Inference and Representations
Problem Set 1

Zhuoru Lin
zlin@nyu.edu

1 Hidden Markov Model

a

Since we know $X_1 = Happy$, $p(X1 = Happy) = 1$. Therefore:

$$\begin{aligned}
p(X_2 = Happy) &= p(X_2 = Happy, X_1 = Happy) + p(X_2 = Happy, X_1 = Angry) \\
&= p(X_2 = Happy \mid X_1 = Happy)p(X_1 = Happy) + \\
&p(X_2 = Happy \mid X_1 = Angry)p(X_1 = Angry) \\
&= p(X_2 = Happy \mid X_1 = happy) \\
&= 0.9
\end{aligned}$$

b

$$\begin{aligned}
p(Y_2 = frawn) &= p(Y_2 = frawn \mid X_2 = Happy)p(X_2 = Happy) + p(Y_2 = frawn \mid X_2 = Angry)p(X_2 = Angry) \\
&= 0.1 \times 0.9 + 0.6 \times 0.1 \\
&= 0.15
\end{aligned}$$

c

$$\begin{aligned}
p(X_2 = Happy \mid Y_2 = frawn) &= \frac{p(X_2 = Happy, Y_2 = frawn)}{p(Y_2 = frawn)} \\
&= \frac{p(Y_2 = frawn \mid X_2 = Happy)p(X_2 = Happy)}{p(Y_2 = frawn)} \\
&= \frac{0.1 * 0.9}{0.15} \\
&= 0.6
\end{aligned}$$

d

We know that $p(X_1 = Happy) = 1$. It can be shown by induction that:

$$p(X_t = Happy) = a^{t-1}p(X_1 = Happy) + b(a^{t-2} + a^{t-1} + ... + a + 1). \tag{1}$$

where $a = p(X_t = Happy|X_{t-2} = Happy) - p(X_t = Happy|X_{t-2} = Angry) = 0.8$ and $b = p(X_t = Happy|X_{t-2} = Angry) = 0.1$ is true for $t = 2, ..n$. The proving steps of induction is shown at the end of this question. Approximate $a^{79} \approx 0$. Equation (1) becomes:

$$\begin{aligned}
p(X_t = Happy) &= b(a^{t-2} + a^{t-1} + ... + a + 1) \\
&= b\frac{a^{79} - 1}{a - 1} \\
&\approx \frac{b}{1 - a} \\
&= 0.5
\end{aligned}$$

Then we get:

$$\begin{aligned}
p(Y_80 = yell) &= p(Y_{80} = yell \mid X_{80} = Happy) + p(Y_{80} = yell \mid X_{80} = Angry) \\
&= 0.1 \times 0.5 + 0.2 \times 0.5 \\
&= 0.15
\end{aligned}$$

Proof of induction step

Base case: when $t = 2$, $p(X_2 = Happy) = 0.8 * p(X_1 = Happy) + 0.1 = 0.9$
Induction Step:
Suppose (1) is true for $t = 2, 3, 4, ..., n$:

$$\begin{aligned}
&p(X_{n+1} = Happy) = \\
&p(X_{t+1} = Happy|X_t = Happy)p(X_t = Happy) + p(X_{t+1} = Happy|X_t = Angry)(1 - p(X_t = Happy)) \\
&= (p(X_{t+1} = Happy|X_t = Happy) - p(X_{t+1} = Happy|X_t = Angry))p(X_t = Happy) \\
&+ p(X_{t+1} = Happy|X_t = Angry) \\
&= ap(X_t = Happy) + b \\
&= a(a^{t-1}p(X_1 = Happy) + b(a^{t-2} + a^{t-1} + ... + a + 1)) + b \\
&= a^t p(X_1 = Happy) + b(a^t + a^{t-1} ... + a + 1)
\end{aligned}$$

(e)

Let x_t, y_t be the realization of X_t and Y_t. $x_1 = Happy$ and $y_1 = y_2 = ... = y_5 = frown$ Since:

$$\begin{aligned}
p(x_1, x_2, ..., x_5|y_1, y_2, ..., y_5) &= \frac{p(x_1, x_2, .., x_5, y_1, y_2, ..., y_5)}{p(y_1, y_2, ..., y_5)} \qquad (2) \\
&= \frac{\prod_{v \in \text{nodes}} p(v \mid \pi(v))}{p(y_1, y_2, ..., y_5)} \qquad (3)
\end{aligned}$$

$p(y_1, y_2, ..., y_5)$ is a constant. Hence: argmax $p(x_1, x_2, ..., x_5|y_1, y_2, ..., y_5) =$ argmax $p(x_1, x_2, .., x_5, y_1, y_2, ..., y_5)$.
For HMM :

$$\prod_{v \in \text{nodes}} p(v \mid \pi(v)) = p(x_1)p(y_1|x_1) \prod_{t=2}^{5} p(x_t|x_t - 1)p(y_t|x_t) \tag{4}$$

$$= p(x_1, x_2, x_3, y_1, y_2, y_3) \prod_{t=4}^{5} p(x_t|x_t - 1)p(y_t|x_t). \tag{5}$$

It can be calculated (Calculation omitted for conciseness) that argmax $p(x_1, x_2, x_3, y_1, y_2, y_3)$ is $x_1 = Happy$, $x_2 = Angry$ and $x_3 = Angry$.

We also have $p(x_t = Angry|x_{t-1} = Angry)p(y_t = frown|x_t = Angry) = 0.9 \times 0.6 = 0.54$. This is larger than Any of $p(x_t = Happy|x_{t-1} = Angry)p(y_t = frown|x_t = Happy) = 0.1 \times 0.1 = 0.01$, $p(x_t = Happy|x_{t-1} = Happy)p(y_t = frown|x_t = Happy) = 0.9 \times 0.1 = 0.09$ or $p(x_t = Angry|x_{t-1} = Happy)p(y_t = frown|x_t = Angry) = 0.1 \times 0.6 = 0.06$. Start with $X_3 = Angry$, if x_1, x_2, x_3 are MAE, $x_4 = Angry$ must be MAE too.

Hence we can keep obtain maximum likelihood by letting $x_t = Angry$ for $t > 3$.

Therefore the MAE estimations are:
$x_1 = Happy, x_2 = x_3 = x_4 = x_5 = Angry.$

2 Bayesian Networks must be acyclic

To show that f **MAY** no longer define proper probability distribution. We show a counter example as below:

Consider three binary random variable X_1, X_2 and X_3 which form a directed circle. We can imagine a probability distributions that is only possible when $X_1 = X_2 = X_3$. In this case $p_{X_i \mid X_j}(a, b) = 1$ when $a = b$ and $p_{X_i \mid X_j}(a, b) = 0$ when $a \neq b$. Let $f_{x_i}(x_i|x_j) = p_{X_i \mid X_j}(a, b)$ for $x_i = a$ and $x_j = b$. One can verify that this is totally valid by our definition that $\sum_{x \in Vals(X_v)} f_v(x_v|_{pa(v)}) = 1$ because a children is either same or different from its parent.

Now we must have:

$$\begin{aligned} f(0,0,0) &= f_{x_1}(x_1|x_3)f_{x_2}(x_2|x_1)f_{x_3}(x_3|x_2) \\ &= 1 \times 1 \times 1 \\ &= 1 \end{aligned}$$

and

$$\begin{aligned} f(1,1,1) &= f_{x_1}(x_1|x_3)f_{x_2}(x_2|x_1)f_{x_3}(x_3|x_2) \\ &= 1 \times 1 \times 1 \\ &= 1 \end{aligned}$$

This shows that $\sum_{x_1,x_2,x_3} f(x1, x2, x3) > 1$

3 D-separation

(a)

Using Bayes Ball algorithm without shading any random variables, d-separation infers marginal independence.
Then we can get $X_i \perp X_j$ for all (i,j) in $(1,2),(1,3),(1,5),(1,7),(1,8),$
$(1,9),(1,10),(2,7),(2,8),(3,7),(3,8),(4,8),(6,7),(6,8),(7,8),(7,10),(8,10)$.

(b)

According to Bayes Ball algorithm, X_3, X_5, X_7, X_8, X_10 are d-separated. Therefore $A = 3,5,7,8,10$.

4 X,Y,Z

By writing out all $p(x,y,z)$ one can shows that $Pr(x,y,z)$ correspond to $p_X(x) = p_Y(y) = p_Z(z) = \frac{1}{2}$ and $p_{X,Y}(x,y) = p_{X,Z}(x,z) = p_{Y_Z}(y,z) = \frac{1}{4}$.

This implies $p_{X,Y} = p_X p_Y$, $p_{X,Z} = p_X p_Z$ and $p_{Y,Z} = p_Y p_Z$. Therefore X, Y and Z are mutual independent.

Suppose there exists a directed acyclic graph G such that $I_{d-sep}(G) = I(Pr)$. By $X \perp Y$, G must not contain edge between X and Y since X, Y, Z must be a V-structure. However by $X \perp Z$, X and G must have a shared children in G. This contradicts with the fact that there missing edge between X and Y which is inferred by $X \perp Y$.

p5

September 18, 2017

In [168]:
```
# import package
import numpy as np
from collections import OrderedDict
from collections import defaultdict
from collections import Counter
import matplotlib.pyplot as plt
%matplotlib inline
```

In [169]:
```
# Load data
train_lines = open('./data/train.txt','r').readlines()
test_lines = open('./data/test','r').readlines()
```

1 (a) Data preprocessing

In [170]:
```
# Preprocess data
# Tranfer line from raw txt line to default dict containing
# id: id number is_spam: 1 represent spam word_count: word count in docu
def preprocess(lines):
    output_list = []
    for line in lines:
        doc_dict = {}
        splitted = line.split()
        doc_dict['id'] = splitted[0]
        doc_dict['is_spam'] = splitted[1]=='spam'
        doc_dict['word_count'] = OrderedDict(zip(splitted[2:][::2], np.ar
        output_list.append(doc_dict)
    return output_list
```

In [171]:
```
# Get training and test documents
train_docs = preprocess(train_lines)
test_docs = preprocess(test_lines)
```

In [172]:
```
# Filter to get spam and ham in training sample
train_spam_docs = list(filter(lambda x: x['is_spam'], train_docs))
train_ham_docs = list(filter(lambda x: not x['is_spam'], train_docs))
```

2 (b) What is p(spam) in training data

In [173]:
```
p_spam_train = len(train_spam_docs)/len(train_docs)
print('Probability of spam: %.4f' % p_spam_train)
```

```
Probability of spam: 0.5737
```

In [174]:
```
print('Probability of ham: %.4f' % (1-p_spam_train))
```

```
Probability of ham: 0.4263
```

3 (c) Determine $p(w_i|spam)$

Get vocabulary counts that in spam and all training documents

In [175]:
```
# Function to get a vocabulary dict with key=word value=count of word in
def get_vocabulary_count(doc_list):
    output_vocabulary_dict = defaultdict(lambda :0)
    for doc_dict in doc_list:
        for word, word_count in doc_dict['word_count'].items():
            output_vocabulary_dict[word]+=word_count
    return output_vocabulary_dict
```

In [176]:
```
# Get word count in all trainign documents and spam training documents
train_vocab_count = get_vocabulary_count(train_docs)
train_spam_vocab_count = get_vocabulary_count(train_spam_docs)
train_ham_vocab_count = get_vocabulary_count(train_ham_docs)
```

Apply m-estimate and get $p(wi|spam)$ or $p(wi|ham)$

In [189]:
```
# Apply m-estimate and get p(wi|spam) or p(wi|ham)
def get_p_wi_spam(w, train_vocab_count, subset_vocab_count, m_multiplier=
    output_dict = defaultdict()
    n = np.sum(list(subset_vocab_count.values()))
    vocab_sum = np.sum(list(train_vocab_count.values()))
    #vocab_sum = len(train_vocab_count)
    p = 1.0/vocab_sum
    m = m_multiplier*vocab_sum
    for w_i in w:
        n_c = subset_vocab_count[w_i]
        output_dict[w_i]=(n_c + m*p)/(n+m)
    return output_dict
```

In [190]:
```
words = [key for key in train_vocab_count.keys()]
p_w_given_spam = get_p_wi_spam(words, train_vocab_count, train_spam_vocab
p_w_given_ham = get_p_wi_spam(words, train_vocab_count, train_ham_vocab_c
```

```
In [191]: print('The top 5 most likely word in spam are:\n\n%s' %
                '\n'.join([word for word, prob in Counter(p_w_given_spam).most_comm
```

```
The top 5 most likely word in spam are:

enron
a
corp
the
to
```

```
In [192]: print('The top 5 most likely word in ham are:\n\n%s' %
                '\n'.join([word for word, prob in Counter(p_w_given_ham).most_commo
```

```
The top 5 most likely word in ham are:

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
enron
the
to
a
```

4 (d) Classifier and accuracy

A classifier that make predicison by comparing the $log(p(w, spam))$ and $log(p(w, ham))$

```
In [193]: # Code of Naive Bayes Classifier
          def classify(test_docs, train_docs, m_multiplier = 1):
              train_spam_docs = list(filter(lambda x: x['is_spam'], train_docs))
              train_ham_docs = list(filter(lambda x: not x['is_spam'], train_docs))

              p_spam_train = len(train_spam_docs)/len(train_docs)
              p_ham_train = 1-p_spam_train

              train_vocab_count = get_vocabulary_count(train_docs)
              train_spam_vocab_count = get_vocabulary_count(train_spam_docs)
              train_ham_vocab_count = get_vocabulary_count(train_ham_docs)

              preds = []
              for doc in test_docs:
                  word_count_dict = doc['word_count']
                  words = list(word_count_dict.keys())

                  p_w_given_spam = get_p_wi_spam(words, train_vocab_count, train_sp
                  p_w_given_ham = get_p_wi_spam(words, train_vocab_count, train_ham
```

```
                log_likelihood_spam = np.log(p_spam_train)
                log_likelihood_ham = np.log(p_ham_train)
                for word, word_count in word_count_dict.items():
                    log_likelihood_spam += np.log(p_w_given_spam[word])*word_cour
                    log_likelihood_ham += np.log(p_w_given_ham[word])*word_count

                preds.append(log_likelihood_spam>log_likelihood_ham)
            return preds
```

```
In [194]: # predictions
          preds = classify(test_docs, train_docs,1)
```

```
In [195]: # Evaluation of predictions results
          def evaluate_predictions(preds, test_docs):
              true_labels = np.array([doc['is_spam'] for doc in test_docs])
              return (preds==true_labels).sum()/len(test_docs)
```

```
In [196]: print('Accuracy: %.3f'% evaluate_predictions(preds, test_docs))
```

```
Accuracy: 0.914
```

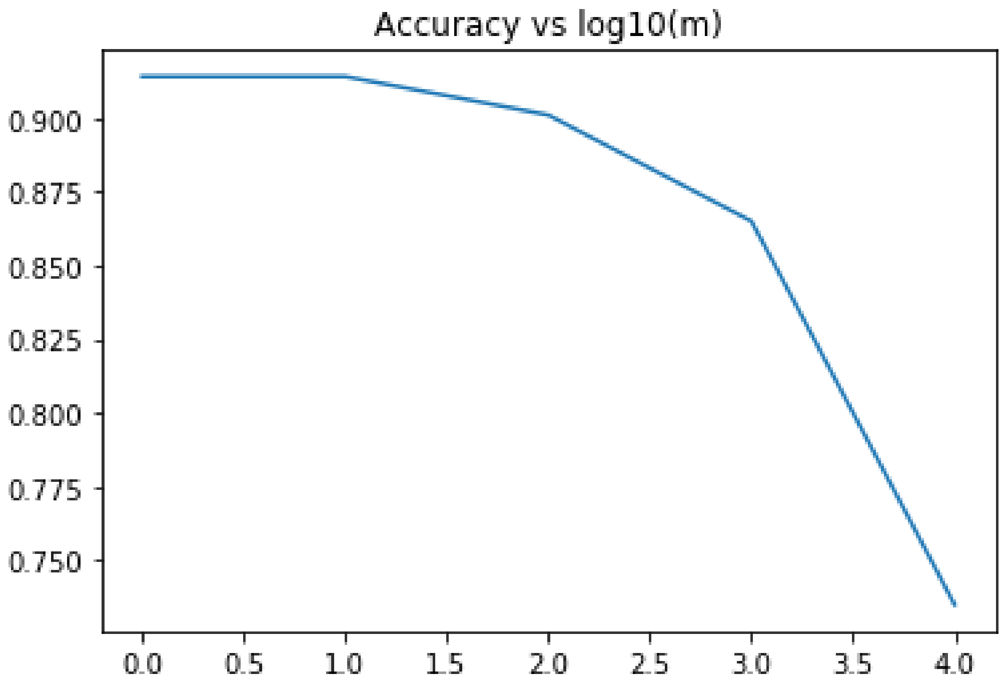
```
In [197]: accuracy = []
          m_multiplier_list = [1,10,100,1000,10000]
          for m_multiplier in m_multiplier_list:
              preds = classify(test_docs, train_docs, m_multiplier)
              accuracy.append(evaluate_predictions(preds,test_docs))
```

5 (e) Vary m parameter

Plot are shown below

```
In [198]: plt.plot(np.log10(m_multiplier_list), accuracy)
          plt.title('Accuracy vs log10(m)')
```

```
Out[198]: <matplotlib.text.Text at 0x10edb7dd8>
```



5.1 What does m assume?

m represents the size of the imaginary training data in which word distribution follow practitioner defined priors $p(w_i|spam)$. The larger the m is, the more weight you put in the prior. More 'counts' are assigned to unobserved word relative to observed word.

A small m assume that training samples are very good representations of global samples.

A large m assume that training samples are less representative and we believe the prior distribution more.

In our case the a large m harms test accuracy.

6 (f) What to do if I am a spammer?

If the spam detector is a naive bayes classifer. We should avoid using spam-common words. And we should make our spam email long and make the percentage of common and ham-common words higher. In a word, we should write a spam that contains spam message, while seems like ham in bag of words.