HW1_A20439949_Soutonglang

November 12, 2023

1 CS 585 - HW 1 - Getting Started

Tania Soutonglang A20439949

[1]:
```
#NLTK setup - uncomment and run first time you import NLTK
import nltk
nltk.download('punkt')

import pandas as pd
from nltk.tokenize import word_tokenize
from csv import QUOTE_NONE

import numpy as np
from nltk import probability
import math
import string
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\tsout\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[2]:
```
# import sst dataset
df_sst = pd.read_csv("SST-2/train.tsv",delimiter="\t")
df_sst.head(3)
```

[2]:
```
                                            sentence  label
0       hide new secretions from the parental units       0
1               contains no wit , only labored gags       0
2  that loves its characters and communicates som…      1
```

[3]:
```
# import qnli dataset
df_qnli = pd.read_csv("QNLI/dev.tsv",delimiter="\t",quoting=QUOTE_NONE)
df_qnli.head(3)
```

[3]:
```
   index                                          question  \
0      0  What came into force after the new constitutio…
1      1  What is the first major city in the stream of …
2      2  What is the minimum required if you want to te…
```

```
                                            sentence          label
0  As of that day, the new constitution heralding…      entailment
1  The most important tributaries in this area ar…  not_entailment
2  In most provinces a second Bachelor's Degree s…  not_entailment
```

1.1 Problem 1 - Representing English Text

- Read in these two GLUE datasets (see section "DATA" above). Also convert alphabetical characters to lower case:

[4]:
```
# create sst series with only sentence column
df_sstData = df_sst[['sentence']].dropna()
df_sstData = df_sstData['sentence'].str.lower()
df_sstData.head(3)
```

[4]:
```
0         hide new secretions from the parental units
1                  contains no wit , only labored gags
2    that loves its characters and communicates som…
Name: sentence, dtype: object
```

[5]:
```
# create qnli series with only sentence column
df_qnliData = df_qnli[['sentence']].dropna()
df_qnliData = df_qnliData['sentence'].str.lower()
df_qnliData.head(3)
```

[5]:
```
0    as of that day, the new constitution heralding…
1    the most important tributaries in this area ar…
2    in most provinces a second bachelor's degree s…
Name: sentence, dtype: object
```

- Convert each dataset into a single list of tokens by applying the function "word_tokenize()" in the NLTK :: nltk.tokenize package. We will use these lists represent two distributions of English text.
- To show you have finished this step, print the first 10 tokens from each dataset.

[6]:
```
sst_tokens = []

# create token list for sst data
for sentence in df_sstData:
    # tokenize the sentence
    tokens = word_tokenize(sentence)

    # filter out the punctuation
    filtered_tokens = [token for token in tokens if token not in string.
 ↪punctuation]

    # append tokens to list
    sst_tokens.append(filtered_tokens)
```

```
temp = []
for words in sst_tokens:
    temp.extend(words)
sst_vocab = np.unique(temp)

for token in range(10):
    print(sst_vocab[token])
```

```
''
'30s
'40s
'50s
'53
'60s
'70s
'80s
'90s
'd
```

[7]:

```
qnli_tokens = []

# create token list for qnli data
for sentence in df_qnliData:
    # tokenize the sentence
    tokens = word_tokenize(sentence)

    # filter out the punctuation
    filtered_tokens = [token for token in tokens if token not in string.
 ↪punctuation]

    # append tokens to list
    qnli_tokens.append(filtered_tokens)

temp = []
for words in qnli_tokens:
    temp.extend(words)
qnli_vocab = np.unique(temp)

for token in range(10):
    print(qnli_vocab[token])
```

```
''
'aided
'apothecary
'bath
'bends
'bucks
```

```
'carry
'chares
'church
'comb
```

1.2 Problem 2 - Word Probability

- Write a python function that creates a probability distribution from a list of tokens. This function should return a dictionary that maps a token to a probability (I.e., maps a string to a floating-point value)

[8]:
```python
def create_probDist(tokenList):
    # get the total number of tokens
    count = len(tokenList)

    # create a frequency distribution of the token list
    freqDist = probability.FreqDist(tokenList)

    # calculate the probability of each token
    probDist = {token: freq / count for token, freq in freqDist.items()}

    return probDist
```

- Apply your function to the list created in Problem 1 to create SST and QNLI distributions.
- Show that both probability distributions sum to 1, allowing for some small numerical rounding error. Or, if they do not, add a comment in your notebook to explain why.

[9]:
```python
sst_probDist = create_probDist(sst_vocab)
print(sum(sst_probDist.values()))
```

```
0.999999999999606
```

[10]:
```python
qnli_probDist = create_probDist(qnli_vocab)
print(sum(qnli_probDist.values()))
```

```
1.0000000000001854
```

1.3 Problem 3 - Entropy

- Write a python function that computes the entropy of a random variable, input as a probability distribution.

[11]:
```python
def calc_entropy(probDist):
    entropy = 0.0

    # calculate the entropy of each word
    for prob in probDist.values():
        if prob > 0:
            entropy += -prob * math.log2(prob)
```

```
    return entropy
```

- Use this function to compute the word-level entropy of SST and QNLI, using the distributions you created in Problem 2. Show results in your notebook.

[12]:
```
sst_entropy = calc_entropy(sst_probDist)
print(sst_entropy)
```

```
13.851944197996536
```

[13]:
```
qnli_entropy = calc_entropy(qnli_probDist)
print(qnli_entropy)
```

```
13.944071453008073
```

1.4 Problem 4 - KL Divergence

- Write a python function to compute the KL divergence between two probability distributions.

[14]:
```
def calc_KLDivergence(probDist_a, probDist_b):
    kl_divergence = 0.0

    for key, prob_a in probDist_a.items():
        # Get the corresponding probability from the 2nd list, otherwise␣
↪default to 0 if it doesn't exist
        prob_b = probDist_b.get(key, 0)

        # calculate the KL divergence
        if prob_b > 0:
            kl_divergence += prob_a * math.log2(prob_a / prob_b)

    return kl_divergence
```

- Apply this function to the distributions you created in Problem 2 to show that KL divergence is not symmetric.

[15]:
```
sst_KLqnli = calc_KLDivergence(sst_probDist, qnli_probDist)
print(sst_KLqnli)
```

```
0.032000918551245115
```

[16]:
```
qnli_KLsst = calc_KLDivergence(qnli_probDist, sst_probDist)
print(qnli_KLsst)
```

```
-0.030021291903982113
```

1.5 Problem 5 - Entropy Rate

- Write a python function that computes the per-word entropy rate of a message relative to a specific probability distribution.

[17]:
```
def calc_perWordEntropyRate(doc, probDist):
    # split the inputted document into tokens
    tokens = doc.split()
    tokens = [token for token in tokens if token not in string.punctuation]
    token_count = len(words)

    entropy_rate = 0.0

    for word in tokens:
        # get the probability of the word, otherwise it will be 0
        probability = probDist.get(word, 0)
        if probability > 0:
            entropy_rate += -probability * math.log2(probability)

        # find the per-word rate
        if token_count > 0:
            entropy_rate /= token_count

    return entropy_rate
```

- Find a recent movie review online (any website) and compute the entropy rates of this movie review using the distributions you created for both SST and QNLI datasets. Show results in your notebook.

[18]:
```
# movie review of The Nun II from Rotten Tomatoes (https://www.rottentomatoes.
↪com/m/the_nun_ii/reviews)
review = "A narratively bland sequel that grants star Taissa Farmiga a bit more␣
↪agency (while wasting Storm Reid), The Nun II proves that while there's␣
↪plenty of box office in The Conjuring Universe, the storytelling techniques␣
↪are phoning it in."
review = review.lower()
```

[19]:
```
# per-word entropy using sst probability distribution
sst_perWord = calc_perWordEntropyRate(review, sst_probDist)
print(sst_perWord)
```

```
5.576440220966119e-07
```

[20]:
```
# per-word entropy using qnli probability distribution
qnli_perWord = calc_perWordEntropyRate(review, qnli_probDist)
print(qnli_perWord)
```

```
5.266264845240931e-07
```

1.6 Problem 6 - Observed Entropy Rate

- Refer to your results from Problem 5. Which distribution gives you the lowest entropy rate for your movie review? Does this match what you expected? Why or why not?

Entropy rate with SST: 4.836273055258784e-07 Entropy rate with QNLI: 4.5657562774588297e-07 The QNLI distribution gave the lowest entropy rate for my movie review. I expected this only because the probability distribution of this dataset had lower numbers compared to the SST dataset, therefore anything calculated with that probability distribution would be lower.

1.7 Problem 7 – Zero probabilities

- Problem 5 required that you handle "zero probabilities" cases, where a token occurred in one dataset but not the other. How did you handle these tokens? (Hint: Dropping the word from both probability distributions is not an ideal solution).

I handled these cases by making the probability of the missing token 0.0. This allowed me to still calculate the word if it appeared in one dataset, but also factored in the fact that it does not exist in the other (therefore the 0% possibility in the probability distribution).

[ ]:

```

```