NAME: Ayush Srivastava, Karim Sabar, Dhruv Sandasera

UTEID: as79973, kas5852, djs3967

Problem 1

Read Shannon's 1948 paper 'A Mathematical Theory of Communication'. Focus on pages 1-19 (up to Part II), the remaining part is more relevant for communication.
http://math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf
(http://math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf) Summarize what you learned briefly (e.g. half a page).

The paper talks about various communication systems and provides us with mathematical models on how take into account noise in the channel and interpret he original message from the final destination. It starts off with an introduction to the essential parts of a communication system: information source, transmitter, channel, receiver and destination. Part I mainly focuses on discrete noiseless systems such as teletype and telegraphy where initially we learn about the symbols used to transmit information and how to calculate the capacity of a channel. The channel capacity C is equal to log_W where W is the largest real root of the determinant equation:

|$\sum W_{ij}^{b}$ - φ_{ij} = 0

We learn about stochastic processes and try many types using different criteria for transition probabilities and different orders of approximation (n-gram structures). This leads to a discussion about how Markoff and in particular the 'ergodic' processes represent a discrete source of information. Next we learn about the entropy (H) of a set of probabilities and the assumptions it must satisfy:

H = $-K\sum_{i=1}^{n} p_i logp_i$

We consider a finite discrete source and the definition for entropy of a source as the weighted average of the individual entropies and find that H is thus approximately the logarithm of the reciprocal probability of a typical long sequence divided by the number of symbols in the sequence. We also get an understanding of what relative entropy and redundancy mean through real life examples.

While understanding the operations performed by the transmitter and the receiver (transducers) we learn about non-singular and inverse transducers and the two functions they are described by. We then see how H can be interpreted as the rate of generating information by proving that H determined the channel capacity with the most efficient coding. Considering real life examples we learn how the entropy of a source determines the channel capacity which is necessary and sufficient.

In [45]:
```
import requests
from bs4 import BeautifulSoup
import shutil
import PyPDF2
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import numpy as np
from io import StringIO
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
import os
import sys, getopt
from collections import Counter
```

Problem 2: Scraping, Entropy and ICML papers.

ICML is a top research conference in Machine learning. Scrape all the pdfs of all ICML 2018 papers from http://proceedings.mlr.press/v80/ (http://proceedings.mlr.press/v80/).

In [60]:
```
# specify the URL of the archive here
archive_url = "http://proceedings.mlr.press/v80/"

def get_pdf_links():

    # create response object
    r = requests.get(archive_url)

    # create beautiful-soup object
    soup = BeautifulSoup(r.content)

    # find all links on web-page
    links = soup.findAll('a',text="Download PDF")

#     print(links)

    # filter the link sending with .mp4
    pdf_links = [ link['href'] for link in links if link['href'].endswith('pd
f')]

    return pdf_links


def download_pdf(pdf_links):

    for link in pdf_links:

        '''iterate through all links in video_links
        and download them one by one'''

        # obtain filename by splitting url and getting
        # last string
        file_name = link.split('/')[-1]

        # create response object
        r = requests.get(link, stream = True)


        with open("C:\\Users\\Karim Sabar\\Desktop\\PDFs\\"+file_name,"wb") as
pdf:
            shutil.copyfileobj(r.raw,pdf)
#             for chunk in r.iter_content(chunk_size=1024):
#                 if chunk:
#                     pdf.write(chunk)



    print("All PDFs downloaded!")
    return

if __name__ == "__main__":

    pdf_links = get_pdf_links()

    download_pdf(pdf_links)
```

```
All PDFs downloaded!
```

1. What are the top 10 common words in the ICML papers?

In [61]:
```
def convert( pages=None):
    if not pages:
        pagenums = set()
    else:
        pagenums = set(pages)

    text=""
    fileList= os.listdir('C:\\Users\\Karim Sabar\\Desktop\\PDFs2\\')
    fileList.sort()
    print("Starting Convert")

    for filename in fileList:
        if filename.endswith('.pdf'):
            output = StringIO()
            manager = PDFResourceManager()
            converter = TextConverter(manager, output, laparams=LAParams())
            interpreter = PDFPageInterpreter(manager, converter)

#             print("Starting: "+filename)
            infile = open("C:\\Users\\Karim Sabar\\Desktop\\PDFs2\\"+filename,
'rb')
            for page in PDFPage.get_pages(infile, pagenums):
                interpreter.process_page(page)
            infile.close()
            converter.close()
            text2 = output.getvalue()
#             with open("./TXTs2/"+filename+".txt","wb") as txt:
#                 txt.write(text2.encode())

            text=text+" "+text2
            output.close

    return text
```

In [62]:
```
#Used only used between 100-150 PDFs because it was taking wayyy too long
if __name__ == "__main__":
    tokens=[""]
    text=convert()
    print("Done Converting")
    token = word_tokenize(text)
    for word in token:
        tokens.append(word)
    punctuations = ['(',')',';',':','[',']',',','.','=','The']

    stop_words = stopwords.words('english')

    keywords = [word for word in tokens if not word in stop_words and not word
in punctuations and len(word)>2 and word.isalpha()]

    print("Starting to count")
    counter = Counter(keywords)

    most_occur = counter.most_common(10)
    print(most_occur)
```

```
Starting Convert
Done Converting
Starting to count
[('model', 2573), ('learning', 2434), ('data', 1957), ('algorithm', 1954),
('function', 1825), ('set', 1578), ('using', 1575), ('This', 1571), ('distrib
ution', 1562), ('Learning', 1468)]
```

1. Let Z be a randomly selected word in a randomly selected ICML paper. Estimate the entropy of Z.

In [63]:
```
    word_probs = []
    unique_words = []
    entrophy = 0

    for word in counter:
        unique_words.append(word)
        entrophy+= -1*(counter[word]/len(keywords))*(np.log2(counter[word]/len
(keywords)))
        word_probs.append(counter[word]/len(keywords))

    print(entrophy)
```

```
12.042583709771046
```

1. Synthesize a random paragraph using the marginal distribution over words.

In [64]:
```
    for i in range (80):
        print(np.random.choice(unique_words, p=word_probs), end = " ")
```

```
loss ones sparsity uniform RBMn test dots given However Similar resort applie
s extended mal Figure Method symmetric General tuning Unbiased outlined under
lying domain designed strategy Mooij probability test measure data locally co
mplexity policies preprint occur code thresholding models use Predict policy
considered MLE prior sake mild RNNs studied satisfy distribution huge convolu
tional Further adversarially per algorithm want matching goal Rademacher obje
ctive prediction next order provide capturing achieve consider Kalisch simple
Boston sufficient taxonomy starts centroids Andrew von inequality reliable rew
ard
```

In [ ]:

In [ ]: