COGSCI131–Spring 2019 — Homework 10Solutions

Yuchen Zhou, SID 3034489901

1.

By definition, the entropy over words in English is the expectation of surprisal of all words in dictionary.
The meaning of entropy over words in English is the average bits that we have to use to determine the exact word we are referring to.

2.

- Compute the entropy from the given data set.
- The entropy over this data set is 8.93. So use 9 bits (ask 9 questions) we can always determine which word we are referring to.
- In other words, 20 questions game is not a fair game because the answerer will always win if he plays optimally.

```
import pandas as pd
import numpy as np

def H(p):
    s=0
    for i in p:
        s+=-i*np.log2(i)
    return s

data=pd.read_csv("Assignment10-WordFrequencies.csv",header=None)
shape=int(data.shape[0])
dic=[]
frec=[]
prob=[]
sum=0
for i in range(shape):
    dic.append(str(data.loc[i][0]))
    frec.append(float(data.loc[i][1]))
    sum+=frec[i]
for i in range(shape):
    prob.append(frec[i]/sum)

print(H(prob))
```

3.

- Making plot of 26 bars as follows:

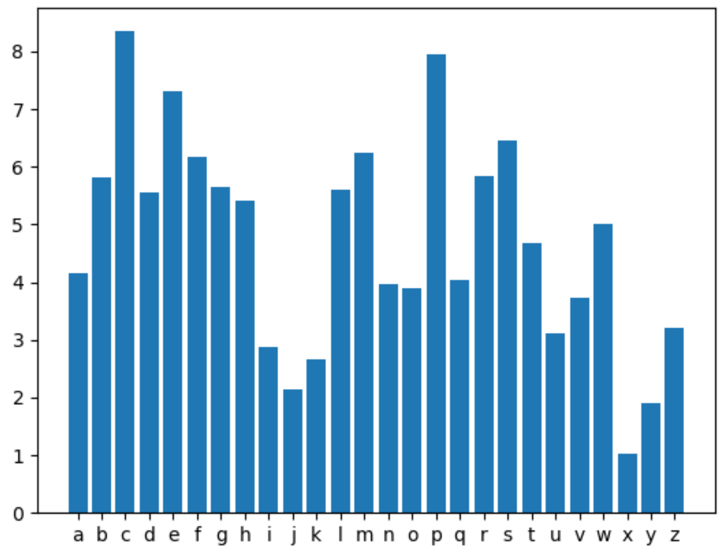

Figure 1: conditional entropy conditioning on the first character

- We can find that words starting by "c", "e" and "p" need relatively more bits to determine, in other words, these starting characters carry more information than other characters.

- Code:

```
import pandas as pd
import csv
import numpy as np
import matplotlib.pyplot as plt

def H(p):
    s=0
    for i in p:
        s+=-i*np.log2(i)
    return s

data=pd.read_csv("Assignment10-WordFrequencies.csv",header=None)
shape=int(data.shape[0])
dic=[]
frec=[]
prob=[]
sum=0
for i in range(shape):
    dic.append(str(data.loc[i][0]))
    frec.append(float(data.loc[i][1]))
    sum+=frec[i]
for i in range(shape):
    prob.append(frec[i]/sum)
```

```
condh=[]
letter=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r
                                        ','s','t','u','v','w','x','y','z']
for i in letter:
    sump=0
    newp=[]
    for j in range(shape):
        if dic[j][0] == i:
            sump+=prob[j]
    for j in range(shape):
        if dic[j][0] == i:
            newp.append(prob[j]/sump)
    condh.append(H(newp))

fig=plt.figure()
ax=fig.add_subplot(111)
ax.bar(letter,condh)
plt.show()
```

4.

- I think the most important one is the first character of a word, because when we loop up in the dictionary, we generally find the word by its first character.
- In this question we are asked to plot the mutual information. Plot three bars representing three conditions respectively.

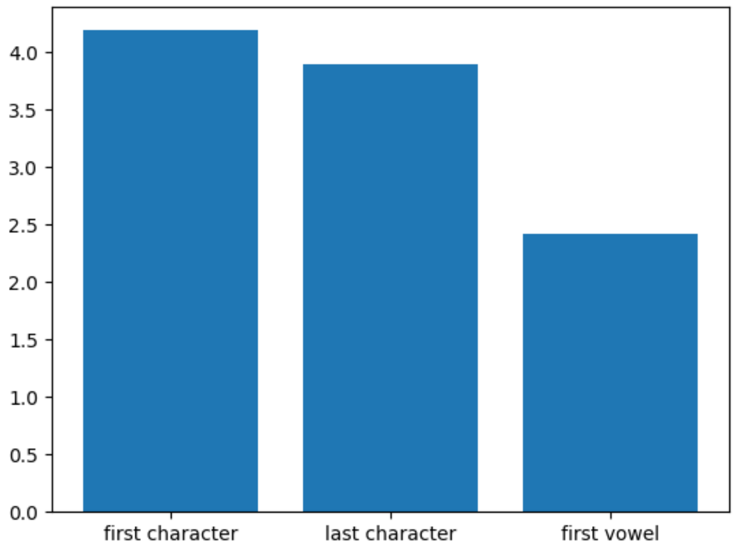

Figure 2: mutual information conveyed under different conditions

- From the plot we know that our prediction is reasonable because the mutual information conditioning on the first character do convey more information.
- Code as follows, in case 3 remember to compute those words with no vowel in it.

```
import pandas as pd
import csv
import numpy as np
import matplotlib.pyplot as plt

def H(p):
    s=0
    for i in p:
        s+=-i*np.log2(i)
    return s


data=pd.read_csv("Assignment10-WordFrequencies.csv",header=None)
shape=int(data.shape[0])
#shape=1000
dic=[]
frec=[]
prob=[]
```

```
sum=0
for i in range(shape):
    dic.append(str(data.loc[i][0]))
    frec.append(float(data.loc[i][1]))
    sum+=frec[i]
for i in range(shape):
    prob.append(frec[i]/sum)

info=[0]*3

pp=[]
condh=[]
letter=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r
    ','s','t','u','v','w','x','y','z']
for i in letter:
    sump=0
    newp=[]
    for j in range(shape):
        if dic[j][0] == i:
            sump+=prob[j]
    for j in range(shape):
        if dic[j][0] == i:
            newp.append(prob[j]/sump)
    condh.append(H(newp))
    pp.append(sump)
info[0]=H(prob)
for i in range(len(condh)):
    info[0]-=pp[i]*condh[i]

pp=[]
condh=[]
for i in letter:
    sump=0
    newp=[]
    for j in range(shape):
        if dic[j][-1] == i:
            sump+=prob[j]
    for j in range(shape):
        if dic[j][-1] == i:
            newp.append(prob[j]/sump)
    condh.append(H(newp))
    pp.append(sump)
info[1]=H(prob)
for i in range(len(condh)):
    info[1]-=pp[i]*condh[i]

vowel=['a','e','i','o','u']
pp=[]
condh=[]
for i in vowel:
    sump=0
    newp=[]
    for j in range(shape):
        k=0
        if (dic[j][k] in vowel) and (dic[j][k] == i):
            sump += prob[j]
        k+=1
        while k<len(dic[j]) and (dic[j][k-1] not in vowel):
            if (dic[j][k] in vowel) and (dic[j][k] == i):
                sump+=prob[j]
            k+=1
    for j in range(shape):
        k=0
        if (dic[j][k] in vowel) and (dic[j][k] == i):
            newp.append(prob[j] / sump)
        k+=1
        while k<len(dic[j]) and (dic[j][k-1] not in vowel):
            if (dic[j][k] in vowel) and (dic[j][k] == i):
                newp.append(prob[j] / sump)
            k+=1
    condh.append(H(newp))
    pp.append(sump)
sump=0
newp=[]
for j in range(shape):
    if set(dic[j])-set(vowel)==set(dic[j]):
        sump += prob[j]
for j in range(shape):
    if set(dic[j]) - set(vowel) == set(dic[j]):
        newp.append(prob[j] / sump)
condh.append(H(newp))
pp.append(sump)
print(np.sum(pp))
info[2]=H(prob)
for i in range(len(condh)):
    info[2]-=pp[i]*condh[i]
print(info)

fig=plt.figure()
ax=fig.add_subplot(111)
ax.bar(['first character','last character','first vowel'],info)
plt.show()
```

5.

- Actually the average surprisal of each word length is exactly the conditional entropy conditioning on word length.

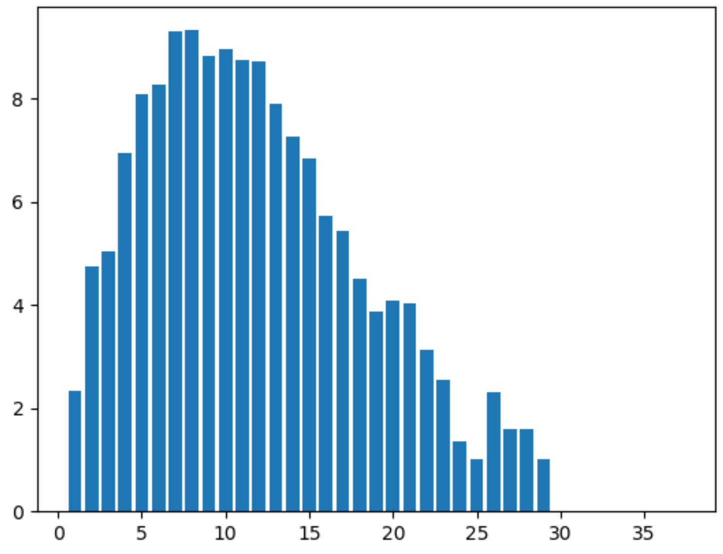

Figure 3: average surprisal over different word lengths

- If English were an efficient code, this plot should be linear.
- Proof: Denote the number of elements in alphabet as n, then for word length l, all possible representing ways of a word is n^l. If each word of length l are of same frequency, the conditional entropy is

$$n^l \cdot \frac{1}{n^l} \cdot \log \frac{1}{\frac{1}{n^l}} = l \log n$$

  which is linear to l. If we denote $k := \log n$, then $H(l) = kl$.

- Hence my plot does not agree with an efficient code because in fact we do not utilize all possible combinations with fewer word length instead of longer one.
- Code:

```
import pandas as pd
import csv
import numpy as np
import matplotlib.pyplot as plt

def H(p):
    s=0
    for i in p:
        s+=-i*np.log2(i)
    return s
```

```
data=pd.read_csv("Assignment10-WordFrequencies.csv",header=None)
shape=int(data.shape[0])
#shape=500
dic=[]
frec=[]
prob=[]
sum=0
for i in range(shape):
    dic.append(str(data.loc[i][0]))
    frec.append(float(data.loc[i][1]))
    sum+=frec[i]
for i in range(shape):
    prob.append(frec[i]/sum)

maxlen=0
for i in range(shape):
    if len(dic[i])>maxlen:
        maxlen=len(dic[i])
print(maxlen)

condh=[]
for l in range(maxlen):
    sump=0
    newp=[]
    for j in range(shape):
        if len(dic[j]) == l+1:
            sump+=prob[j]
    for j in range(shape):
        if len(dic[j]) == l+1:
            newp.append(prob[j]/sump)
    condh.append(H(newp))
ll=[i+1 for i in range(maxlen)]
fig=plt.figure()
ax=fig.add_subplot(111)
ax.bar(ll,condh)
plt.show()
```

6.

- Based on the ideal solutions we mentioned in 5), let the loss of word length l to be

$$loss(l) = l \log n - H_{actual}(l)$$

  where $n = 26$.
  The measure is:

$$LOSS(N) = \frac{\sum_{l=1}^{maxlen(N)} loss(l)}{maxlen(N)}$$

- Run procedure for $N = 10, 20, \ldots$ till all words are taken into consideration. Plot as shown below.

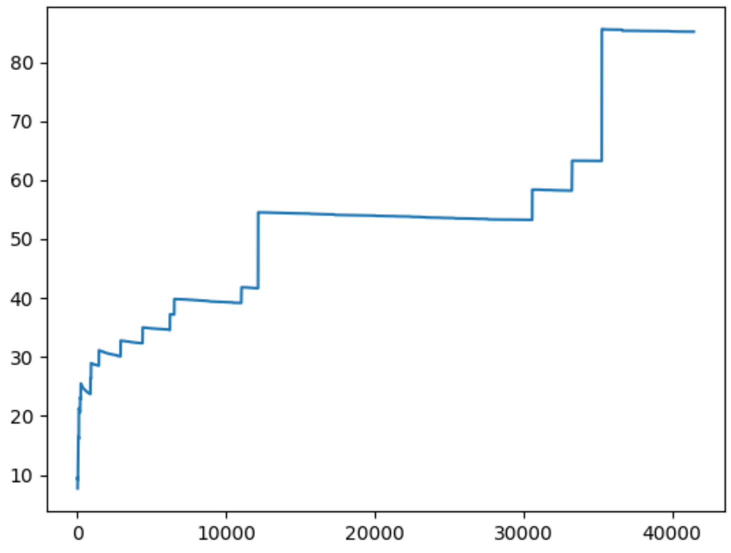

Figure 4: losses for different N

- From this plot we can conclude that it's true that most frequent used words are somehow "more optimized" because the normalized loss is blowing up when N grows.

- Code:

```
import pandas as pd
import csv
import numpy as np
import matplotlib.pyplot as plt

def H(p):
    s=0
    if p==[]:
        return 0
    for i in p:
        s+=-i*np.log2(i)
    return s

data=pd.read_csv("Assignment10-WordFrequencies.csv",header=None)
shape=int(data.shape[0])
#shape=500
dic=[]
frec=[]
prob=[]
sum=0
for i in range(shape):
    dic.append(str(data.loc[i][0]))
    frec.append(float(data.loc[i][1]))
    sum+=frec[i]
for i in range(shape):
    prob.append(frec[i]/sum)

dicc=dic.copy()
probb=prob.copy()
diccc=[]
probbb=[]
def find(N):
    for nn in range(N):
        maxx=0
        for mm in range(len(dicc)):
            if frec[maxx]<frec[mm]:
                maxx=mm
        diccc.append(dicc.pop(maxx))
        probbb.append(probb.pop(maxx))
    return diccc,probbb

loss = []
for i in range(int(shape/10)):
    inn=(i+1)*10
    idd,ipp=find(10)
    maxlen = 0
    for i in range(inn):
        if len(idd[i]) > maxlen:
            maxlen = len(idd[i])
    print(inn,"maxlen:",maxlen)
    ls = 0
    for l in range(maxlen):
        sump = 0
        newp = []
        for j in range(inn):
            if len(idd[j]) == l + 1:
                sump += ipp[j]
        for j in range(inn):
            if len(idd[j]) == l + 1:
                newp.append(ipp[j] / sump)
        ls += (l + 1) * np.log2(26) - H(newp)
    loss.append(ls/maxlen)

ll=[(i+1)*10 for i in range(int(shape/10))]
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(ll,loss)
plt.show()
```