INTRODUCTION TO MACHINE LEARNING

First Homework

Julian Ewaied and Namir Ballan

The University of Haifa – Spring Semester of 2023

CONTENTS

QUESTION 2

Let S be the event of the tested person being sick, and T be the event of the test being positive. We're given a prior $P(S) = \frac{1}{1000}$. We're also given the conditional probability $P(T|\bar{S}) = 0.01$, $P(\bar{T}|S) = 0$.

a. We want to calculate the conditional probability $P(S|T)$. We can use Bayes rule for that:

$$P(S|T) = P(T|S) \cdot \frac{P(S)}{P(T)} = \left(1 - P(\bar{T}|S)\right) \cdot \frac{P(S)}{P(T|\bar{S})P(\bar{S}) + P(T|S)P(S)}$$

$$P(S|T) = 1 \cdot \frac{10^{-3}}{0.01 \cdot (1 - 0.001) + 1 \cdot 0.001} = 0.091$$

b. It is more probable I don't have the disease since $P(S|T) < P(\bar{S}|T)$.

c. Obviously yes, since the maximum likelihood would ignore the strong prior we have, which makes it almost sure that we have the disease (the likelihood $P(T|S) = 1 - P(\bar{T}|S) = 1$).

QUESTION 4

We wanted to implement a naïve Bayes classifier using MAP. For that purpose, we had a huge dataset of labels and sentences. We defined a class 'NBClassifier'. The constructor of the class receives an array-like object where each entry of it is an array of words (split string), called texAll, and respectively the sentences' label, called lbAll. In addition, we had all possible categories and our vocabulary. Then, we built counts – the number of appearances of each label, in a dictionary, and totals – a dictionary to save the number of words for each label.

Then , we apply train(), which learns the data by finding the class conditionals $P(x|\omega_i)$, and the priors $P(\omega_i)$ using MLE as we proved in class. we apply logarithms on all numbers to avoid floating point underflow, and for convenience we do it with base 2 (because the smaller the base, the more precise the numbers).

Now that we have the class conditionals and priors, we can write 'calc_posterior()', which receives a sentence and a label, and calculates the log of the posterior

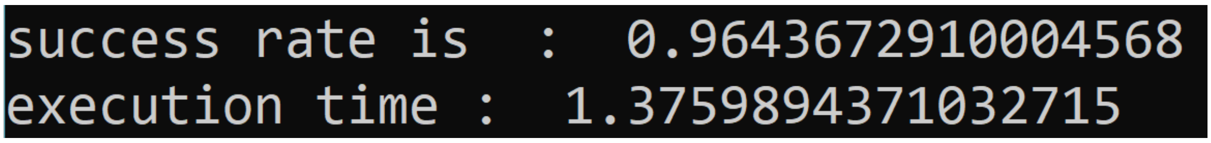
$$\log\left(P(label)P(sentence|label)\right) = \log\left(P(label)\right) + \Sigma_{w \in sentence} \log\left(P(w|label)\right)$$

Keeping in consideration that some words might not be in the vocabulary or might not appear in some label, and therefore we should use 'totals' to calculate the answer with Laplace smoothing (only for words whose probability is 0). The probability of a word with a zero-probability is:

$$P(word|label) = \frac{1}{N_{label} + N_{voc}}$$

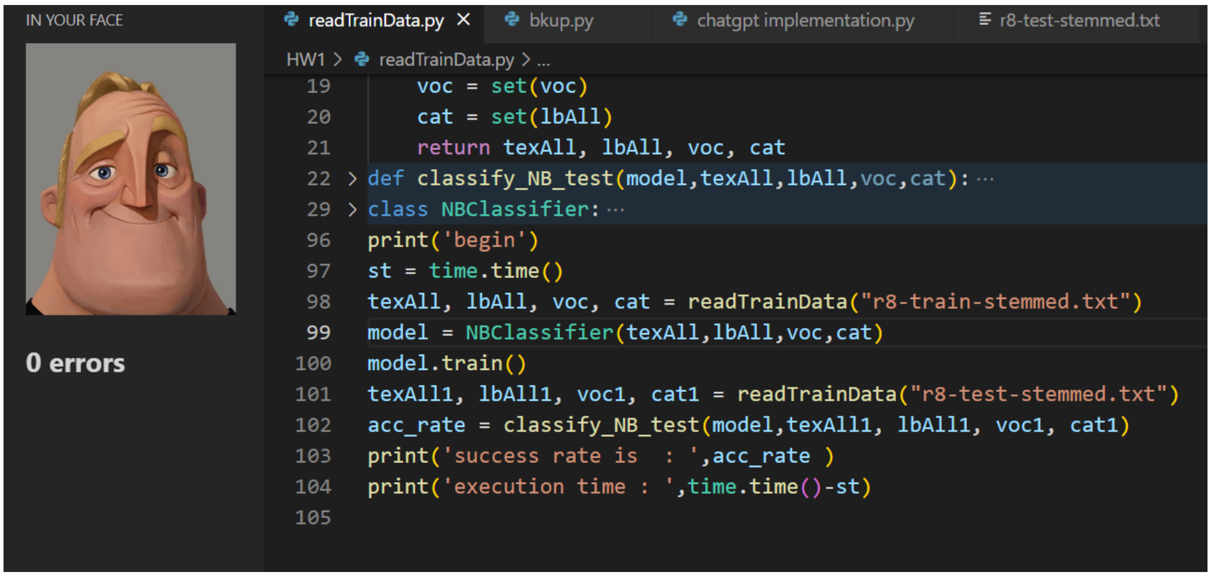
This makes sense since the more we know words, the less it's probable to get a new word, and the more we've seen words in a label, the less it's probable to miss a word. Therefore, we've minimized the effect of new words but still didn't ignore them.

And then, classify can make use of them to calculate the MAP output. The final step is to test the classifier, which can be done using the test dataset, which is done by the function 'NB_classify_test()', as we read from the test file and compare our classifier's answer to the actual answer. Eventually we receive the success rate and print it, which is in this case and for the input we've received:



```
success rate is  :  0.9643672910004568
execution time :  1.3759894371032715
```

Which is:



```python
    voc = set(voc)
    cat = set(lbAll)
    return texAll, lbAll, voc, cat
def classify_NB_test(model,texAll,lbAll,voc,cat): ...
class NBClassifier: ...
print('begin')
st = time.time()
texAll, lbAll, voc, cat = readTrainData("r8-train-stemmed.txt")
model = NBClassifier(texAll,lbAll,voc,cat)
model.train()
texAll1, lbAll1, voc1, cat1 = readTrainData("r8-test-stemmed.txt")
acc_rate = classify_NB_test(model,texAll1, lbAll1, voc1, cat1)
print('success rate is  : ',acc_rate )
print('execution time : ',time.time()-st)
```

Figure 1: very good!!

For more explanation check the attached code.

QUESTION 5

Let's look at the output of the Bayesian classifier:

$$c_i = \arg\max_{\alpha_i} -R(\alpha_i|x) = \arg\max_{\alpha_i} -\sum_{c_i} \lambda(\alpha_i|c_j)P(c_j|x) = \arg\max_{\alpha_i} -\sum_{i\neq j} P(c_j|x)$$

Where the last transition is simply substituting the loss function in.

$$c_i = \arg\max_{\alpha_i}\left(1 - \sum_{i\neq j} P(c_j|x)\right) = \arg\max_{\alpha_i}(P(c_i|x))$$

We only added 1 which doesn't change the argument maximum, and reached a new probability argmax which is the MAP output.