John Soderstrom
CS 4342
Project 1

Over 200 statements were selected. No more than 3 came from a single chapter, or from the full source in case of smaller sources. Before selecting any statements, I decided to start from the third sentence of the first paragraph, the second sentence of the second paragraph, and the first sentence of the third paragraph where possible. I ended each selection at 15 words or the end of the paragraph, whichever came first. This was not always possible, particularly when selecting data from non-CS sources.

99 statements were selected from CS sources: an AI textbook and a compilers textbook. More diverse statements could have been selected from CS material covering different topics. Statements from other CS material may also not be classified correctly as often.

Statements from non-CS sources were obtained from a wider variety of material, including random Wikipedia pages, news articles, a textbook, and novels. I used a greater number of statements from this classification to try to handle the wider spread of keywords that may or may not exist compared to the CS sources.

My original set of keywords was insufficient to test the data - most of them barely appeared in the training set after I finished preparing the data. In order to improve the keywords, I tallied every word in the training data without seeing their classifications. I believe this let me obtain more meaningful keywords without adding bias, but it may still be a mistake. Even if it added bias, I believe it limited the effect of it.

Begin Source Code

```
from pprint import pprint as pp

#Returns a list of probabilities for each keyword given a numeric list
#The first interior list contains probabilities of each keyword when the
#label is 1 (or True/CS) and the probability of the label being 1.
#The second interior list contains probabilities of each keyword when the
#label is 0 (or False/Not CS) and the probability of the label being 0.
def getTrainingProbability(trainingResults):
    #Create a list of probabilities to determine expected test results
    probList = [[], []]
    for i in range(len(trainingResults[0]) - 1):
        probList[0].append(0)
        probList[1].append(0)

    #Counts the number of 1 and 0 classifications from passed data
    #Used to calculate probability of both values across all data.
    countYes = 0
    countNo = 0
    for index, line in enumerate(trainingResults):
        probLine = 0
        if line[-1][0] is 1:
            countYes += 1
        else:
            countNo += 1
            probLine = 1
        for inIndex, element in enumerate(line[:-1]):
            probList[probLine][inIndex] += element

    #Calculates probability that each keyword exists in each classification
    #Probabilities are separated based on 1 and 0 classification
    for index, line in enumerate(probList):
        for inIndex, element in enumerate(line):
            #Check a need for smoothing
            if probList[0][inIndex] is 0 or probList[1][inIndex] is 0:
                if index is 0:
                    line[inIndex] = (element + 1) / (countYes + 2)
                else:
                    line[inIndex] = (element + 1) / (countNo + 2)
            #Don't apply smoothing if both entries have something already
            else:
                if index is 0:
                    line[inIndex] /= countYes
                else:
                    line[inIndex] /= countNo

    #Adds the final probability that a randomly selected line will
    #be classified as either 1 or 0
    probList[0].append([countYes / (countYes + countNo)])
    probList[1].append([countNo / (countYes + countNo)])
    return probList

#Calculates required parts of probability that a given line
#is either 0 or 1 (CS or not CS in this project, but usable outside of it)
#Appends the expected classification to the list containing the
#actual classification of the line.
def getSingleTestResult(probList, testLine):
    #Final probability will always include P(y=1) or P(y=0), so start with them
    probYes = probList[0][-1][0]
    probNo = probList[1][-1][0]
    #Iterates through list, multiplying probabilities by given y=1 and y=0
    #respectively. Index is aligned with keyword values.
    for index, element in enumerate(testLine):
        if element is 1:
            probYes *= probList[0][index]
            probNo *= probList[1][index]
        elif element is 0:
            probYes *= 1 - probList[0][index]
            probNo *= 1 - probList[1][index]

    #Compares final probabilities of both classifications. While not
    #true probabilities, the ratios are the same and they may be
    #compared safely.
    if probYes > probNo:
        testLine[-1].append(1)
    else:
        testLine[-1].append(0)

#Passed both the training and test data indicating presence of
#keywords in each line, expecting the final values of both
#to be a list containing only the actual classification
#Will modify the test data to append both the predicted classification
#and the error value to the list containing actual classification
def getTestResults(training, test):
    probList = getTrainingProbability(training)
    count = 0
    #Runs each line through getting expected classification and
    #appends the error value
    for line in test:
        getSingleTestResult(probList, line)
        result = 0
        if line[-1][0] is not line[-1][1]:
            result = 1
        line[-1].append(result)

#Cleans a string for learning, removing non-alphanumeric characters
#and changing case to lower.
def cleanString(string):
    string = string.lower()
    firstIndex = 0
    secondIndex = 0
    newString = ""
    for char in string:
        if not (char.isalnum() or ord(char) is 32):
            newString += string[firstIndex:secondIndex]
            firstIndex = secondIndex + 1
        secondIndex += 1
    return newString

#Use for deciding keywords, modifies a dictionary to contain each
#word in the training data with a count, ignoring classification
```

```
def potentialKeywords(keywordDict, line):
    tempLine = line.split(' ')
    for word in tempLine:
        if word in keywordDict:
            keywordDict[word] += 1
        else:
            keywordDict[word] = 1

#Includes two lines to uncomment to generate word counts for potential
#keywords.
#Stores the presence of keywords across all lines in training data
#with labels. Returns a list aligned with the keywords list passed in.
def storeResults(fileName, keywords, warningLabel):
    try:
        with open(fileName) as inFile:
            #Uncomment to store potential keywords
##          potentialKey = {}
            results = []
            #Splits input lines one by one and scans for presence of keywords
            #Appends class label after scanning of a line is complete
            for line in inFile:
                #I was getting strange behavior and could not get it
                #to stop without adding this if statement
                if line[-1] is not '\n':
                    line += '\n'
                line = cleanString(line)
                checkLine = line.rstrip('\n').rsplit(' ', 1)
                #Uncomment to add potential keywords to dictionary
##              potentialKeywords(potentialKey, checkLine[0])
                results.append(testLineKeywords(checkLine[0], keywords))
                results[-1].append([int(checkLine[1])])
            #Uncomment to print potential keyword dictionary
##          pp([(k, v) for k, v in sorted(potentialKey.items(),
##                                key=lambda x: x[1], reverse=True)])
            return results
    except IOError:
        print("Unable to open " + fileName + ".")

#Test a line of text for presence of keywords in given list
#Returns a list of 0/1 describing their presence
def testLineKeywords(line, keywords):
    iterList = []
    tempLine = line.split(' ')
    for keyword in keywords:
        #Testing for presence, not number of times keyword exists
        if keyword in tempLine:
            iterList.append(1)
        else:
            iterList.append(0)
    return iterList

#Stores a list of keywords based on an input file
#Expects each keyword on its own line
#Returns a list of keywords
def storeKeywords(fileName, warningLabel):
    try:
        keywords = []
        with open(fileName) as inFile:
            for line in inFile:
                keywords.append(line.rstrip('\n'))
            if not keywords:
                print("Warning: " + warningLabel + " is empty.")
        return keywords
    except IOError:
        print("Unable to open " + fileName + ".")

#Gets error rate on test data through error results added to test data
def getErrorRate(test):
    #Count the number of lines in test data and the number of found errors
    errorCount = 0
    totalCount = 0
    for line in test:
        totalCount += 1
        errorCount += line[-1][-1]

    #Returns the error rate
    return errorCount / totalCount

#Print statement numbers with actual and expected classification
def printResults(testResults, label1, label0):
    count = 1
    for line in testResults:
        if line[-1][0] is 1:
            actLabel = label1
        else:
            actLabel = label0
        if line[-1][1] is 1:
            expLabel = label1
        else:
            expLabel = label0
        print("Statement #{}".format(count))
        print("Expected Classification: {}".format(expLabel))
        print("Actual Classification: {}".format(actLabel))
        if actLabel is expLabel:
            print("No error\n")
        else:
            print("Error\n")
        count += 1

#Set initial filenames
trainingFileName = "trainingData.txt"
testFileName = "testData.txt"
keywordFileName = "keywords.txt"

#Store data for testing, keywords and training data
keywords = storeKeywords(keywordFileName, "Keyword Data")
trainingResults = storeResults(trainingFileName, keywords, "Training Data")

#Store test data related to keywords and get results
testResults = storeResults(testFileName, keywords, "Test Data")
getTestResults(trainingResults, testResults)
printResults(testResults, "CS material", "Non-CS material")

#Run training data as test and get the error rate
altTrainingResults = storeResults(trainingFileName, keywords, "Training Data")
getTestResults(trainingResults, altTrainingResults)
errorRate = getErrorRate(altTrainingResults)
print(errorRate)
```

End Source Code

Begin Training Data

Partial CS-sourced statements, classified by '1' at the end
They make strong claims about how the intelligence of humans is achieved - not by purely 1
For example, the transition model for the 8-puzzle - knowledge of what the actions do - is 1
Chapter 6 introduced the idea of representing states as assignments of values to variables; this 1
Unfortunately, propositional logic is too puny a language to represent knowledge of complex environments in 1
Our discussion motivates the development of first-order logic, a much more expressive language than the 1
Programming languages (such as C++ or Java or Lisp) are by far the largest class 1
Section 9.1 introduces inference rules for quantifiers and shows how to reduce first-order inference to 1
We begin with some simple inference rules that can be applied to sentences with quantifiers 1
Let us begin with universal quantifiers. Suppose our knowledge base contains the standard folkloric axiom 1
In this chapter we introduce a representation for planning problems that scale up to problems 1
Section 10.2 shows how forward and backward search algorithms can take advantage of this representation, 1

Partial non-CS-sourced statements, classified by '0' at the end
Eventually, other schools in the area began to request performances. These small cafeteria performances grew 0
After a few months after its launch, the song reached number one on Colombia's national 0
Soon after, the group recorded a new single entitled "Bien Cerquita", which features Dominican artist 0
For a more conventional President, that might mean putting the spectacle of impeachment behind him 0
And he has long believed the federal government — and the outer orbit of the White 0
Trump has launched a season of getting even, lashing out at those who spoke out 0
The jump in new cases—more than 14,800 in the province at the epicenter of 0
Confirmed cases will now be based on clinical diagnosis, including the use of CT scans. 0
“In terms of monitoring the progression of the outbreak of infections, it’s quite a sensible 0
On February 7, a crucial witness in the House Impeachment Inquiry last fall, Lt. Col. 0
From 1905 to 1925, Einstein not only conceived the theory of relativity, but he made 0
Rejected by academia, Einstein went his own way and took a job at the Swiss 0
Professorships at leading universities soon came his way, and as his fame grew so did 0
For some, there are obligations of social class to satisfy, family expectations to consider, or 0
Given a number of choices determined by all the other factors - national origin, social background 0
After all, what do we mean when we say that a person is, or is 0
Even more than the husband-wife relationship, the parent-child relationship has this serious factor of interpersonal 0

End Training Data

Begin Testing Data

Non-CS-sourced statements, classified by '0' at the end
Lawmakers are grappling over several approaches to curtail the practice that can leave patients on 0
She said she does not suffer from asthma or have a history of respiratory issues. 0
Shipwrecks along the southern Lake Superior coast known as the "Graveyard of the Great Lakes" 0

CS-sourced statements, classified by '1' at the end
Some of the material in this chapter is fairly mathematical, although the general lessons can 1
Assignment of program variables to registers across basic blocks is an example of a rather 1

Test data output
Statement #1: Lawmakers are grappling over several approaches to curtail the practice that can leave patients on
Expected Classification: Non-CS material
Actual Classification: Non-CS material
No error

Statement #2: She said she does not suffer from asthma or have a history of respiratory issues.
Expected Classification: Non-CS material
Actual Classification: Non-CS material
No error

Statement #3: Shipwrecks along the southern Lake Superior coast known as the "Graveyard of the Great Lakes"
Expected Classification: Non-CS material
Actual Classification: Non-CS material
No error

Statement #4: Some of the material in this chapter is fairly mathematical, although the general lessons can
Expected Classification: CS material
Actual Classification: CS material
No error

Statement #5: Assignment of program variables to registers across basic blocks is an example of a rather
Expected Classification: CS material
Actual Classification: CS material
No error

Training data error rate
0.11818181818181818

End Testing Data