Lab Class ML:IV

Until **Monday, Jan. 6th, 2020, 11:00 am CET**, the following exercises must be submitted:

Students receiving 4.5 ECTS: 2, and 3.
Students receiving 6 ECTS: 1, 2, and 3.

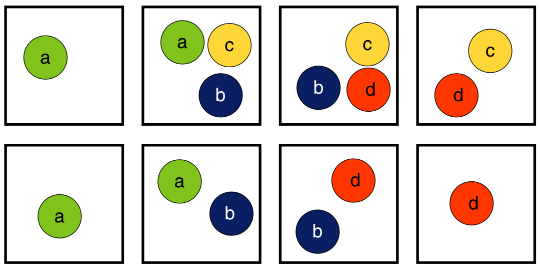
Lab Class General Instructions. Exercises should be completed in groups of three students; which and how many exercises you must submit depends on the number of ECTS credits awarded for the course, based on which degree programme you're studying:

	4.5 ECTS	6 ECTS
Degree programmes	CS4DM, CSM, MI, HCI*	DE, HCI*
	*(HCI students can choose number of credits in electives module)	
For admission to the final exam:		
Must reach at least	80 points	120 points
Out of maximum	120 points	180 points

For each exercise sheet, all required exercises must be submitted until the stated deadline. Each individual exercise will receive a score between zero and ten. Submissions received after the deadline will not be considered for grading. In order to be admitted to the final exams, you must reach the minimum total score shown.
Upload your solutions to Moodle as a single `.pdf` file, plus one `.ipynb` file for any programming exercises. Make it clear which solution corresponds to which exercise. Programming exercises are marked with $\boxed{\text{P}}$ – their solutions must be documented extensively with docstrings and/or inline comments. ml.weimar.webis.de

Exercise 1 : Probability Basics (Conditional Independence)

There are eight boxes containing different colored balls as shown in the illustration below:



The balls can be green, blue, yellow, or red (also marked a, b, c, d in the figure in case the colors are hard to distinguish). When picking one of the eight boxes at random, let A refer to the event "box contains a green ball," B to the event "box contains a blue ball, " C to the event "box contains a yellow ball," and D to the event "box contains a red ball." Hence, $A \cap B$ is the event "box contains both a green and a blue ball," and so on.

(a) Calculate $P(A)$, $P(B)$, $P(C)$, and $P(D)$.

(b) Calculate $P(A \cap B)$, $P(A \cap C)$, $P(B \cap C)$, and $P(B \cap D)$.

(c) Check all that apply:

☐ The events A and B are statistically independent.
☐ The events A and C are statistically independent.

☐ The events B and C are statistically independent.
☐ The events B and D are statistically independent.

(d) Calculate $P(A \mid C)$, $P(B \mid C)$, and $P(A \cap B \mid C)$.

(e) Calculate $P(B \mid D)$, $P(C \mid D)$, and $P(B \cap C \mid D)$

(f) Check all that apply:

☐ The events A and B are conditionally independent given C.
☐ The events B and C are conditionally independent gived D.

Exercise 2 : Bayes

A hospital database contains diagnoses (diseases) along with observed symptoms, collected during the past years. Let following representative dump be given, where the diseases are sorted temporally according to their appearances. Note, that the symptoms for a disease can change over different time periods.

		Symptom	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
Year	Diagnosis										
2001	D_1		X		X		X				
2002	D_2			X		X	X		X		
2003	D_3		X		X			X		X	
2004	D_4			X		X	X		X		
2005	D_3		X		X					X	
2006	D_5						X				X
2007	D_3		X		X			X			
2008	D_2			X					X		

(a) Compute the prior probabilities $P(D_i)$.

(b) Compute the posterior probabilities $P(D_i \mid S_4)$ of the diagnoses D_i given symptom S_4.

Exercise 3 : $\boxed{\text{P}}$ Content-based Spam Filtering with Naive Bayes

Your task is to implement a simple content-based spam email filter using the Naive Bayes approach: given a set of spam and non-spam ("ham") emails, your program should learn to compute the probability that a previously unseen email is spam based on the occurrence of words and other tokens. For the purpose of our implementation, we will consider the presence or absence of words w_i in an email as binary events, which are assumed to be statistically independent according to the Naive Bayes assumption discussed in the lecture.

For training and testing our classifier, we will use the SpamAssassin public mail corpus.

(a) First, Read the article "A Plan for Spam" from http://paulgraham.com/antispam.html, to familiarize yourself with the problem and basic terminology.

(b) Download the following two datasets of spam and non-spam emails:

- http://spamassassin.apache.org/old/publiccorpus/20050311_spam_2.tar.bz2 (containing $\approx$ 1400 spam emails)
- http://spamassassin.apache.org/old/publiccorpus/20030228_easy_ham_2.tar.bz2 (containing $\approx$ 1400 ham emails)

The example emails are provided as individual files within a gzipped tar archive, many of them using the Quoted-printable MIME encoding. Starter code for extracting and decoding the individual emails will be given in lab class, so that you can get each email as a string out of these files.

(c) With the help of the Python standard library, write a functions `tokenize` that splits an email into individual words and other tokens. How you handle special characters such as punctuation is up to you – you might decide to delete them completely, treat them as separate tokens, etc. Special processing of email headers is also optional. In any case, you implementation should be able to handle at least the following simple test case:

```
>>> tokenize("this is a test string")
['this', 'is', 'a', 'test', 'string']
```

(d) Develop a class `SpamClassifier` which implements a simple Naive Bayes spam filter. The constructor of your class should take two parameters: a collection of spam emails, and a collection of ham emails—both already split into their constituent tokens—which are then used to train the classifier. During training, your classifier should estimate the probability $P(w_i|\textsf{spam})$, for each word w_i which occurs in either a spam or a non-spam email. Also consider how you deal with words that do not occur in the training data. Your class should provide a method `predict`, which takes the words w_j of a single email as an argument, and returns the probability $P(\textsf{spam}|w_j)$ that this email is spam.

The interface to your classifier should work as follows:

```
>>> cls = SpamClassifier(spam=[["this", "is", "spam"], ["more", "spam"]],
                         ham=[["this", "is", "ham"], ["more", "ham"]])
>>> cls.predict(["is", "this", "spam", "or", "not"])
1.0
```

Hint: the article mentioned in (a) gives some practical advice on tweaking the implementation. Depending on which of these hints you decide to implement, you may get different results to the one above. Don't consider this a strict test case.

(e) Train and test your spam filter using the SpamAssassin corpus. Select 100 random examples each of the `spam` and `ham` classes to use as a validation set, and exclude them from the training data. Report the misclassification rate of your classifier on this validation set, and print out a confusion matrix with the corresponding number of examples filled in for a, b, c, d, in each cell as follows:

	predicted class “spam”	predicted class “ham”
true class “spam”	a	b
true class “ham”	c	d

(f) Train your classifier on the entire dataset (including the validation set) and examine the conditional spam probabilities for individual words that your classifier computes during training. Which three words or tokens are the strongest evidence that an email is spam, which three for ham? Which six words provide the weakest evidence for either class? Test the trained classifier using the following additional datasets:

- spam: https://spamassassin.apache.org/old/publiccorpus/20030228_spam.tar.bz2 and
- ham: https://spamassassin.apache.org/old/publiccorpus/20030228_hard_ham.tar.bz2

What is the misclassification performance now?