CS 5342 Project 5: Tyler Martin and Joel Fecke

Imports

In [1]:
```python
import numpy as np
import math
import os
import re
import nltk
from nltk.corpus import stopwords
import pandas as pd

#global definition for stopwords
nltk.download('stopwords')
sw = stopwords.words('english')
pattern = re.compile(r'\b(' + r'|'.join(sw) + r')\b\s*')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/administrator/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Global Variables

In [2]:
```python
non_cs_lines = []
wordList = []
cs_lines = []
Xs_CS = []
Xs_Non = []
n = 4
```

Creating Word Lists For Case 1

In [3]:
```python
#=========================
#
# Case 1
#
#=========================
def Case1(csName, nonName):

    #call globals
    global non_cs_lines
    global wordList
    global cs_lines

    #make word list for cs file
    with open(csName, 'r') as file:
        for line in file.readlines():
            line = line.lower().split()
            line_words = []
            for word in line:
                word = re.sub(r'[^a-z]', '', word)
                if len(word) > n and word not in cs_lines:
                    cs_lines.append(word)
                if word not in wordList and len(word) > n:
                    wordList.append(word)

    #make word list for non-cs file
    with open(nonName, 'r') as file:
        for line in file.readlines():
            line = line.lower().split()
            line_words = []
            for word in line:
                word = re.sub(r'[^a-z]', '', word)
                if len(word) > n and word not in cs_lines:
                    non_cs_lines.append(word)
                if word not in wordList and len(word) > n:
                    wordList.append(word)
                    word = re.sub(r'[^a-z]', '', word)

#Make word lists for computer science and non computer science
Case1("trainData_cs", "trainData_non")
```

Creating Word Counts For Case 1

In [4]:
```python
#Getting word occurrences
for i, line in enumerate(cs_lines):
    wordCount_cs = {}
    wordCount_non = {}
    for word in wordList:
        if word in cs_lines[i]:
            wordCount_cs[word] = 1
        else:
            wordCount_cs[word] = 0

    Xs_CS.append(wordCount_cs)

for i, line in enumerate(non_cs_lines):
    wordCount_cs = {}
    wordCount_non = {}
    for word in wordList:
        if word in non_cs_lines[i]:
            wordCount_non[word] = 1
        else:
            wordCount_non[word] = 0

    Xs_Non.append(wordCount_non)

#Get P(X|CS) for each word
df_CS = pd.DataFrame(Xs_CS)
df_non = pd.DataFrame(Xs_Non)
CS_sums = dict(df_CS.sum())
Non_sums = dict(df_non.sum())

#Calculate P(CS)
total = len(cs_lines) + len(non_cs_lines)
p_CS = len(cs_lines)/total
```

Function For Making a Smoothing Variable

In [5]:
```python
def laplace_estimate(word, y):

    weight = 1

    # Number of training instances where
    # Xi = c and Y = y
    nc = 0

    # numnber of training instances
    if y == 1:
        n = len(cs_lines)
    else:
        n = len(non_cs_lines)

    # Number of dimensions (CS and Non)
    v = 2 * weight
    v = len(wordList)

    return (nc + weight)/(n+v)
```

Bayes for Case 1

In [6]:
```python
def Bayes1(testLine):
    p_X_CS = 1
    P_X = 1
    normalize = 0

    line = testLine.lower().split()
    for word in line:

        if word in cs_lines or word in non_cs_lines:

            word = re.sub(r'[^a-z]', '', word)

            if word in cs_lines:
                p_X_CS *= CS_sums[word]/len(cs_lines)
            else:
                p_X_CS *= laplace_estimate(word, 1)

            if word in wordList:
                normalize += 1
                if word in cs_lines and non_cs_lines:
                    num = (CS_sums[word] + Non_sums[word])
                    denom = (len(cs_lines) + len(non_cs_lines))
                    P_Y = num/denom
                elif word in non_cs_lines:
                    num = Non_sums[word]
                    denom = (len(cs_lines) + len(non_cs_lines))
                    P_Y = num/denom
                else:
                    num = CS_sums[word]
                    denom = (len(cs_lines) + len(non_cs_lines))
                    P_Y = num/denom

                P_X *= P_Y

            else:
                P_X *= 1

    norm_value = normalize

    P = (p_X_CS*(p_CS)**norm_value)/(P_X)


    return P
```

Creating Word Lists & Counts For Case 2

In [7]:
```python
#=========================
#
# Case 2
#
#=========================
def Case2(name, type):

    #dict to store words from numeric run
    classification = dict()

    #signal whether to use file or string
    if (type == 1):
        #get all words from classifiers file
        with open(name, 'r') as file:
            for line in file.readlines():

                #remove stopwords
                line = pattern.sub('', line)

                words = line.lower().split()
                for word in words:
                    word = re.sub(r'[^a-z]', '', word)
                    if word in classification:
                        classification[word] += 1
                    else:
                        classification[word] = 1

    elif (type == 2):
        #get all words from input string
        for word in name.split():
            word = re.sub(r'[^a-z]', '', word)
            if word in classification:
                classification[word] += 1
            else:
                classification[word] = 1

    #return dict
    return classification

#getting results for computer science CASE 1 and CASE 2
cs_case_Two = Case2("trainData_cs", 1)

#getting results for normal CASE 1 and CASE 2
non_case_Two = Case2("trainData_non", 1)
```

Determining distributions for word counts

In [8]:
```python
#creating a distribution from our dictionary
def create_distribution(dict):

    #get sum of keys of input
    total = sum(dict.values())

    #weighting
    for key, value in dict.items():
        dict[key] = value/total

    return dict

#make distributions for numeric
cs_case_Two_dist = create_distribution(cs_case_Two)
non_case_Two_dist = create_distribution(non_case_Two)
```

Bayes for Case 2

In [9]:
```python
#data sets given labels
numeric_dist = {'Computer Science': non_case_Two_dist,
                'Non-Computer Science': cs_case_Two_dist}

#function to do actual bayes calcs
def Bayes2(input, dist):

    attr_dist = {cName: cProb for cName, cProb in dist.items()}

    #dictionary to hold predictions
    predictions = dict()

    #calculate likelyhood that the
    #input sentence is CS or non-CS
    for t in dist.keys():

        #use array to pass values for calculation
        #without raising dict errors
        valpass =[]
        attr = attr_dist[t]
        results = 1

        #avoid keyerrors
        for i in input:
            if i in attr.keys():
                valpass.append(attr[i])
            else:
                valpass.append(1)

        #calc products
        for x in valpass:
            results *= x

        #add to final dict
        predictions.update({t: results})

    #weight results and return
    return create_distribution(predictions)
```

Running Both Bayes and Classifying Sentences on Test Data

In [10]:
```python
#assigning strings to short names so pdf conversion works
cs = "Computer Science"
ncs = "Non-Computer Science"

#call our functions to make predictions from testData file
file = open("testData")
testData = file.readlines()

#data structures to track correct guesses
binGuesses = []
numGuesses = []
realResults = [cs, cs, ncs, ncs, cs, cs, ncs, ncs]

#results and printing
for data in testData:

    #remove stopwords
    data = pattern.sub('', data)

    #calc
    binaryResults = Bayes1(data)
    binaryResults = dict({cs: binaryResults, ncs: 1-binaryResults})
    numericResults = create_distribution(Case2(data.lower(), 2))
    numericResults = Bayes2(numericResults, numeric_dist)

    #storing results in short names so the ipynb to pdf
    #conversion works.
    ncsbr = binaryResults.get(ncs)
    csbr = binaryResults.get(cs)
    ncsn = numericResults.get(ncs)
    csn = numericResults.get(cs)

    #add result to guesses
    if csbr > ncsbr:
        binGuesses.append(cs)
    else:
        binGuesses.append(ncs)

    if csn > ncsn:
        numGuesses.append(cs)
    else:
        numGuesses.append(ncs)

    #print results
    print('=' * 86)
    print("Input String: ", data)
    print("Binary: ", binaryResults)
    print("Numeric: ", numericResults)
    print("\nDetermination (Bin): ", binGuesses[-1])
    print("Determination (Num): ", numGuesses[-1])
    print('=' * 86)

#printing overall Accuracy
correct = 0
total = 0
for i in range(len(realResults)):
    if (numGuesses[i] == binGuesses[i] == realResults[i]):
        correct += 1
    total += 1

totalAcc = math.floor((correct/total)*100)
print("Overall Naiive Bayes Implementation Accuracy: ", totalAcc, "%")
```

```
======================================================================================
Input String:  It unrealistic expect data perfect. There may problems due human error,
limitations measuring devices, flaws data collection process.

Binary:  {'Computer Science': 0.36212624584717606, 'Non-Computer Science': 0.637873754
1528239}
Numeric:  {'Computer Science': 0.9999973977604031, 'Non-Computer Science': 2.602239596
965131e-06}

Determination (Bin):  Non-Computer Science
Determination (Num):  Computer Science
======================================================================================
======================================================================================
Input String:  Even data present looks fine, may inconsistencies—person height 2 meter
s, weighs 2 kilograms

Binary:  {'Computer Science': 1.0, 'Non-Computer Science': 0.0}
Numeric:  {'Computer Science': 0.9832828707063822, 'Non-Computer Science': 0.016717129
29361779}

Determination (Bin):  Computer Science
Determination (Num):  Computer Science
======================================================================================
======================================================================================
Input String:  An anxiety-provoking aspect parenting regards choices prepare children
future world vaguely imagine curriculum parents sometimes understand

Binary:  {'Computer Science': 0.0008598248918017464, 'Non-Computer Science': 0.9991401
751081983}
Numeric:  {'Computer Science': 1.0099856342558805e-05, 'Non-Computer Science': 0.99998
99001436575}

Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science
======================================================================================
======================================================================================
Input String:  Elementary school STEM immersive education science design. Learners enc
ouraged practically engage develop behaviors go numerate, critical collaborative probl
em solvers

Binary:  {'Computer Science': 0.36212624584717606, 'Non-Computer Science': 0.637873754
1528239}
Numeric:  {'Computer Science': 0.00025503698036215245, 'Non-Computer Science': 0.99974
49630196378}

Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science
======================================================================================
======================================================================================
Input String:  After counting supports, algorithm eliminates candidate itemsets whose
support counts less

Binary:  {'Computer Science': 1.0000000000000002, 'Non-Computer Science': -2.220446049
250313e-16}
Numeric:  {'Computer Science': 0.9948717948717949, 'Non-Computer Science': 0.005128205
128205128}

Determination (Bin):  Computer Science
Determination (Num):  Computer Science
======================================================================================
======================================================================================
Input String:  In Apriori algorithm, candidate itemsets partitioned different buckets
stored hash tree

Binary:  {'Computer Science': 1.0000000000000002, 'Non-Computer Science': -2.220446049
250313e-16}
Numeric:  {'Computer Science': 0.9999734304009353, 'Non-Computer Science': 2.656959906
4750113e-05}

Determination (Bin):  Computer Science
Determination (Num):  Computer Science
======================================================================================
======================================================================================
Input String:  Really, would completely necessary doubt genuineness reply. It might al
so useful point agreed (sticking example) perhaps unusual, controversial, speaker anti
cipates surprise agreement.

Binary:  {'Computer Science': 0.12070874861572535, 'Non-Computer Science': 0.879291251
3842747}
Numeric:  {'Computer Science': 0.009800949782762452, 'Non-Computer Science': 0.9901990
502172375}

Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science
======================================================================================
======================================================================================
Input String:  This 'actually referring back "number", thing person.
Binary:  {'Computer Science': 0.36212624584717606, 'Non-Computer Science': 0.637873754
1528239}
Numeric:  {'Computer Science': 1.717679991583967e-11, 'Non-Computer Science': 0.999999
9999828232}

Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science
======================================================================================
Overall Naiive Bayes Implementation Accuracy:  87 %
```

Running Both Bayes on Initial Training Data

In [11]:
```python
#assigning strings to short names so pdf conversion works
cs = "Computer Science"
ncs = "Non-Computer Science"

for i in range(2):
    #call our functions to make predictions from training files
    if (i == 0):
        print("Classifying CS Training Data")
        print('=' * 86)
        file = open("trainData_cs")
        testData = file.readlines()

    else:
        print("Classifying Non-CS Training Data")
        print('=' * 86)
        file = open("trainData_non")
        testData = file.readlines()


    #data structures to track correct guesses
    binGuesses = []
    numGuesses = []

    #results and printing
    for data in testData:

        #remove stopwords
        data = pattern.sub('', data)

        #calc
        binaryResults = Bayes1(data)
        binaryResults = dict({cs: binaryResults, ncs: 1-binaryResults})
        numericResults = create_distribution(Case2(data.lower(), 2))
        numericResults = Bayes2(numericResults, numeric_dist)

        #storing results in short names so the ipynb to pdf
        #conversion works.
        ncsbr = binaryResults.get(ncs)
        csbr = binaryResults.get(cs)
        ncsn = numericResults.get(ncs)
        csn = numericResults.get(cs)

        #add result to guesses
        if csbr > ncsbr:
            binGuesses.append(cs)
        else:
            binGuesses.append(ncs)

        if csn > ncsn:
            numGuesses.append(cs)
        else:
            numGuesses.append(ncs)

        #print results
        print("Binary: ", binaryResults)
        print("Numeric: ", numericResults)
        print("Determination (Bin): ", binGuesses[-1])
        print("Determination (Num): ", numGuesses[-1], "\n")
```

```
Classifying CS Training Data
======================================================================================
Binary:  {'Computer Science': 0.6666666666666667, 'Non-Computer Science': 0.3333333333
3333326}
Numeric:  {'Computer Science': 0.9982953680514002, 'Non-Computer Science': 0.001704631
9485998327}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.0000000000000009, 'Non-Computer Science': -8.881784197
001252e-16}
Numeric:  {'Computer Science': 1.0, 'Non-Computer Science': 5.7829718791826386e-43}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.0000000000000004, 'Non-Computer Science': -4.440892098
500626e-16}
Numeric:  {'Computer Science': 1.0, 'Non-Computer Science': 3.905062765067777e-19}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.0000000000000007, 'Non-Computer Science': -6.661338147
750939e-16}
Numeric:  {'Computer Science': 1.0, 'Non-Computer Science': 1.384456909698153e-29}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.000000000000001, 'Non-Computer Science': -1.1102230246
251565e-15}
Numeric:  {'Computer Science': 1.0, 'Non-Computer Science': 3.516081040503245e-31}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.000000000000001, 'Non-Computer Science': -1.1102230246
251565e-15}
Numeric:  {'Computer Science': 1.0, 'Non-Computer Science': 1.5081736813034335e-39}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.0000000000000004, 'Non-Computer Science': -4.440892098
500626e-16}
Numeric:  {'Computer Science': 0.9999999999999999, 'Non-Computer Science': 1.640126361
3284663e-16}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.0000000000000009, 'Non-Computer Science': -8.881784197
001252e-16}
Numeric:  {'Computer Science': 1.0, 'Non-Computer Science': 1.903033552849694e-31}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.0000000000000007, 'Non-Computer Science': -6.661338147
750939e-16}
Numeric:  {'Computer Science': 1.0, 'Non-Computer Science': 9.504545484215315e-40}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.0000000000000009, 'Non-Computer Science': -8.881784197
001252e-16}
Numeric:  {'Computer Science': 1.0, 'Non-Computer Science': 9.02444392017644e-24}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.000000000000001, 'Non-Computer Science': -1.1102230246
251565e-15}
Numeric:  {'Computer Science': 1.0, 'Non-Computer Science': 3.584944715970697e-35}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Binary:  {'Computer Science': 1.0000000000000009, 'Non-Computer Science': -8.881784197
001252e-16}
Numeric:  {'Computer Science': 1.0, 'Non-Computer Science': 7.416017188137394e-21}
Determination (Bin):  Computer Science
Determination (Num):  Computer Science

Classifying Non-CS Training Data
======================================================================================
Binary:  {'Computer Science': 4.9286589637458975e-06, 'Non-Computer Science': 0.999995
0713410363}
Numeric:  {'Computer Science': 3.491718517555136e-26, 'Non-Computer Science': 1.0}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 1.7553755253156718e-06, 'Non-Computer Science': 0.999998
2446244747}
Numeric:  {'Computer Science': 4.590042117819418e-41, 'Non-Computer Science': 1.0}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 0.00040831000406078404, 'Non-Computer Science': 0.999591
6899959392}
Numeric:  {'Computer Science': 4.114391353290184e-31, 'Non-Computer Science': 1.0}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 0.0010378838671804027, 'Non-Computer Science': 0.9989621
161328196}
Numeric:  {'Computer Science': 2.4200129123525434e-19, 'Non-Computer Science': 1.0}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 0.0005732165945344975, 'Non-Computer Science': 0.9994267
834054655}
Numeric:  {'Computer Science': 5.15144127056684e-17, 'Non-Computer Science': 1.0}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 0.36212624584717606, 'Non-Computer Science': 0.637873754
1528239}
Numeric:  {'Computer Science': 3.6443135407020614e-07, 'Non-Computer Science': 0.99999
96355686458}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 0.000136103334686928, 'Non-Computer Science': 0.99986389
66653131}
Numeric:  {'Computer Science': 3.7175349375224613e-19, 'Non-Computer Science': 1.0}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 0.002866082972672488, 'Non-Computer Science': 0.99713391
70273276}
Numeric:  {'Computer Science': 2.414481454267166e-18, 'Non-Computer Science': 1.0}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 0.0008166200081215681, 'Non-Computer Science': 0.9991833
799918785}
Numeric:  {'Computer Science': 8.999274529781274e-31, 'Non-Computer Science': 1.0}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 0.0062273032030824155, 'Non-Computer Science': 0.9937726
967969176}
Numeric:  {'Computer Science': 9.48645061642196e-16, 'Non-Computer Science': 0.9999999
999999991}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 0.18106312292358812, 'Non-Computer Science': 0.818936877
0764118}
Numeric:  {'Computer Science': 1.4135554846415428e-12, 'Non-Computer Science': 0.99999
99999985865}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science

Binary:  {'Computer Science': 0.04371180597712314, 'Non-Computer Science': 0.956288194
0228769}
Numeric:  {'Computer Science': 1.126758011991344e-16, 'Non-Computer Science': 0.999999
9999999998}
Determination (Bin):  Non-Computer Science
Determination (Num):  Non-Computer Science
```