Assignment 2 – Solution

Machine Learning
MSc Business Analytics

1 Individual Assignment

1.1 Bayes' Theorem: Dining Safely

Using C for 'clean restaurant' (and the complementary event $\overline{C}$ for 'dirty restaurant') and S for 'falling sick' (as well as the complementary event $\overline{S}$ for 'not falling sick'), the prior probabilities are $\mathbb{P}(C) = 0.95$ and $\mathbb{P}(\overline{C}) = 0.05$. From the text we also know that $\mathbb{P}(S|C) = 0.02$ and $\mathbb{P}(S|\overline{C}) = 0.3$.

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

From the law of total probability, we obtain

$$\mathbb{P}(S) = \mathbb{P}(S|C) \cdot \mathbb{P}(C) + \mathbb{P}(S|\overline{C}) \cdot \mathbb{P}(\overline{C}) = 0.02 \cdot 0.95 + 0.3 \cdot 0.05 = 0.034.$$

We then obtain that

$$\mathbb{P}(C|S) = \frac{\mathbb{P}(C)}{\mathbb{P}(S)} \cdot \mathbb{P}(S|C) = \frac{0.95}{0.034} \cdot 0.02 = 0.56$$

as well as

$$\mathbb{P}(\overline{C}|S) = \frac{\mathbb{P}(\overline{C})}{\mathbb{P}(S)} \cdot \mathbb{P}(S|\overline{C}) = \frac{0.05}{0.034} \cdot 0.3 = 0.44.$$

Thus, it is more likely that the restaurant practices clean hygiene.

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?

We denote by S_1 and S_2 the two events 'sick for the first time' and 'sick for the second time', respectively. From the law of total probability, we obtain

$$\begin{aligned}
\mathbb{P}(S_1 \wedge S_2) &= \mathbb{P}(S_1 \wedge S_2|C) \cdot \mathbb{P}(C) + \mathbb{P}(S_1 \wedge S_2|\overline{C}) \cdot \mathbb{P}(\overline{C}) \\
&= \mathbb{P}(S_1|C) \cdot \mathbb{P}(S_2|C) \cdot \mathbb{P}(C) + \mathbb{P}(S_1|\overline{C}) \cdot \mathbb{P}(S_2|\overline{C}) \cdot \mathbb{P}(\overline{C}) \\
&= 0.02^2 \cdot 0.95 + 0.3^2 \cdot 0.02 \;=\; 0.0049.
\end{aligned}$$

We then obtain that

$$\begin{aligned}\mathbb{P}(C|S_1 \wedge S_2) &= \frac{\mathbb{P}(C)}{\mathbb{P}(S_1 \wedge S_2)} \cdot \mathbb{P}(S_1 \wedge S_2|C) \\ &= \frac{\mathbb{P}(C)}{\mathbb{P}(S_1 \wedge S_2)} \cdot \mathbb{P}(S_1|C) \cdot \mathbb{P}(S_2|C) \\ &= \frac{0.95}{0.0049} \cdot 0.02^2 \;=\; 0.077\end{aligned}$$

as well as

$$\begin{aligned}\mathbb{P}(\overline{C}|S_1 \wedge S_2) &= \frac{\mathbb{P}(\overline{C})}{\mathbb{P}(S_1 \wedge S_2)} \cdot \mathbb{P}(S_1 \wedge S_2|\overline{C}) \\ &= \frac{\mathbb{P}(\overline{C})}{\mathbb{P}(S_1 \wedge S_2)} \cdot \mathbb{P}(S_1|\overline{C}) \cdot \mathbb{P}(S_2|\overline{C}) \\ &= \frac{0.05}{0.0049} \cdot 0.3^2 \;=\; 0.9222.\end{aligned}$$

Thus, with 92.2% probability the restaurant does *not* practice clean hygiene!

1.2 Building a Spam Filter

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.)

The priors are $\mathbb{P}(\text{spam}) = \frac{2}{4} = 0.5$ and $\mathbb{P}(\text{ham}) = \frac{2}{4} = 0.5$.

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 $\mathbb{P}(\neg W|\text{spam})$.

The likelihoods (truncated to the $[0.05, 0.95]$ interval) are given in the following table:

W	$\mathbb{P}(W\|\text{ham})$	$\mathbb{P}(W\|\text{spam})$	$\mathbb{P}(\neg W\|\text{ham})$	$\mathbb{P}(\neg W\|\text{spam})$
i	0.5	0.05	0.5	0.95
am	0.5	0.05	0.5	0.95
going	0.5	0.05	0.5	0.95
shopping	0.5	0.05	0.5	0.95
well	0.5	0.05	0.5	0.95
done	0.5	0.05	0.5	0.95
do	0.05	0.5	0.95	0.5
you	0.05	0.5	0.95	0.5
need	0.05	0.5	0.95	0.5
bitcoins	0.05	0.5	0.95	0.5
win	0.05	0.5	0.95	0.5
free	0.05	0.5	0.95	0.5
tickets	0.05	0.5	0.95	0.5

Now we can make predictions. For 'Going out', we first calculate

$$\begin{aligned}
&\mathbb{P}(\neg\text{i} \wedge \neg\text{am} \wedge \text{going} \wedge \neg\text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\
&\qquad \wedge \neg\text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}) \\
&= \mathbb{P}(\neg\text{i} \wedge \neg\text{am} \wedge \text{going} \wedge \neg\text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\
&\qquad \wedge \neg\text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}|\text{ham}) \cdot \mathbb{P}(\text{ham}) + \\
&\quad \mathbb{P}(\neg\text{i} \wedge \neg\text{am} \wedge \text{going} \wedge \neg\text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\
&\qquad \wedge \neg\text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}|\text{spam}) \cdot \mathbb{P}(\text{spam}) \\
&= \mathbb{P}(\neg\text{i}|\text{ham}) \cdot \ldots \cdot \mathbb{P}(\neg\text{tickets}|\text{ham}) \cdot \mathbb{P}(\text{ham}) + \\
&\quad \mathbb{P}(\neg\text{i}|\text{spam}) \cdot \ldots \cdot \mathbb{P}(\neg\text{tickets}|\text{spam}) \cdot \mathbb{P}(\text{spam}) \\
&= 0.5^6 \cdot 0.95^7 \cdot 0.5 + 0.95^5 \cdot 0.5^7 \cdot 0.05 \cdot 0.5 \;=\; 0.00561.
\end{aligned}$$

We now calculate

$$\begin{aligned}
&\mathbb{P}(\text{ham}|\neg\text{i} \wedge \neg\text{am} \wedge \text{going} \wedge \neg\text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\
&\qquad \wedge \neg\text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}) \\
&= \frac{\mathbb{P}(\text{ham})}{\mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets})} \cdot \mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets}|\text{ham}) \\
&= \frac{\mathbb{P}(\text{ham})}{\mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets})} \cdot \mathbb{P}(\neg\text{i}|\text{ham}) \cdot \ldots \cdot \mathbb{P}(\neg\text{tickets}|\text{ham}) \\
&= \frac{0.5}{0.00561} \cdot 0.5^6 \cdot 0.95^7 \;=\; 0.97305
\end{aligned}$$

and

$$\begin{aligned}
&\mathbb{P}(\text{spam}|\neg\text{i} \wedge \neg\text{am} \wedge \text{going} \wedge \neg\text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\
&\qquad \wedge \neg\text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}) \\
&= \frac{\mathbb{P}(\text{spam})}{\mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets})} \cdot \mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets}|\text{spam}) \\
&= \frac{\mathbb{P}(\text{spam})}{\mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets})} \cdot \mathbb{P}(\neg\text{i}|\text{spam}) \cdot \ldots \cdot \mathbb{P}(\neg\text{tickets}|\text{spam}) \\
&= \frac{0.5}{0.00561} \cdot 0.95^5 \cdot 0.5^7 \cdot 0.05 \;=\; 0.02695.
\end{aligned}$$

Thus, the message should considered to be 'ham'.

For 'Get bitcoins now', we first calculate

$$\begin{aligned}
&\mathbb{P}(\neg\text{i} \wedge \neg\text{am} \wedge \neg\text{going} \wedge \neg\text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\
&\qquad \wedge \text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}) \\
&= \mathbb{P}(\neg\text{i} \wedge \neg\text{am} \wedge \neg\text{going} \wedge \neg\text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\
&\qquad \wedge \text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}|\text{ham}) \cdot \mathbb{P}(\text{ham}) + \\
&\quad \mathbb{P}(\neg\text{i} \wedge \neg\text{am} \wedge \neg\text{going} \wedge \neg\text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\
&\qquad \wedge \text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}|\text{spam}) \cdot \mathbb{P}(\text{spam}) \\
&= \mathbb{P}(\neg\text{i}|\text{ham}) \cdot \ldots \cdot \mathbb{P}(\neg\text{tickets}|\text{ham}) \cdot \mathbb{P}(\text{ham}) + \\
&\quad \mathbb{P}(\neg\text{i}|\text{spam}) \cdot \ldots \cdot \mathbb{P}(\neg\text{tickets}|\text{spam}) \cdot \mathbb{P}(\text{spam}) \\
&= 0.5^6 \cdot 0.95^6 \cdot 0.05 \cdot 0.5 + 0.95^6 \cdot 0.5^7 \cdot 0.5 \;=\; 0.00316.
\end{aligned}$$

We now calculate

$$\begin{aligned}
&\mathbb{P}(\text{ham}|\neg\text{i} \wedge \neg\text{am} \wedge \neg\text{going} \wedge \neg\text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\
&\qquad \wedge \text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}) \\
&= \frac{\mathbb{P}(\text{ham})}{\mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets})} \cdot \mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets}|\text{ham}) \\
&= \frac{\mathbb{P}(\text{ham})}{\mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets})} \cdot \mathbb{P}(\neg\text{i}|\text{ham}) \cdot \ldots \cdot \mathbb{P}(\neg\text{tickets}|\text{ham}) \\
&= \frac{0.5}{0.00316} \cdot 0.5^6 \cdot 0.95^6 \cdot 0.05 \;=\; 0.0909
\end{aligned}$$

and

$$\begin{aligned}
&\mathbb{P}(\text{spam}|\neg\text{i} \wedge \neg\text{am} \wedge \neg\text{going} \wedge \neg\text{shopping} \wedge \neg\text{well} \wedge \neg\text{done} \wedge \neg\text{do} \wedge \neg\text{you} \wedge \neg\text{need} \\
&\qquad \wedge \text{bitcoins} \wedge \neg\text{win} \wedge \neg\text{free} \wedge \neg\text{tickets}) \\
&= \frac{\mathbb{P}(\text{spam})}{\mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets})} \cdot \mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets}|\text{spam}) \\
&= \frac{\mathbb{P}(\text{spam})}{\mathbb{P}(\neg\text{i} \wedge \ldots \wedge \neg\text{tickets})} \cdot \mathbb{P}(\neg\text{i}|\text{spam}) \cdot \ldots \cdot \mathbb{P}(\neg\text{tickets}|\text{spam}) \\
&= \frac{0.5}{0.00316} \cdot 0.95^6 \cdot 0.5^7 \;=\; 0.9091.
\end{aligned}$$

Thus, the message should considered to be 'spam'.

2 Group Assignment

We proceed as follows:

1. Load the *.csv* files into Python data frames and the *.txt* files into lists. (Please do NOT shuffle the data frames)

This can be done with the following functions (which will be incorporated in 2.):

```
import pandas as pd
def load_csv(filepath):
        data_raw = pd.read_csv(filepath, header=0, usecols=[0,1], encoding="
            latin-1")
        data_raw.columns = ['label', 'sms']
        return data_raw
def load_censored(filepath):
        with open(filepath,'r') as f:
                return f.read().splitlines()
```

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!)

You can create new columns by operating on the existing ones. For example, you can create a new column with the length of each message in the DataFrame as follows:

```
df['length'] = df.sms.apply (lambda x : len (x))
```

Lambda expressions provide a convenient way to create ad-hoc functions in Python. The 'apply' method applies a function to every element of a DataFrame column. We can now pre-process the SMS messages as follows:

```
whitelist = set ('abcdefghijklmnopqrstuvwxy ')
df['sms'] = df.sms.apply (lambda x : ''.join (filter (whitelist.__contains__,
   x.lower())))
```

The following code loads and preprocesses all data, except for the censored lists which are already preprocessed

```
def load_sms_lbls(filepath):
        df   = load_csv(filepath)
        whitelist = set ('abcdefghijklmnopqrstuvwxyz ')
        df['sms'] = df.sms.apply (lambda x : ''.join (filter (whitelist.
            __contains__, x.lower())))
        msgs = list(df.sms)
        lbls = list(df.label)
        return df, msgs, lbls

train, trainms, trainlbs = load_sms_lbls('data/training.csv')
val, valms, vallbs       = load_sms_lbls('data/validation.csv')
test1, test1ms, test1lbs = load_sms_lbls('data/test1.csv')
test2, test2ms, test2lbs = load_sms_lbls('data/test2.csv')
c_list1 = load_censored('data/censored_list_test1.txt')
c_list2 = load_censored('data/censored_list_test2.txt')
```

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?

The functions 'train' and 'train2' calculate and store the prior probabilities and likelihoods. In Naïve Bayes this is all the training phase does. The 'predict' function repeatedly applies Bayes' Theorem to every word in the constructed dictionary, and based on the posterior probability it classifies the message as 'spam' or 'ham'. The 'score' function calls 'predict' for multiple messages and compares the outcomes with the supplied 'ground truth' labels and thus evaluates the classifier. It also computes and returns a confusion matrix.

The difference between 'train' and 'train2' is that the latter function only takes into account words that are 20 times more likely to appear in a spam message than a ham message. Not all words are equally informative. Using 'train2' will decrease the size of the dictionary significantly, thus making the classifier more efficient. There are multiple other ideas that one can use to construct more informative dictionaries. For example, you can treat words with the same root ('go', 'goes', 'went', 'gone', 'going') as the same word.

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.

The training functions require the ham and spam messages to be passed on separately:

```
hammsgs = [m for (m, l) in zip (trainms, trainlbs) if 'ham' in l]
spammsgs = [m for (m, l) in zip (trainms, trainlbs) if 'spam' in l]
print len (hammsgs)
print len (spammsgs)
```

We have 1,752 ham messages and 248 spam messages in our training set. Now we can create a classifier with

```
clf = NaiveBayesForSpam()
```

and train it via

```
clf.train (hammsgs, spammsgs)
```

or

```
clf.train2 (hammsgs, spammsgs)
```

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

We can do this via:

```
score,confusion = clf.score (valms, vallbs)
print score
print confusion
```

If we use the 'train' function, the confusion matrix is

	ham	**spam**
predicted ham	844	29
predicted spam	16	111

with an overall performance of 0.955.

Using 'train2', we obtain the confusion matrix

	ham	**spam**
predict ham	857	35
predict spam	3	105

with an overall performance of 0.962.

Our data is not equally divided into the two classes. As a baseline, let's see what the success rate would be if we always guessed 'ham':

```
print('basescore', len([1 for l in vallbs if 'ham' in l]) / float (len (vallbs
    )))
```

This gives a performance of 0.86, which is inferior. We can also calculate our in sample error:

```
score, confusion = clf.score (trainms, trainlbs)
print score
print confusion
```

If we use the 'train' function (this might take a long time!), the confusion matrix is

	ham	**spam**
predict ham	1,719	21
predict spam	33	227

with an overall performance of 0.973. Using 'train2', which is much faster, we obtain

	ham	**spam**
predict ham	1,751	41
predict spam	1	207

with an overall performance of 0.979.

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

We have answered this question already under point 5: The difference between 'train' and 'train2' is that the latter function only takes into account words that are 20 times more likely to appear in a spam message than a ham message. This makes the dictionary much smaller and hence speeds up the algorithm.

However, why would a simpler model (less words) give a better performance — not just out of sample (in the validation set), where overfitting may deteriorate the quality of more sophisticated methods, but also in sample? The reason is the conditional independence assumption of Naïve Bayes. With more words in our dictionary, the number of correlated words is larger and therefore the violation of our theoretical assumption stronger. It appears that 'train2' mediates this effect.

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)?

We can achieve this by changing the following line in the 'predict' function

```
if (posteriors[0] > 0.5):
        return ['ham', posteriors[0]]
```

to something like

```
if (posteriors[0] > 0.4):
        return ['ham', posteriors[0]]
```

There does not seem to be any difference. How about:

```
if (posteriors[0] > 0.001):
        return ['ham', posteriors[0]]
```

Using 'train2', we obtain the following confusion matrix for the validation set:

	ham	**spam**
predict ham	859	53
predict spam	1	87

with an overall performance of 0.946. With these settings we lose only one email in our validation set.

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

The probabilities of missing values are marginalized to be equal to 1. The posteriors formula now reads:

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

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'.

The previous equation essentially tells us to leave the posteriors unchanged, whenever we encounter a word in a vocabulary that is listed as "censored". We can create two new functions in the provided Naïve Bayes classifier:

```
def predict_censored (self, message, censored_words):
        posteriors = np.copy (self.priors)
        for i, w in enumerate (self.words):
                if w in censored_words: #do nothing
                        continue
```

```
                elif w in message.lower():
                posteriors *= self.likelihoods[:,i]
                else:
                        posteriors *= np.ones (2) - self.likelihoods[:,i]
                posteriors = posteriors / np.linalg.norm (posteriors) #
                   normalise
        if posteriors[0] > 0.5:
                return ['ham', posteriors[0]]
        return ['spam', posteriors[1]]

def score_censored (self, messages, labels, censored_words):
        confusion = np.zeros(4).reshape (2,2)
        for m, l in zip (messages, labels):
                if self.predict_censored(m,censored_words)[0] == 'ham' and l
                   == 'ham':
                        confusion[0,0] += 1
                elif self.predict_censored(m,censored_words)[0] == 'ham' and l
                    == 'spam':
                        confusion[0,1] += 1
                elif self.predict_censored(m,censored_words)[0] == 'spam' and
                   l == 'ham':
                        confusion[1,0] += 1
                elif self.predict_censored(m,censored_words)[0] == 'spam' and
                   l == 'spam':
                        confusion[1,1] += 1
        return (confusion[0,0] + confusion[1,1]) / float (confusion.sum()),
           confusion
```

Using 'train', we obtain the following confusion matrix:

	ham	**spam**
predict ham	1098	29
predict spam	12	146

with an overall performance of 0.968 on test1. Using 'train2', we obtain the following confusion matrix:

	ham	**spam**
predict ham	1106	29
predict spam	4	146

with an overall performance of 0.974 on test1.