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.

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.

# 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.

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.

# 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’.

# 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

Kumar Dhital –

Matthew Stock –

# 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

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

for element in text\_list:

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

plt.plot(x,y)

plt.show()

### Source Code for kmeans\_graph\_vector.py

# -\*- coding: utf-8 -\*-

"""

Created on Sun Nov 25 02:03:02 2018

@author: Afnan

"""

import pandas as pd

import csv

import numpy as np

import matplotlib.pyplot as plt

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.cluster import KMeans

csv.field\_size\_limit(100000000)

def word\_count(str):

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

train\_x = pd.read\_csv("place\_bag\_train0" + ".csv", sep=',', encoding = "ISO-8859-1", engine='python')

vectorizer = CountVectorizer(stop\_words='english')

text\_list = train\_x.text.tolist()

master\_word\_list = []

for element in text\_list:

array = word\_count(element) # vector for that country

#print(array)

i = 0

for word in array:

#print(word)

if word not in master\_word\_list:

master\_word\_list.append(word)

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

np\_list = []

for element in text\_list:

array = word\_count(element) # vector for that country

#for word in array:

dict((master\_word\_dict[key], value) for (key, value) in array.items())

'''

new\_word = ""

if word in master\_word\_dict:

#word = master\_word\_dict[word]

array[master\_word\_dict[word]] = array[word]

del array[word]

np\_list.append(new\_word)

'''

# 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

plt.plot(x,y)

plt.show()

X = np.array(np\_list)

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 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>