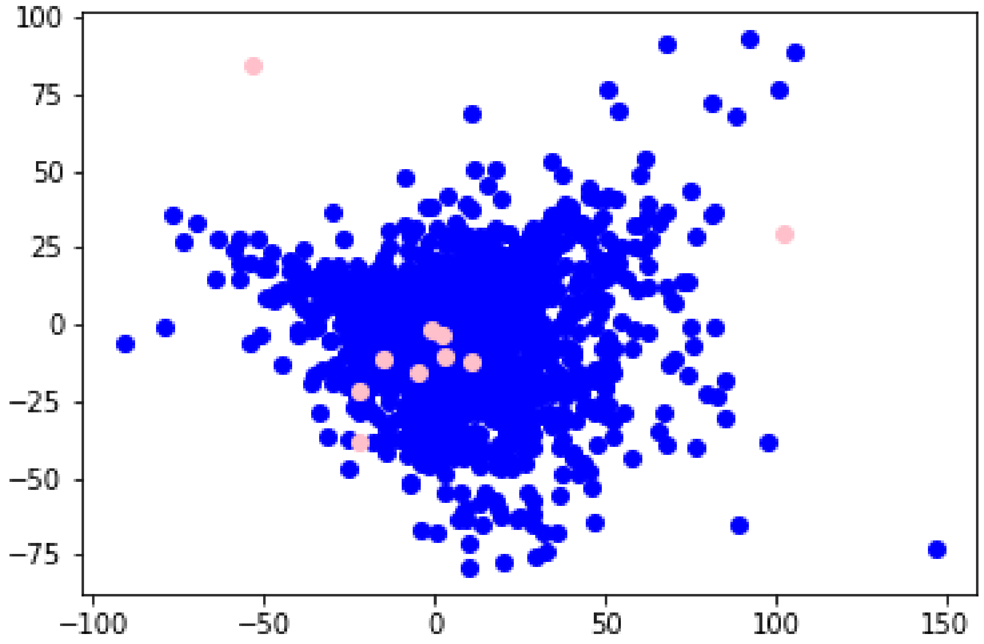

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

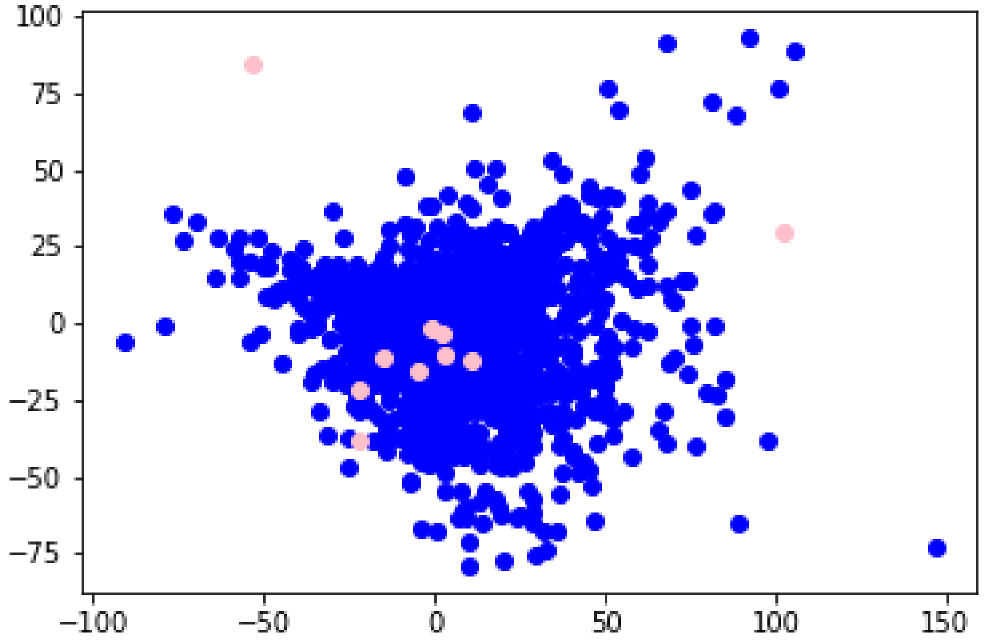

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

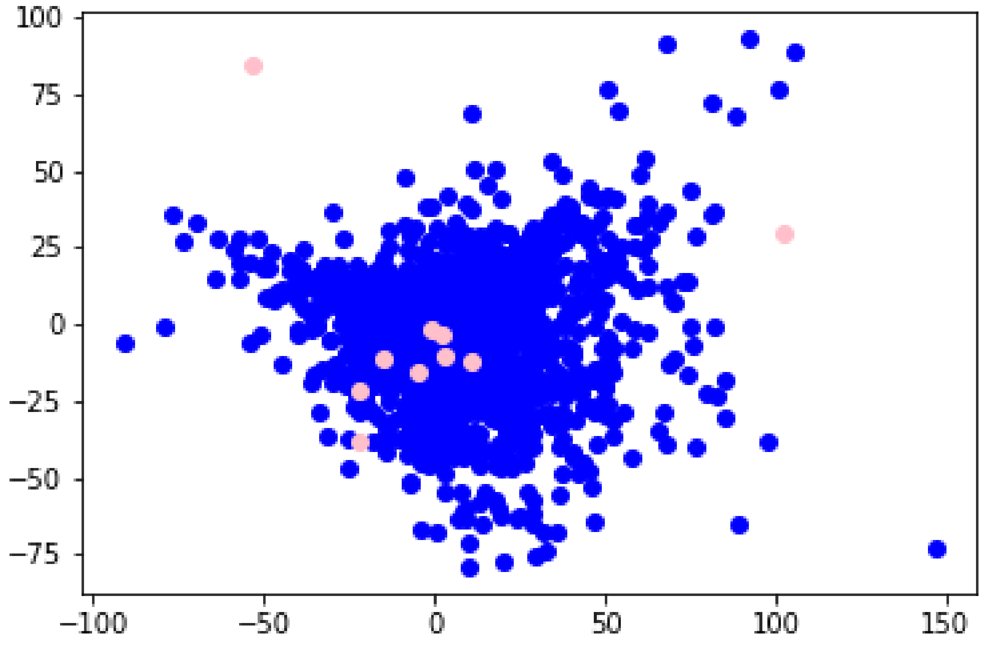

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

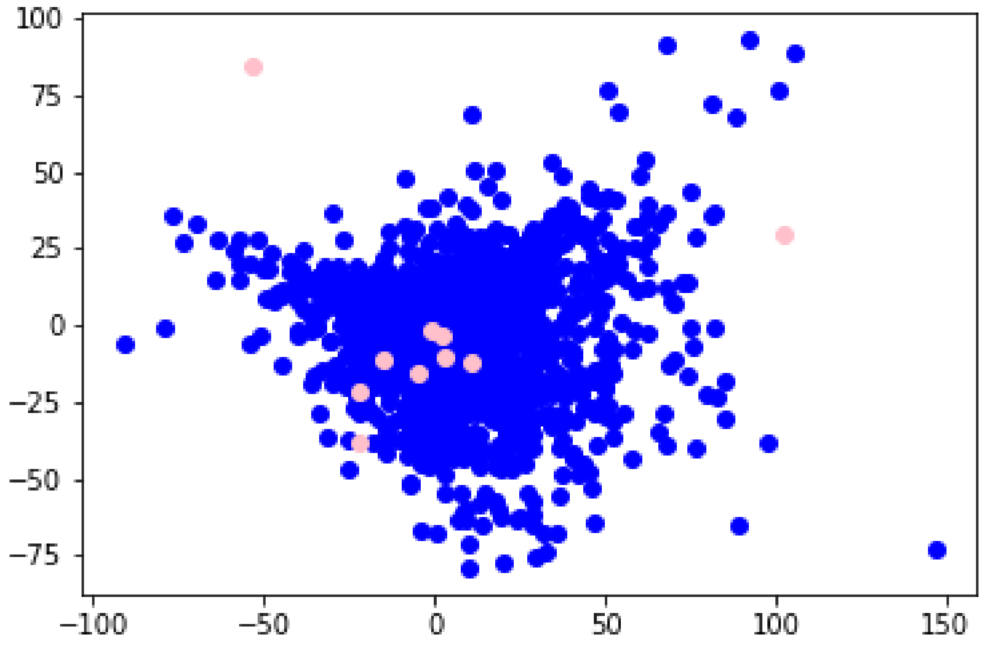

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

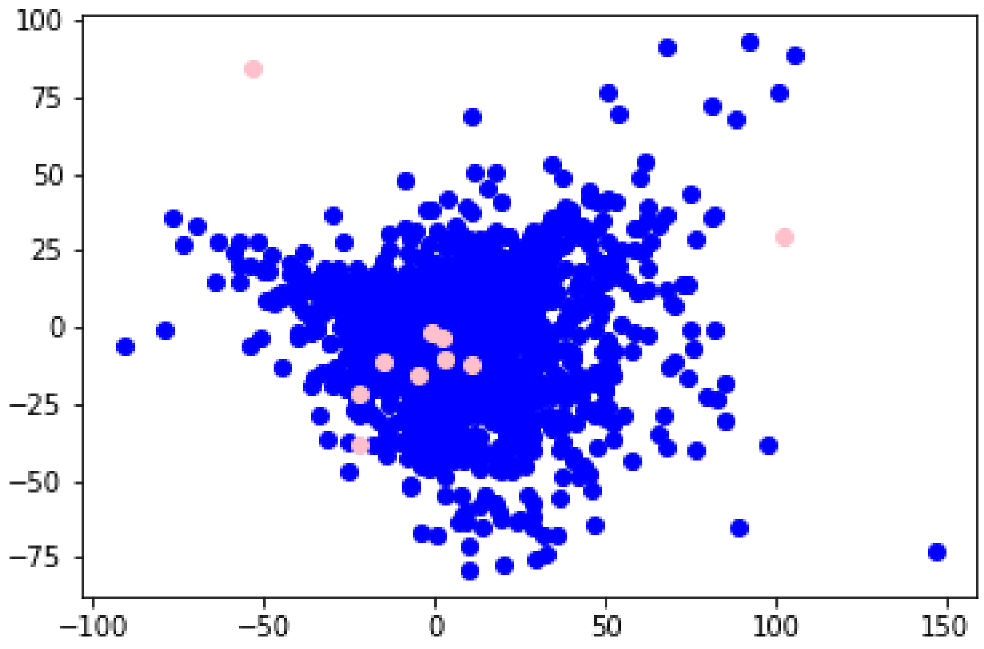

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

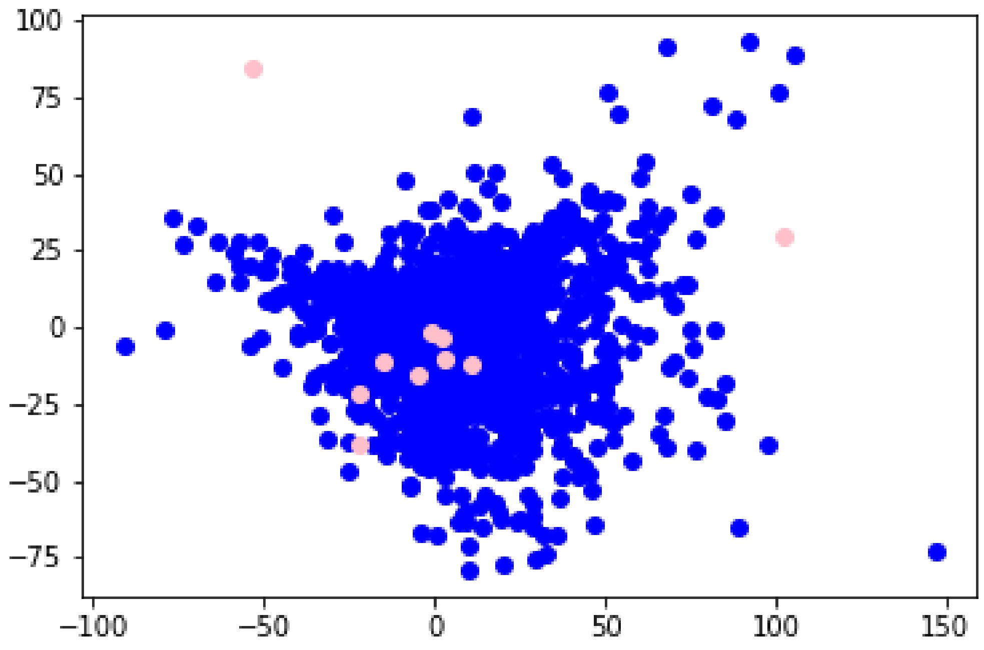

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

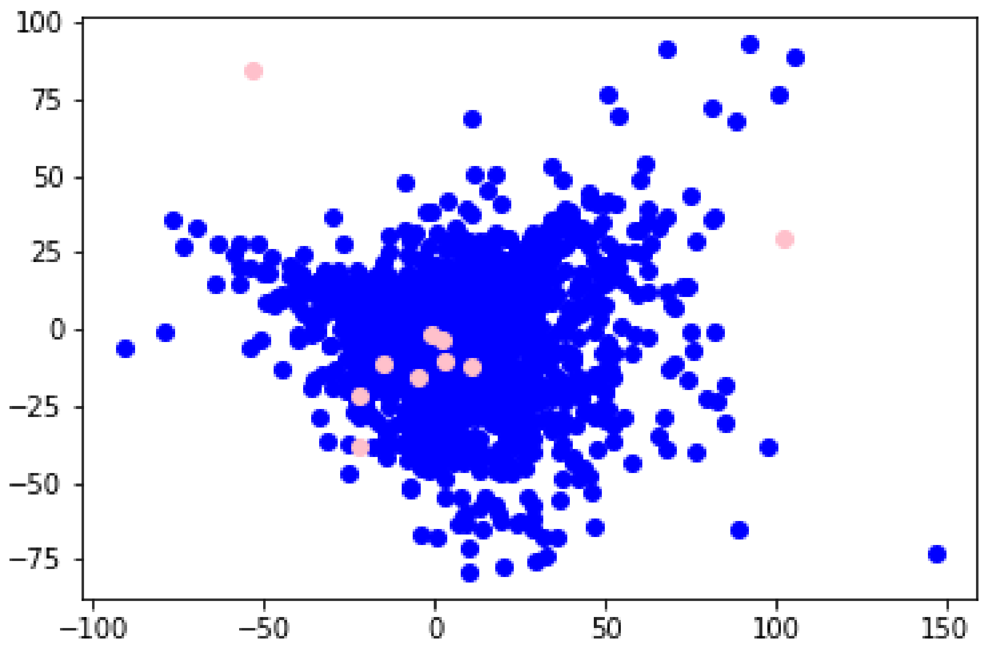

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

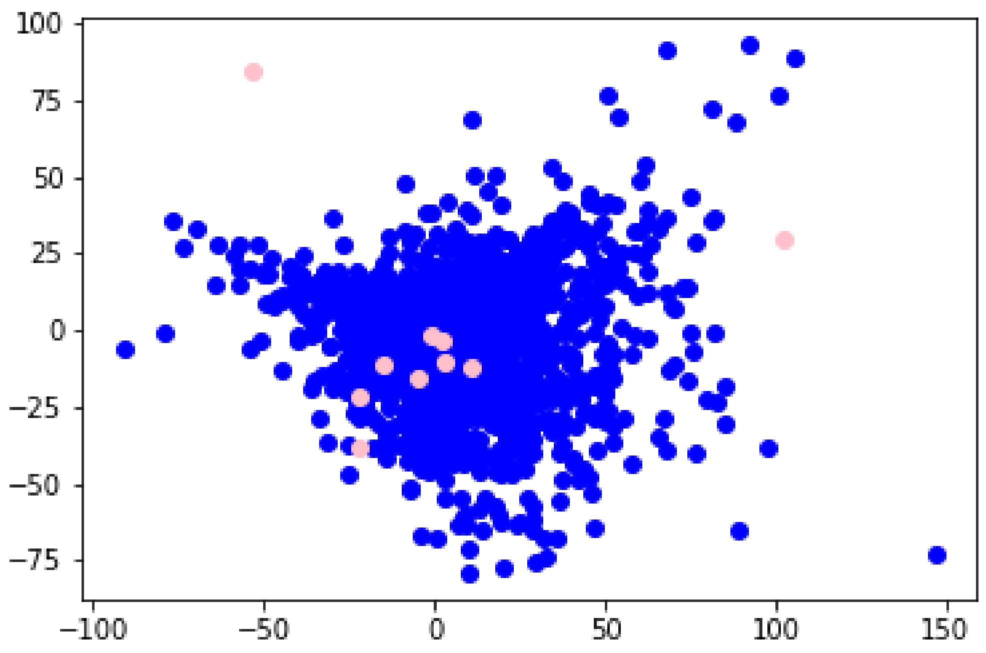

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

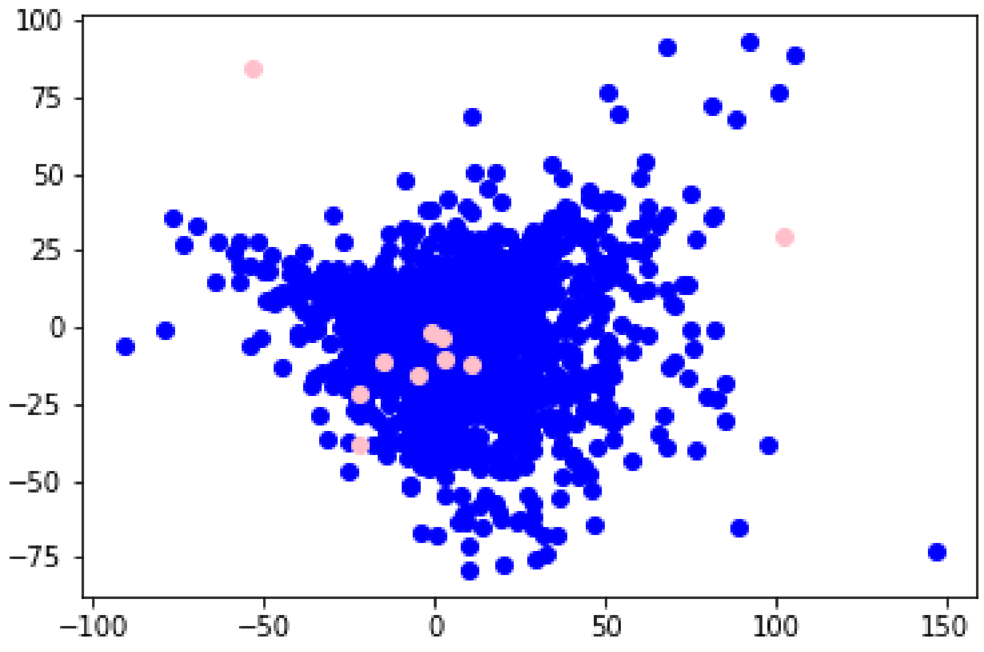

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150

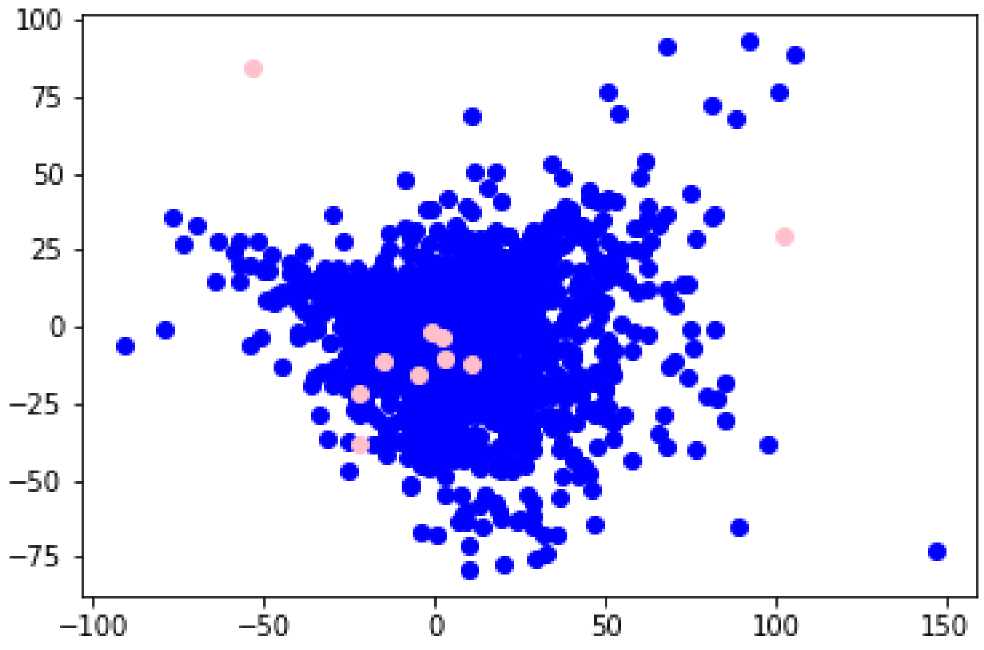

100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150


100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150


100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150


100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
150





[43]:
```
#from sklearn.cluster import KMeans

#report the top 10 words of each cluster in order of the largest positive␣
 ↪distance from the average value across all data.
```

```
def Report_Top_Ten(data_word_doc, n, vocab):
    kmeans = KMeans(n_clusters = k, init='k-means++').fit(data_word_doc)
    prediction = kmeans.predict(data_word_doc)

    TopTen = {}

    for clusters in range (n):
        ind_ = [i for i, x in enumerate (prediction) if x == clusters]
        distant = kmeans.transform(data_word_doc)
        dist_ = distant[:,clusters]
        frequency = np.argsort(dist_)[::]
        RankIndex = [index for index in frequency if index in ind_][:10]
        TopTen[clusters + 1] = [vocab[ind] for ind in RankIndex]

    return TopTen

# Report terms closest to the center
Report_Top_Ten(data_word_doc, 10, vocab)
```

[43]:
```
{1: ['varying'],
 2: ['recalls',
  'clinton',
  'fight',
  'prize',
  'spending',
  'hes',
  'rights',
  'pay',
  'chair',
  'told'],
 3: ['org',
  'sciencemag',
  'vol',
  'thymocytes',
  'endothelial',
  'myeloid',
  'caspase',
  'agonists',
  'nmda',
  'immunoreactive'],
 4: ['ribbon',
  'refinement',
  'helices',
  'refined',
  'reflections',
  'helical',
  'crystallographic',
  'polypeptide',
  'hydrophobic',
  'stabilize'],
 5: ['correspondence',
  'addressed',
  'email',
  'fig',
  'reports',
  'indicated',
  'shown',
  'respectively',
  'observed',
  'indicate'],
 6: ['latitude',
  'geophys',
  'uncertainties',
  'modeled',
  'cooling',
  'uncertainty',
  'variability',
  'intervals',
  'inferred',
  'averaged'],
 7: ['outside'],
 8: ['excitations',
  'coulomb',
  'insulating',
  'spins',
  'resonant',
  'coherence',
  'isotropic',
  'magnetization',
  'anisotropic',
  'fermi'],
 9: ['cdna',
  'blot',
  'incubated',
  'kinase',
  'assays',
  'promoter',
  'intracellular',
  'staining',
  'stained',
  'induction'],
 10: ['lcts',
  'aptamers',
  'neas',
  'trxr',
  'dnag',
  'proteorhodopsin',
  'doxy',
  'lg268',
  'nompc',
  'kcv']}
```

Clustering by terms of words provides a direct insight of the usage of the words. This algorithm might be helpful when we want to figure out the trends of the documents or some specific usage of words.

```
[ ]:
```

3a. k-means is a special case of EM

Set the probability of hidden data given y_i and parameter θ

For E-step computes prob on responsibility to classes

$$r_{ic} = \frac{\pi_c N(x_i | \mu_c, \Sigma_c)}{\sum_{k=1}^{K} \pi_k N(x_i | \mu_k, \Sigma_k)}$$

$$r_k = I(\arg\min_{1 \le k \le K} \| x_i - x_k \|^2 = k), \quad i = 1, 2, \cdots, N$$

For M-step computes means updated

$$m_c = \sum_i r_{ic} \quad \pi_c = \frac{m_c}{m} \quad \mu_c = \frac{1}{m_c} \sum_i r_i x_i$$

$$\Sigma_c = \frac{1}{m_c} \sum_i r_{ic} (x_i - \mu_c)^T (x_i - \mu_i)$$

$$\ln p(X | \pi, \mu, \Sigma) = \sum_{i=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k N(x_i | \mu_k, \Sigma_k) \right)$$

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} \gamma_{ik} y_i}{\sum_{i=1}^{N} \gamma_{ik}}$$

When ~~σ~~ $\sigma = 0$, π_k converges to 1 with the class with the highest probability. π_k converges to 0 o.w. r_k could belong to $\{0, 1\}$ in E or M steps as well.

Q3

November 14, 2019

1 3. EM algorithm and implementation

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import random,math
from sklearn import preprocessing
from sklearn.cluster import KMeans
```

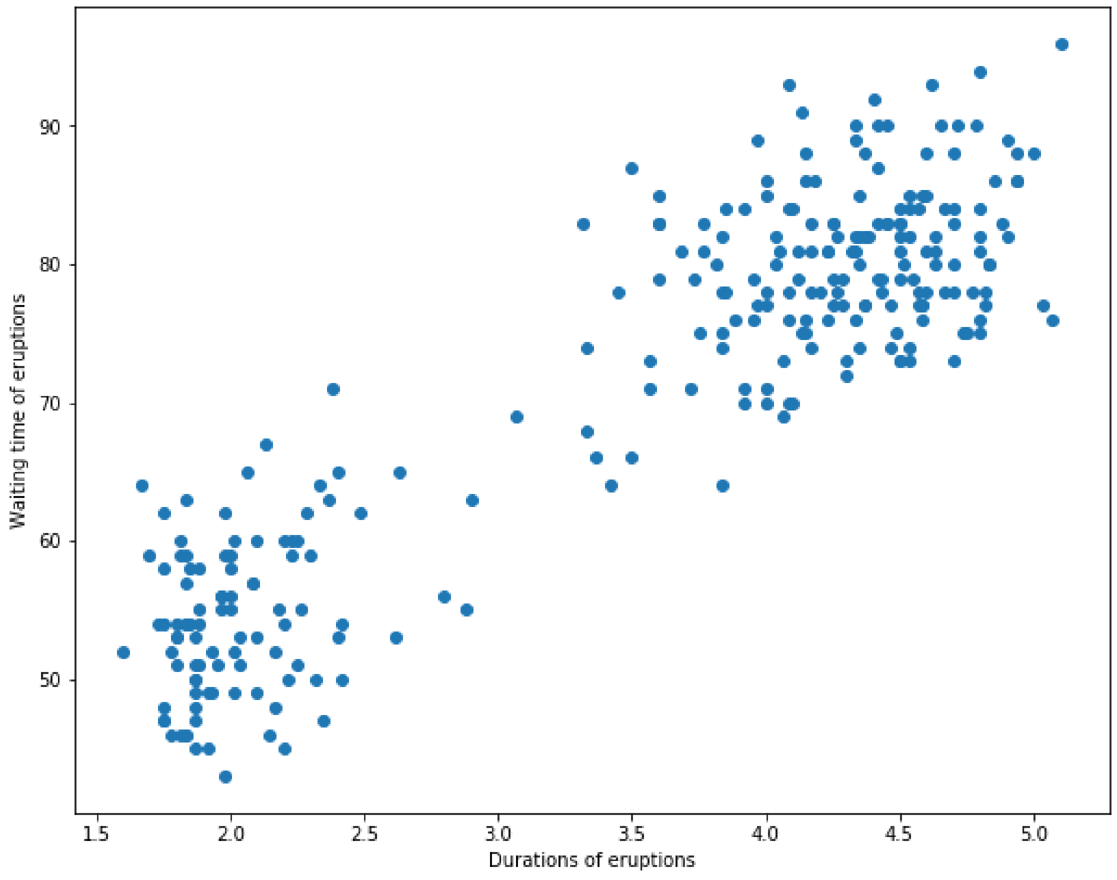
(a) The parameters of Gaussian Mixture Model (GMM) can be estimated via the EM algorithm. Show that the alternating algorithmfor k-means (in Lec. 11) is a special case of the EMalgorithmand show the corresponding objective functions for E-step and M-step.

[ ]:
```
```

(b) Download the Old Faithful Geyser Dataset. The data file contains 272 observations of (eruption time, waiting time). Treat each entry as a 2 dimensional feature vector. Parse and plot all data points on 2-D plane.

[2]:
```
#import numpy as np
#import matplotlib.pyplot as plt
#%matplotlib inline

#load the dataset
data = np.loadtxt('faithful.txt')
#print(data.shape)
#print(data)
#272 observations

#featured as 2d vectors
#Durations of eruptions = data[;,1]
x = data[:,1]
#Waiting time of eruptions = data[;,2]
y = data[:,2]

#plot waiting time between eruptions and durations of eruptions
```

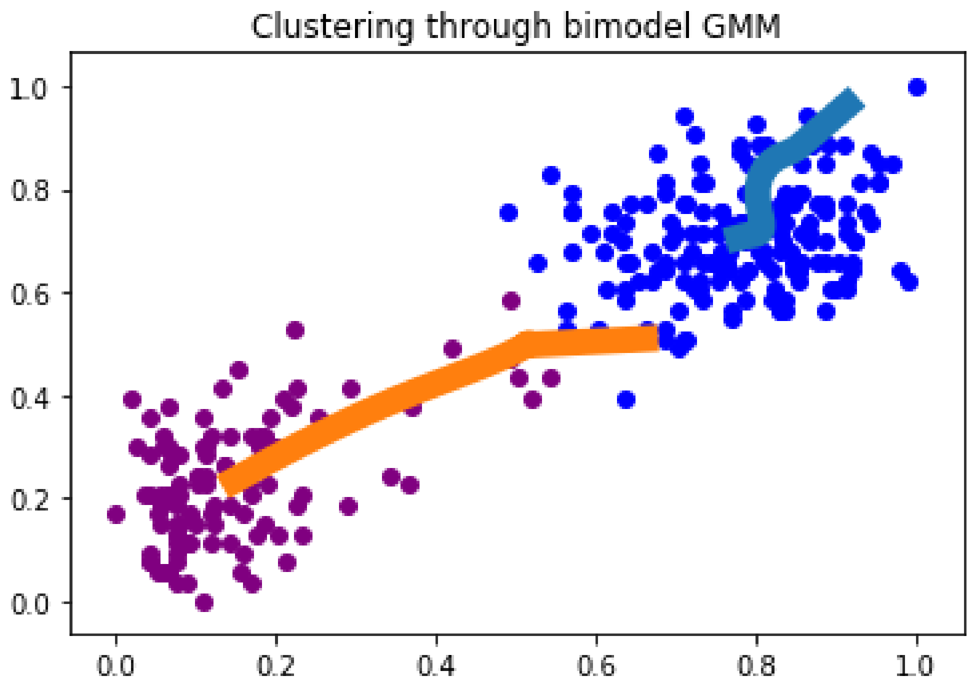
```
fig, ax = plt.subplots()
fig.set_size_inches(10, 8)
ax.scatter(x, y)
plt.xlabel("Durations of eruptions")
plt.ylabel("Waiting time of eruptions")
plt.show()
```



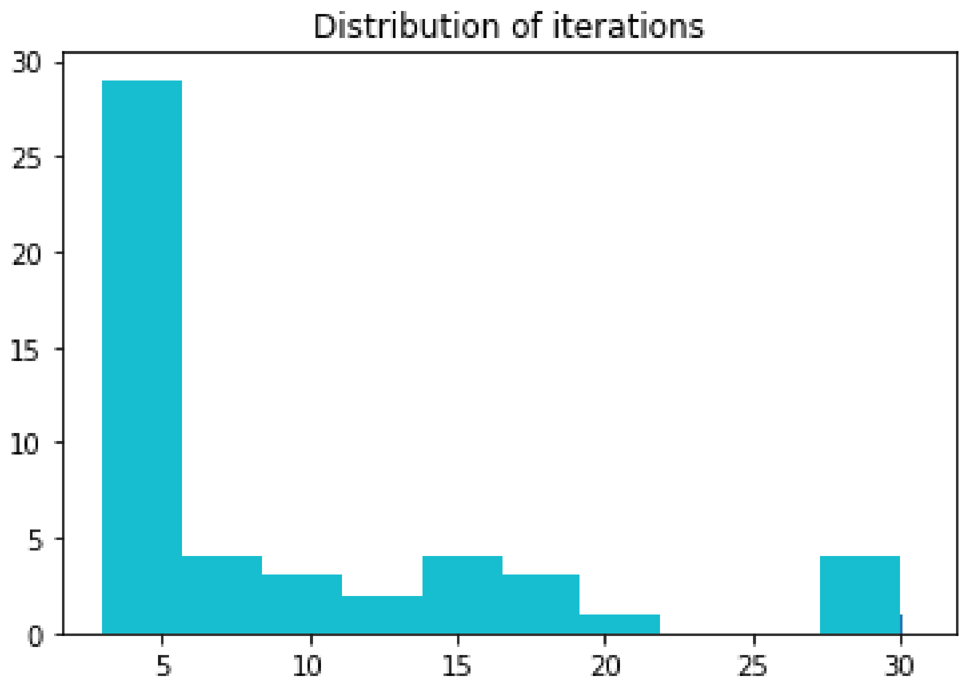
(c) Implement a bimodal GMMmodel to fit all data points using EMalgorithm. Explain the reasoning behind your termination criteria. For this problem, we assume the covariance matrix is spherical (i.e., it has the formof ¿2I for scalar ¿) and you can randomly initialize Gaussian parameters. For evaluation purposes, please submit the following figures: Plot the trajectories of two mean vectors in 2 dimensions (i.e., coordinates vs. iteration). Run your program for 50 times with different initial parameter guesses. Show the distribution of the total number of iterations needed for algorithmto converge.

Reason for termination: For each (G) of the data points, set it to (G-Gmin)/(Gmax-Gmin). Let the threshold to be as small as 1e-20, so when the difference of the log likelihood is smaller than the threshold, we will break.

[3]:
```
#import random,math
#from sklearn import preprocessing

#implement a biomodal GMMmodel
def GMM(G,M):              #M = 2
    NumofObservations, Dimension = np.shape(G)
    N_of_O = 272                                                      ␣
 ↪#NumofObservations = 272
    D = 2                                                                 ␣
 ↪#Dimension = 2
    idx_ = random.sample(range(N_of_O), D)            #Take random example     ␣
 ↪
    centriod = G[idx_]
    vector_1 = []
    vector_1.append(centriod[0])
    vector_2 = []
    vector_2.append(centriod[1])

    [means,Pi,covs] = take_initial(G, centriod, M, NumofObservations, Dimension)

    # Normalize so that the responsibility matrix is row stochastic
    while True:
        #calculate Gaussian posterior probability = gpp
        #import numpy as np
        gpp = GPP(means,covs,G,M,NumofObservations,Dimension)
        gamma = gpp * np.tile(Pi,(NumofObservations,1))
        gamma = gamma  / np.tile((np.sum(gamma,axis=1)),(M,1)).T
        Num_k = np.sum(gamma,axis=0)
        means = np.dot(np.dot(np.diag(1 / Num_k),gamma.T), G)
        vector_1.append(means[0])
        vector_2.append(means[1])

        Pi = np.sum(gamma,axis=0) / NumofObservations
        #covariances
        for mp in range(M):
            new = G - np.tile(means[mp],(NumofObservations,1))
            covs[:,:,mp] = (np.dot(np.dot(new.T,np.diag(gamma[:,mp])),new)) /␣
 ↪Num_k[mp]

        #set the log likelihood to be -infinite
        Logli = -np.inf
        #termination/check for convergence
        L = np.sum(np.log(gpp*(Pi.T)))
        if L-Logli < 1e-20:
            break
        Logli = L
    return gpp, vector_1, vector_2

#define Gaussian posterior probability
def GPP(means, covs, G, M, NumofObservations, Dimension):
    gpp = np.zeros((NumofObservations, M))
    for m in range(M):
        new = G - np.tile(means[m],(NumofObservations,1))
        new_cov = np.linalg.pinv(covs[:,:,m])
        temp = np.sum(np.dot(new,new_cov) * new, axis = 1)
        coeficient = (2*np.pi)**(-Dimension/2) * np.sqrt(np.linalg.det(new_cov))
        gpp[:,m] = coeficient * np.exp(-0.5 * temp)
    return gpp

#define randomly initialize Gaussian parameters
def take_initial(G, centriod, M, NumofObservations, Dimension):
    means = centriod;
    Pi = np.zeros([1, M])
    #cov matrix of k components
    covs = np.zeros([Dimension, Dimension, M])
    dist = np.tile(np.sum(G * G, axis = 1),(M,1)).T + np.tile(np.sum(means *␣
 ↪means,axis = 1).T,(NumofObservations,1)) - 2 * np.dot(G,means.T)
    row_index = np.argmin(dist,axis = 1)

    #covariances
    for m in range(M):
        xm = G[row_index==m]
        Pi[0][m] = float(np.shape(xm)[0]) / NumofObservations
        covs[:,:,m] = np.cov(xm.T)

    return means,Pi,covs

#from sklearn import preprocessing
X = data[:,[1,2]]
X = preprocessing.MinMaxScaler().fit_transform(X)
gpp_1, vector_1, vector_2 = GMM(X,2)
index = np.argmax(gpp_1,axis=1)

#plot the dataset
plt.scatter(X[index==0][:,0],X[index==0][:,1],c="blue")
plt.scatter(X[index==1][:,0],X[index==1][:,1],c="purple")

#plot the trajectories of two mean vectors in 2 dimensions
plt.plot(np.mat(vector_1)[:,0],np.mat(vector_1)[:,1],linewidth = 10)
plt.plot(np.mat(vector_2)[:,0],np.mat(vector_2)[:,1],linewidth = 10)
plt.title("Clustering through bimodel GMM")
plt.show()
```



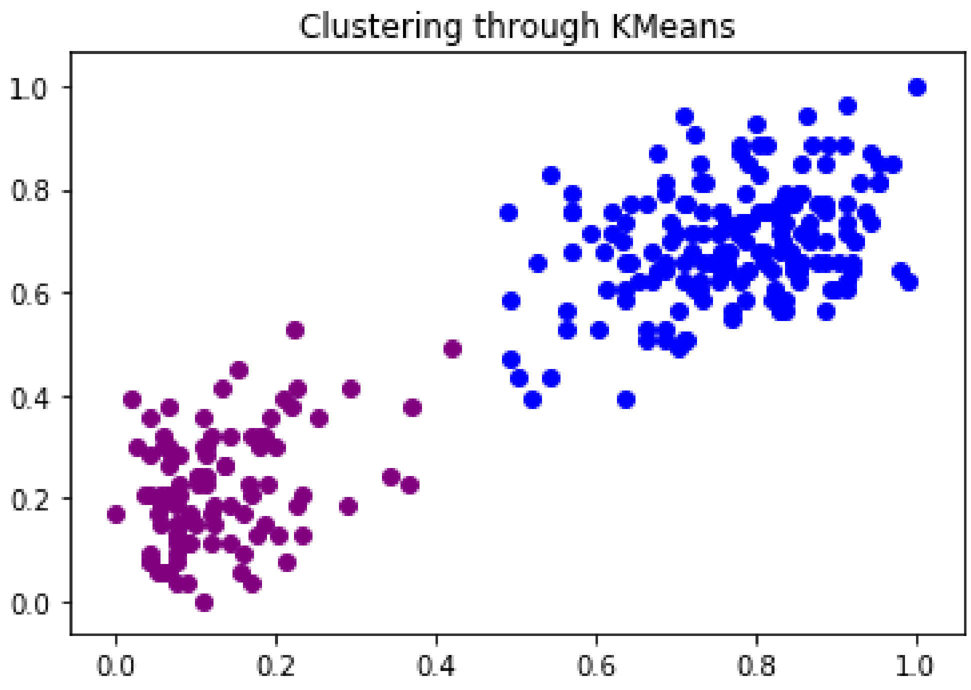
[7]:
```
#Run for 50 times with different initial parameter guesses.
#Show the distribution of the total number of iterations needed for algorithmto␣
↪converge
dist_it = []
for i in range(50):
    gpp_1, vector_1, vector_2 = GMM(X,2)
    dist_it.append(np.mat(vector_1).shape[0])
    plt.title("Distribution of iterations")
    plt.hist(dist_it)
```



(d) Repeat the task in (c) but with the initial guesses of the parameters generated from the following process: Run a k-means algorithm over all the data points with K Æ 2 and label each point with one of the two clusters. Estimate the first guess of the mean and covariance matrices using maximumlikelihood over the labeled data points. Compare the algorithm performances of (c) and (d).

[8]:
```
#Clustering through KMeans
#from sklearn.cluster import KMeans
#Run a k-means algorithm over all the data points with K Æ 2 and label each␣
 ↪point with one of the two clusters
X = data[:,[1,2]]
X = preprocessing.MinMaxScaler().fit_transform(X)
y_prediction = KMeans(n_clusters=2, random_state=150).fit_predict(X)
plt.scatter(X[y_prediction==0][:,0],X[y_prediction==0][:,1],c = 'blue')
plt.scatter(X[y_prediction==1][:,0],X[y_prediction==1][:,1],c = "purple")
plt.title("Clustering through KMeans")
```

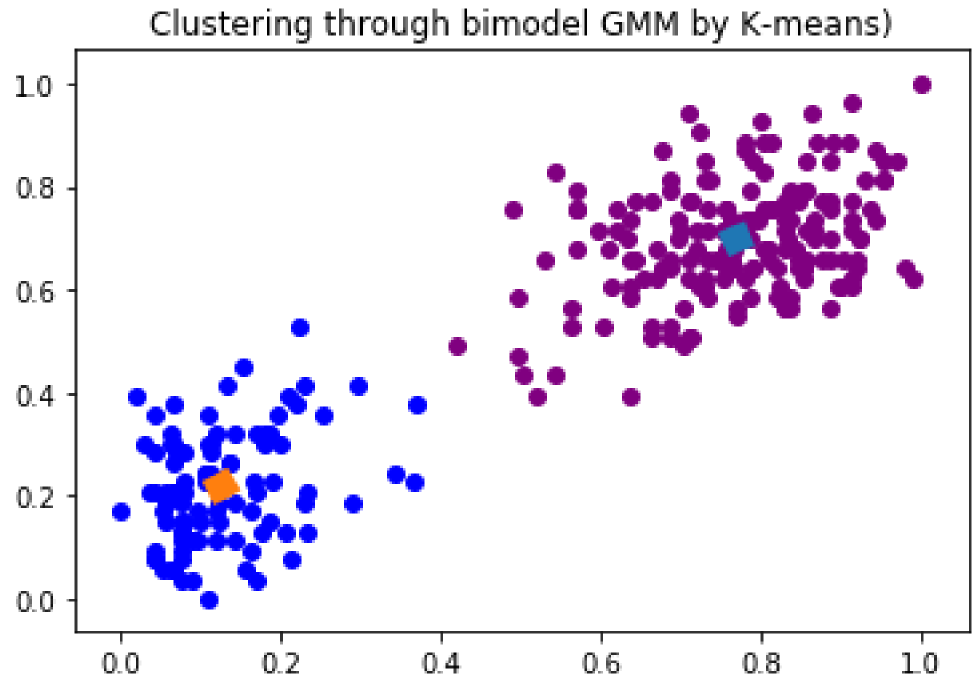
[8]:
```
Text(0.5, 1.0, 'Clustering through KMeans')
```

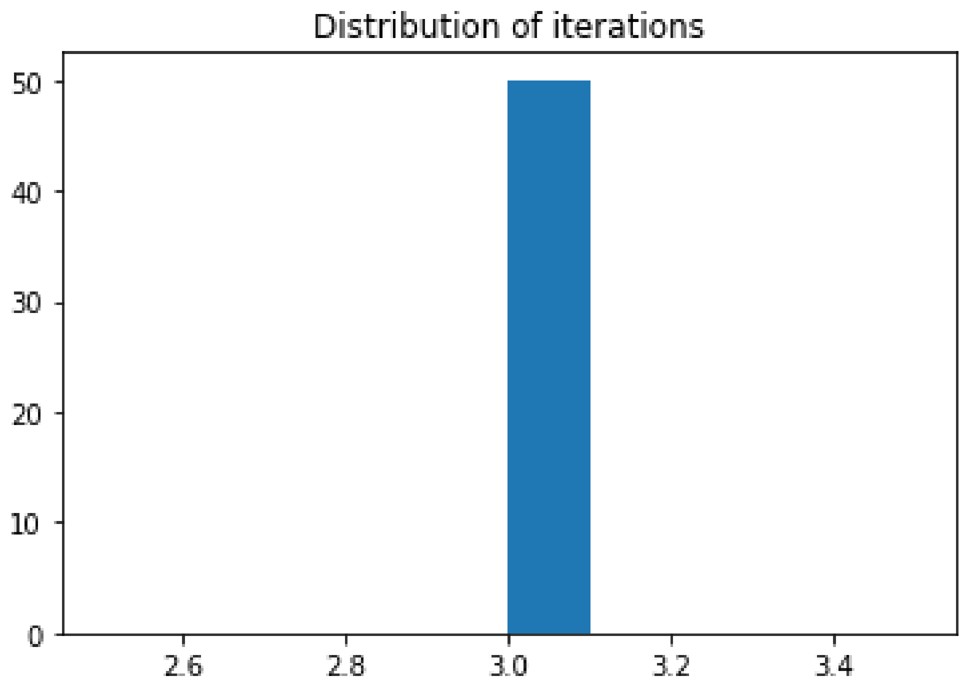


[10]:
```
def kmeans(G,y_prediction,M,NumofObservations,Dimension):
    means = np.zeros([M, 2])
    Pi = np.zeros([1, M])
    covs = np.zeros([Dimension, Dimension, M])
    for m in range(M):
        Gm = G[y_prediction == m]
        means[m,:] = np.mean(G[y_prediction == m],axis = 0)
        Pi[0][m] = float(np.shape(Gm)[0]) / NumofObservations
        covs[:,:,m] = np.cov(Gm.T)

    return means,Pi,covs


#Repeat the task in (c) but with the initial guesses of the parameters
#Gaussian posterior probability
def GPP(means,covs,G,M,NumofObservations,Dimension):
    gpp = np.zeros((NumofObservations,M))
    for m in range(M):
        new = G - np.tile(means[m],(NumofObservations,1))#X-means
        new_cov = np.linalg.pinv(covs[:,:,m])
        temp = np.sum(np.dot(new,new_cov) * new,axis=1)
        coeficient = (2*np.pi)**(-Dimension/2) * np.sqrt(np.linalg.det(new_cov))
        gpp[:,m] = coeficient * np.exp(-0.5 * temp)
    return gpp
```

```
#implement a biomodal GMMmodel by Kmeans
def GMM(G,M):
    NumofObservations, Dimension = np.shape(G)
    y_prediction = KMeans(n_clusters=M, random_state=150).fit_predict(G)
    [means,Pi,covs] = kmeans(G,y_prediction,M,NumofObservations,Dimension)

    vector_1 = []
    vector_2 = []

    vector_1.append(means[0])
    vector_2.append(means[1])


    # Normalize so that the responsibility matrix is row stochastic
    while True:
        gpp = GPP(means,covs,G,M,NumofObservations,Dimension)
        gamma = gpp * np.tile(Pi,(NumofObservations,1))
        gamma = gamma / np.tile((np.sum(gamma,axis=1)),(M,1)).T
        Nm = np.sum(gamma,axis=0)
        means = np.dot(np.dot(np.diag(1 / Nm),gamma.T),G)
        vector_1.append(means[0])
        vector_2.append(means[1])
        Pi = Nm / NumofObservations
        for km in range(M):
            new = G - np.tile(means[km],(NumofObservations,1))
            covs[:,:,km] = (np.dot(np.dot(new.T,np.diag(gamma[:,km])),new)) /␣
 ↪Nm[km]

        #set the log likelihood to be -infinite
        Logli = -np.inf
        L = np.sum(np.log(gpp*(Pi.T)))
        if L-Logli < 1e-20:
            break
        Logli = L

    return gpp, vector_1, vector_2



#from sklearn import preprocessing
X = data[:,[1,2]]
NumofObservations, Dimension = np.shape(X)
X = preprocessing.MinMaxScaler().fit_transform(X)

gpp_1, vector_1, vector_2 = GMM(X,2)
index = np.argmax(gpp_1,axis = 1)
plt.figure(1)
```

```
#plot the dataset
plt.scatter(X[index==0][:,0],X[index==0][:,1],c = 'blue')
plt.scatter(X[index==1][:,0],X[index==1][:,1],c = 'purple')
plt.title("Clustering through bimodel GMM by K-means")
#plot the trajectories of two mean vectors in 2 dimensions
plt.plot(np.mat(vector_1)[:,0],np.mat(vector_1)[:,1],linewidth = 10)
plt.plot(np.mat(vector_2)[:,0],np.mat(vector_2)[:,1],linewidth = 10)

#Run for 50 times with different initial parameter guesses.
#Show the distribution of the total number of iterations needed for algorithm␣
 ↪to converge
dist_it = []
    for i in range(50):
        gpp_1, vector_1, vector_2 = GMM(X,2)
        dist_it.append(np.mat(vector_1).shape[0])

    plt.figure(2)
    plt.title("Distribution of iterations")
    plt.hist(dist_it)
```





Compare the algorithm performances of (c) and (d): Obviously, the distribution of iterations by kmeans is more nomarlized than that by randomly GMM. The GMM algorithms implemented by kmeans tends to be more stable and requires less iterations to reach its convergence.

```
[ ]:
```

hw3_q4

November 13, 2019

[79]:
```
import numpy as np
import numpy as np
from sklearn.manifold import MDS
from sklearn.cluster import KMeans
import seaborn as sns; sns.set()
%matplotlib inline
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
import random
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as shc
import numpy.linalg as la
from sklearn.metrics.pairwise import euclidean_distances
import sys
import warnings
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

[80]:
```
data = np.load("mds-population.npz")
print(data['D']) # Distance matrix
print(data['population_list']) # List of populations
```

```
[[  0.  87.  12. … 462. 559. 553.]
 [ 87.   0.  87. … 210. 359. 285.]
 [ 12.  87.   0. … 317. 401. 377.]
 …
 [462. 210. 317. …   0. 226. 173.]
 [559. 359. 401. … 226.   0. 127.]
 [553. 285. 377. … 173. 127.   0.]]
[b'Bantu' b'E. Afncan' b'Nilo-Saharan' b'W. African' b'San' b'Barter'
 b'Mbuti' b'Indian' b'Iranian' b'Near Eastern' b'Uralic' b'Ainu'
 b'Japanese' b'Korean' b'Mon Khmer' b'Thai' b'Dravidian' b'Mongol Tungus'
 b'Tibetan' b'Indonesian' b'Malaysian' b'Filipino' b'N. Turkic'
 b'S. Chinese' b'Basque' b'Lapp' b'Sardinian' b'Danish' b'English'
 b'Greek' b'Italian' b'C Amerind' b'Eskimo' b'Na-Dene' b'N. American'
 b'S. American' b'Chukchi' b'Melanesian' b'Micronesian' b'Polynesian'
 b'New Guinean' b'Australian']
```

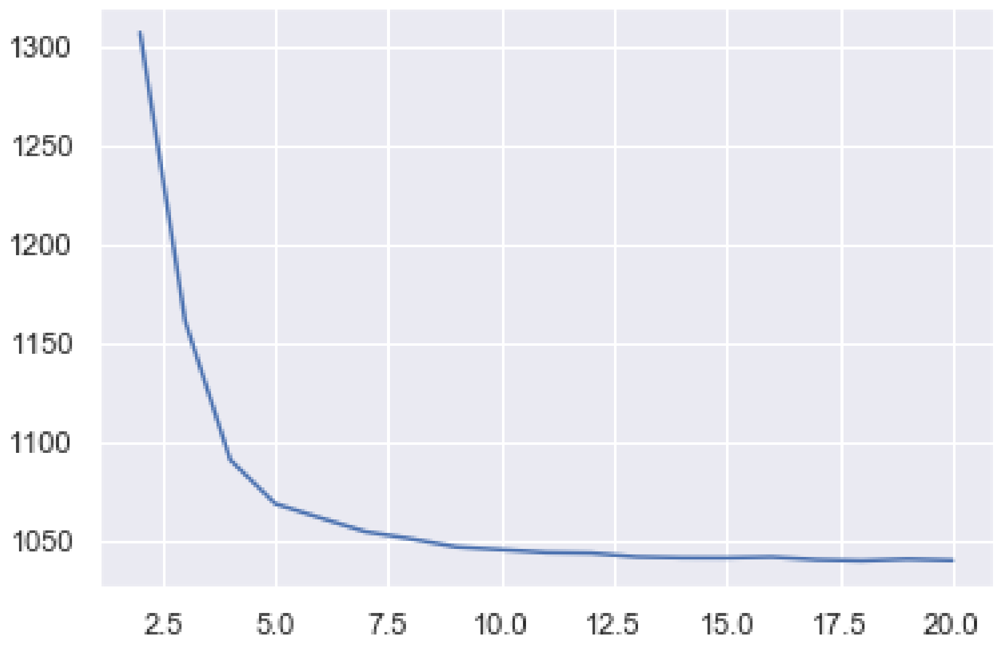
1 a)

i:What assumptions are being made? Under what circumstances could this fail? How could we measure how much information is being lost?

Assumptions: L2 norm could approximate Nei's distance. We are considering linearembedding. We can compare euclidean distance matrix of MDS and original matrix and compare their difference. We first project original dataset to a high dimention space and then use PCA to get major dimentions.

[107]:
```
ls = []
D_norm = la.norm(data['D'])
def mds(m):
    embedding = MDS(n_components=m)
    X = embedding.fit_transform(data['D'])
    D = np.matmul(X, X.T)
    loss = la.norm(data['D'] - D) / D_norm
    ls.append(loss)
for m in range (2, 21):
    mds(m)
```

[109]:
```
fig, ax = plt.subplots()
plt.plot(list(range(2,21)), ls)
plt.show()
```

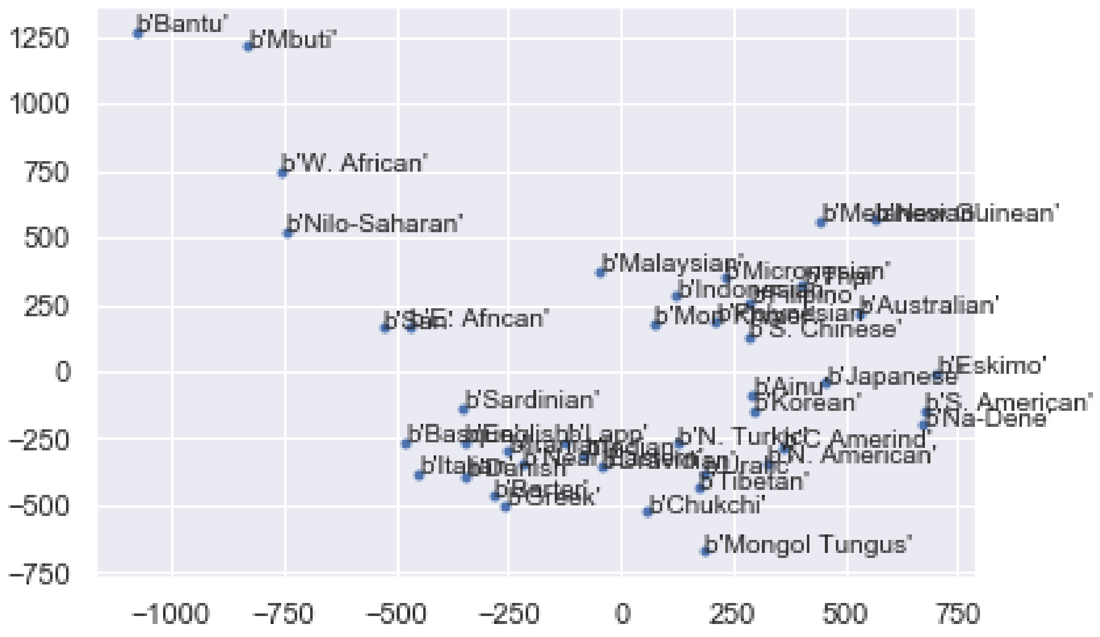


ii We can conclude that 4 dimensions are necessary to capture most of the variation in the data.

iii)

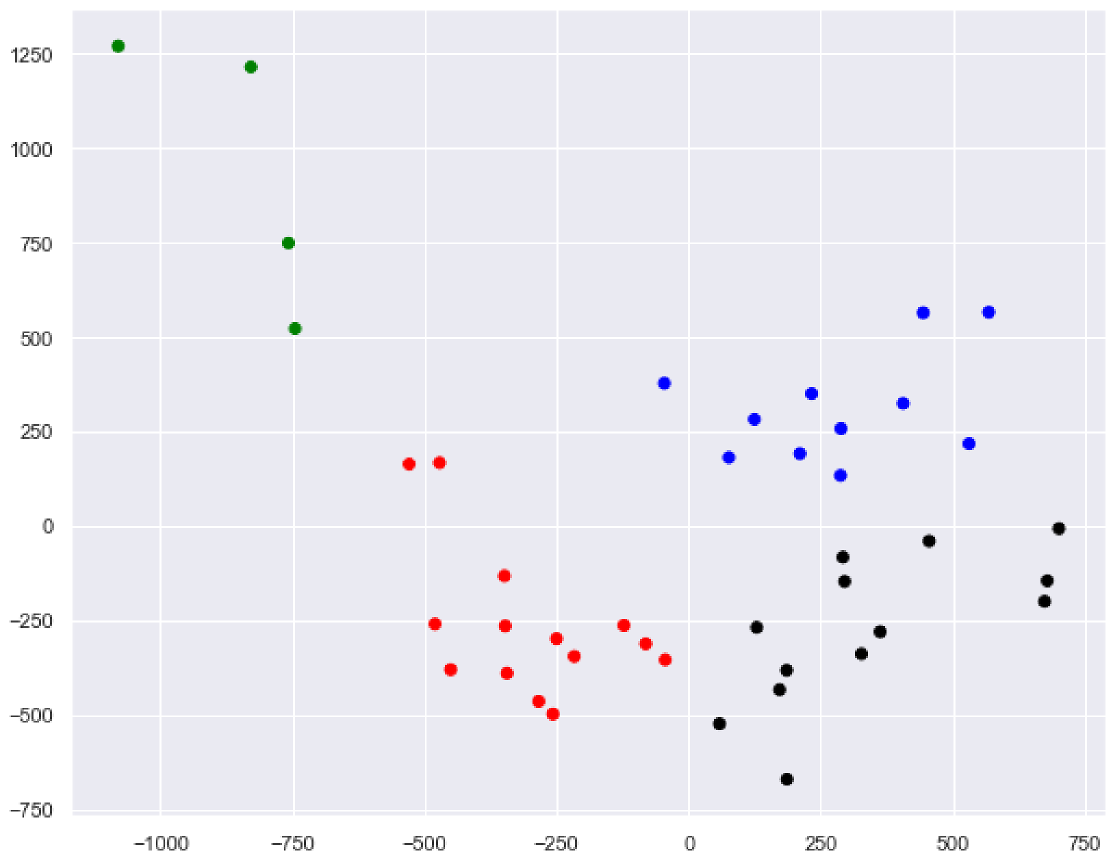
[83]:
```
embedding = MDS(2)
new_data = embedding.fit_transform(data['D'])
```

[84]:
```
fig, ax = plt.subplots()
plt.plot(new_data[:,0],new_data[:,1],'.')
for i, txt in enumerate (data['population_list']):
    ax.annotate(txt, (new_data[i,0],new_data[i,1]))
plt.show()
```



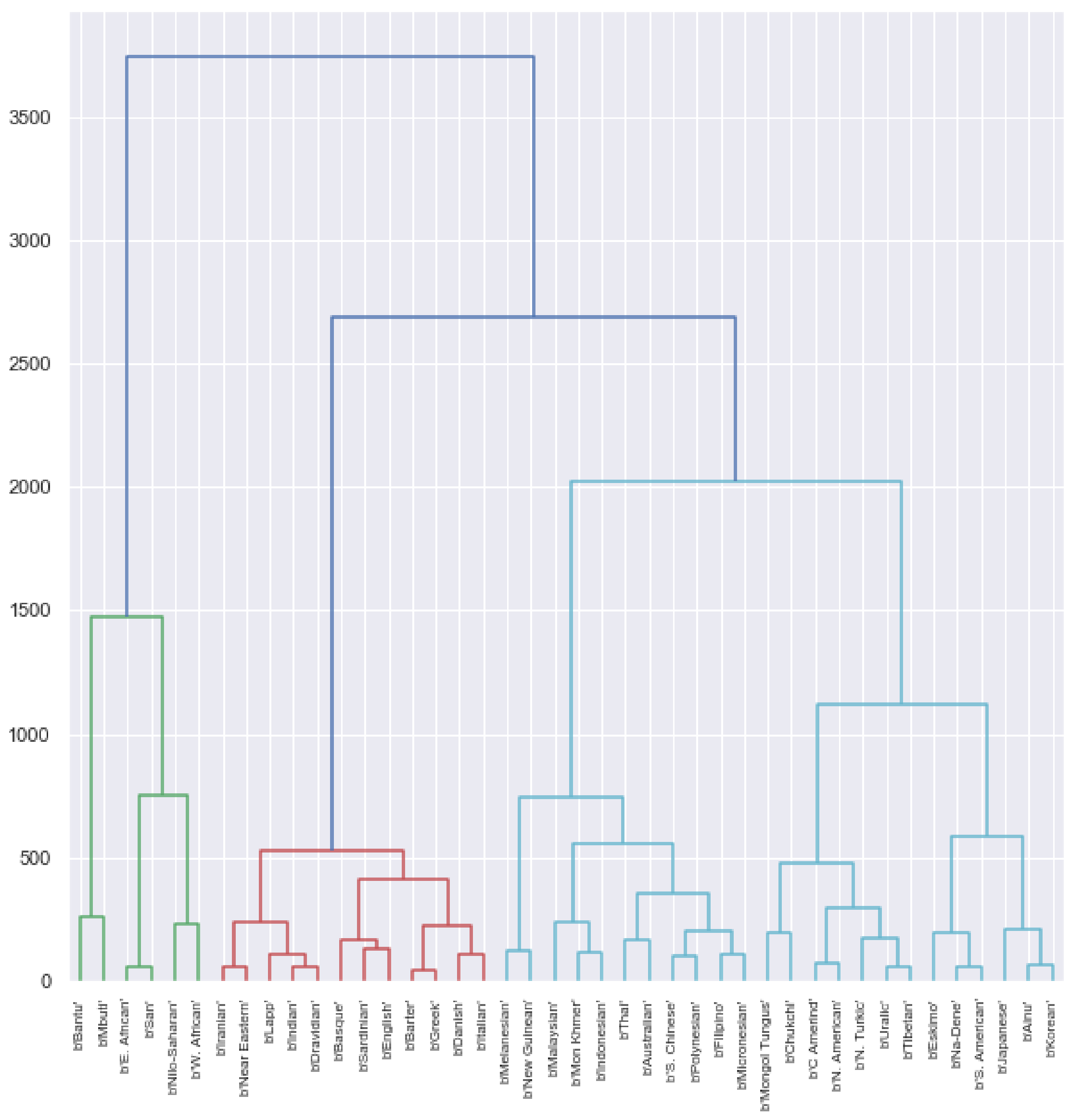
b) k-means on 2D embedding

[100]:
```
kmeans = KMeans(n_clusters = 4, random_state = 0)
labels = kmeans.fit_predict(new_data)
fig, ax = plt.subplots()
fig.set_size_inches(10,8)

colordic = {1:'red',2:'blue',0:'green',3:'black'}
col = [colordic[i] for i in labels]

plt.scatter(new_data[:,0], new_data[:,1], c = col)
plt.show()
```

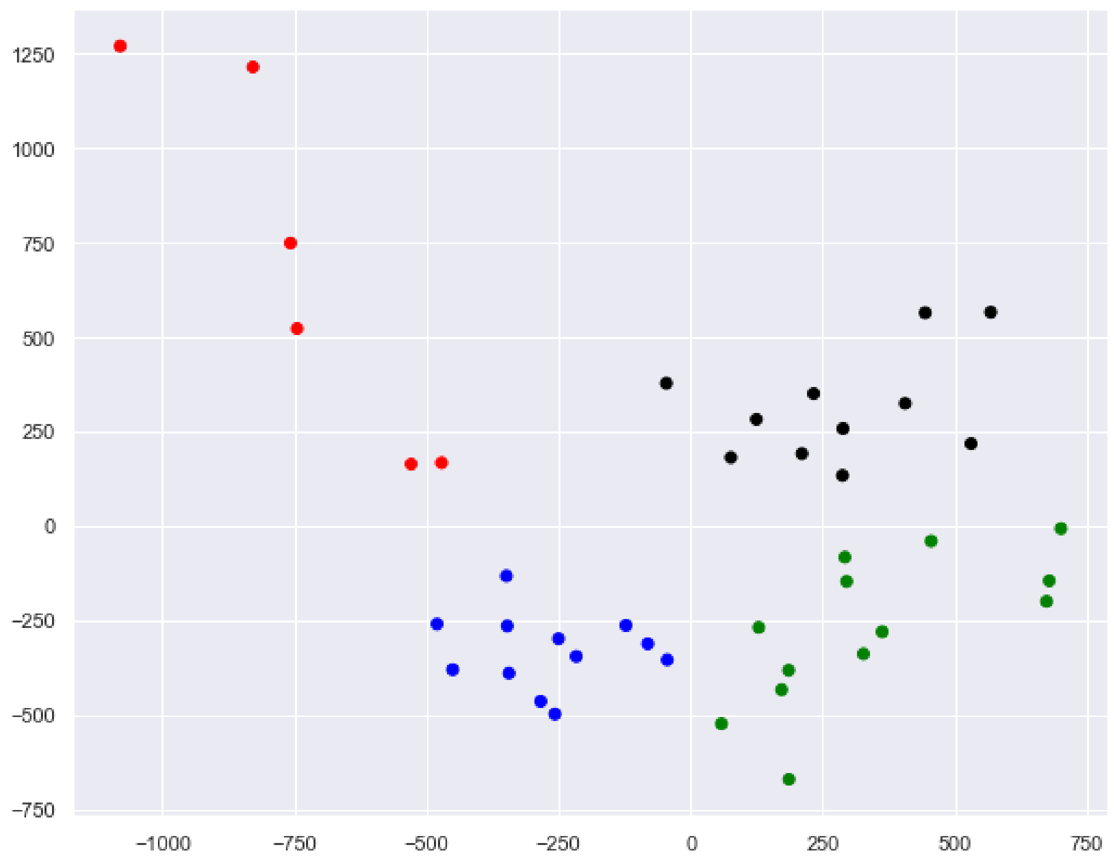


c) Comparing hierarchical clustering with K-Means.

```
[92]: plt.figure(figsize=(10,10))
Z = shc.linkage(new_data, method='ward')
dendro = shc.dendrogram(shc.linkage(new_data, method='ward'), labels =␣
 ↪data['population_list'])
```



[99]:
```python
rt = fcluster(Z, t= 1500, criterion='distance')
fig, ax = plt.subplots()
fig.set_size_inches(10, 8)

colordic = {1:'red',2:'blue',4:'green',3:'black'}
col = [colordic[i] for i in rt]

plt.scatter(new_data[:, 0], new_data[:, 1], c=col)
plt.show()
```

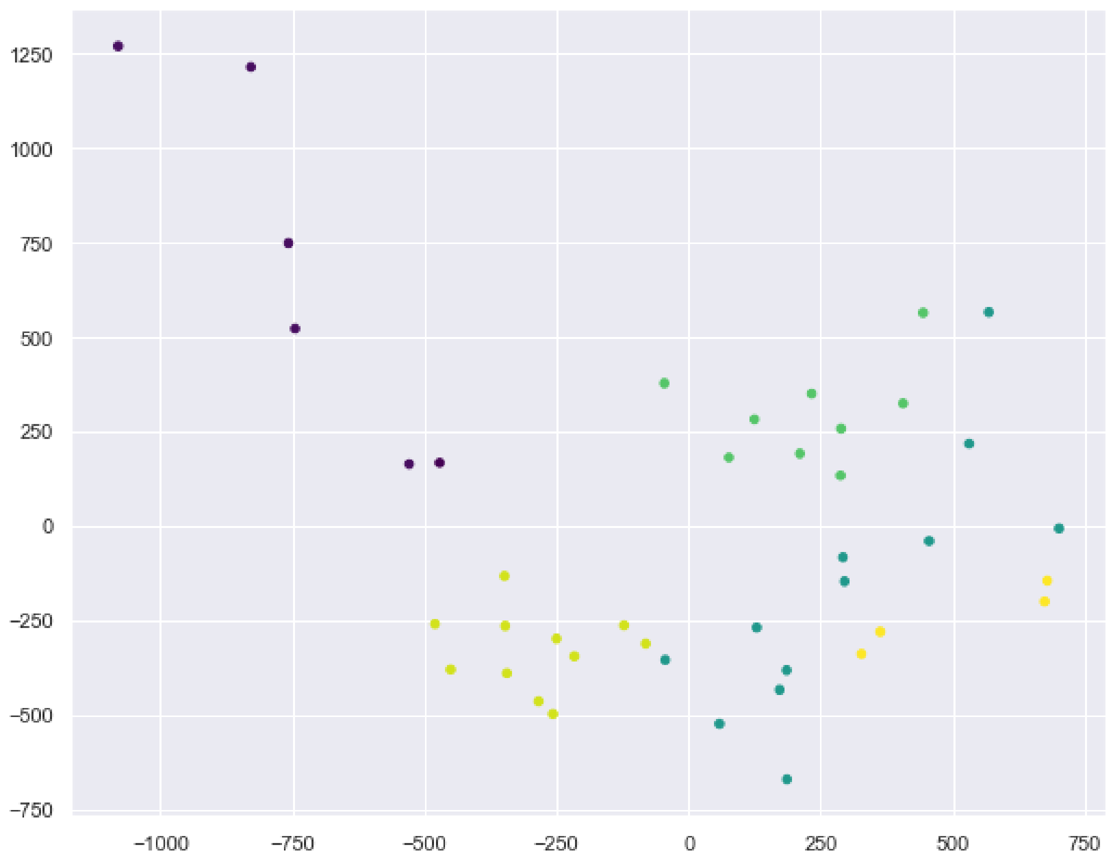


This cluster is similar to k means clustering.

d)Compare k-medoids with k-means

[101]:
```
## Code from: https://github.com/salspaugh/machine_learning/blob/master/
↪clustering/kmedoids.py
def cluster(distances, k):

    m = distances.shape[0] # number of points

    # Pick k random medoids.
    curr_medoids = np.array([-1]*k)
    while not len(np.unique(curr_medoids)) == k:
        curr_medoids = np.array([random.randint(0, m - 1) for _ in range(k)])
    old_medoids = np.array([-1]*k) # Doesn't matter what we initialize these to.
    new_medoids = np.array([-1]*k)

    # Until the medoids stop updating, do the following:
    while not ((old_medoids == curr_medoids).all()):
        # Assign each point to cluster with closest medoid.
        clusters = assign_points_to_clusters(curr_medoids, distances)
        # Update cluster medoids to be lowest cost point.
        for curr_medoid in curr_medoids:
            cluster = np.where(clusters == curr_medoid)[0]
            new_medoids[curr_medoids == curr_medoid] =␣
 ↪compute_new_medoid(cluster, distances)

        old_medoids[:] = curr_medoids[:]
        curr_medoids[:] = new_medoids[:]

    return clusters, curr_medoids

def assign_points_to_clusters(medoids, distances):
    distances_to_medoids = distances[:,medoids]
    clusters = medoids[np.argmin(distances_to_medoids, axis=1)]
    clusters[medoids] = medoids
    return clusters

def compute_new_medoid(cluster, distances):
    mask = np.ones(distances.shape)
    mask[np.ix_(cluster,cluster)] = 0.
    cluster_distances = np.ma.masked_array(data=distances, mask=mask,␣
 ↪fill_value=10e9)
    costs = cluster_distances.sum(axis=1)
    return costs.argmin(axis=0, fill_value=10e9)
```

[103]:
```
labels, medoids = cluster(data['D'],6)
fig, ax = plt.subplots()
fig.set_size_inches(10, 8)
ax.scatter(new_data[:,0],new_data[:,1], c= labels, s=20, cmap='viridis',␣
 ↪zorder=2)
plt.show()
```



There are no significant differences between the clustering chosen by k-medoids compared to k-means.

Written Exercises

1. Decision Trees

a) Suppose left split has more p_1, right split has more p_2.

based on the tree-growing algorithm, the left split are categorized as p_1

and the error for left split is n_1 ; the right split are categorized as p_2.

and the error for right split is n_2. Thus, the mistaken number

for tree-growing algorithm is n_1+n_2.

$p_1 > n_1$, $p_2 > n_2$ weighted impurity :

$$(p_1+n_1)\min\left(\frac{p_1}{p_1+n_1}, \frac{n_1}{p_1+n_1}\right) + (p_2+n_2)\min\left(\frac{p_2}{p_2+n_2}, \frac{n_2}{p_2+n_2}\right) = n_1+n_2$$

Thus, the number of training mistakes made by this truncated tree is exactly equal to the weighted impurity.

b) Gini Index $1-r^2-(1-r)^2 = 2r(1-r)$

For a_1:

if $a_1=1$ $\quad 1-(\frac{1}{6}^2+\frac{5}{6}^2) = 0.278$

if $a_1=0$ $\quad 1-(\frac{2}{4}^2+\frac{2}{4}^2) = 0.5$

$6\times 0.278 + 4\times 0.5 = 3.668$

For a_2:

if $a_2=1$ $\quad 1-(\frac{3}{4}^2-\frac{1}{4}^2) = 0.375$

$a_2=0$ $\quad 1-(\frac{2}{6}^2-\frac{4}{6}^2) = 0.444$

$4\times 0.375 + 6\times 0.444 = 4.164$

For a_3:

if $a_3=1$ $\quad 1-(\frac{3}{3}^2+\frac{0}{3}^2) = 0$

$a_3=0$ $\quad 1-(\frac{3}{7}^2+\frac{4}{7}^2) = 0.490$

$0\times 3 + 7\times 0.490 = 3.43$

a_3 will be chosen since is has the smallest gini index impurity.

By Min-error impurity function.

For a_1 : $(2+2)\min(\frac{2}{4},\frac{2}{4}) + (1+5)\min(\frac{1}{6},\frac{5}{6}) = 3$.

a_2 : $(1+3)\min(\frac{1}{4},\frac{3}{4}) + (2+4)\min(\frac{2}{6},\frac{4}{6}) = 3$

a_3 : $(3+4)\min(\frac{3}{7}, \frac{4}{7}) + (3+0)\min(\frac{3}{3}, \frac{0}{3}) = 3$

all a_1, a_2, a_3 can be chosen

c) $N = n_1 + n_2 + p_1 + p_2$

We want min-error impurity before the split

$>$ weighted min-error after the split

$N \cdot \min(\frac{p_1+p_2}{N}, \frac{n_1+n_2}{N}) > (n_1+p_1)\cdot\min(\frac{p_1}{n_1+p_1}, \frac{n_1}{n_1+p_1}) + (n_2+p_2)\cdot\min(\frac{p_2}{n_2+p_2}, \frac{n_2}{n_2+p_2})$

$\min(p_1+p_2, n_1+n_2) > \min(n_1, p_1) + \min(n_2, p_2)$

if $p_1 < n_1$ $p_2 > n_2$

$\min(p_1+p_2, n_1+n_2) > p_1 + n_2$

set $n_1 + n_2 < p_1 + p_2$

$\Rightarrow n_1 > p_1$ ✓

Thus: $p_1 < n_1$, $p_2 > n_2$, $p_1 + p_2 > n_1 + n_2$

or

$p_1 > n_1$, $p_2 < n_2$, $p_1 + p_2 < n_1 + n_2$

d) It suggests that min-error impurity is simpler in calcutation but gives less information

2. $\lim_{n\to\infty}(1-\frac{1}{n})^n = \frac{1}{e} = 0.3679$