Dashboard / My courses / COSC367-2020S2 / Weekly quizzes / 9. Machine learning with naive Bayes nets

Started on	Friday, 2 October 2020, 2:58 AM
State	Finished
Completed on	Friday, 2 October 2020, 5:06 PM
Time taken	14 hours 7 mins
Marks	5.00/5.00
Grade	**100.00** out of 100.00

Information

Representation of belief networks in Python

A belief (or Bayesian) network is represented by a dictionary. The keys are the names of variables. The values are dictionaries themselves. The second level dictionaries have two keys: `'Parents'` whose value is a list of the names of the variables that are the parents of the current variable, and `'CPT'` whose value is a dictionary again. The keys of the third level dictionaries are tuples of Booleans which correspond to possible assignments of values to the parents of the current node (in the order they are listed) and the values are real numbers representing the probability of the current node being true given the specified assignment to the parents.

Notes

- Variable names are case sensitive.
- If a node does not have any parents, the value of `'Parents'` must be an empty list and the only key of the third level dictionary is the empty tuple.
- For simplicity, we assume that all the variables are Boolean.

Example

The following is the representation of the *alarm network* presented in the lecture notes.

```
network = {
    'Burglary': {
        'Parents': [],
        'CPT': {
            (): 0.001,
         }
    },

    'Earthquake': {
        'Parents': [],
        'CPT': {
            (): 0.002,
        }
    },

    'Alarm': {
        'Parents': ['Burglary','Earthquake'],
        'CPT': {
            (True,True): 0.95,
            (True,False): 0.94,
            (False,True): 0.29,
            (False,False): 0.001,
        }
    },

    'John': {
        'Parents': ['Alarm'],
        'CPT': {
            (True,): 0.9,
            (False,): 0.05,
        }
    },

    'Mary': {
        'Parents': ['Alarm'],
        'CPT': {
            (True,): 0.7,
            (False,): 0.01,
        }
    },
}
```

Question **1**

Correct

Mark 1.00 out of 1.00

Suppose we want to predict the value of variable `Y` based on the values of variables `X1`, `X2`, and `X3`. Assuming that we want to use a Naive Bayes model for this purpose, create a belief net for the model called `network`. The probabilities must be learnt by using the dataset given below. Use Laplacian smoothing with a pseudocount of 2.

X1	X2	X3	Y
T	T	F	F
T	F	F	F
T	T	F	F
T	F	F	T
F	F	F	T
F	T	F	T
F	F	F	T

Notes

- Node names are case sensitive.
- Since we are using Python syntax, you can use fraction expressions if you wish. For example you can use `3/4` or even `(2+1)/(2+1+0+1)` which will be evaluated at runtime.

For example:

Test	Result

<pre>from student_answer import network
from numbers import Number
 	
# Checking the overall type-correctness of the network 	
# without checking anything question-specific 	
 	
assert type(network) is dict
for node_name, node_info in network.items():
    assert type(node_name) is str
    assert type(node_info) is dict
    assert set(node_info.keys()) == {'Parents', 'CPT'}
    assert type(node_info['Parents']) is list
    assert all(type(s) is str for s in node_info['Parents'])
    for assignment, prob in node_info['CPT'].items():
        assert type(assignment) is tuple
        assert isinstance(prob, Number)
 	
print("OK")</pre>
	OK

Answer: (penalty regime: 0, 10, ... %)

```
network = {
    'Y': {
        'Parents': [],
        'CPT': {
            (): 6/11,
        }
    },
    'X1': {
        'Parents': ['Y'],
        'CPT': {
            (True,): 3/8,
            (False,): 5/7,
        }
    },
    'X2': {
        'Parents': ['Y'],
        'CPT': {
            (True,): 3/8,
            (False,): 4/7,
        }
    },
    'X3': {
        'Parents': ['Y'],
        'CPT': {
            (True,): 2/8,
            (False,):2/7,
        }
    }

}

```

	Test	Expected	Got	

✔	`from student_answer import network`
	`from numbers import Number`
	
	`# Checking the overall type-correctness of the network`
	`# without checking anything question-specific`
	
	`assert type(network) is dict`
	`for node_name, node_info in network.items():`
	`    assert type(node_name) is str`
	`    assert type(node_info) is dict`
	`    assert set(node_info.keys()) == {'Parents', 'CPT'}`
	`    assert type(node_info['Parents']) is list`
	`    assert all(type(s) is str for s in node_info['Parents'])`
	`    for assignment, prob in node_info['CPT'].items():`
	`        assert type(assignment) is tuple`
	`        assert isinstance(prob, Number)`
	
	`print("OK")`
		OK	OK	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Information

Representation of naïve Bayes models

Naïve Bayes models can be represented with belief networks. However, since they all have a very simple topology (a directed tree of depth one where the root is the class variable and the leaves are the input features), we can use a more compact representation that is only concerned with the values of CPTs.

We assume that all the variables in a naïve Bayes network are binary. For a network with *n* binary input features X[1] to X[n], we represent the conditional probability tables (CPTs) that are required in the network, with the following two objects:

- `prior`: a real number representing *p(Class=true)*. The probability *p(Class=false)* can be obtained by `1 - prior`.
- `likelihood`: a tuple of length *n* where each element is a pair of real numbers such that `likelihood[i][False]` is *p(X[i]=true|C=false)* and `likelihood[i][True]` is *p(X[i]=true|C=true)*. That is, likelihood contains the *2*n* CPTs that are required at leaf nodes.

Note: indexing sequences with booleans is not generally a good programming practice. We used them here instead of 0 and 1 to bring the appearance of the code closer to the equations we use.

Question **2**

Correct

Mark 1.00 out of 1.00

Write a function `posterior(prior, likelihood, observation)` that returns the posterior probability of the class variable being true, given the observation; that is, it returns *p(Class=true|observation)*. The argument `observation` is a tuple of *n* Booleans such that `observation[i]` is the observed value (True or False) for the input feature `X[i]`. The arguments `prior` and `likelihood` are as described above.

Notes

1. Example 8.35 in the textbook is relevant.
2. The model used in the test cases is according to this network. You can download and explore the model in the belief network applet.

For example:

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td><pre>
from student_answer import posterior

prior = 0.05
likelihood = ((0.001, 0.3),(0.05,0.9),(0.7,0.99))

observation = (True, True, True)

class_posterior_true = posterior(prior, likelihood, observation)
print("P(C=False|observation) is approximately {:.5f}"
      .format(1 - class_posterior_true))
print("P(C=True |observation) is approximately {:.5f}"
      .format(class_posterior_true))
</pre></td><td><pre>
P(C=False|observation) is approximately 0.00248
P(C=True |observation) is approximately 0.99752
</pre></td></tr>
<tr><td><pre>
from student_answer import posterior

prior = 0.05
likelihood = ((0.001, 0.3),(0.05,0.9),(0.7,0.99))

observation = (True, False, True)

class_posterior_true = posterior(prior, likelihood, observation)
print("P(C=False|observation) is approximately {:.5f}"
      .format(1 - class_posterior_true))
print("P(C=True |observation) is approximately {:.5f}"
      .format(class_posterior_true))
</pre></td><td><pre>
P(C=False|observation) is approximately 0.29845
P(C=True |observation) is approximately 0.70155
</pre></td></tr>
<tr><td><pre>
from student_answer import posterior

prior = 0.05
likelihood = ((0.001, 0.3),(0.05,0.9),(0.7,0.99))

observation = (False, False, True)

class_posterior_true = posterior(prior, likelihood, observation)
print("P(C=False|observation) is approximately {:.5f}"
      .format(1 - class_posterior_true))
print("P(C=True |observation) is approximately {:.5f}"
      .format(class_posterior_true))
</pre></td><td><pre>
P(C=False|observation) is approximately 0.99454
P(C=True |observation) is approximately 0.00546
</pre></td></tr>
<tr><td><pre>
from student_answer import posterior

prior = 0.05
likelihood = ((0.001, 0.3),(0.05,0.9),(0.7,0.99))

observation = (False, False, False)

class_posterior_true = posterior(prior, likelihood, observation)
print("P(C=False|observation) is approximately {:.5f}"
      .format(1 - class_posterior_true))
print("P(C=True |observation) is approximately {:.5f}"
      .format(class_posterior_true))
</pre></td><td><pre>
P(C=False|observation) is approximately 0.99987
P(C=True |observation) is approximately 0.00013
</pre></td></tr>
</table>

Answer: (penalty regime: 0, 15, ... %)

```
def posterior(prior, likelihood, observation):
    '''d'''

    true = prior
    false = 1 - prior
    for i in range(len(observation)):
        #print(likelihood[i][False])
```

```
        if observation[i]:
            false *= likelihood[i][False]
            true *= likelihood[i][True]
        else:
            true *= (1 - likelihood[i][True])
            false *= (1 - likelihood[i][False])
    result = true / (true + false)
    return result
```

	Test	Expected	Got	

✔	from student_answer import posterior
	
	prior = 0.05
	likelihood = ((0.001, 0.3),(0.05,0.9),(0.7,0.99))
	
	observation = (True, True, True)
	
	class_posterior_true = posterior(prior, likelihood, observation)
	print("P(C=False|observation) is approximately {:.5f}"
	      .format(1 - class_posterior_true))
	print("P(C=True |observation) is approximately {:.5f}"
	      .format(class_posterior_true))
		P(C=False|observation) is approximately 0.00248
		P(C=True |observation) is approximately 0.99752
			P(C=False|observation) is approximately 0.00248
			P(C=True |observation) is approximately 0.99752
				✔

✔	from student_answer import posterior
	
	prior = 0.05
	likelihood = ((0.001, 0.3),(0.05,0.9),(0.7,0.99))
	
	observation = (True, False, True)
	
	class_posterior_true = posterior(prior, likelihood, observation)
	print("P(C=False|observation) is approximately {:.5f}"
	      .format(1 - class_posterior_true))
	print("P(C=True |observation) is approximately {:.5f}"
	      .format(class_posterior_true))
		P(C=False|observation) is approximately 0.29845
		P(C=True |observation) is approximately 0.70155
			P(C=False|observation) is approximately 0.29845
			P(C=True |observation) is approximately 0.70155
				✔

<table>
<tr><th></th><th>Test</th><th>Expected</th><th>Got</th><th></th></tr>
<tr><td>✔</td><td><pre>from student_answer import
posterior

prior = 0.05
likelihood = ((0.001, 0.3),
(0.05,0.9),(0.7,0.99))

observation = (False, False, True)

class_posterior_true =
posterior(prior, likelihood,
observation)
print("P(C=False|observation) is
approximately {:.5f}"
      .format(1 -
class_posterior_true))
print("P(C=True |observation) is
approximately {:.5f}"

.format(class_posterior_true))</pre></td><td><pre>P(C=False|observation) is
approximately 0.99454
P(C=True |observation) is
approximately 0.00546</pre></td><td><pre>P(C=False|observation) is
approximately 0.99454
P(C=True |observation) is
approximately 0.00546</pre></td><td>✔</td></tr>
<tr><td>✔</td><td><pre>from student_answer import
posterior

prior = 0.05
likelihood = ((0.001, 0.3),
(0.05,0.9),(0.7,0.99))

observation = (False, False,
False)

class_posterior_true =
posterior(prior, likelihood,
observation)
print("P(C=False|observation) is
approximately {:.5f}"
      .format(1 -
class_posterior_true))
print("P(C=True |observation) is
approximately {:.5f}"

.format(class_posterior_true))</pre></td><td><pre>P(C=False|observation) is
approximately 0.99987
P(C=True |observation) is
approximately 0.00013</pre></td><td><pre>P(C=False|observation) is
approximately 0.99987
P(C=True |observation) is
approximately 0.00013</pre></td><td>✔</td></tr>
</table>

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Information

Bayesian Spam Filter

In the next three questions, you are asked to develop the learning and classification components of a naive Bayes classifier (a spam filter).

The file spam-labelled.csv describes 200 emails, labelled as spam or non-spam by human users. Each email is specified by 12 binary attributes, indicating the presence of features such as “Lottery”, “MILLION DOLLARS”, significant amounts of text in CAPS, an invalid reply-to address, and so on.

The layout of the data is that each row is an example (one email), and columns correspond to attributes (features), which are binary. There are 12 input features (X1 to X12). The last (right-most) column is the class label where 1 means the example is spam (positive) and 0 means non-spam (negative).

Note that there are 2^12 =4096 possible input patterns; in other words, the data set only contains a small proportion of all possible input patterns. This is a common scenario in machine learning.

The file has Unix-like line breaks. Windows users need to open the file in a proper text editor that supports different line endings. You do not need a spreadsheet to open the file.

In Python, the csv module may come in handy. You can load the content of the file as a list of tuples using the following:

```
with open(file_name) as in_file:
        training_examples = [tuple(row) for row in csv.reader(in_file)]
```

In the next three questions, the above file is available on the server (in the current directory). Therefore a statement like the one above, would read the file. Your function will be tested on the same file format and header but the content (the examples) may vary. Make sure your solution works with the original copy of the file, not your own format.

Question **3**

Correct

Mark 1.00 out of 1.00

Write a function `learn_prior(file_name, pseudo_count=0)` that takes the file name of the training set and an optional pseudo-count parameter and returns a real number that is the prior probability of spam being true. The parameter `pseudo_count` is a non-negative integer and it will be the same for all the attributes and all the values.

Notes

- Pseudo-counts are described in the lecture notes and section 7.2.3 of the textbook.
- Although you see high values of pseudo-count in some test cases, in practice small values are mostly used.

For example:

Test	Result

`from student_answer import learn_prior`
 	
`prior = learn_prior("spam-labelled.csv")`
`print("Prior probability of spam is {:.5f}.".format(prior))`
	`Prior probability of spam is 0.25500.`

`from student_answer import learn_prior`
 	
`prior = learn_prior("spam-labelled.csv")`
`print("Prior probability of not spam is {:.5f}.".format(1 - prior))`
	`Prior probability of not spam is 0.74500.`

`from student_answer import learn_prior`
 	
`prior = learn_prior("spam-labelled.csv", pseudo_count = 1)`
`print(format(prior, ".5f"))`
	`0.25743`

`from student_answer import learn_prior`
 	
`prior = learn_prior("spam-labelled.csv", pseudo_count = 2)`
`print(format(prior, ".5f"))`
	`0.25980`

`from student_answer import learn_prior`
 	
`prior = learn_prior("spam-labelled.csv", pseudo_count = 10)`
`print(format(prior, ".5f"))`
	`0.27727`

`from student_answer import learn_prior`
 	
`prior = learn_prior("spam-labelled.csv", pseudo_count = 100)`
`print(format(prior, ".5f"))`
	`0.37750`

`from student_answer import learn_prior`
 	
`prior = learn_prior("spam-labelled.csv", pseudo_count = 1000)`
`print(format(prior, ".5f"))`
	`0.47773`

Answer: (penalty regime: 0, 10, ... %)

```
import csv
def learn_prior(file_name, pseudo_count=0):
    '''d'''
    with open(file_name) as in_file:
        training_examples = [tuple(row) for row in csv.reader(in_file)]
    is_spam = 0
    not_spam = 0
    for item in training_examples:
        if item[-1] == '1':
            is_spam += 1
        if item[-1] == '0':
            not_spam += 1
    count_true = is_spam + pseudo_count
    count_false = not_spam + pseudo_count
    total = count_true + count_false
    return count_true/total
```

	Test	Expected	Got	

	Test	Expected	Got	
✔	from student_answer import learn_prior prior = learn_prior("spam-labelled.csv") print("Prior probability of spam is {:.5f}.".format(prior))	Prior probability of spam is 0.25500.	Prior probability of spam is 0.25500.	✔
✔	from student_answer import learn_prior prior = learn_prior("spam-labelled.csv") print("Prior probability of not spam is {:.5f}.".format(1 - prior))	Prior probability of not spam is 0.74500.	Prior probability of not spam is 0.74500.	✔
✔	from student_answer import learn_prior prior = learn_prior("spam-labelled.csv", pseudo_count = 1) print(format(prior, ".5f"))	0.25743	0.25743	✔
✔	from student_answer import learn_prior prior = learn_prior("spam-labelled.csv", pseudo_count = 10) print(format(prior, ".5f"))	0.27727	0.27727	✔
✔	from student_answer import learn_prior prior = learn_prior("spam-labelled.csv", pseudo_count = 100) print(format(prior, ".5f"))	0.37750	0.37750	✔
✔	from student_answer import learn_prior prior = learn_prior("spam-labelled.csv", pseudo_count = 1000) print(format(prior, ".5f"))	0.47773	0.47773	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 1.00 out of 1.00

Write a function `learn_likelihood(file_name, pseudo_count=0)` that takes the file name of a training set (for the spam detection problem) and an optional pseudo-count parameter and returns a sequence of pairs of likelihood probabilities. As described in the representation of likelihood, the length of the returned sequence (list or tuple) must be 12. Each element in the sequence is a pair (tuple) of real numbers such that `likelihood[i][False]` is *P(X[i]=true|Spam=false)* and `likelihood[i][True]` is *P(X[i]=true|Spam=true)*.

For example:

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td>from student_answer import learn_likelihood

likelihood = learn_likelihood("spam-labelled.csv")
print(len(likelihood))
print([len(item) for item in likelihood])</td><td>12
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]</td></tr>
<tr><td>from student_answer import learn_likelihood

likelihood = learn_likelihood("spam-labelled.csv")

print("P(X1=True | Spam=False) = {:.5f}".format(likelihood[0][False]))
print("P(X1=False| Spam=False) = {:.5f}".format(1 - likelihood[0][False]))
print("P(X1=True | Spam=True) = {:.5f}".format(likelihood[0][True]))
print("P(X1=False| Spam=True) = {:.5f}".format(1 - likelihood[0][True]))</td><td>P(X1=True | Spam=False) = 0.35570
P(X1=False| Spam=False) = 0.64430
P(X1=True | Spam=True) = 0.66667
P(X1=False| Spam=True) = 0.33333</td></tr>
<tr><td>from student_answer import learn_likelihood

likelihood = learn_likelihood("spam-labelled.csv", pseudo_count=1)

print("With Laplacian smoothing:")
print("P(X1=True | Spam=False) = {:.5f}".format(likelihood[0][False]))
print("P(X1=False| Spam=False) = {:.5f}".format(1 - likelihood[0][False]))
print("P(X1=True | Spam=True) = {:.5f}".format(likelihood[0][True]))
print("P(X1=False| Spam=True) = {:.5f}".format(1 - likelihood[0][True]))</td><td>With Laplacian smoothing:
P(X1=True | Spam=False) = 0.35762
P(X1=False| Spam=False) = 0.64238
P(X1=True | Spam=True) = 0.66038
P(X1=False| Spam=True) = 0.33962</td></tr>
</table>

Answer: (penalty regime: 0, 15, ... %)

```
import csv
def learn_likelihood(file_name, pseudo_count=0):
    '''d'''
    with open(file_name) as in_file:
        training_examples = [tuple(row) for row in csv.reader(in_file)][1:]
    likelihoods = []
    for i in range(12):
        likelihoods.append([pseudo_count, pseudo_count])
    spam_true = 0
    spam_false = 0
    prior = 0
    for row in training_examples:
        spam = int(row[-1])
        if spam == 1:
            spam_true += 1
        if spam == 0:
            spam_false += 1
        for i in range(12):
            likelihoods[i][spam] += int(row[i])
    #print(likelihoods)
    #print(spam_true,'spam true')
    #print(spam_false,'spam false')
    count_true = spam_true + pseudo_count
    #print(count_true, 'count true')
    count_false = spam_false + pseudo_count
    #print(count_false, 'countfalse')

    for i in range(len(likelihoods)):
        #likelihoods[i][True] += pseudo_count
        likelihoods[i][True] /= count_true + pseudo_count
        #likelihoods[i][False] += pseudo_count
        likelihoods[i][False] /= count_false + pseudo_count
    return likelihoods
```

	Test	Expected	Got	
✔	from student_answer import learn_likelihood likelihood = learn_likelihood("spam-labelled.csv") print(len(likelihood)) print([len(item) for item in likelihood])	12 [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]	12 [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]	✔
✔	from student_answer import learn_likelihood likelihood = learn_likelihood("spam-labelled.csv") print("P(X1=True | Spam=False) = {:.5f}".format(likelihood[0][False])) print("P(X1=False| Spam=False) = {:.5f}".format(1 - likelihood[0][False])) print("P(X1=True | Spam=True) = {:.5f}".format(likelihood[0][True])) print("P(X1=False| Spam=True) = {:.5f}".format(1 - likelihood[0][True]))	P(X1=True | Spam=False) = 0.35570 P(X1=False| Spam=False) = 0.64430 P(X1=True | Spam=True) = 0.66667 P(X1=False| Spam=True) = 0.33333	P(X1=True | Spam=False) = 0.35570 P(X1=False| Spam=False) = 0.64430 P(X1=True | Spam=True) = 0.66667 P(X1=False| Spam=True) = 0.33333	✔
✔	from student_answer import learn_likelihood likelihood = learn_likelihood("spam-labelled.csv", pseudo_count=1) print("With Laplacian smoothing:") print("P(X1=True | Spam=False) = {:.5f}".format(likelihood[0][False])) print("P(X1=False| Spam=False) = {:.5f}".format(1 - likelihood[0][False])) print("P(X1=True | Spam=True) = {:.5f}".format(likelihood[0][True])) print("P(X1=False| Spam=True) = {:.5f}".format(1 - likelihood[0][True]))	With Laplacian smoothing: P(X1=True | Spam=False) = 0.35762 P(X1=False| Spam=False) = 0.64238 P(X1=True | Spam=True) = 0.66038 P(X1=False| Spam=True) = 0.33962	With Laplacian smoothing: P(X1=True | Spam=False) = 0.35762 P(X1=False| Spam=False) = 0.64238 P(X1=True | Spam=True) = 0.66038 P(X1=False| Spam=True) = 0.33962	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **5**

Correct

Mark 1.00 out of 1.00

Write a function `nb_classify(prior, likelihood, input_vector)` that takes the learnt prior and likelihood probabilities and classifies an (unseen) input vector. The input vector will be a tuple of 12 integers (each 0 or 1) corresponding to attributes X1 to X12. The function should return a pair (tuple) where the first element is either `"Spam"` or `"Not Spam"` and the second element is the certainty. The certainty is the (posterior) probability of spam when the instance is classified as spam, or the probability of 'not-spam' otherwise. If spam and 'not spam' are equally likely (i.e. p=0.5) then choose 'not spam'.

This is a very simple function to implement as it only wraps the `posterior` function developed earlier.

Supply the following functions you developed earlier: `learn_prior` and `learn_likelihood`. Also include import statements and any other function that you may be using (e.g. `posterior`).

For example:

<table>
<tr><th>Test</th><th>Result</th></tr>
<tr><td><pre>
from student_answer import learn_prior, learn_likelihood,
nb_classify

prior = learn_prior("spam-labelled.csv")
likelihood = learn_likelihood("spam-labelled.csv")

input_vectors = [
    (1,1,0,0,1,1,0,0,0,0,0,0),
    (0,0,1,1,0,0,1,1,1,0,0,1),
    (1,1,1,1,1,0,1,0,0,0,1,1),
    (1,1,1,1,1,0,1,0,0,1,0,1),
    (0,1,0,0,0,0,1,0,1,0,0,0),
    ]

predictions = [nb_classify(prior, likelihood, vector)
               for vector in input_vectors]

for label, certainty in predictions:
    print("Prediction: {}, Certainty: {:.5f}"
          .format(label, certainty))
</pre></td><td><pre>
Prediction: Not Spam, Certainty:
0.99351
Prediction: Spam, Certainty: 0.57441
Prediction: Spam, Certainty: 0.59337
Prediction: Spam, Certainty: 0.83465
Prediction: Not Spam, Certainty:
0.99140
</pre></td></tr>
<tr><td><pre>
from student_answer import learn_prior, learn_likelihood,
nb_classify

prior = learn_prior("spam-labelled.csv", pseudo_count=1)
likelihood = learn_likelihood("spam-labelled.csv",
pseudo_count=1)

input_vectors = [
    (1,1,0,0,1,1,0,0,0,0,0,0),
    (0,0,1,1,0,0,1,1,1,0,0,1),
    (1,1,1,1,1,0,1,0,0,0,1,1),
    (1,1,1,1,1,0,1,0,0,1,0,1),
    (0,1,0,0,0,0,1,0,1,0,0,0),
    ]

predictions = [nb_classify(prior, likelihood, vector)
               for vector in input_vectors]

for label, certainty in predictions:
    print("Prediction: {}, Certainty: {:.5f}"
          .format(label, certainty))
</pre></td><td><pre>
Prediction: Not Spam, Certainty:
0.99213
Prediction: Spam, Certainty: 0.57759
Prediction: Spam, Certainty: 0.59073
Prediction: Spam, Certainty: 0.83059
Prediction: Not Spam, Certainty:
0.98989
</pre></td></tr>
</table>

Answer: (penalty regime: 0, 15, ... %)

```
import csv

def posterior(prior, likelihood, observation):
    '''d'''

    true = prior
    false = 1 - prior
    for i in range(len(observation)):
        #print(likelihood[i][False])
        if observation[i]:
            false *= likelihood[i][False]
            true *= likelihood[i][True]
        else:
            true *= (1 - likelihood[i][True])
            false *= (1 - likelihood[i][False])
    result = true / (true + false)
    return result

def learn_prior(file_name, pseudo_count=0):
```

```python
    '''d'''
    with open(file_name) as in_file:
        training_examples = [tuple(row) for row in csv.reader(in_file)]
    is_spam = 0
    not_spam = 0
    for item in training_examples:
        if item[-1] == '1':
            is_spam += 1
        if item[-1] == '0':
            not_spam += 1
    count_true = is_spam + pseudo_count
    count_false = not_spam + pseudo_count
    total = count_true + count_false
    return count_true/total

def learn_likelihood(file_name, pseudo_count=0):
    '''d'''
    with open(file_name) as in_file:
        training_examples = [tuple(row) for row in csv.reader(in_file)][1:]
    likelihoods = []
    for i in range(12):
        likelihoods.append([pseudo_count, pseudo_count])
    spam_true = 0
    spam_false = 0
    prior = 0
    for row in training_examples:
        spam = int(row[-1])
        if spam == 1:
            spam_true += 1
        if spam == 0:
            spam_false += 1
        for i in range(12):
            likelihoods[i][spam] += int(row[i])
    #print(likelihoods)
    #print(spam_true,'spam true')
    #print(spam_false,'spam false')
    count_true = spam_true + pseudo_count
    #print(count_true, 'count true')
    count_false = spam_false + pseudo_count
    #print(count_false, 'countfalse')

    for i in range(len(likelihoods)):
        #likelihoods[i][True] += pseudo_count
        likelihoods[i][True] /= count_true + pseudo_count
        #likelihoods[i][False] += pseudo_count
        likelihoods[i][False] /= count_false + pseudo_count
    return likelihoods


def nb_classify(prior, likelihood, input_vector):
    '''s'''
    true = posterior(prior, likelihood, input_vector)
    #print(true, 'true')
    false = 1 - posterior(prior, likelihood, input_vector)
    #print(false, 'false')
    if true > false:
        return ("Spam", true)
    elif true == false:
        return ("Not Spam", false)
    else:
        return ("Not Spam", false)
```

	Test	Expected	Got	

	Test	Expected	Got	

✔	`from student_answer import learn_prior, learn_likelihood, nb_classify`
	
	`prior = learn_prior("spam-labelled.csv")`
	`likelihood = learn_likelihood("spam-labelled.csv")`
	
	`input_vectors = [`
	`    (1,1,0,0,1,1,0,0,0,0,0,0),`
	`    (0,0,1,1,0,0,1,1,1,0,0,1),`
	`    (1,1,1,1,1,0,1,0,0,0,1,1),`
	`    (1,1,1,1,1,0,1,0,0,1,0,1),`
	`    (0,1,0,0,0,0,1,0,1,0,0,0),`
	`    ]`
	
	`predictions = [nb_classify(prior, likelihood, vector)`
	`               for vector in input_vectors]`
	
	`for label, certainty in predictions:`
	`    print("Prediction: {}, Certainty: {:.5f}"`
	`          .format(label, certainty))`
		Prediction: Not Spam, Certainty: 0.99351
		Prediction: Spam, Certainty: 0.57441
		Prediction: Spam, Certainty: 0.59337
		Prediction: Spam, Certainty: 0.83465
		Prediction: Not Spam, Certainty: 0.99140
			Prediction: Not Spam, Certainty: 0.99351
			Prediction: Spam, Certainty: 0.57441
			Prediction: Spam, Certainty: 0.59337
			Prediction: Spam, Certainty: 0.83465
			Prediction: Not Spam, Certainty: 0.99140
				✔

✔	`from student_answer import learn_prior, learn_likelihood, nb_classify`
	
	`prior = learn_prior("spam-labelled.csv", pseudo_count=1)`
	`likelihood = learn_likelihood("spam-labelled.csv", pseudo_count=1)`
	
	`input_vectors = [`
	`    (1,1,0,0,1,1,0,0,0,0,0,0),`
	`    (0,0,1,1,0,0,1,1,1,0,0,1),`
	`    (1,1,1,1,1,0,1,0,0,0,1,1),`
	`    (1,1,1,1,1,0,1,0,0,1,0,1),`
	`    (0,1,0,0,0,0,1,0,1,0,0,0),`
	`    ]`
	
	`predictions = [nb_classify(prior, likelihood, vector)`
	`               for vector in input_vectors]`
	
	`for label, certainty in predictions:`
	`    print("Prediction: {}, Certainty: {:.5f}"`
	`          .format(label, certainty))`
		Prediction: Not Spam, Certainty: 0.99213
		Prediction: Spam, Certainty: 0.57759
		Prediction: Spam, Certainty: 0.59073
		Prediction: Spam, Certainty: 0.83059
		Prediction: Not Spam, Certainty: 0.98989
			Prediction: Not Spam, Certainty: 0.99213
			Prediction: Spam, Certainty: 0.57759
			Prediction: Spam, Certainty: 0.59073
			Prediction: Spam, Certainty: 0.83059
			Prediction: Not Spam, Certainty: 0.98989
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

◄ 8. Belief networks and probabilistic inference

Jump to...

10. Machine learning with kNN and basic neural networks ►