## Problem 1: Natural Language Processing

1. 168,253 total words. 18,787 computer word types.
2. 20 Most common words:

('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),

('at', 1137),

('I', 1126),

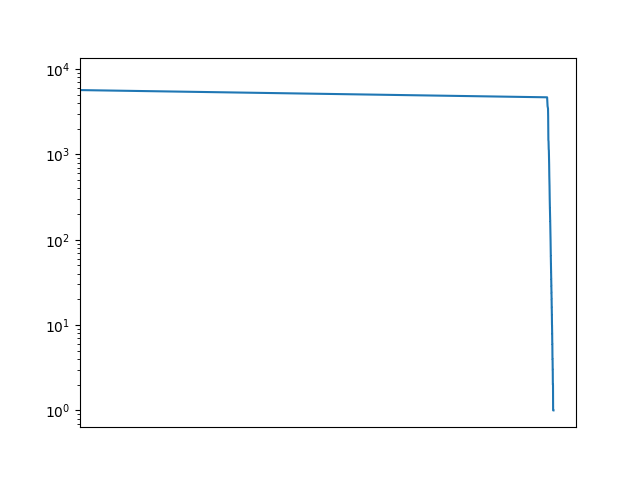
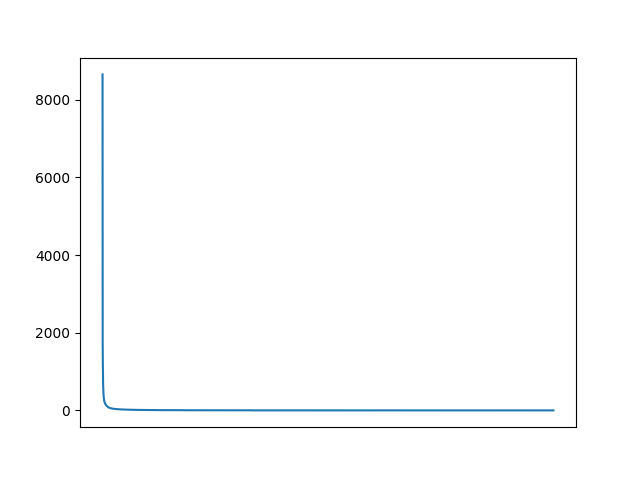
('his', 1111),

('that', 1060),

('has', 965),

('be', 950),

('but', 931)

1. The first graph shows how only a few words occur often, and most words only occur seldomly. The second graph shows how, for the most part, the ratio between the rank and count of any word is constant across all words in the corpus.
2. tf-idf of ‘contract’: 0.06256452604679522

ten most common words by tf-idf:

[('Ronaldo', 0.08369754249382252),

('contract', 0.06256452604679522),

('United', 0.04821441865216133),

('Trafford.', 0.0442295715617284),

('five-year-deal,', 0.0442295715617284),

('first-team.', 0.0442295715617284),

('World.', 0.0442295715617284),

('tomorrow.', 0.0442295715617284),

('knows,"', 0.0442295715617284),

('club.', 0.04398404786205042)]

* 1. Cosine similarity of BoW vectors: 0.5819694066677242
  2. Cosine similarity of tf-idf vectors: 0.0533427215549849
  3. These similarities are not the same since tf-idf prioritizes words that have less frequency in other documents. In this way it can sort of skip over the typical stopwords. The words that remain have a more profound effect on the cosine similarity. Given that the two articles are about different things, it makes more sense that they are 5% similar rather than 58%.

1. One of the major issues with this form of computer words is that there is no preprocessing done to them. This means that words that do not take the exact same form (i.e. start of a sentence, end of a sentence, has adjacent punctuation) will not be identified as the same. If an article had a word at the beginning of a sentence half of the time and not for the other half, the importance of the word would be skewed because its count has been split in two. Additionally, our computer words do not take context into account. The meaning of some words can change drastically depending on the context. Without this, the overall meaning of words and entire articles can be lost.

## Problem 2: Principal Component Analysis

1. The vector is 11-dimension.

Mean of retail: 32511.33146067416

Mean of horsepower: 213.2191011235955

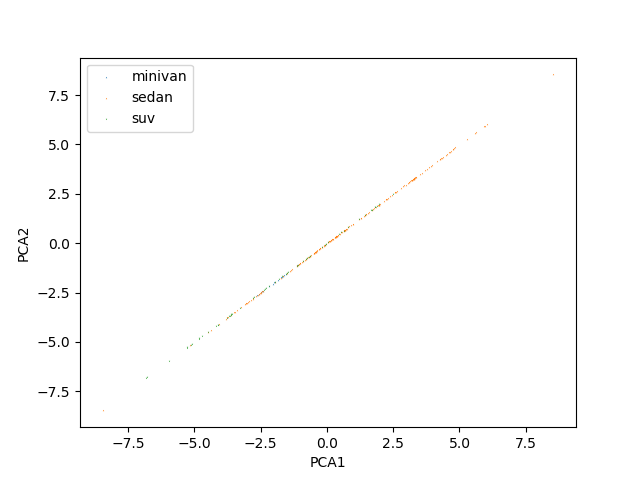
2. Eigenvector with largest eigenvalue:

[-0.27526177 0.44414216 0.25904398 -0.27683893 0.10048994 0.01031016 0.22926907 -0.71010606 -0.03948592 -0.04431689 0.11261683]

Eigenvector with third largest eigenvalue:

[-0.34518922 -0.0135008 0.06436867 0.53355309 -0.02940948 0.09136302 -0.0188246 0.01105705 -0.05070988 0.40895466 0.64213375]

3. In the first eigenvector, coordinates 2, 3, 5, 6, 7, and 11 are positive. The first eigenvector indicates the direction in real space that the first principal component would have. Thus those coordinates being positive indicate that the first principal component would be positive in that dimension.

4. 

5. Based on the plot, the minivans are clustered the most strongly. From this plot, Bob can conclude that most of the difference between sedans and suvs will be along this axis, but most of the difference for minivans may be elsewhere.

# Appendix

nlp.py

*"""HW6"""*

import collections

import os

import matplotlib.pyplot as plt

import math

class CounterSource:

def \_\_init\_\_(self, source):

self.count = 0

self.sources = [source]

self.numSources = 1

def \_\_lt\_\_(self, other):

return self.count < other.count

def \_\_gt\_\_(self, other):

return self.count > other.count

def \_\_eq\_\_(self, other):

return self.count == other.count

def \_\_ne\_\_(self, other):

return self.count != other.count

def add(self, source):

self.count += 1

if source not in self.sources:

self.sources.append(source)

self.numSources += 1

def dot\_product(A, B):

return sum(i[0] \* i[1] for i in zip(A, B))

def cosine\_similarity(A, B):

return dot\_product(A, B) / (math.sqrt(dot\_product(A, A)) \* math.sqrt(dot\_product(B, B)))

def count\_all\_words():

ctr = {}

for filename in os.listdir('news'):

with open('news/' + filename, 'r') as file:

for word in file.read().split():

if word in ctr.keys():

ctr[word].add(filename)

else:

ctr[word] = CounterSource(filename)

print("Counted all words.")

return ctr

def rc():

*"""RC"""*

*# Tally occurrences of words in a file*

ctr = collections.Counter()

for filename in os.listdir('news'):

with open('news/' + filename, 'r') as file:

for word in file.read().split():

ctr[word] += 1

print(sum(ctr.values()))

print(len(ctr.items()))

print(ctr.most\_common(20))

*#reorganize data to be by most common*

rc\_data = []

ordered\_keys = []

for word in ctr.most\_common():

ordered\_keys.append(word[0])

rc\_data.append(word[1])

*# RC plot*

*# Shows that there are only a few words that are used many times,*

*# the majority of words is only used a few times.*

fig, ax = plt.subplots()

ax.plot(ordered\_keys, rc\_data)

ax.get\_xaxis().set\_visible(False)

plt.savefig('rc.png')

*# LogR LogC plot*

*# Shows that the ratio between rank and count is, for the most part,*

*# the same across all words*

fig, ax = plt.subplots()

ax.set\_yscale('log')

ax.set\_xscale('log')

ax.plot(ordered\_keys, rc\_data)

ax.get\_xaxis().set\_visible(False)

plt.savefig('logrlogc.png')

def analyze\_file(filename, all\_words):

*"""TF\_IDF"""*

bow\_ctr = collections.Counter()

tf\_idf\_ctr = collections.Counter()

target\_count = 0

doc\_word\_count = 0

with open('news/' + filename) as file:

*# initialize counters*

for word in all\_words.keys():

bow\_ctr[word] = 0

print("Initialized BoW counters.")

*# compute word counts*

for word in file.read().split():

bow\_ctr[word] += 1

doc\_word\_count += 1

print("Completed BoW counting.")

*# convert to if-idf format*

for word in all\_words.keys():

tf = bow\_ctr[word] / float(doc\_word\_count)

idf = math.log(511 / float(all\_words[word].numSources))

tf\_idf\_ctr[word] = tf \* idf

print("Completed tf-idf computation.")

return bow\_ctr, tf\_idf\_ctr

def vector\_compare():

all\_ctr = count\_all\_words()

v1\_bow, v1\_tf\_idf = analyze\_file('098.txt', all\_ctr)

v2\_bow, v2\_tf\_idf = analyze\_file('297.txt', all\_ctr)

print(cosine\_similarity(v1\_bow.values(), v2\_bow.values()))

print(cosine\_similarity(v1\_tf\_idf.values(), v2\_tf\_idf.values()))

def main():

*#rc()*

*# tf\_idf()*

vector\_compare()

if \_\_name\_\_ == '\_\_main\_\_':

main()

pca.py

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

import pandas as pd

*# Formate csv data into matrix*

data = pd.read\_csv('cardata.csv', ',').iloc[:, 2:13].values

print("Mean of retail: ", np.mean(data[:,0]))

print("Mean of horsepower: ", np.mean(data[:,4]))

*#Centralize and normalize data*

data\_std = StandardScaler().fit\_transform(data)

*# Get covariance matrix from standardized data*

cov = np.cov(data\_std.T)

*# Get eigenvectors and eigenvalues*

eigen = np.linalg.eig(cov)

e\_values = eigen[0]

e\_vectors = eigen[1]

*# Sort eigenvalues by descending value*

print("E-values pre sort: ", e\_values)

e\_values[::-1].sort()

print("Biggest three E-values:", e\_values[0:3])

print("First E-vector:", e\_vectors[0])

print("Third E-vector:", e\_vectors[2])

*# Get first to PCAs from some of standardized data*

pca = PCA(n\_components=2).fit\_transform(data\_std)

*#Filter out data*

minivan\_x = []

minivan\_y = []

sedan\_x = []

sedan\_y = []

suv\_x = []

suv\_y = []

with open('cardata.csv', 'r') as file:

next(file)

line\_num = 0

for line in file:

words = line.rstrip().split(',')

if words[1] == 'minivan':

minivan\_x.append(pca[line\_num][0])

minivan\_y.append(pca[line\_num][0])

elif words[1] == 'sedan':

sedan\_x.append(pca[line\_num][0])

sedan\_y.append(pca[line\_num][0])

elif words[1] == 'suv':

suv\_x.append(pca[line\_num][0])

suv\_y.append(pca[line\_num][0])

line\_num += 1

plt.scatter(minivan\_x, minivan\_y, label='minivan', marker='^', linewidths=0, s=1)

plt.scatter(sedan\_x, sedan\_y, label='sedan', marker='^', linewidths=0, s=1)

plt.scatter(suv\_x, suv\_y, label='suv', marker='v', linewidths=0, s=1)

plt.xlabel('PCA1')

plt.ylabel('PCA2')

plt.legend()

plt.savefig('pca.png')