Data Mining Assignment 1

Zhengyuan Zhu
CSE5334 - Data Mining
University of Texas at Arlington

March 3, 2020

Problem 1 solution guideline

- Use numpy to generate fake data.
- Use numpy to implement K-means algorithm.
- Test K-means algorithm by different settings.
- Visualize results via matplotlib.

Solution for 1.1

First, set up all the parameters for the 2-D Gaussian distribution.

```
mul1, sigma1 = [1, 0], [[0.9, 0.4], [0.4, 0.9]]
mul2, sigma2 = [0, 1.5], [[0.9, 0.4], [0.4, 0.9]]
size = 500
```

Then use ‘numpy.random‘ to generate some fake data.

```
dummyData1 = np.random.multivariate_normal(mean=mul1, cov=sigma1, size=
                                         size)
dummyData2 = np.random.multivariate_normal(mean=mul2, cov=sigma2, size=
                                         size)
```

The two figures below are the visualization of fake data:

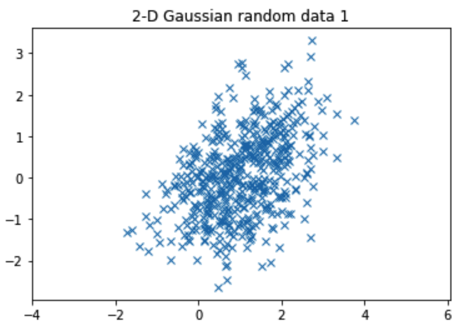


Figure 1: Fake data1 with μ_1, σ_1

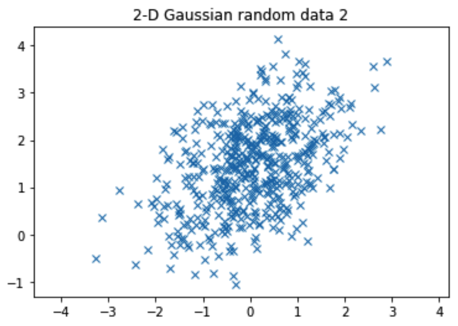


Figure 2: Fake data2 with μ_2, σ_2

Then start to write the cluster function. First, it will take 3 parameters as input when initialize itself.

```
class mykmeans():
    def __init__(self, X, k, c):
        self.X = X
        self.k = k
        self.c = c
        self.i = 0
```

To define the termination of iteration, I set iteration to 10000 and use np.linalg.norm to compute the l_2-norm between a previous center and an updated center.

```
def __call__(self, max_iter=10000):
    for i in range(max_iter):
        ...

    if np.linalg.norm(prev_c-self.c)<1e-3:
        break
```

The core logic of K-means is simple: assign each data point to the cluster whose mean has the least square Euclidean distance. In our problem, distance is calculated by scipy.spatial.distance.cdist. And then assign data point to the nearest class through index.

$$S_i^{(t)} = \{x_p : ||x_p - m_i^t||^2 \leq ||x_p - m_j^t||^2\}, \forall j, 1 \leq j \leq k$$

Then in the update step, calculate the new means(centers) of the observations in the new clusters through the simple formula:

$$m_i^{(t+1)} = \frac{1}{|s_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

The code implementation is below:

```
diag = np.eye(self.k)
for i in range(max_iter):
    prev_c = np.copy(self.c)
    dist = cdist(self.X, self.c)
    cluster_idx = np.argmin(dist, axis=1)
    cluster_idx = diag[cluster_idx]
    self.c = np.sum(self.X[:, None, :] * cluster_idx[:, :, None], axis=0)/
             np.sum(cluster_idx, axis=0)[:, None]
```

Solution for 1.2

If apply k=2 and initial centers $c_1 = (10, 10)$ and $c_2 = (-10, -10)$ to the data generated above:

```
center1 = np.array([[10., 10.], [-10., -10.]])
k2_1 = mykmeans(X=dummyData1, k=2, c=center1)
k2_1.__call__()
k2_2 = mykmeans(X=dummyData2, k=2, c=center1)
k2_2.__call__()
```

The centers and iterations are shown in Figure 3 and 4

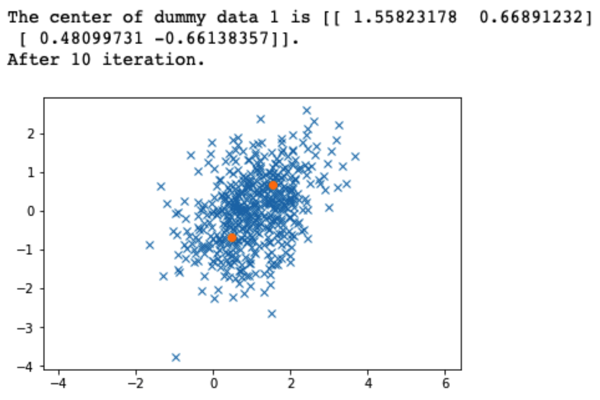

```
The center of dummy data 1 is [[ 1.55823178  0.66891232]
 [ 0.48099731 -0.66138357]].
After 10 iteration.
```

Figure 3: Find centers after 10 iterations

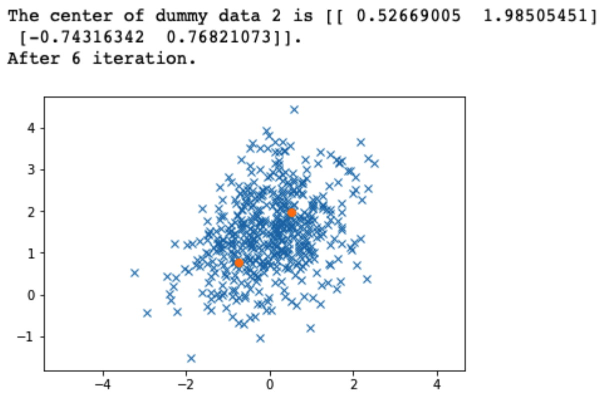

```
The center of dummy data 2 is [[ 0.52669005  1.98505451]
 [-0.74316342  0.76821073]].
After 6 iteration.
```

Figure 4: Find centers after 6 iterations

Solution for 1.3

If apply k=4 and initial centers $c_1 = (10, 10), c_2 = (-10, -10), c_3 = (10, -10), c_4 = (-10, 10)$ to the data generated above:

```
center2 = np.array([[10., 10.], [-10., -10.], [10., -10.], [-10., 10.]])
k4_1 = mykmeans(X=dummyData1, k=4, c=center2)
k4_1.__call__()
k4_2 = mykmeans(X=dummyData2, k=4, c=center2)
k4_2.__call__()
```

The centers and iterations are shown in Figure 5 and 6

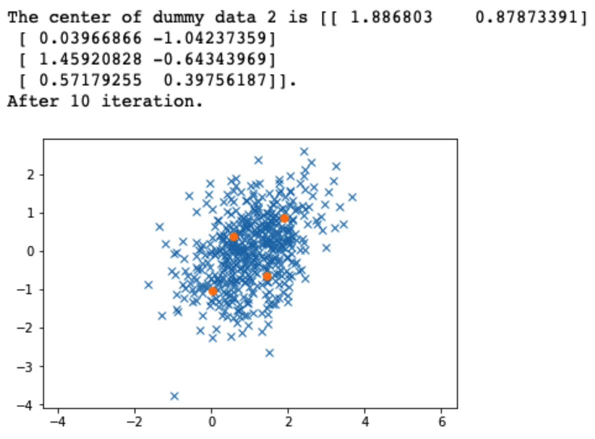

```
The center of dummy data 2 is [[ 1.886803    0.87873391]
 [ 0.03966866 -1.04237359]
 [ 1.45920828 -0.64343969]
 [ 0.57179255  0.39756187]].
After 10 iteration.
```

Figure 5: Find centers after 10 iterations

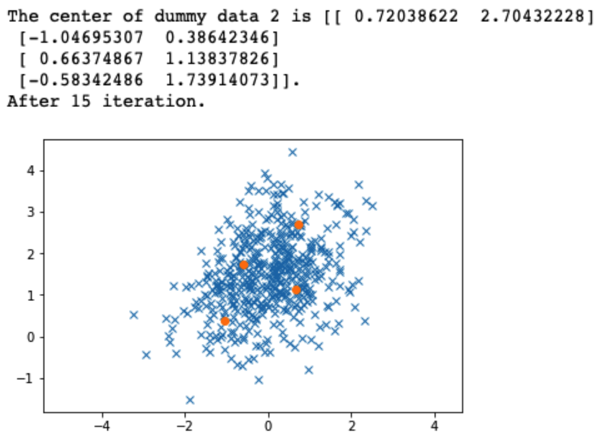

```
The center of dummy data 2 is [[ 0.72038622  2.70432228]
 [-1.04695307  0.38642346]
 [ 0.66374867  1.13837826]
 [-0.58342486  1.73914073]].
After 15 iteration.
```

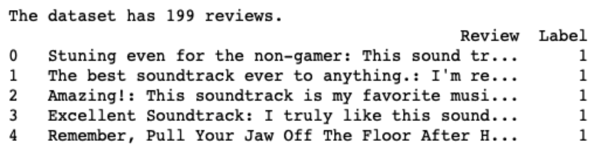
Figure 6: Find centers after 6 iterations

Problem 2 Solution

Solution 2.1

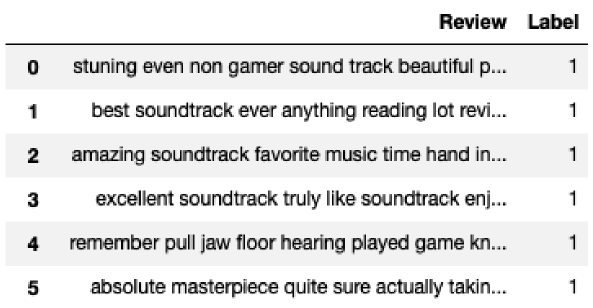
To solve the problem, the first thing is to read data from csv through pandas. And print information and samples of Amazon-reviews dataset shown in Figure 7.

```
review=pd.read_csv('./data/Amazon_Reviews.csv')
review['Label']=review['Label'].map({'__label__2 ':1,'__label__1 ':0})
docs_num = review.shape[0]
```



```
The dataset has 199 reviews.
                                              Review  Label
0  Stuning even for the non-gamer: This sound tr...      1
1  The best soundtrack ever to anything.: I'm re...      1
2  Amazing!: This soundtrack is my favorite musi...      1
3  Excellent Soundtrack: I truly like this sound...      1
4  Remember, Pull Your Jaw Off The Floor After H...      1
```

Figure 7: Samples of raw data



	Review	Label
0	stuning even non gamer sound track beautiful p...	1
1	best soundtrack ever anything reading lot revi...	1
2	amazing soundtrack favorite music time hand in...	1
3	excellent soundtrack truly like soundtrack enj...	1
4	remember pull jaw floor hearing played game kn...	1
5	absolute masterpiece quite sure actually takin...	1

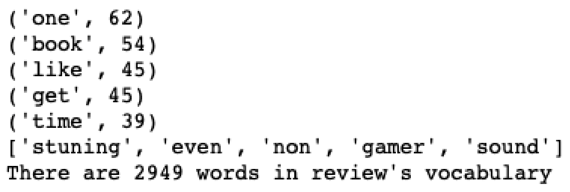
Figure 8: Samples of processed data

Then pre-process reviews through tokenize, stopwords removal and lemmatization by NLTK.

- Use RegexpTokenizer matches the regular expression to delete the punctuations.
- Use stop words from nltk to help us delete the trivial words.
- Use word net to stem all the words from review.

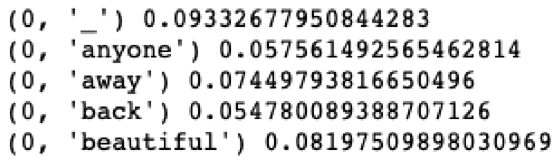
After pro-precess, the reviews will look cleaner in Figure 8.
Then it is necessary to calculate document frequency for TF-IDF through build a dictionary that key is word and value is all the documents that contain the key. The length of values is document frequency for specific word. For the term frequency, it can easily calculated by a Counter class. As a result, in tf-idf dictionary, the key is document index along with word and the value is word weight. The visualization is very huge, so I save it to *word_matrix.png* in the attachment.

```
# document frequency
DF = {}
for idx, row in review.iterrows():
    for w in row['Review'].split():
        try:
            DF[w].add(idx)
        except:
            DF[w] = {idx}
DF = {k: len(v) for k, v in DF.items()}
# TF-IDF
TFIDF = {}
for idx, row in review.iterrows():
    rw = row['Review'].split()
    cnt = Counter(rw)
    for w in np.unique(rw):
        tf = cnt[w]/len(rw)
        df = DF[w]
        idf = np.log((docs_num+1)/(df+1))
        TFIDF[(idx, w)] = tf*idf
```



```
('one', 62)
('book', 54)
('like', 45)
('get', 45)
('time', 39)
['stuning', 'even', 'non', 'gamer', 'sound']
There are 2949 words in review's vocabulary
```

Figure 9: Samples of document frequency and vocabulary size



```
(0, '_') 0.09332677950844283
(0, 'anyone') 0.057561492565462814
(0, 'away') 0.07449793816650496
(0, 'back') 0.054780089388707126
(0, 'beautiful') 0.08197509898030969
```

Figure 10: Samples of tf-idf weight

Solution 2.2

For the purpose of indicating if a product is good or bad. The best way to select positive and negative words is manually picking them according to frequency.

- Positive words(frequency): like(54) great(38) good(35) love(33) best(20)
- Negative words(frequency): however(15) disappointed(11) waste(10) poor(10) hard(8)

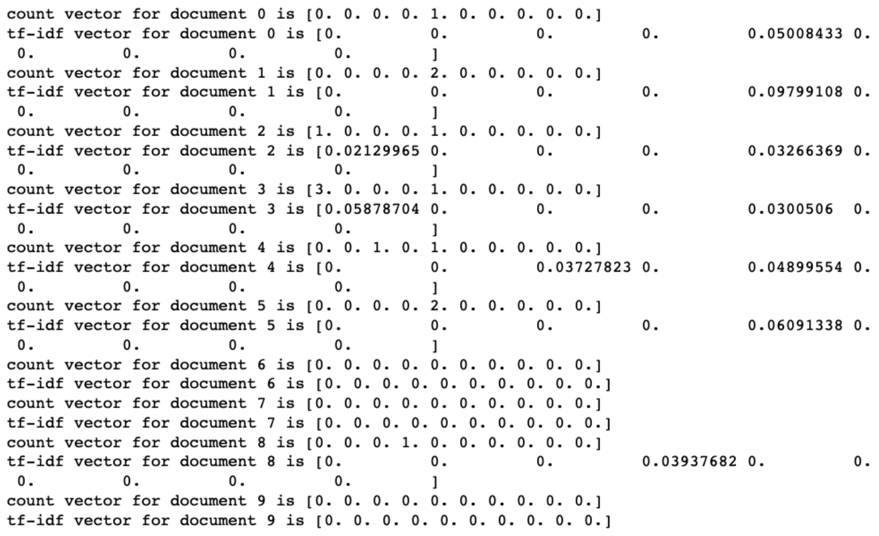
To represent each review in a vector space of ten words and tf-idf weight matrix, we just need to save all the information in two lists:

```
def gen_vector(idx, review):
    Q = np.zeros(10)
    P = np.zeros(10)
    ten_words = ['like', 'great', 'good', 'love', 'best',
                 'however', 'disappointed', 'waste', 'poor', 'hard']
    for w in ten_words:
        for k, v in TFIDF.items():
            if k[0]==idx and k[1]==w:
                Q[ten_words.index(w)] = v
        for word in review:
            if word==w:
                P[ten_words.index(w)] += 1

    return P, Q

tfidf_vectors = []
count_vectors = []
for idx, row in review.iterrows():
    rw = row['Review'].split()
    P, Q = gen_vector(idx, rw)
    tfidf_vectors.append(Q)
    count_vectors.append(P)
```

Some of the results are in the figure 11 below:



```
count vector for document 0 is [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
tf-idf vector for document 0 is [0.         0.         0.         0.         0.05008433 0.
 0.         0.         0.         0.        ]
count vector for document 1 is [0. 0. 0. 0. 2. 0. 0. 0. 0. 0.]
tf-idf vector for document 1 is [0.         0.         0.         0.         0.09799108 0.
 0.         0.         0.         0.        ]
count vector for document 2 is [1. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
tf-idf vector for document 2 is [0.02129965 0.         0.         0.         0.03266369 0.
 0.         0.         0.         0.        ]
count vector for document 3 is [3. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
tf-idf vector for document 3 is [0.05878704 0.         0.         0.         0.0300506  0.
 0.         0.         0.         0.        ]
count vector for document 4 is [0. 0. 1. 0. 1. 0. 0. 0. 0. 0.]
tf-idf vector for document 4 is [0.         0.         0.03727823 0.         0.04899554 0.
 0.         0.         0.         0.        ]
count vector for document 5 is [0. 0. 0. 0. 2. 0. 0. 0. 0. 0.]
tf-idf vector for document 5 is [0.         0.         0.         0.         0.06091338 0.
 0.         0.         0.         0.        ]
count vector for document 6 is [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
tf-idf vector for document 6 is [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
count vector for document 7 is [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
tf-idf vector for document 7 is [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
count vector for document 8 is [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
tf-idf vector for document 8 is [0.         0.         0.         0.03937682 0.         0.
 0.         0.         0.         0.        ]
count vector for document 9 is [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
tf-idf vector for document 9 is [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

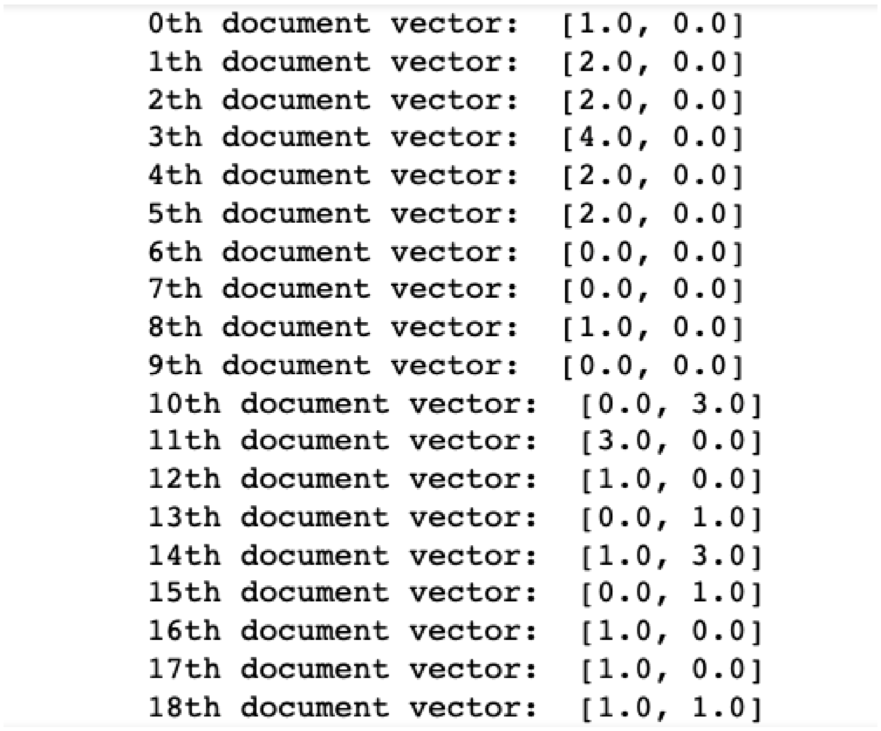
Figure 11: Samples of count and tf-idf vectors

Solution 2.3

After sum up the frequency of "positive" words and "negative" words. The reviews as vectors of length 2 can be processed by code:

```
review_vectors = []
pos = ['like', 'great', 'good', 'love', 'best']
neg = ['however', 'disappointed', 'waste', 'poor', 'hard']
for idx, row in review.iterrows():
    rw = row['Review'].split()
    pos_cnt, neg_cnt = 0, 0
    for w in rw:
        if w in pos:
            pos_cnt += 1
        elif w in neg:
            neg_cnt += 1
    review_vectors.append([float(pos_cnt), float(neg_cnt)])
```

The result is in Figure 12.



```
0th document vector:  [1.0, 0.0]
1th document vector:  [2.0, 0.0]
2th document vector:  [2.0, 0.0]
3th document vector:  [4.0, 0.0]
4th document vector:  [2.0, 0.0]
5th document vector:  [2.0, 0.0]
6th document vector:  [0.0, 0.0]
7th document vector:  [0.0, 0.0]
8th document vector:  [1.0, 0.0]
9th document vector:  [0.0, 0.0]
10th document vector:  [0.0, 3.0]
11th document vector:  [3.0, 0.0]
12th document vector:  [1.0, 0.0]
13th document vector:  [0.0, 1.0]
14th document vector:  [1.0, 3.0]
15th document vector:  [0.0, 1.0]
16th document vector:  [1.0, 0.0]
17th document vector:  [1.0, 0.0]
18th document vector:  [1.0, 1.0]
```

Figure 12: Document vectors represented by 2-D vectors

At last, we apply code from Problem 1 to this 2D data with k = 2,3,4 with randomly initialized centers.

```
k1 = K_means(X=review_vectors, k=2, c=None)
k1.__call__()
plotAns(review_vectors, k1.c)
print(k1.i)
```

The results are in the Figure 13, 14 and 15.

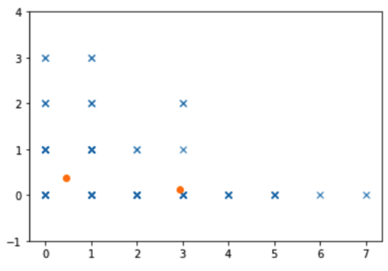


Figure 13: Centers found after 4 iterations

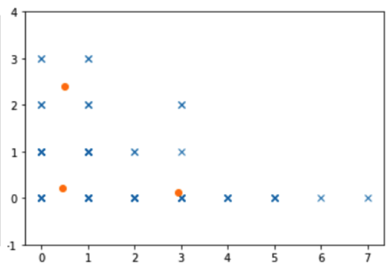


Figure 14: Centers found after 5 iterations

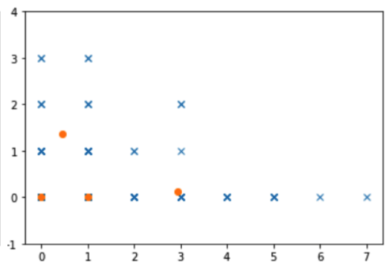


Figure 15: Centers found after 2 iterations

Reference

- https://en.wikipedia.org/wiki/K-means_clustering#Standard_algorithm_(naive_k-means)
- https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.norm.html
- https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089