CS 540-3: Introduction to Artificial Intelligence
Homework Assignment #5

Assigned: Monday, April 20
Due: Friday, May 1 by 11:59 p.m.

Hand-in Instructions

This homework includes two written questions and a programming problem. Hand in all parts electronically by copying them to the Moodle dropbox called "HW5 Hand-In," including your answers to the written problems. Your answers to each written problem should be turned in as separate pdf files called **P1.pdf** and **P2.pdf**. The first page of each pdf 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 problem must be written in Java, and all files needed to run your program, including any support (**do** also include unmodified files in the provided code) files but not input/output/debug files, should be submitted. Finally, create a folder called <**wisc username**>_**HW5**, put **all your code** as well as **P1.pdf** and **P2.pdf** into the folder and compress it as <**wisc username**>_**HW5.zip**. This final zip file should then be uploaded to the course Moodle website.

Late Policy

All assignments are due at 11:59 p.m. 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 11:59 p.m., and it is handed in between Wednesday 11:59 p.m. and Thursday 11:59 p.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, regardless of whether any free late days are used. Written problems and programming problems have the same deadline. A total of three (3) free late days may be used throughout the semester without penalty. Free late days will be used automatically on a first-needed basis. **Note that no submission will be accepted after three late days even if you have free late days left.**
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

Problem 1. Probabilities [20]

The following two tables provide the full joint probability distribution for three Boolean variables, X, Y, and Z.

Z

	Y	$\neg Y$
X	0.20	0.12
$\neg X$	0.04	0.24

$\neg Z$

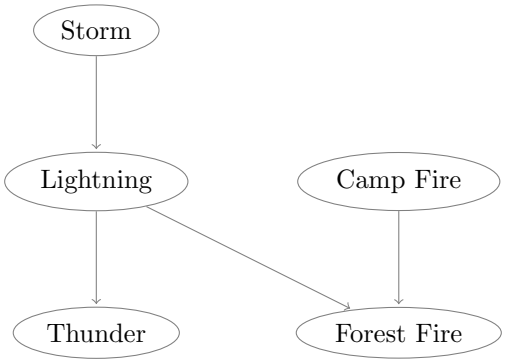
	Y	$\neg Y$
X	0.10	0.08
$\neg X$	0.06	0.16

Answer each of the following questions, showing all your work to receive full credit.

(a) [4] What is $P(\neg Y)$?
(b) [4] What is $P(X|\neg Y)$?
(c) [4] What is $P(\neg Z|\neg X, Y)$?
(d) [4] Verify whether or not X and Y are independent.
(e) [4] Verify whether or not Y and Z are independent.

Problem 2. Bayesian Networks [30]

Consider the following Bayesian Network containing five Boolean random variables. In this Bayesian Network, Camp Fire is independent of Storm, and of Lightning, and of Thunder.



The conditional probability tables (CPTs) associated with these five nodes are given by:

P(Storm=T)	P(Storm=F)
0.1	0.9

P(Camp Fire=T)	P(Camp Fire=F)
0.75	0.25

Storm	P(Lightning=T)	P(Lightning=F)
T	0.5	0.5
F	0.05	0.95

Lightning	P(Thunder=T)	P(Thunder=F)
T	0.95	0.05
F	0.20	0.80

Lightning	Camp Fire	P(Forest Fire=T)	P(Forest Fire=F)
T	T	0.5	0.5
T	F	0.4	0.6
F	T	0.1	0.9
F	F	0.01	0.99

Answer each of the following questions, showing all your work to receive full credit.

(a) [6] What is the probability of a forest fire?
(b) [6] What is the probability of a forest fire given thunder?
(c) [6] What is the probability that there is a storm given that there is a forest fire?
(d) [6] What is the probability of thunder given that there is no storm?
(e) [6] What is the probability of a camp fire and a forest fire?

Problem 3. Programming Naive Bayes Classifiers [50]

One application of Naive Bayes classifiers is for spam filtering e-mail messages. 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 Naive Bayes classifier to identify messages as either SPAM or HAM (i.e., not SPAM). The data set we are providing to you consists of 5,580 text messages that have been labelled as either SPAM or HAM and we have split them into a training set and a testing set. If you open the files, you will see that the first word in each line is the label and the remainder of the line is the text message.

Methods to Implement

We have provided code for you that will open a file, parse it, pass it to your classifier, and output the results. What you have to focus on is the implementation in the file `NaiveBayesClassifierImpl.java`. If you open that file, you will see the following four methods that 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 the section on **Smoothing** below.)
- `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`.
- `double p_w_given_l(String word, Label label)`
  This method should return the conditional probability of `word` given `label`. That is, return $P(word|label)$. To compute this probability, you will use smoothing. Please read the note on how to implement this below.
- `ClassifyResult classify(String[] words)`
  This method returns the classification result for a single message.
- `public ConfusionMatrix calculateConfusionMatrix(Instance[] testData)`
  This method takes a set of test instances and calculates the confusion matrix. The return type is `ConfusionMatrix`. The member variables of that class should be self explanatory. The following table explains the different cells of the confusion matrix.

Table 1: Confusion Matrix

	True SPAM	True HAM
Classified SPAM	True Positive	False Positive
Classified HAM	False Negative	True Negative

We have also defined three class types to assist you in your implementation. 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;
}
```

The `Label` class is an enumeration of our class labels:

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

The `ClassifyResult` class is a data structure holding a label and two log probabilities (whose values are described in the log probabilities section below):

```
public class ClassifyResult {
    public Label label;
    public double log_prob_spam;
    public double log_prob_ham;
}
```

The only provided file you are allowed to edit is `NaiveBayesClassifierImpl.java`. But you are allowed to add extra class files if you like.

Smoothing

There are two concepts we use here:

- Word token: an occurrence of a given word.
- Word type: a unique word as a dictionary entry.

For example, "the dog chases the cat" has 5 word tokens but 4 word types; there are two tokens of the word type "the". Thus, when we say a word "token" in the discussion below, we mean the number of words that occur and **NOT** 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. For example, if the word "lol" appeared 5 times, we say there were 5 tokens of the word type "lol".

The conditional probability $P(w|l)$, where w represents some word token and l is a label, is a multinomial random variable. If there are $|V|$ possible word types that might occur, imagine a $|V|$-sided die. $P(w|l)$ is the likelihood that this die lands with the w-side up. You will need to estimate two such distributions: $P(w|Spam)$ and $P(w|Ham)$.

One might consider estimating the value of $P(w|Spam)$ by simply counting the number of w tokens and dividing by the total number of word tokens in all messages in the training data labelled as Spam, but this method is not good in general because of the "unseen event problem," i.e., the possible presence of events in the test data that did not occur at all in the training data. For example, in our classification task consider the word "pale." Say "pale" does not appear in our training data but does occur in our test data. What probability would our classifier assign to $P(pale|Spam)$ and $P(pale|Ham)$? The probability would be 0, and because we are taking the sum of the logs of the conditional probabilities for each word and $\log 0$ is undefined, the expression whose maximum we are computing would be undefined. This wouldn't be a good classifier.

What we do to get around this is that we pretend we actually did see some (possibly fractionally many) tokens of the word type "pale." This goes by the name of Laplace smoothing or add-δ smoothing, where δ is a parameter. We write:

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

where $l \in \{Spam, Ham\}$, and $C_l(w)$ is the number of times the token w appears in messages labeled l in the training data. As above, $|V|$ is the size of the total vocabulary we assume we will encounter (i.e., the dictionary size). Thus it forms a superset of the words used in the training and test sets. The value $|V|$ will be passed to the train method of your classifier as the argument `int v`. For this assignment,

use the value $\delta = 0.00001$. With a little reflection, you will see that if we estimate our distributions in this way, we will have $\sum_{w \in V} P(w|l) = 1$. Use the equation above for $P(w|l)$ to calculate the conditional probabilities in your implementation.

Log Probabilities

The second gotcha that any implementation of a Naive Bayes classifier must contend with is underflow. Underflow can occur when we take the product of a number of very small floating-point values. Fortunately, there is a workaround. Recall that a Naive Bayes classifier computes

$$f(w) = \arg\max_l \left[P(l) \prod_{i=1}^{k} P(w_i|l) \right]$$

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

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

What this means for you is that in your implementation you should compute the $g(w)$ formulation of the function above rather than the $f(w)$ formulation. Use the Java function `log(x)` which computes the natural logarithm of its input. This will result in code that avoids errors generated by multiplying very small numbers. Note, however, that your methods `p_l` and `p_w_given_l` should return the true probabilities themselves and **NOT** the logs of the probabilities.

This is what you should return in the `ClassifyResult` class: `log_prob_spam` is the value $\log P(l) + \sum_{i=1}^{k} \log P(w_i|l)$ with $l = Spam$ and `log_prob_ham` is the value with $l = Ham$. The label returned in this class corresponds to the output of $g(w)$.

Testing

We will test your program on multiple training and testing sets, and the format of testing commands will be like this:

```
java HW5 trainingFilename testFilename
```

which trains the classifier using the data in `trainingFilename` and tests it using the messages in `testFilename`, outputting the classifier's prediction for the label and the true label for each message in the test set, in the same order as it occurs in `testFilename`. It will also print the confusion matrix that you calculated.

In order to facilitate your debugging, we are providing to you a sample input file and its corresponding output file. They are called `train0.txt` and `test0.txt` in the zip file. So, here is an example command:

```
java HW5 train0.txt test0.txt
```

As part of our testing process, we will unzip the file you turn in, remove any class files, call `javac HW5.java` to compile your code, and then call the main method HW5 with parameters of our choosing. Make sure your code runs on one of the computers in the department because we will conduct our tests on these computers.

Deliverables

1. Hand in (see the cover page for details) your modified version of the code skeleton that implements your Naive Bayesian classifier. Also include any additional java class files needed to run your program.
2. Optionally, submit a file called **P3.pdf** containing any comments about your program that you would like us to know.