Homework 3 Report

Part 1：

Documents explanation:

FetchDoc.py: Fetch documents from Bing according to queries (texas aggies, texas longhorns, duke blue devils, dallas cowboys, dallas mavericks). After running this file, docs.json, which is used for storing title and contents of a doc, and title.json, which is used for storing titles of the retrieved docs, will be created in the current directory.

document-generation.py: This file is where we should start.

kmeans.py: Module for K-Means algorithm. It will create title_cluster.json, which stores all titles of retrieved docs under five clusters.

Purity.py: Module for calculating purity and RI of K-Means result

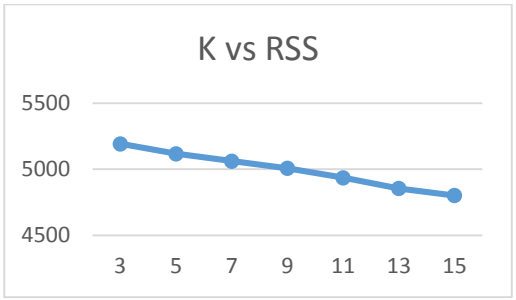
RSS.py: Module for calculating Residual Sum of Squares

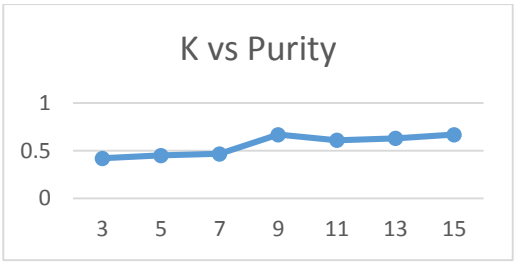
How to run my program:

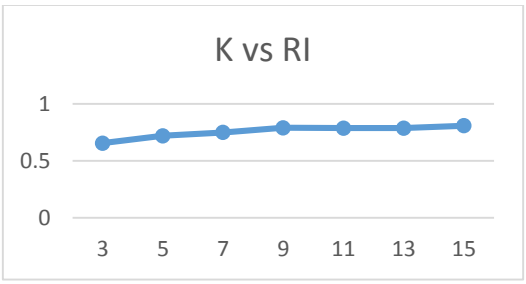
Execute document-generation.py, input a k you want:

```
>>> ================================ RESTART ================================
>>>
Please input a value for k: 3
```

After running it from k=3 to k =10 and get the best result (lowest RSS) in each case, the following graph can be output:







According to my observation, in general, RI and Purity increases as K increases. And RSS decreases as K increases. This is totally in accordance with my expectation. However, I did not find a clear knee in the k vs RSS graph. It seems to be decreasing monotonously.

How I calculated Purity and RI:

Purity Calculation:

$$\text{purity}(\Omega, \Gamma) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|$$

$\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$ is the set of clusters and
$\Gamma = \{c_1, c_2, \ldots, c_J\}$ is the set of classes.

After running my program, clustering result will be written into title_cluster.json. My Purity-Calculation algorithm is as follows (Suppose k is 5 and title_cluster stores the K-Means result),

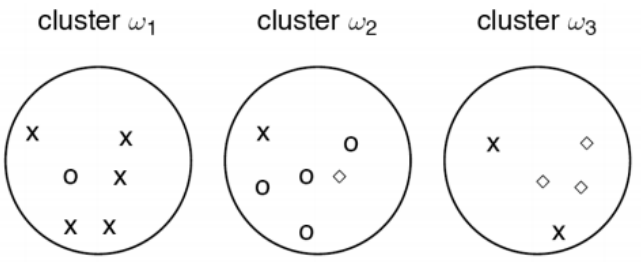
```
initialize an accumulative value as 0.
    For every title in cluster from 0 to 4 in title_cluster.json,
        if title in title['texas aggies']
            cluster_result['texas aggies'] += 1
        else if title in title[ 'texas longhorns' ]:
            cluster_result[ 'texas longhorns' ] += 1
        else if title in title[ 'duke blue devils' ]:
            cluster_result[ 'duke blue devils' ] += 1
        else if title in title[ 'dallas cowboys' ]:
            cluster_result[ 'dallas cowboys' ] += 1
        else if title in title[ 'dallas mavericks' ]:
            cluster_result[ 'dallas mavericks' ] += 1
    then add the maximum value of the five cateogories in cluster_result to the accumulative value
    Reset cluster_result, go back to second row and start calculation for next cluster.
Divide the accumulative value by 5 to obtain Purity value
```

```
result = 1.0
for cluster_num in title_cluster:
    cluster_result = defaultdict( int )
    for doc_title in title_cluster[ cluster_num ]:
        if doc_title in title[ 'texas aggies' ]:
            cluster_result[ 'texas aggies' ] += 1
        elif doc_title in title[ 'texas longhorns' ]:
            cluster_result[ 'texas longhorns' ] += 1
        elif doc_title in title[ 'duke blue devils' ]:
            cluster_result[ 'duke blue devils' ] += 1
        elif doc_title in title[ 'dallas cowboys' ]:
            cluster_result[ 'dallas cowboys' ] += 1
        elif doc_title in title[ 'dallas mavericks' ]:
            cluster_result[ 'dallas mavericks' ] += 1
    result += findMax( cluster_result )
    RI_List.append( cluster_result )

purity = 1.0 * result / doc_num
print "Purity is " + str(purity)
```

Rand Index Calculation:

The cluster_result in Purity calculation can be reused in RI calculation.



For convenient explanation, let's use the diagram in lecture note. There are five cluster_result in total. They have the following structure: { 'texas aggies' : 2, 'texas longhorns' : 5,… }. They can be seen as the circle in the above diagram. My RI-Calculation Algorithm is as follows,

For each cluster
    Count the number of titles (denoted by n) under each of the five queries=>tp += C_n^2
    Count the total number of titles (dentoed by f) in the cluster=>tp+fp += C_f^2
total number of titles N in the whole collection = Sum up the total number of titles in each cluster
tp+tn+fp+fn = C_N^2
tn+fn = C_N^2 – (tp+fp)
For the number of titles under each query in each cluster,
    multiply the number of titles under the same query in other clusters and sum them up to get fn
tn = (tn+fn)-fn
Then RI = (tp+tn)/(tp+tn+fp+fn)

```
for cluster in RI_List:
    cluster_total_point = 0
    if (len(cluster) == 1) and (cluster.values()[0] == 1):
        cluster_total_point += 1
    else:
        for query_class in cluster:
            cluster_total_point += cluster[ query_class ]
            if cluster[ query_class ] > 1:
                TP += C( cluster[ query_class ],2 )
        TPFP += C( cluster_total_point,2 )
    total_point += cluster_total_point

TPTNFPFN = C(total_point,2)
TNFN = TPTNFPFN - TPFP

'''
    Calculate FN
'''
i = 0
while i < len(RI_List):
    j = i + 1
    while j < len(RI_List):
        for key in RI_List[ i ]:
            if key in RI_List[ j ]:
                FN += RI_List[ i ][ key ] * RI_List[ j ][ key ]
        j += 1
    i += 1

TN = TNFN - FN

return 1.0 * ( TP + TN ) / TPTNFPFN
```

Part 2

Documents explanation:

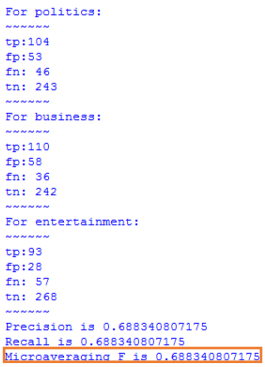
FetchDoc.py, TestDataRetrieval.py: According to the queries, fetch documents from Bing. After running these two file, docs.json and TestDocs.json will be created in current directory.

Training.py: Training stage. After running this file, TrainingResult.json will be created in the current directory.

Testing.py: This is where we should start given that all preparation work have been done.

How to run my program:

Input "python Testing.py" in terminal, you will get:



This is shown at the end of the execution. As we can see, Microaveraging F value is approximately 0.69.

classified / reality	politics	not politics
politics	104	46
non politics	53	243

classified / reality	business	not business
business	110	36
not business	58	242

classified / reality	entertainment	not entertainment
entertainment	93	57
not entertainment	28	268

Aggregate TP,FP,TN,FN is the sum of tp,fp,tn,fn in the three charts above.

How do I calculate F1 value?

My algorithm is as follows,

```
Input：classification_dict. key is reality, value is a list of classified value.
For category in classification_dict:
    for classification in category:
        if classification is category, tp += 1
        else fn += 1
for classification in not category:
    if classification is category:
        fp += 1
tn = total number of docs in category – (tp+fp+fn)
Then we get all four values in the chart above for one category.
```

Sum up all tp,fp,tn,fn in each category to calculate Precision and Recall using

$$P = TP / (TP + FP)$$
$$R = TP / (TP + FN)$$

Then F1 value can be obtained by calculating:

$$F_1 = \frac{1}{\frac{1}{2}\frac{1}{P} + \frac{1}{2}\frac{1}{R}} = \frac{2PR}{P + R}$$

Part 3

K-Means clustering optimization strategies:

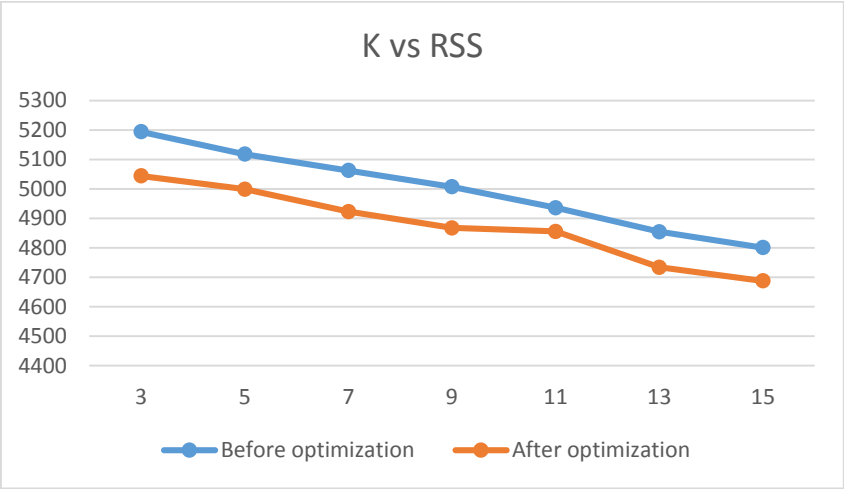
Open folder “Part 3/Part 1 optimization”, you will see two sub-folders, one is “Bisecting_K_Means”, the other is “No Stop-Word and Cosine Similarity”. The following performance enhancement presentation is based on the program in “No Stop-Word and Cosine Similarity”. The execution procedure is just as mentioned in Part 1. As for the “Bisecting_K_Means”, there seems to be no performance enhancement. Detailed program execution procedure of how to run them will be included later.

(1) Ignore stop words:
This is mainly for program efficiency. Ignoring stop words will decrease the dimension of the vector space. In my program, all stop words are stored in a list.

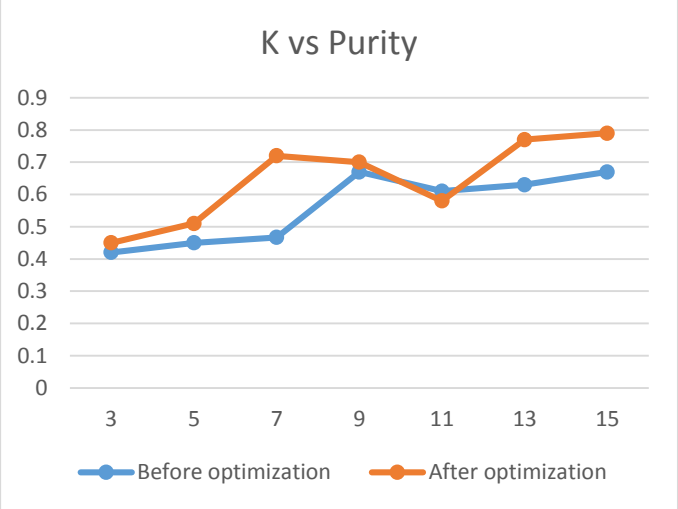
(2) Use Cosine similarity to determine similarity instead of Euclidean distance:
This will increase the performance of K-Means. For example, in a three dimensional vector space model, it is more reasonable for (1,1,1) and (100,100,100) to be in the same cluster because they are talking about the same topic. The three words appear in both documents and have similar weight in the documents. Let’s say, another document (1,0,0) should not be in the same cluster as (1,1,1) because of the same reason. However, if we use Euclidean distance, (1,1,1) and (1,0,0) will definitely be clustered together because they are closer to each other. This is inappropriate.

(3) Try bisecting K-Means:
My intuition is that bisecting K-Means, as a hierarchical divisive improved version of K-Means, will guarantee that each cluster has an averaged number of documents and it should be no worse than normal K-means.

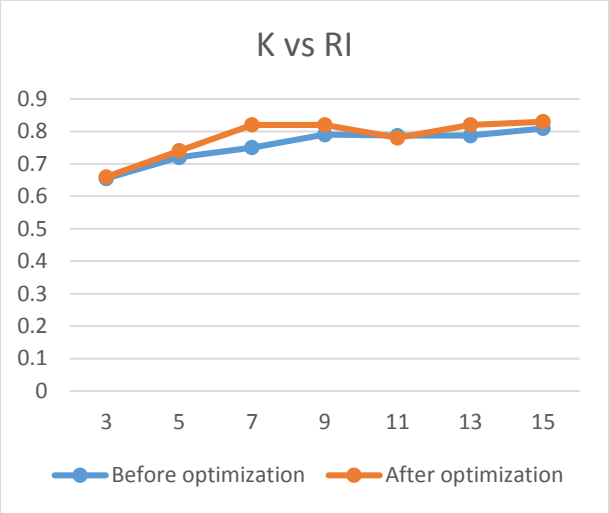
Performance Comparison between before optimization and after optimization:



After optimization, when the same k is selected, RSS is lower.



After optimization, in general, Purity is higher compared with before optimization.



After optimization, RI is slightly higher than before optimization.

How to run Bisecting_K_Means.py:
Input “python Bisecting_K_Means.py” in terminal

```
Please input how many clusters you want:5
Stable
RSS is 4439.48195205
Satisfied with this RSS?(Y/N)
```

Here we set k = 5. According to Bisecting K-Means algorithm, at least 4 times normal K-Means algorithm will be run. Each time, the program will ask whether user satisfies with the result according to RSS value. Usually, we need to try a few times to select the best RSS possible. Input “Y” to continue to next bisecting. Input “N” to executing K-Means to current cluster again. After input “Y” for four times, we got the result indicating Purity is about 53%, nearly the same as mentioned above in the graph.

```
--------------------
[u'OL Cuney is No. 9 for Longhorns for 2014', u'Duke-Notre Dame Preview', u'Iris
h women a big challenge for Duke', u'Skylar Diggins leads Notre Dame past Duke,
to Final Four', u'Blue Devils headed to NCAA Elite Eight with win over Nebraska'
, u'Duke vs. Davidson 1 6 PM 1 Durham Bulls Athletic Park', u'Duke falls short a
gain, losing 87-76 to Notre Dame', u'Duke Blue Devils versus Louisville Cardinal
s Pick Prediction NCAA Tournament College Basketball Betting Lines Odds Preview
3-31-2013', u'Defense spearheads Duke Sweet 16 win over Cornhuskers', u"UofL bea
ts Duke 85-63, despite Kevin Ware's compound fracture", u'Duke Tops Michigan Sta
te 71-61: Seth Curry Leads Blue Devils Past Spartans', u"Cesena: Aggie women's e
arly exit from tournament not good enough for fans", u'Twitter Reactions After K
evin Ware Breaks Leg During Louisville-Duke Game', u'Notre Dame women defeat Duk
e 87-76 to reach Final 4', u'McCallie prepares Duke to continue along road to Fi
nal Four', u'Duke Blue Devils', u"Pressel and Dubreuil record low finishes for N
o. 16 Women's Golf at Farms Invitational", u'Duke Basketball: Top 5 Highlights o
f the 2012-13 Season for the Blue Devils', u'Duke tangles with Louisville in Mid
west Regional final', u"Duke Women's Puts The Clamps On Nebraska To Become Elite
", u'College Football Nation: Duke Blue Devils', u'2015 DT Bryce English commits
 to the Texas Longhorns', u'In 2013-14, Duke Blue Devils will be smaller, more a
thletic', u'Duke women fall to Notre Dame, 87-76, in NCAA basketball tourney', u
'Blue Devils Top Wildcats for Fifth Straight Win', u"Curry's hot shooting propel
s Duke to Elite Eight"]
30
43
11
20
26
130
0.530769230769
```

Naïve Bayes classification optimization strategies:

Open folder “Part 3/Part 2 optimization”, you will see three .py files, NB_Training.py, Rocchio_Training.py and Testing.py.

How to run my program:

(1) Input “python NB_Training.py” in terminal
(2) Input “python Rocchio_Training.py” in terminal

(3) Input “python Testing.py” in terminal, and it will take no more than one minute to output the result.

Here is the result:

```
For politics:
~~~~~~
tp:111
fp:54
fn: 39
tn: 242
~~~~~~
For business:
~~~~~~
tp:107
fp:56
fn: 39
tn: 244
~~~~~~
For entertainment:
~~~~~~
tp:93
fp:25
fn: 57
tn: 271
~~~~~~
Precision is 0.69730941704
Recall is 0.69730941704
Microaveraging F is 0.69730941704
```

Compared with the F1 score (68.8) in part 2, I got a one percent tiny improvement.

To achieve the tiny performance improvement, I firstly took bi-word phrase and tri-word phrase into consideration in Naïve Bayes training process. More words mean more dimensions in vector space. More dimensions may give me a better discrimination among test docs.

The code snippet in Naïve Bayes training stage is as follows,

```
bi_word = ""
tri_word = ""
for doc in class_list:
    for word in re.findall( r'[0-9a-z]+', doc.lower() ):
        if word in stop_word_list:
            continue
        vacabulary_dict[ word ] += 1

        bi_word += word
        if len(bi_word.split()) == 2:
            vacabulary_dict[ bi_word ] += 1
            bi_word = bi_word.split()[1]
        bi_word += " "

        tri_word += word
        if len(tri_word.split()) == 3:
            vacabulary_dict[ tri_word ] += 1
            tri_word = tri_word.split()[1] + " " + tri_word.split()[2]
        tri_word += " "

return vacabulary_dict
```

Secondly, I combined Naïve Bayes with another classification algorithm---TF-IDF based Rocchio algorithm.

Although Naïve Bayes will generally produce a better classification result than Rocchio, under some specific cases, Rocchio may be better. Therefore, my thought is to take Rocchio classification rank and Naïve Bayer classification rank as features in final decision. For example,

Rocchio classification result:{0:’politics’,1:’business’,2:’entertainment’}

NB classification result:{0:’business’,1:’ entertainment’,2:’politics’}

Here 0: classified_as, 1: second classified_as candidate, 2: probably not in the category

final score:

{‘business’: (0+1), ‘politics’: (0+2), ‘entertainment’: (1+2)}

The one with lowest score is final decision. Here, it means the doc will be classified as “business”.

When two category has the same final score, just pick one randomly.

My code snippet is as follows,

```
combined_score = defaultdict(int)
combined_score[ NB_Result[0] ] += 0
combined_score[ NB_Result[1] ] += 1
combined_score[ NB_Result[2] ] += 2
combined_score[ Rocchio_Result[0] ] += 0
combined_score[ Rocchio_Result[1] ] += 1
combined_score[ Rocchio_Result[2] ] += 2

sorted_combined_score = sorted(combined_score.values())
val = sorted_combined_score[0]
for cate in combined_score:
    if combined_score[cate] == val:
        return cate
```