Assignment 2

Machine Learning
MSc Business Analytics

1 Individual Assignment

Instructions: *This exercise should be done "by hand", that is, not using Python. All necessary calculations should be included in the submission, as well as brief explanations of what you do.*

1.1 Bayes' Theorem: Dining Safely

95% of the restaurants in Kentish Town practice good hygiene. Each time you eat in a clean restaurant, there is a 2% chance that you will get sick, independent of your previous visits. Each time you eat in a restaurant that does not practice good hygiene, on the other hand, there is a 30% chance that you will get sick, independent of your previous visits.

1. You eat at a restaurant in Kentish Town and you get sick. What is the probability that the restaurant practices good hygiene?

Hint: *Using the law of total probability, first calculate the probability of the event 'falling sick'. You can use this probability in your subsequent application of Bayes' Theorem.*

2. You go to the same restaurant for a second time, and you get sick again. What is the probability of the restaurant practicing good hygiene now?

Hint: *Using the law of total probability, first calculate the probability of the event 'falling sick twice'. In this calculation, you can use the conditional independences*

$$\mathbb{P}(\text{falling sick twice}|\text{restaurant clean}) = \mathbb{P}(\text{falling sick once}|\text{restaurant clean})^2$$

and

$$\mathbb{P}(\text{falling sick twice}|\text{restaurant not clean}) = \mathbb{P}(\text{falling sick once}|\text{restaurant not clean})^2.$$

You can use these probabilities in your subsequent application of Bayes' Theorem, where you can again use the same conditional independence.

1.2 Building a Spam Filter

Consider the following 4 SMS messages and their classification as ‘ham’ or ‘spam’ messages:

message	**label**
I am going shopping	**ham**
Well done	**ham**
Do you need bitcoins	**spam**
win free tickets	**spam**

Using the above messages, construct a Naïve Bayes classifier to classify new SMS messages. In particular:

1. Compute the prior probabilities of a new SMS message being ‘spam’ or ‘ham’. (These are just the empirical probabilities from the four messages above.)
2. For each de-capitalised keyword that appears in your training set (that is, ‘i’, ‘am’, ‘going’, ‘shopping’, ‘well’, ‘done’, ‘do’, ‘you’, ‘need’, ‘bitcoins’, ‘win’, ‘free’ and ‘tickets’), build a frequency table that records the likelihoods $\mathbb{P}(W|\text{ham})$, $\mathbb{P}(\neg W|\text{ham})$, $\mathbb{P}(W|\text{spam})$ and $\mathbb{P}(\neg W|\text{spam})$.

Tip: Remember Laplace’s estimator!

Just like a likelihood of 0, a likelihood of 1 is troublesome as its negation is 0. This is not a concern for big data sets as no word is likely to appear in every message. To avoid this issue in our toy example, replace any likelihood smaller than 0.05 (larger than 0.95) with 0.05 (0.95). This can be regarded as a variant of the Laplace estimator.

Now you can use the Naïve Bayes algorithm to make predictions. Predict if the following two SMS messages are ham or spam:

message	**label**
Going out	?
Get bitcoins now	?

This actually turns out to be more cumbersome than expected: For the first SMS message, for example, you need to use the law of total probability as well as the conditional independence to compute the quantity

$$\mathbb{P}(\neg\text{i} \wedge \neg\text{am} \wedge \neg\text{going} \wedge \text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\ \wedge \neg\text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}).$$

Afterwards you can compute the quantities

$$\mathbb{P}(\text{ham}|\neg\text{i} \wedge \neg\text{am} \wedge \neg\text{going} \wedge \text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\ \wedge \neg\text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets})$$

and

$$\begin{aligned}\mathbb{P}(\text{spam}|\neg\text{i} \wedge \neg\text{am} \wedge \neg\text{going} \wedge \text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need}\\ \wedge \neg\text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets})\end{aligned}$$

using Bayes' Theorem and the conditional independence.

Alternatively, you can exploit (as explained in the slides) the fact that

$$\mathbb{P}(\text{ham}|\neg\text{i} \wedge \neg\text{am} \wedge \ldots) \ \propto \ \mathbb{P}(\text{ham}) \cdot \mathbb{P}(\neg\text{i}|\text{ham}) \cdot \mathbb{P}(\neg\text{am}|\text{ham}) \cdot \ldots$$

and

$$\mathbb{P}(\text{spam}|\neg\text{i} \wedge \neg\text{am} \wedge \ldots) \ \propto \ \mathbb{P}(\text{spam}) \cdot \mathbb{P}(\neg\text{i}|\text{spam}) \cdot \mathbb{P}(\neg\text{am}|\text{spam}) \cdot \ldots$$

to calculate propensities that you subsequently scale to probabilities by dividing them with the sum

$$\mathbb{P}(\text{ham}) \cdot \mathbb{P}(\neg\text{i}|\text{ham}) \cdot \mathbb{P}(\neg\text{am}|\text{ham}) \cdot \ldots \ + \ \mathbb{P}(\text{spam}) \cdot \mathbb{P}(\neg\text{i}|\text{spam}) \cdot \mathbb{P}(\neg\text{am}|\text{spam}) \cdot \ldots$$

You can use a spreadsheet to simplify computations if you wish. Make sure that you explain every step, though!

2 Group Assignment

For this group assignment, you are given four datasets of labelled SMS messages and two lists of words:

- *training.csv, validation.csv, test1.csv, test2.csv*
- *list_censored_test1.txt, list_censored_test2.txt*

Follow the steps below to build a Naïve Bayes spam filter.

1. Load the *.csv* files into Python data frames and the *.txt* files into lists. (Please do NOT shuffle the data frames)
2. Pre-process the SMS messages: Remove all punctuation and numbers from the SMS messages, and change all messages to lower case. (Please provide the Python code that achieves this!)
3. While Python's SciKit-Learn library has a Naïve Bayes classifier, it works with continuous probability distributions and assumes numerical features. Although it is possible to transform categorical variables into numerical features using a binary encoding, we will instead build a simple Naïve Bayes classifier from scratch:

```
import numpy as np
import pandas as pd
class NaiveBayesForSpam:
    def train (self, hamMessages, spamMessages):
        self.words = set ('␣'.join (hamMessages + spamMessages).split())
        self.priors = np.zeros (2)
        self.priors[0] = float (len (hamMessages)) / (len (hamMessages) +
            len (spamMessages))
        self.priors[1] = 1.0 - self.priors[0]
        self.likelihoods = []
        for i, w in enumerate (self.words):
            prob1 = (1.0 + len ([m for m in hamMessages if w in m])) /
                len (hamMessages)
            prob2 = (1.0 + len ([m for m in spamMessages if w in m])) /
                len (spamMessages)
            self.likelihoods.append ([min (prob1, 0.95), min (prob2,
                0.95)])
        self.likelihoods = np.array (self.likelihoods).T

    def train2 (self, hamMessages, spamMessages):
        self.words = set ('␣'.join (hamMessages + spamMessages).split())
        self.priors = np.zeros (2)
        self.priors[0] = float (len (hamMessages)) / (len (hamMessages) +
            len (spamMessages))
        self.priors[1] = 1.0 - self.priors[0]
        self.likelihoods = []
        spamkeywords = []
        for i, w in enumerate (self.words):
            prob1 = (1.0 + len ([m for m in hamMessages if w in m])) /
                len (hamMessages)
            prob2 = (1.0 + len ([m for m in spamMessages if w in m])) /
                len (spamMessages)
            if prob1 * 20 < prob2:
                self.likelihoods.append ([min (prob1, 0.95), min (prob2,
                    0.95)])
                spamkeywords.append (w)
        self.words = spamkeywords
        self.likelihoods = np.array (self.likelihoods).T

    def predict (self, message):
        posteriors = np.copy (self.priors)
        for i, w in enumerate (self.words):
            if w in message.lower():  # convert to lower-case
                posteriors *= self.likelihoods[:,i]
            else:
                posteriors *= np.ones (2) - self.likelihoods[:,i]
            posteriors = posteriors / np.linalg.norm (posteriors, ord =
                1)  # normalise
        if posteriors[0] > 0.5:
            return ['ham', posteriors[0]]
        return ['spam', posteriors[1]]
    def score (self, messages, labels):
        confusion = np.zeros(4).reshape (2,2)
        for m, l in zip (messages, labels):
            if self.predict(m)[0] == 'ham' and l == 'ham':
                confusion[0,0] += 1
            elif self.predict(m)[0] == 'ham' and l == 'spam':
                confusion[0,1] += 1
            elif self.predict(m)[0] == 'spam' and l == 'ham':
                confusion[1,0] += 1
            elif self.predict(m)[0] == 'spam' and l == 'spam':
                confusion[1,1] += 1
        return (confusion[0,0] + confusion[1,1]) / float (confusion.sum()
            ), confusion
```

4. Explain the code: What is the purpose of each function? What do ’train’ and ‘train2’ do, and what is the difference between them? Where in the code is Bayes’ Theorem being applied?

5. Use your training set to train the classifiers ‘train’ and ‘train2’. Note that the interfaces of our classifiers require you to pass the ham and spam messages separately.

6. Using the validation set, explore how each of the two classifiers performs out of sample.

7. Why is the ‘train2’ classifier faster? Why does it yield a better accuracy both on the training and the validation set?

8. How many false positives (ham messages classified as spam messages) did you get in your validation set? How would you change the code to reduce false positives at the expense of possibly having more false negatives (spam messages classified as ham messages)?

In this next section you are going to use the test sets and lists of censored words.
10% and 30% of the keywords obtained from the ‘train’ function have been removed (*”censored”*) from the messages in *test1* and *test2* respectively, therefore, you have no knowledge whether the test sets contain these words or not. The censored words, listed in the *.txt* files, must then be treated as missing values.

9. Assuming the missing words are $(X_j = x_j, \dots, X_k = x_k)$ with $k \leq p$, how would you change the formula given as hint to calculate $\mathbb{P}(Y = C_j | X_1 = x_1, \dots, X_p = x_p)$?

10. Modify the ‘predict’ function in the code to implement the change in the previous point, and use it to report accuracies on *test1*, with both ‘train’ and ‘train2’.

11. Repeat the previous point using *test2*. Briefly report your findings.

Hint: *In the lecture we have discussed the formula*

$$\mathbb{P}(Y = C_j | X_1 = x_1, \ldots, X_p = x_p) \propto \mathbb{P}(Y = C_j) \prod_{i=1}^{p} \mathbb{P}(X_i = x_i | Y = C_j)$$

for the Naïve Bayes algorithm. While mathematically correct, the formula cannot be used 'as is' in an implementation since the product of many probabilities leads to a very small number that causes numerical issues. Hence, the code above conducts a normalisation after each multiplication.