Q1) Natural Language Processing

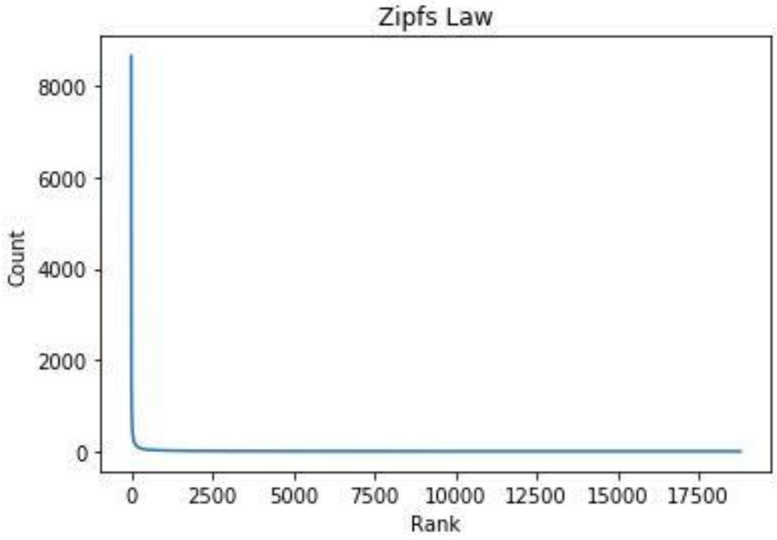
1) Total word and token 18787 and 168253

2) Top 20 types with count

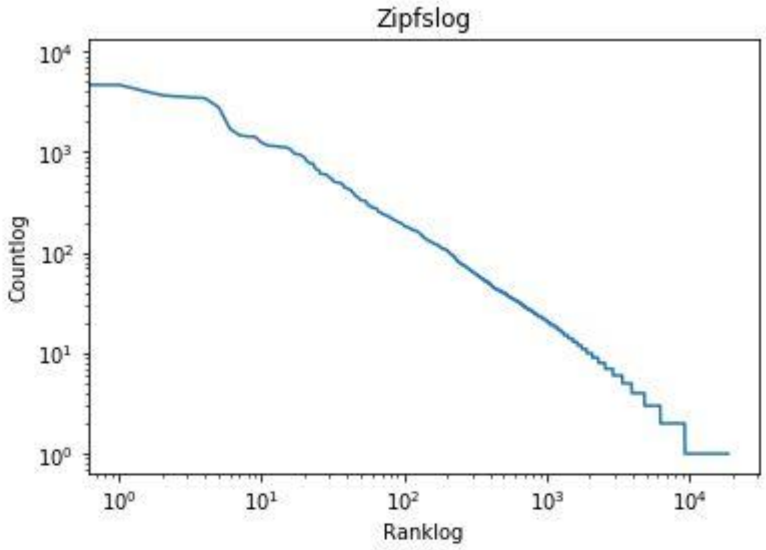
```
[('the', 8651), ('to', 4663), ('a', 3673), ('in', 3521), ('and',
3446), ('of', 2792), ('for', 1711), ('is', 1470), ('on', 1432),
('was', 1421), ('he', 1244), ('with', 1166), ('have', 1152), ('a
t', 1137), ('I', 1126), ('his', 1111), ('that', 1060), ('has', 9
65), ('be', 950), ('but', 931)]
```

3)

3.1)



3.2)



3.3) Observation from the curve showed that this dataset Holds Zipf's law.
The more occurrence of dataset, more insignificant the type is.

4)

```
{'Ronaldo': 0.03394311933338285, 'contract': 0.025335234491445598,
'United': 0.01950804832931014, 'Trafford.': 0.01921468057429667, 'f
ive-year-deal,': 0.01921468057429667, 'first-team.': 0.019214680574
29667, 'World.': 0.01921468057429667, 'tomorrow.': 0.01921468057429
667, 'knows,"': 0.01921468057429667, 'club.': 0.01860042485537295}
```

5)
```
0.966392440693141
0.34291035547055376
```

The two cosine similarity are not the same. Since the calculation consider only current document, The types that considered important are not the same as the importance calculated from over all token distribution.

6) The major issue :-

1. Word with punctuation marks are counted as different words.

The non-important word can be count as different word so importance is increased by not adding two types of together.
Ex: for low repeating word like 'car.' and 'car' can be counted as different type, making the same type less likely to become the important info of the text.

2. Word with uppercase are different from lowercase.

The non-important word can be count as different word so importance is increased by not adding two types of together.
Ex: For high repeating word like 'The' and 'the', are same non-important words but, not taking them together make each of them twice important.

Q2) Principal Component Analysis

1) Dimension of vector: 356*1
   Retail Mean: 32511.33146067416
   Horsepower Mean: 0.75

2) First: [-0.01382535 0.62459663 0.74745736 -0.04927839 -0.09408548 0.15688077 0.07951538 -0.01130479 -0.08826296 0.00343817 -0.02916711]

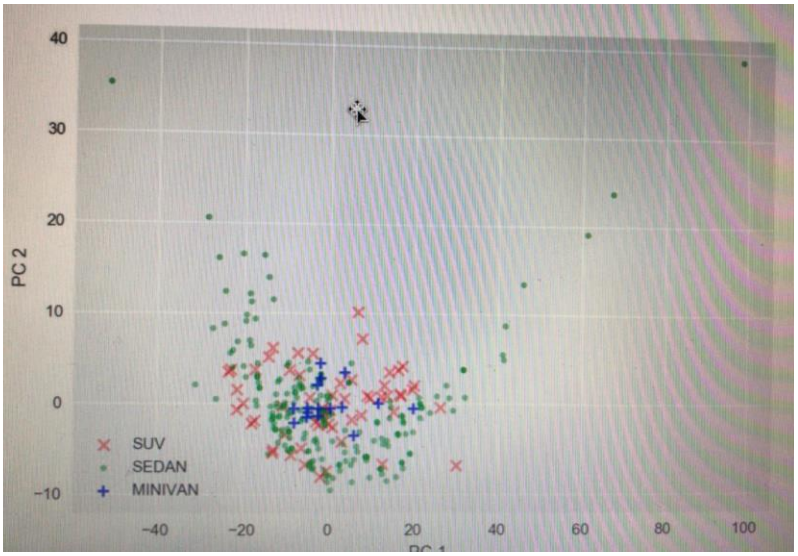
   Third: [0.01700774 0.54540029 -0.60623619 -0.36830448 -0.2802661 0.26585027 0.17801119 0.01087792 -0.1302184 0.00695671 -0.03227632]

3) It means that horsepower is always a positive factor in the eigenvector.

The sign (positive or negative) tells you the direction that a given variable in that PC is going on a single dimension vector.

Y coordinates are positive. Refers to horsepower.

4)



5)

Bob can conclude that Sedan are clustered most strongly as horsepower increases, increasing the retail to a level.

Appendix

Q1)

```
import collections
import math
import matplotlib.pyplot as plt

#cosine similarity
def simulator(x, y):
    dX = 0
    dY = 0
    dXY = 0
    newlength = len(x)
    if len(x) > len(y):
        newlength = len(y)
    for i in range(newlength):
        dX = dX + x[i] * x[i]
        dY = dY + y[i] * y[i]
        dXY = dXY + x[i] * y[i]
    normX = math.sqrt(dX)
    normY = math.sqrt(dY)

    return dXY / (normX * normY)

wdCt1 = 0
wdCt2 = 0
sbCrps1  = collections.Counter()
```

```
sbCrps2  = collections.Counter()
tknCt = 0
bow1 =[]
bow2 = []
v1 = []
v2 = []
corpus = collections.Counter()

for i in range(1,512):
    if i < 10:
        fileName = "00"+str(i)
    elif i < 100:
        fileName = "0"+str(i)
    else:
        fileName = str(i)

    f = open("files/news/{}.txt".format(fileName),"r+",encoding="utf-8");

    for i,line in enumerate(f):
        Split = line.strip().split()
        for word in Split:
            tknCt += 1;
            corpus[word] +=1;
            if fileName  == '098':
                wdCt1 += 1;
                sbCrps1[word] += 1;
            elif fileName == '287':
                wdCt2 += 1;
                sbCrps2[word] += 1;
    f.close()


#Q1-1
print("Total word and token {0} and {1}".format(len(corpus),tknCt))
#Q1-2
print()
print("Top 20 types with count")
print(corpus.most_common(20))
#Q1-3
#1
rnList = [ i for i,data in enumerate(corpus.most_common())]
coutList = [data[1] for data in corpus.most_common()]
fig1 = plt
fig1.plot(rnList, coutList)
fig1.ylabel('Count')
fig1.xlabel('Rank')
fig1.title('Zipfs Law')
fig1.show()
#2
fig2 = plt
fig2.plot(rnList, coutList)
fig2.yscale('Log')
fig2.xscale('Log')
fig2.ylabel('Countlog')
fig2.xlabel('Ranklog')
fig2.title('Zipfslog')
fig2.show()
```

```
#Q1-4
tiDict1 = {}
for tup in sbCrps1.most_common():
    HitSet = set();
    bow1.append(tup[1]/wdCt1)
    for i in range(1,512):
        if i < 10:
            fileName = "00"+str(i)
        elif i < 100:
            fileName = "0"+str(i)
        else:
            fileName = str(i)
        f = open("files/news/{}.txt".format(fileName),"r+",encoding="utf-8");

        for i,line in enumerate(f):
            Split = line.strip().split()
            if tup[0] in line:
                HitSet.add(fileName)
        f.close()
    tf = sbCrps1[tup[0]]/wdCt1
    if len(HitSet) == 0:
        idf = math.log(512/(len(HitSet)+1),10)
    else:
        idf = math.log(512/len(HitSet),10)
    tfidf = tf * idf
    tiDict1[tup[0]] = tfidf
    v1.append(tfidf)
topRate = dict(collections.Counter(tiDict1).most_common(10))
print(topRate)
```

```
#Q1-5
tiDict2 = {}
for tup in sbCrps2.most_common():
    HitSet = set();
    bow2.append(tup[1]/wdCt2)
    for i in range(1,512):
        if i < 10:
            fileName = "00"+str(i)
        elif i < 100:
            fileName = "0"+str(i)
        else:
            fileName = str(i)

        f = open("files/news/{}.txt".format(fileName),"r+",encoding="utf-8");

        for i,line in enumerate(f):
            Split = line.strip().split()
            if tup[0] in line:
                HitSet.add(fileName)
        f.close()
    tf = sbCrps2[tup[0]]/wdCt1
    if len(HitSet) == 0:
        idf = math.log(512/(len(HitSet)+1),10)
    else:
        idf = math.log(512/len(HitSet),10)
    tfidf = tf * idf
    tiDict2[tup[0]] = tfidf
    v2.append(tfidf)
```

```
Bowsim = simulator(bow1,bow2)

TIsimulator = simulator(v1,v2)

print(Bowsim)

print(TIsimulator)
```

Q2)

```
import numpy as np

import sys

import csv

import pandas as pd

from itertools import islice

from collections import defaultdict

import math

import matplotlib.pyplot as plt


# def readfile(file):

def mean(df, str):

   sum = 0

   count = 0

   list = df[str]


   for val in list:

      sum = sum + val

      count = count + 1

   return sum / count


def stdev(df, str):

   list = df[str]

   stdev = np.std(list)
```

```
    return stdev

def allstdev(df):
    stdev_list = []

    for column in df.columns[2:]:
        stdev_list.append(find_stdev(df, column))
    return stdev_list

def allmean(df):

    mean_list = []

    for column in df.columns[2:]:
        mean_list.append(find_mean(df, column))
    return mean_list

def manipu(df, label, features):
    stdev_list = all_stdev(df)
    mean_list = all_mean(df)
    norm_features = features
    count = 0

def normal(val, stdev, mean):
    norm = (val - mean)
    norm = norm / stdev
    return norm
```

```
def eigenv(df, val):


def main():
    df = pd.read_csv('venv/cardata.csv')
    label = df['Category']
    features = df.loc[:, 'Retail($)':]
    print(find_mean(df, 'Retail($)'))  # finds non-normalized mean of price
    print(find_mean(df, 'Horsepower'))  # finds non-normalized mean of horsepower

    manipulate(df, label, features)

    # find_eigenvalue(1)
    #find_eigenvalue(3)

    # Read car file into list of tuples
    # readfile(read)
```