CSCI 6515 Assignment 2 – part 1

Enron Email Dataset Assignment

Deepak Munjal
B00748375

Introduction:-

This assignment includes work on the Enron Email Dataset. First of all preprocessing needs to be done, which includes removing data which is not of much use in terms of clustering and further analysis. This is followed by data discovery i.e. finding the statistics of data. Then applying k-means to understand the clustering in emails and LDA to understand topic model of emails.

Data Preprocessing and Data Discovery:-
This step includes cleaning data. In this all header information, attachment information, stopping words are removed and stemming is applied on the data. In this I began with traversing all files and using some regular expressions to remove the unwanted data. After removing that, other data i.e. email sent count, name of sender, Average length of an email etc. is calculated as shown in the code below:-

```
import nltk
import os
import time
import re
import pandas as pd
from nltk.corpus import stopwords
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
from email.parser import Parser
from nltk.stem.porter import *
from stemming.porter2 import stem

f_name=""
check='True'
a=-1
for route,directories,files in os.walk('/media/deepak/temp/'):
  if os.path.dirname(os.path.dirname(route))=='/media/deepak/temp/':
    counter_words=0
    counter_emails=0
  for file in files:
    main_path=os.path.dirname(os.path.join(route,file))
    for check in 'True':
      if os.path.dirname(main_path)=='/media/deepak/temp/':
        break
      main_path=os.path.dirname(main_path)
    f_name=main_path.split('/')
```

```
      main_path=main_path+".txt"
      openFile=open(main_path,'a+')
      with open(os.path.join(route,file)) as fl:
        for lines in fl:
            counter_words=counter_words+lines.__len__()
            lines=re.sub('Subject:','',lines)
            lines=re.sub('(http(s)?://)?www\..*|[^a-zA-Z0-9 ]|[\w\-\W]+:.*|[?|$|-|.]|.*Forwarded by.*|The following section of this message contains a file attachment.*|.*Internet MIME message format.*|.*MIME-compliant system.*|.*save it or view it from within your mailer.*|.*please ask your system administrator.*|\-+ [\w\d\s]+ \-+|\=+ [\w\d\s]+ \=+|\-\s*[\d\-\w\W]+.\w+','',lines)
            stopping_words_removed=" ".join([word for word in lines.split() if word not in set(stopwords.words('english'))])
            print stopping_words_removed
            #stemmed_line =" ".join([stem(word) for word in stop_line.split()])
            #print stemmed_line
            openFile.write(stopping_words_removed)
      openFile.close()
      counter_emails+=1
      f_name=f_name[-1]
    if os.path.dirname(route)=='/media/deepak/temp/':
      a+=1
      if counter_words>0 and a>0:
        print("email sent count is %d,name is %s,Average length of an email is %d\n" % (counter_emails,f_name,counter_words/counter_emails))
```

Data Analysis:-

k-means:-

In this step, first the data is processed to feed it to the kmeans for clustering. The path is traversed and for each user all the data is converted to lower text and is splitted into words. Then a dictionary is made, which contains key, value pairs containing the word and its counter. Only the words having length more than four are considered. And then most frequently used 300 words are found. And total number of words are calculated and then using this data, probability of a word occurring is found. The code is as shown below:-

```
import re
import os
import sys
from collections import OrderedDict
from itertools import islice
```

```
for route,directories,files in os.walk('/media/deepak/file_folder'):

    for file in files:
        f_name=os.path.join(route,file).split('/')
        openfile=open(os.path.join(route,file), "r+")
        data_file=openfile.read()
        openfile.close()
        data_file=re.sub(r'\W+', ' ',data_file)
        data_file=data_file.lower()
        all_words=data_file.split(' ')
        d={}
        file_created="/media/deepak/"+f_name[-1]+"data_words.csv"
        file_w=open(file_created, 'a+')
        for eachword in all_words:
          if eachword.__len__()>4:
            if eachword in d:
              d[eachword]+=1
            else:
              d[eachword]=1
        total_word_count=0
        k=0
        d=OrderedDict(sorted(d.items(),key=lambda x: x[1], reverse=True))
        for keys,values in d.items():
            total_word_count=values+total_word_count
            print(keys)
            print(values)
            k=k+1
            if k==300:
              break
        k=0
        for keys,values in d.items():
           prob_of_word_occuring=float(float(values)/total_word_count)
           print "%d%s%d,%f\n" % (len(keys),',',values,prob_of_word_occuring)
           file_w.write("%d%s%d,%f\n" % (len(keys),',',values,prob_of_word_occuring))
           k=k+1
           if k==300:
             break
        file_w.close()
```

And then this data which contains length of a word, its frequency of occurrence and its probability of occurring is saved in a csv, which acts as input for kmeans and LDA.

The processed data so far, is sent to kmeans, which evaluates the centers of clusters. In this assignment k is considered as 4. This analysis helps in understanding the clustering in emails.

#This has been taken and edited as per task from
http://spark.apache.org/docs/latest/mllib-clustering.html#k-means

```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array
from math import sqrt
from pyspark import SparkContext
import sys
import os

sc =SparkContext()
for route,directories,files in os.walk('/media/deepak/data_words.csv'):
    for file in files:
        f_name=os.path.join(route,file).split('/')
        # Load and Parse the data
        data = sc.textFile(os.path.join(route,file))
        dataParsed = data.map(lambda line: array([float(x) for x in line.split(',')]))
        # Build the model (cluster the data)
        clusters = KMeans.train(dataParsed, 4, maxIterations=10,
                runs=10, initializationMode="random")
        # Evaluate clustering by computing Within Set Sum of Squared Errors
        def error(point):
            center = clusters.centers[clusters.predict(point)]
            return sqrt(sum([x**2 for x in (point - center)]))
        WSSSE = dataParsed.map(lambda point: error(point)).reduce(lambda x, y: x + y)
        print(f_name[-1])
        print('\n')
        print("Within Set Sum of Squared Error is " + str(WSSSE))
        print('\n')
        print(clusters.clusterCenters)
        # Save and load model
        clusters.save(sc, "/media/deepak/Kmean_output/"+f_name[-1])
        sameModel = KMeansModel.load(sc, "/media/deepak/Kmean_output/"+f_name[-1])
```

LDA:-

The processed data so far, is sent to LDA also, and which returns the topics as distributions over vocabulary. This shows why some parts of data are similar. This helps to understand topic model of emails.

```
#This has been taken and edited as per task from
http://spark.apache.org/docs/latest/mllib-clustering.html#latent-dirichlet-allocation-lda

import os
import sys
from pyspark.mllib.clustering import LDA,LDAModel
from pyspark.mllib.linalg import Vectors
from pyspark import SparkContext

sc =SparkContext()
for route,directories,files in os.walk('/media/deepak/data_words.csv'):
   for file in files:
      f_path=os.path.join(route,file)
      f_name=os.path.join(route,file).split('/')
      # Load and parse the data
      data = sc.textFile(f_path)
      dataParsed = data.map(lambda line: Vectors.dense([float(x) for x in line.strip().split(',')]))
      # Index documents with unique IDs
      corpus = dataParsed.zipWithIndex().map(lambda x: [x[1], x[0]]).cache()
      # Cluster the documents into three topics using LDA
      ldaModel = LDA.train(corpus, k=3)
      # Output topics. Each is a distribution over words (matching word count vectors)
      print(f_name[-1])
      print("Learned topics (as distributions over vocab of " + str(ldaModel.vocabSize()) + "
words):")
      topics = ldaModel.topicsMatrix()
      for topic in range(3):
         print("Topic " + str(topic) + ":")
         for word in range(0, ldaModel.vocabSize()):
            print(" " + str(topics[word][topic]))
      # Save and load model
      ldaModel.save(sc, "/media/deepak/lda_output/"+f_name[-1])
      sameModel = LDAModel.load(sc, "/media/deepak/lda_output/"+f_name[-1])
```

Technical problems faced:-

In the data preprocessing, I encountered problem in cleaning data. While doing header information removal, as there were lots of patters and other text, and it was bit difficult to write a single regex for all these, so I tried covering as many scenarios as possible. But it did made the regex part little bit heavy.

Other issue I encountered was during data stemming. Data Stemming converts each word to its root word. I tried to write the code, but it started showing some error and after investigation and analysis also, I was not able to resolve the issue in data stemming. Hence the code which I wrote has been included in the assignment but is commented.
Moreover, I faced few problems while running my code on AWS, and hence I implemented this on my local machine. And then while doing Data analysis for kmeans and LDA, statistics related to a user was successfully extracted but I was not able to extract communication (number of emails sent and received from a particular user) of a user with all other users i.e. their contacts. In kmeans and LDA, I was successfully able to implement it and create output files.