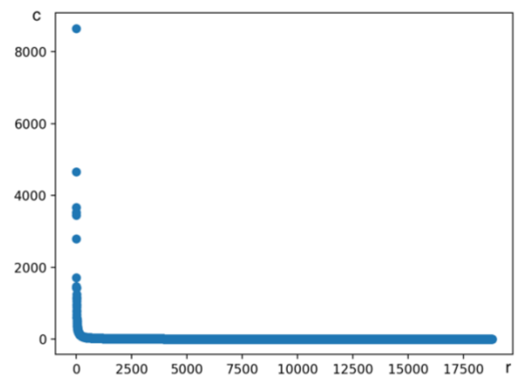
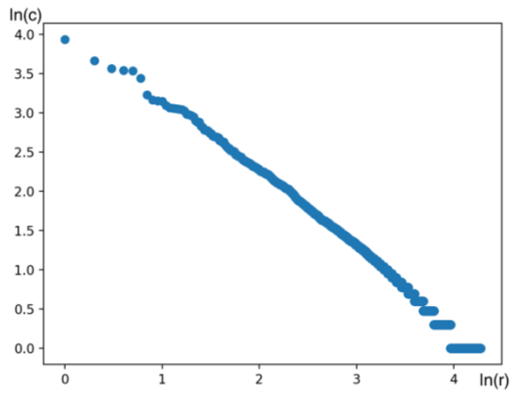
Yuhao Liu

# CS 540: Introduction to Artificial Intelligence
# Homework #6

## Problem 1: Natural Language Processing

1. Total occurrence = 168253
   Word types = 18787
2. Ranking top-20 word types and their respective counts
   1) ('the', 8651)
   2) ('to', 4663)
   3) ('a', 3673)
   4) ('in', 3521)
   5) ('and', 3446)
   6) ('of', 2792)
   7) ('for', 1711)
   8) ('is', 1470)
   9) ('on', 1432)
   10) ('was', 1421)
   11) ('he', 1244)
   12) ('with', 1166)
   13) ('have', 1152)
   14) ('at', 1137)
   15) ('I', 1126)
   16) ('his', 1111)
   17) ('that', 1060)
   18) ('has', 965)
   19) ('be', 950)
   20) ('but', 931)
3. Plotting ($r_n$, $c_n$) pairs, where $r_i$ is the rank of ith word type and $c_i$ is the corresponding count of that word type, and n is the number of word types.





The first graph is looks similar to the right half of hyperbola, where c rapidly decreases when r is small and r get bigger, it stays at a minimal value above the x axis.

The second graph is nearly a linear graph, although it concaves down slightly and when ln(r) approaches 4, the graph is discontinuous and shows multiple horizontal segments.

4. The top 10 words with the highest tf-idf value in the file 098.txt are as follows:
   1) ('Ronaldo', 0.7321803857720196)
   2) ('contract', 0.5473102011324537)
   3) ('United', 0.4217764416573958)
   4) ('Trafford.', 0.386917271447816)
   5) ('five-year-deal,', 0.386917271447816)
   6) ('first-team.', 0.386917271447816)
   7) ('World.', 0.386917271447816)
   8) ('tomorrow.', 0.386917271447816)
   9) ('knows,"', 0.386917271447816)
   10) ('club.', 0.3847694468906056)
   The tf-idf value for the word 'contrast' is 0.5473102011324537
5. Between file 098.txt and 297.txt:
   a. Cosine Similarity using BoW: 0.5819694066677241
   b. Cosine Similarity using tf-idf: 0.053342721554948895

The two values are not the same, which is largely due to that the tf-idf model reflect how important a word is to a document in a collection or corpus, while the BoW model only calculates the frequency within a certain document.

6. There are two major issues with my model of computer words. The first of which is that the model does not take capitalization in account, where it treats "The" and "the", for example, as two different word types, yielding a lower frequency. The second issue is that the model is unable to process word tense, again lowering the frequency of words with different tenses.

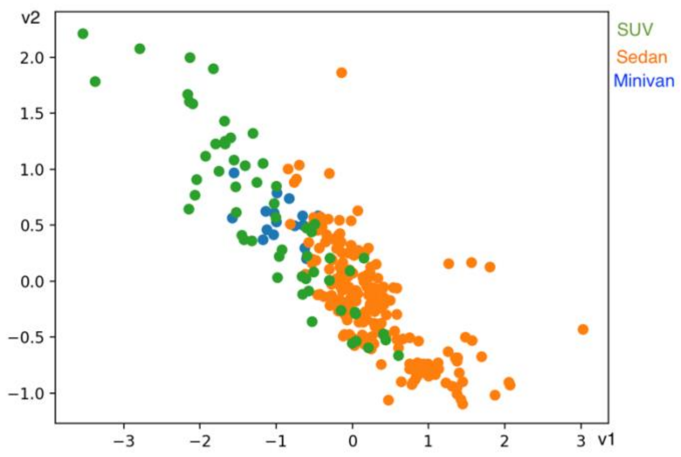
Problem 2: Principle Component Analysis

1. Before normalization:
   a. Dimension of vectors: 11
   b. Mean value of retail: $32511.33146067416
   c. Mean value of horsepower: 213.2191011235955
2. Matrix decomposition:

First eigenvector: [-0.27526177, 0.44414216, -0.25904398, 0.27683893, -0.10048994, -0.01031016, 0.22926907, -0.71010606, -0.03948592, -0.04431689, 0.11261683]
Third eigenvector: [-0.34518922, -0.0135008, -0.06436867, -0.53355309, 0.02940948, -0.09136302, -0.0188246, 0.01105705, -0.05070988, 0.40895466, 0.64213375]

3. Features corresponding to positive eigenvector entries:
   ['Dealer($)', 'Cylinders', 'HighwayMPG', 'Width(in)']
   Positive coordinates means that these features are positively correlated.

4.



5. Sedan is clustered most strongly. It means that sedan has the least variation along the two principle components.

Yuhao Liu

## Appendix

### NLP.py

```
'''
Created on Mar 27, 2019

@author: LiuYuhao
'''
from numpy import sort

''' ROADMAP
Input all files and store in separate classes
Preprocessing
Tokenization
Build bag-of-words representation of each documents using collection.counter
tf-idf
'''
import os
import math
import matplotlib.pyplot as plt
import numpy as np
from numpy import dot
from numpy.linalg import norm
from collections import Counter

docs = []
templ = []


def main():
    #method variables
    global docs
    global templ
    global occurrance
    global wordType

    docs = []
    templ = []
    stopwords = ["a", "the", "of", "with"]
    os.chdir('news')
    index = 1;

    while index <= 511:
```

Yuhao Liu

```
        # open file
        file = open("{:03d}.txt".format(index))
        lines = file.read() # convert all to lowercase; type = str
        # tokenize
        tokens = lines.split()
        # pre-processing
        ct = Counter()
        for token in tokens:
            ct[token]+= 1
        docs.append(ct)
        file.close()
        index+=1
    print("----Finished building all docs----")

    # combine all docs to obtain dictionary
    corpus = Counter()
    for doc in docs:
        corpus += doc
    occurrance = sum(corpus.values())
    wordType =    len(corpus)
    print("Total occurrence = {}".format(occurrance))
    print("Word types = {}".format(wordType))

    # building corpus template
    templ = []
    for x, y in list(corpus.most_common()):
        templ.append(x)

    print('----Ranking Top 20 word types----')
    rankedCorpus = corpus.most_common(20)
    index = 1;
    for word in rankedCorpus:
        print("{}) {}".format(index, word))
        index+= 1;

    # plotting graph
    print('---plotting graph----')
    x = range(1, wordType + 1)
    y = sorted(list(corpus.values()), reverse = True)
    logx = np.log10(x)
    logy = np.log10(y)
    plt.scatter(x, y)
    plt.show()
    plt.scatter(logx, logy)
    plt.show()

    # TF-IDF processing
    v_098= getBoWVectorFor(docs[97])
    v_297= getBoWVectorFor(docs[296])
    v_098_tf= getTfidfVectorFor(docs[97], 'contract')
    v_297_tf= getTfidfVectorFor(docs[296])

    print('----TOP 10 TF-IDF in 098.txt----')
    sortedTuple = sorted(list(zip(templ, v_098_tf)), key = lambda tup: tup[1], reverse =
True)[0:10]
    index = 0
    for entry in sortedTuple:
        print('{}) {}'.format(index+ 1, entry))
        index += 1

    print('----Calculating Cosine Similarity----')
    print('Between file 098.txt and 297.txt:')
    print('Cosine Similarity using BoW: {}'.format(calCosSim(v_098, v_297)))
    print('Cosine Similarity using tf-idf: {}'.format(calCosSim(v_098_tf, v_297_tf)))
    print('-----------------------------')


def calCosSim(v1, v2):
    cos_sim = dot(v1, v2)/(norm(v1)*norm(v2))
    return cos_sim

def getBoWVectorFor(doc):
    token_count = sum(doc.values())
    v = [0] * wordType
    for w in doc:
        c = doc.get(w)
        v[templ.index(w)] = c / token_count
    return v

def getTfidfVectorFor(doc, checkWord = None):
    max_c = doc.most_common(1)[0][1]
    # print(max_c)
    v = [0] * wordType
    for w in doc:
        c = doc.get(w)
        tf_w = c / max_c
        idf_w = math.log(511 / docsContainW(w), 10)
        v[templ.index(w)] = tf_w * idf_w
        if (checkWord != None and w == checkWord):
            print('tf = {}, idf = {}'.format(tf_w, idf_w))

    return v

def docsContainW(w):
    count = 0
    for doc in docs:
        if doc.__contains__(w):
            count += 1
    return count

if __name__ == '__main__':
    main()
```

## PCA.py

```
'''
Created on Mar 31, 2019

@author: LiuYuhao
'''
import pandas as pd
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt

def main():
    cd = pd.read_csv('cardata.csv')

    print('----Basics----')
    d = 13 - 3 + 1
    print('Dimension: {}'.format(d))
    mean_retail = cd['Retail($)'].mean()
    print('Retail Mean: ${}'.format(mean_retail))
    mean_hp = cd['Horsepower'].mean()
    print('Horsepower Mean: ${}'.format(mean_hp))
```

Yuhao Liu

```
    print('----Normalizing Data----')
    normalizedMX = normalize(cd)
    covarMX = normalizedMX.cov()

    print('----Matrix Decomposition----')
    eigVal, eigVec = LA.eig(covarMX)
    eigTup = list(zip(eigVal, eigVec))
    sortedEig = sorted(eigTup, key = lambda x: x[0], reverse = True)
    print("First eigenvector: {}".format(sortedEig[0][1]))
    print("Third eigenvector: {}".format(sortedEig[2][1]))

    print('----Eigenvector Analysis----')
    posEigFeatures = []
    featureList = list(cd)
    index = 0
    for x in sortedEig[0][1]:
        if x>0:
            posEigFeatures.append(featureList[index + 2])
        index += 1
    print('Features corresponding to positive eigenvector entries: \n{}'.format(posEigFeatures))

    print('----Principle Component Analysis----')
    subData = normalize(cd, drop = False)
    setMinivan = dropFrames(subData[subData['Category'].isin(['minivan'])])
    setSedan = dropFrames(subData[subData['Category'].isin(['sedan'])])
    setSUV = dropFrames(subData[subData['Category'].isin(['suv'])])
    global e1
    e1 = sortedEig[0][1]
    global e2
    e2 = sortedEig[1][1]

    plotPCA(setMinivan) #blue
    plotPCA(setSedan) #yellow
    plotPCA(setSUV) #green
    plt.show()

def plotPCA(dataset):
    vec = getVectors(dataset)
    plt.scatter(np.dot(vec, e1), np.dot(vec, e2))

def getVectors(subSet):
    vectors = []
    for i in range(len(subSet)) :
        vector = []
        for j in range(0, 11):
            vector.append(subSet.iloc[i, j])
        vectors.append(vector)
    return vectors

def dropFrames(sets):
    del sets['Model']
    del sets['Category']
    return sets

def normalize(cd, drop = True):
    result = cd.copy()
    if drop:
        del result['Model']
        del result['Category']
    for featureName in result:
        if featureName == 'Model' or featureName == 'Category':
            continue
        mean = result[featureName].mean()
        std = result[featureName].std()
        result[featureName] = (result[featureName] - mean) / std
    return result
if __name__ == '__main__':
    main()
```