CSE 5243 Lab 3 – Clustering of Reuters Article Text Feature Vectors

Afnan Rehman, Matthew Stock, & Kumar Dhital

Section 26469

November 28, 2018

Clustering Methodology:

Given that for our clustering we needed numbers, the “bag of words” vector that we had before needed to be modified to include word counts. To make the code shorter, instead of reusing code we created in the previous lab, we use sklearn’s built in CountVectorizer to accomplish this. We used K-means for the first vector, since we thought it would be interesting to see what kinds of words we got as centroids and which kinds of words would be most frequent in clusters. For the second vector we used a DBSCAN in order to find dense clusters of words which could indicate a relationship between those words and their origin document type.

One method we attempted to implement this was to convert our bag of words vector into a dictionary of pairs that we could graph in two dimensions, in order to visually spot clusters. This is detailed in vector_to_graph.py in the appendix. This was a great way to see how different groups were clustered together, but it proved difficult to then feed this into a K-Means clustering algorithm. K-means allowed us to apply some domain specific knowledge about our dataset in our selection of the appropriate k value and would be efficient to implement over this set of data. Also, K-means was because we could cluster the data into the number of TOPICS and PLACES from the previous labs and determine if the clusters created by this method would line up with our classification results.

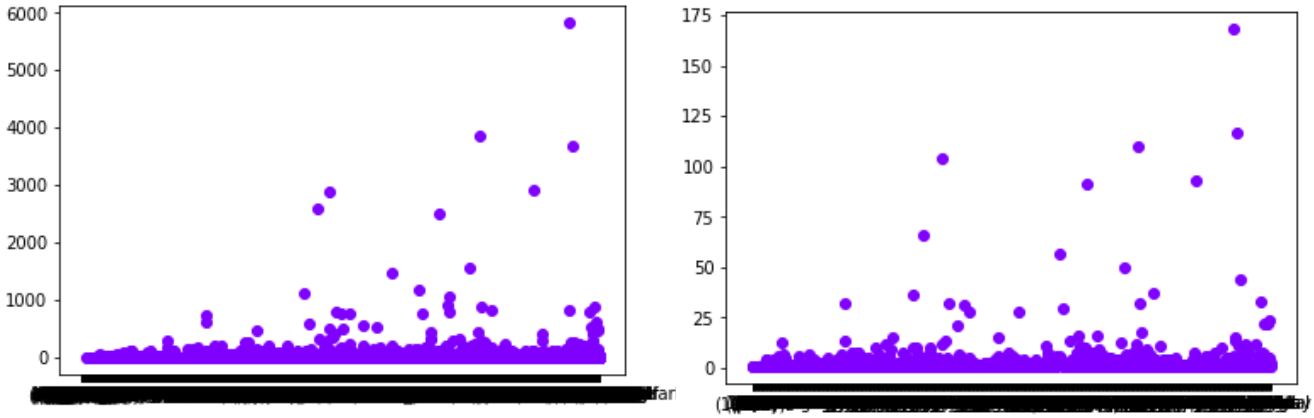
DBSCAN was chosen as the method to use for our second feature vector, the “weighted bag of words” vector because this vector already increases the density of important words by including them more often. This would allow for these words to create dense clusters on the graph. For the weighted bag of words vector from the second lab we also converted the vector of words into dictionary pairs which could be graphed in two dimensions. DBSCAN would allow us to cluster the modified bag of words vector without knowing the number of centroids which would optimally cluster the data. Since specific PLACES and TOPICS were so similar, such as USA and Canada, the DBSCAN method would allow for these two places to be clustered together without forcing an arbitrary separation. Where in K-means we may have to visually inspect the data in order to estimate an appropriate k value, instead we focused on optimizing the radius value and required number of points within the radius. We used the sklearn DBSCAN clustering function, which is similar to the K-means function, in order to preform this clustering and tried many values for the radius and minimum density.

Results:

Using the same accuracy measure we used from lab 2, we ended up with a rather decent accuracy. Since we were able to look at the top centroids for each cluster, we often saw the name of the country mentioned in the centroid list for the cluster, giving us good confidence that the K-means clustering algorithm was working. DBSCAN, on the other hand suffered from the problem that the density of unimportant words caused clusters to be created which did not match well to the classification. Words like ‘it’, ‘a’, and ‘the’ caused clusters to be created which did not provide useful information.

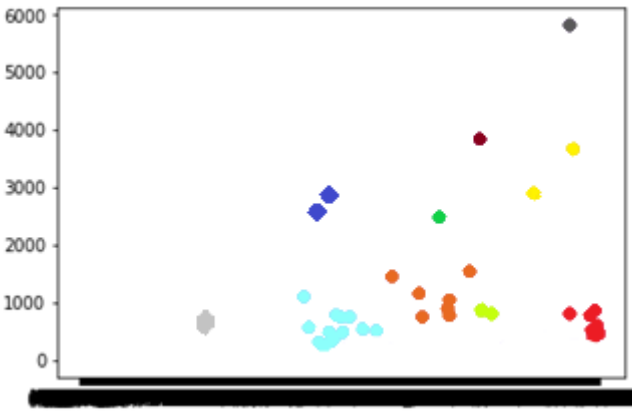
There were a lot of instances of a test vector getting predicted in the same cluster, and this may be due to the testing sample size. Ideally, the prediction results would net a different cluster number for each of the test vector elements. Some of the problems encountered with the prediction was that there existed very similar vectors, such as the vector for USA and UK which contain many of the same words in the bag of words models used by both vectors.

K-Means Clustering



As we can see in the above two graphs, which each represent a specific place (y-axis is word count and x-axis represents a word), the distributions were often very similar and more uniform than telling of a pattern, with peaks at high-frequency words such as ‘the’ or ‘of’. In hindsight, such words should have been removed from the analysis as they are rather useless outliers in the scope of this project. Sparse data from places mentioned less in the article tags often had a better distribution to look at. Given more time we would have tried to look at creating buckets, perhaps of each word so that we could consolidate similar data points and start to notice trends better.

DBSCAN Clustering



As you can see, for the DBSCAN, the occurrences of frequently used unimportant words caused many clusters to be created from the unimportant words that occurred in high frequency. Interesting words that occurred often could be lifted from the data however if these sparse clusters were ignored. The picture above is hand colored with the different clusters the model predicted based on the input feature vector.

Quality of Results:

The K-Means results gave us a decent quality of results, however the we found that some training sets were better than others. The uneven distribution of articles about different places often threw off results somewhat, especially if the held-out validation document had a particularly high count of articles pertaining to one place. We saw this happen often with ‘USA’.

The DBSCAN results were less impressive than the quality of the results obtained with the K-means method. The modified bag of words vector was more suited for the density-based method since it emphasized important words by repeating them multiple times, but it did not completely get rid of the unimportant words. Another pass at pre-processing the data in order to find a way to filter out these sorts of words would have helped to improve the results of the DBSCAN method.

Distribution of work:

Afnan Rehman – Used bag of words vector and sklearn to cluster the different places and topics by K-means clustering, worked on report document formatting, created vector graphing script to attempt to visualize the clusters of words and their counts for different articles.

Kumar Dhital – Researched and helped implement sklearn clustering that performs DBSCAN clustering from vector array or distance matrix.

Matthew Stock – Implemented DBSCAN with optimizations for weighted bag of words vector. Compared results with bag of words vector using K-means clustering.

Appendix:

Source Code for bag_words_kmeans_cluster.py

```
# -*- coding: utf-8 -*-
"""
@author: Afnan, some expressions taken from scikitlearn documentation at:
https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html
"""
import pandas as pd
import csv
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans
csv.field_size_limit(100000000) # Increase field limit to account for large field size

#We use training and test data from lab 2 to cluster and test the results
train_x = pd.read_csv("place_bag_train0" + ".csv", sep=',', encoding = "ISO-8859-1",
engine='python')

# Since word counts were programmed by hand in the last lab, we used the built-in sklearn library
for it this time for brevity
vectorizer = CountVectorizer(stop_words='english')
# This will allow us to use numbers instead of our bag of words approach
X = vectorizer.fit_transform(train_x['text'].values.astype('U')) # Encoding modifier to account
for nulls

# We use the nifty K-means cluster built into sklearn as well here
# K = 147, or the number of countries we are wroking with
model = KMeans(n_clusters=147, init='k-means++', max_iter=300, n_init=1)
model.fit(X)

print("Top terms per cluster:")
# Finding centroids and sorting them
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()

# Here, I print the clusters and their top centroids out, in order to see how words from the
vectors are being clustered
for i in range(147):
    print("Cluster %d:" % i),
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind])

print("\n")
print("Prediction")

# Here we import some premade vectors from our lab 2 solution to use for testing our clustering
prediction
test_x = pd.read_csv("place_bag_test0" + ".csv", sep=',', encoding = "ISO-8859-1",
engine='python')

Y = vectorizer.transform(test_x['text'].values.astype('U'))
prediction = model.predict(Y)
print(prediction)
print(test_x['id'])
```

Source Code for vector_to_graph.py

```
# -*- coding: utf-8 -*-
"""
@author: Afnan
All code in this file is original/programmed from memory and previous labs for this class
"""

import pandas as pd
import csv
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.feature_extraction.text import CountVectorizer
csv.field_size_limit(100000000)

def word_count(str):
    counts = dict()
    words = str.split()
    for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1
    return counts

train_x = pd.read_csv("place_bag_train0" + ".csv", sep=',', encoding = "ISO-8859-1",
engine='python')
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(train_x['text'].values.astype('U'))

text_list = train_x.text.tolist()
master_word_list = []
for element in text_list:
    array = word_count(element) # vector for that country
    i = 0
    for word in array:
        if word[0] not in master_word_list:
            master_word_list.append[word[0]]

master_word_list = sorted(master_word_list)
master_word_dict = {}
index = 0
for element in master_word_list:
    master_word_dict[element] = index
    index += 1
index = 0
colors = cm.rainbow(np.linspace(0, 1, len(text_list)))
for element in text_list:
    c = colors[index]
    array = word_count(element) # vector for that country
    for word in array:
        if word[0] in master_word_dict:
            word[0] = master_word_dict[word[0]]
    # Now feature vector is in coordinate points that we can graph and cluster
    x, y = zip(*array) # unpack a list of pairs into two tuples
    try:
        plt.scatter(x,y,color=c)
    except:
        continue
    index += 1
    plt.show()
```

Source Code for kmeans_graph_vector.py

```
# -*- coding: utf-8 -*-
"""
@author: Afnan
"""

import pandas as pd
import csv
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans
csv.field_size_limit(100000000)

def word_count(str): # this method returns a vector of word counts given a string
    # Since a lot of cleaning was already done in labs 1 and 2, we can simply split the string
and count.
    counts = dict()
    try:
        words = str.split()
    except:
        return []
    for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1
    return counts

# Here I import the training vector from the previous lab
train_x = pd.read_csv("place_bag_train0" + ".csv", sep=',', encoding = "ISO-8859-1",
engine='python')

text_list = train_x.text.tolist()
master_word_list = []
for element in text_list:
    array = word_count(element) # vector for that country
    i = 0
    for word in array:
        if word not in master_word_list: # create a master list of words that exist in the corpus
            master_word_list.append(word)

# In the next few lines, I try to create a large dictionary of words
# This will allow me to plot them in a way where I can track the occurrences
# of each word on a per-country basis. Once this is plotted, it should show
# the "spread", so to speak, of words that are contained in articles pertaining
# to a certain place. I hope this will help create a visual aid of the feature
# vector cluster.
master_word_list = sorted(master_word_list)
master_word_dict = {}
index = 0
for element in master_word_list:
    master_word_dict[element] = index
    index += 1
colors = cm.rainbow(np.linspace(0, 1, len(text_list)))
x_dict = {}
index = 0
for element in text_list:
    array = word_count(element) # vector for that country
    if len(array) > 0:
        dict((master_word_dict[key], value) for (key, value) in array.items()) # Here, I replace
all keys (the individual words) with integer values from the dictionary.
        # The above move will create a dictionary of ordered pairs that I can graph visually and
see how things cluster together.
```

```
        # Now feature vector is in coordinate points that we can graph and cluster
        x, y = array.keys(), array.values() # unpack a list of pairs into two tuples for plotting
        x_dict.update(dict(zip(x, y))) # add x and y to an overal dictionary that could be fed
into the Kmeans model
        # THIS, however, failed to achieve the correct dimensionality, and it would be faster to
just use the CountVectorizer in this case
        try:
            plt.scatter(x,y,color=c) # Plot scatter plot of the data
        except:
            continue
    index += 1
    plt.show()
# Since word counts were programmed by hand in the last lab, we used the built-in sklearn library
for it this time for brevity
vectorizer = CountVectorizer(stop_words='english')
# This will allow us to use numbers instead of our bag of words approach
X = vectorizer.fit_transform(train_x['text'].values.astype('U')) # Encoding modifier to account
for nulls

# Using sklearn, we can immediately run kmeans clustering on the data, with k = 147, the total
number of countries in our vector
model = KMeans(n_clusters=147, init='k-means++', max_iter=300, n_init=1)
model.fit(X)

# Here, it seemed like a nice idea to find the top words in each cluster, to see if the
clustering was on the right path
print("Top words in each cluster:")
# Finding centroids and sorting them
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()

# Here, I print the clusters and their top centroids out, in order to see how words from the
vectors are being clustered
for i in range(147):
    print("Cluster %d:" % i),
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind])

print("Prediction")
plt.show() # Show the plot here so it appears after all of the cluster centroids

# Here we import some premade vectors from our lab 2 solution to use for testing our clustering
prediction
test_x = pd.read_csv("place_bag_test0" + ".csv", sep=',', encoding = "ISO-8859-1",
engine='python')

Y = vectorizer.transform(test_x['text'].values.astype('U'))
prediction = model.predict(Y) # use prediciton feature based on model and test data
print(prediction)
```

Source Code for dbscan_weighted_vector.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 27 02:03:02 2018
@author: Matthew Stock w/ Afnan's K-Means as a basis.
"""
import pandas as pd
import csv
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import DBSCAN
csv.field_size_limit(100000000)

# Import Weighted Bag of Words vector from previous lab.
train_x = pd.read_csv("place_bag_train0" + ".csv", sep=',', encoding = "ISO-8859-1",
engine='python')

# Since word counts were programmed by hand in the last lab, we used the built-in sklearn library
for it this time for brevity
vectorizer = CountVectorizer(stop_words='english')
# This will allow us to use numbers instead of our bag of words approach
X = vectorizer.fit_transform(train_x['text'].values.astype('U')) # Encoding modifier to account
for nulls

model = DBSCAN(eps = 20, min_samples = 3, metric='euclidean', n_jobs = -1)
model.fit(X, y=None, sample_weight=None)

core_samples_mask = np.zeros_like(model.labels_, dtype=bool)
core_samples_mask[model.core_sample_indices_] = True
labels = model.labels_

n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Silhouette Coefficient: %0.3f" % metrics.silhouette_score(X, labels))

terms = vectorizer.get_feature_names()
print(model.labels_)
print(len(model.labels_))

unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col), markeredgecolor='k',
markersize=14)

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col), markeredgecolor='k',
markersize=6)

plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

Source Code Citations:

Trie Source Code: https://towardsdatascience.com/implementing-a-trie-data-structure-in-python-in-less-than-100-lines-of-code-a877ea23c1a1

Naive Bayes Helper Code: https://www.kaggle.com/antmarakis/word-count-and-naive-bayes/notebook

Sklearn Documentation Reference: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html