CS 540: Introduction to Artificial Intelligence
Homework Assignment #4

Assigned: 11/11
Due: 11/25 before class

Hand in your homework:

This homework includes a written portion and a programming portion. Please type the written portion and hand in a printed hardcopy in class. All pages should be stapled together, and the first sheet must include a header with: your name, wisc username, class section, HW #, date -- and, if handed in late, how many days late it is. The programming portion will be written in Java. All files needed to run the code (including any support files you've written) should be collected and compressed as as one zip file named <wisc username>_HW4.zip. This file should then be uploaded to the appropriate place on the course Moodle website.

Late Policy:

All assignments are due **at the beginning of class** on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday 11 a.m, and it is handed in between Wednesday 11 a.m. and Thursday 11 a.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late. Written questions and program submission have the same deadline. A total of two (2) free late days may be used throughout the semester without penalty.

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.

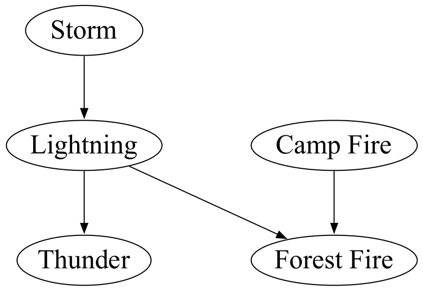
Collaboration Policy:

You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Question 1: Inference in Bayesian Networks
[25] Consider the following Bayesian Network over five boolean-valued random variables



P(Storm=T)	P(Storm=F)
0.11	0.89

P(Camp Fire=T)	P(Camp Fire=F)
0.40	0.60

Storm	P(Lightning=T)	P(Lightning=F)
T	0.83	0.17
F	0.03	0.97

Lightning	P(Thunder=T)	P(Thunder=F)
T	0.99	0.01
F	0.15	0.85

Lightning	Camp Fire	P(Forest Fire=T)	P(Forest Fire=F)
T	T	0.63	0.37
T	F	0.42	0.58
F	T	0.57	0.43
F	F	0.18	0.82

For all of the following questions, to receive full credit, please show your work.

a) [5] What is the probability of hearing thunder?
b) [5] What is the probability of a camp fire given thunder?
c) [5] What is the probability of a storm given camp fire?
d) [5] What is the probability of hearing thunder given no storm?
e) [5] What is the probability of a forest fire given no lightning?

Question 2: Building a Bayes Net and Parameter Learning
[25] You want to build a Bayes Net with the following binary random variables:

C : There is a **C**at in the room
T : A can of **T**una is being opened
M : There is a **M**ouse in the room
H : You **H**ear a "meow"

You know these facts:

$P(T, M) = P(T)\, P(M)$
$P(T, M \mid C) \neq P(T \mid C)\, P(M \mid C)$
$P(H \mid C, M, T) = P(H \mid C)$

a) [15] Using this information, draw the appropriate set of nodes, (directed) edges between nodes and CPTs.

b) [10] Using the following dataset, calculate all necessary CPT values and add them to your Bayes Net. **Do not** perform any smoothing.

```
( T,  M,  C,  H)
( T, ~M,  C, ~H)
(~T, ~M,  C,  H)
( T, ~M,  C, ~H)
(~T, ~M, ~C, ~H)
(~T, ~M,  C,  H)
(~T, ~M,  C, ~H)
(~T, ~M, ~C, ~H)
( T, ~M,  C,  H)
(~T,  M,  C, ~H)
```

Question 3: Programming: Spam Classification with a Naïve Bayes Classifier
[50] One of the famous applications of naïve Bayes classifiers is in spam filtering for e-mail. Can this machine learning method be effective in classifying shorter documents, namely SMS messages (text messages) as well? You are about to find out.

Your task is to implement a naïve Bayes classifier to identify "SPAM" from "HAM". The data set you have been provided, in the file sms.txt consists of 5,580 text messages that have been labeled as either "SPAM" or "HAM". If you open up the file, you will see that the first word per line is the label while the remainder of the line is the text message.

3.1 Methods to implement
We have provided code for you that will open this file, parse it, pass it to your classifier, and output the results. What you will have to focus on is the implementation of the file NaiveBayesClassifierImpl.java. If you open that file, you will see the following 5 methods which you must implement:

`void train(Instance[] trainingData, int v)`

- This method should train your classifier with the training data provided. The integer argument v is the size of the total vocabulary in your model. Please take this argument and store it as a field, as you will need it in computing the smoothed class-conditional probabilities. [See Section 3.3]
- In this method, update the counts that are used to compute the prior probabilities (`p_l`) and class-conditional probability of label (`p_w_given_l`). You may declare appropriate instance variables to store them.
- The instance class is a data structure holding the label and the SMS message as an array of words.

```
public class Instance {
  public Label label;
  public String[] words;
}
```

`double p_l(Label label)`

- This method should return the prior probability of the label in the training set. In other words, return *P*(*Spam*) if `label == Label.SPAM` or *P*(*Ham*) if `label == Label.HAM`.
- Use your count in the train method.

`double p_w_given_l(String word, Label label)`

- This method should return the class-conditional probability of label. In other words, return P(*word*|*label*). To compute this probability, you will use **smoothing**. [See Section 3.3]
- Use your count from the train method.

`Label classify(String[] words)`

- The classify method should return the Label object.
- To decide a Label, refer to the log probabilities section [Section 3.4].
- The Label class is an enumeration of our class labels:

```
public enum Label { SPAM, HAM }
```

void show_informative_5words( )

- This method prints 5 most informative words used by this classifier. Informativeness is defined as the highest value of P(*word* | *label*), for either label, divided by the lowest value of P(*word* | *label*), for either label:

$$Informative(w = \text{great}) = \max\left\{\frac{P(w = \text{great} \mid l = \text{HAM})}{P(w = \text{great} \mid l = \text{SPAM})}, \frac{P(w = \text{great} \mid l = \text{SPAM})}{P(w = \text{great} \mid l = \text{HAM})}\right\}$$

NOTE: The only provided file you are allowed to edit is NaiveBayesClassifierImpl.java. Any changes to the others will be overwritten by the testing script. You are allowed to add extra class files if you would like.

3.2 Command-line arguments

Your code can be tested on the class HW4 with the following calling convention:

```
java HW4 <trainingFilename> <testFilename>
```

which trains the classifier from the data in <trainingFilename> and tests it against <testFilename>, outputting the classifier's prediction for the label and the true label for each instance in the testing data, in the same order as it occurs in <testFilename>.

You do not need to implement these top level functions, that code has already been created by the teaching staff. You need only provide implementations of the four methods listed in the prior section.

3.3 Smoothing

There are two concepts we use here:

- Word token: occurrences of a word.
- Word type: unique word as a dictionary entry.

For example, the sentence *'The dog chases the cat'* has 5 word tokens but 4 word types. There are two tokens of the word type *'the',* one of *'dog'*, etc. Thus, the count of word tokens means the total number of words that occur and the count of word types means the number of unique words. As another example, if an SMS message is 15 words long, we would say that there are 15 word tokens. If the word type *'lol'* appeared 5 times, we would say there are 5 tokens of *'lol'*.

The class-conditional probability $P(w \mid l)$ where w represents some word token and l a label is a multinomial random variable (but in our implementation, it is a binomial random variable). Let V be the set of all word types in our vocabulary. We'll say there are $|V|$ possible word types that might appear. Now imagine a $|V|$-sided die. $P(w \mid l)$ is just the likelihood that this die lands with the w-side **up**, w being one of the words in our vocabulary V. You will need to estimate two such distributions: $P(w \mid Spam)$ and $P(w \mid Ham)$ for every word type w in V.

One way to estimate the value of $P(w \mid Spam)$ is to simply count the number of w tokens and divide by the total number of word tokens in all messages in our training data which are labeled as *Spam.* Of course, this doesn't work for word types which are in our vocabulary but aren't seen in the training data.

Let's be concrete about this in the case of our classification task. Assume the word *pale* does not appear in our training data but occurs in our test data. What probability would our classifier assign to $P(pale \mid Spam)$ and $P(pale \mid Ham)$? The probability in both cases is 0, and we know that this isn't really the case. There is some probability of seeing *pale,* even if it's very small. We'll fix this by doing something called "smoothing".

What we do to get around issue is that we "pretend" we actually did see some (possibly fractionally many) tokens for the word type *pale*. This method is called "Laplace smoothing" or "add-δ" smoothing, where δ is a parameter. Using smoothing, we'll calculate the conditional probability of a word type w as:

$$P(w|l) = \frac{C_l(w) + \delta}{\sum_{v \in V} C_l(v) + |V|\delta}$$

with $l \in \{Spam, Ham\}$, $C_l(w)$ representing the number of times the token w appears in messages labeled l in the training data. As above, $|V|$ represents the size of our vocabulary. It will likely include all word types in our training examples, plus any word types we might guess we'll see in the test set. It can also include a "catch all" word type for any word types in the test set that we didn't anticipate. The value $|V|$ will be passed to the train method of your classifier as the argument int v. With a little reflection, you will see that if we estimate our distributions in this way, we will have $\Sigma_{w \in V} P(w \mid l) = 1$.Please use this equations above for $P(w \mid l)$ to calculate the class-conditional probabilities in your implementation.

IMPORTANT: For this assignment, please use the value $\boldsymbol{\delta} = \mathbf{0.00001}$.

3.4 Log probabilities
The Naïve Bayes classifier must consider underflow. Underflow can occur when we take the product of a number of really small floating-point values. The result is a very, very small number, which might be smaller than what a floating point variable can encode. The value might then get set to 0, instead of just very small. Fortunately, there is a workaround.

Recall that a naïve Bayesian classifier computes:

$$f(w) = \operatorname*{argmax}_{l} \left[P(l) \prod_{i=1}^{k} P(w_i|l) \right]$$

where $l \in \{Spam, Ham\}$ and w_i is the ith word token of your SMS message, numbered 1 to k. Because maximizing a formula is equivalent to maximizing the log value of that formula (log is a monotonic function), $f(w)$ computes the same class as:

$$g(w) = \operatorname*{argmax}_{l} \left[\log P(l) + \sum_{i=1}^{k} \log P(w_i|l) \right]$$

What this means for you is that in your implementation of classify, compute the $g(w)$ value for word type w above rather than the $f(w)$ value. This will result in quicker code and avoid underflow errors generated by multiplying small numbers.

3.5 Submission guidelines
Using Moodle, submit only your new and edited source files (only .java files) as a zip file <wisc_username>_HW4.zip.

Your submitted code must compile on a university computer using javac. This can be done by placing all .java files in a single directory and running the command javac *.java in that directory. Please do NOT put your code in a package. Otherwise, it will fail to compile and that will be a mandatory deduction. Also, do NOT use static members to hold data in your implementation classes. We will be reinstantiating your implementation class through a battery of tests and if you keep data in between runs, your output will very likely be incorrect.

Additions or modifications of this assignment will be posted to the class forum on Piazza. Please post questions about the code there.

3.6 Evaluation

To grade your word we will use your code to train a Naive Bayes classifier on a new training set, and then check the accuracy on another new test set.

Some questions to ask while testing:
- What should the test accuracy be?
- What should the 5 most informative words be?
- What should $P(w\text{=great} \mid l\text{=HAM})$ be?
- What should $P(w\text{=friday} \mid l\text{=SPAM})$ be?
- What should $P(\text{HAM})$ be?
- What should $P(\text{SPAM})$ be?