Programming Assignment
Naive Bayesian Text Classification

1 Introduction to the assignment

The goal of this assignment is to exercise in programming. We assume that you learned the basic programming concepts and have some experience in programming in Java, C or Python. The best way to learn programming is by doing. So why not build a system that does something usefull and that introduces you into the basics of machine learning.

A document or text classifier is a function that assigns to all the texts that are member of a specified corpus one of a fixed set of categories. For example, a spam mail classifier is a text classifier that assigns to the e-mail messages that a person receives in his/her mail-box one of the categories spam or not-spam. A newspaper article classifier assigns to each newspaper article one of the categories in the set C, where $C = \{Sport, Business, Politics, Art, Media\}$. For most news articles it holds that human readers will have no problem in classifying them into one of the categories of C. But can we learn a machine to do that?

In this assignment you will built a Naive Bayesian Text Classifier, that implements a machine learning method for classifying texts based on the statistics about the occurrences of words in texts of the different categories. The first step in building such a classifier is called *modelling*. In this step we specify the properties of a text that are typical for the different categories. The type of properties we use are the occurrence of specific words in a text. If our classifier has to classify a text it will check if certain words occur in the text or not. The second step is *training*. For training we need a labeled corpus, i.e. a set of texts with known categories. Training a classifier basically consists of counting for each word and each category in how many texts of each category the word occurs.

Suppose that the word "soccer" occurs relatively far more often in Sport articles than in Business articles. Then we can use the occurrence of the word "soccer" in a text as an indicator for the text to belong to the class Sport. The text model will contain a feature that says if the word "soccer" does occur in the text or not. When we let the trained classifier classify a new text we check the indicators for the various categories. A classifier will sometimes make an error and predict the wrong class. We can test the quality of our classifier by applying it to a test corpus and check how often the class it predicts equals the correct one.

In the following sections we give a more detailed description.

2 Introducing Naive Bayes Document Classification

Let C be a finite set of categories. Let D be a labeled corpus of documents, i.e. every document in D is assigned one of the categories in C. Let $D(c)$ denote the subset of documents from D of category c.

We model a document as a vector of features X_i.
The vector $X(d) =< X_1(d), X_2(d), ..., X_n(d) >$ is the modelled document d. We also use the notation X or $< X_1, X_2, ...X_n >$ for the feature vector of the document we are talking about.

When we classify d we compute for all classes (categories) $c_j \epsilon C$ the probabilities

$$P(C = c_j | X_1 = x_1, X_2 = x_2, ..., X_n)$$

We use the shorter notation

$$P(C = c_j | X =< x_i >)$$

This is the probability that a document with the given feature values $X =< x_i >$ has category c_j.

A Naive Bayesian Classifier (NBC) chooses that category for document d for which this probability is maximal. Or:

$$NBC(d) = argmax_C P(C = c_j | X =< x_i >)$$

Here $argmax_C(f(c))$ is the function that gives that value of c in the set C for which the value of $f(c)$ is maximal. The function NBC is the basis of the main classify function of the NB classifier. We say that the classifier uses the MAP rule[1].

Because of Bayes' Rule we know that:

$$NBC(d) = argmax_C P(C = c_j | X =< x_i >) =$$

$$argmax_C P(X =< x_i > | C = c_j).P(C = c_j)/P(X =< x_i >)$$

Since we are interested in the argmax over the set of probabilities and since the factor $1/P(X =< x_i >)$ occurs in all of them, we may remove $1/P(X =< x_i >)$ so we can compute instead:

$$NBC(d) = argmax_C P(X =< x_i > | C = c_j).P(C = c_j)$$

The factor $P(C = c_j)$ is the *prior probability* of class c_j. We now take a step that is characteristic for ***naive*** Bayesian inference. We assume that the features X_i are all independent of each other *given the class value* c_i. Or:

$$P(X|C = c_i) = P(X_1, X_2, ..., X_n | C = c_i) = P(X_1|C = c_i).P(X_2|C = c_i)...P(X_n|C = c_i)$$

If we apply this in the formula above for the classify function $NBC(d)$ we obtain:

$$NBC(d) = argmax_C.P(X_1|C = c_j).P(X_2|C = c_j)...P(X_n|C = c_j).P(C = c_j)$$

Various variants of the Naive Bayesian Classification method exist. They differ in the type of features X_i that are used to model a document. In the *multinomial model* the features X_i are functions that *count* how often word w_i from the word list (vocabulary) occurs in the document d. In the *boolean model* the X_i are *boolean* variables that tell if word w_i occurs in d; the values x_i are 1 or 0 then. The NBC that uses the boolean model is called the *binary* variant of NBC. This is the one you will implement and test.

3 Building the Binary Naive Bayesian Classifier

Let $V = w_i$ be a vocabulary of k words w_i $(1 \leq i \leq k)$. In the *binary* variant of the NBC method for text classification a document d is modeled by a vector $X(d) =< x_1, x_2, ..., x_k >$ where the feature x_i denotes whether word w_i occurs in d (value 1) or not (value 0). In this case the conditional probability of X given class c is

$$P(X =< x_i >)|c) = \prod_{i=1}^{i=k} P(w_i|c)^{x_i}.(1 - P(w_i|c))^{1-x_i}$$

How do we know the probabilities $P(w_i|c)$ for each of the words and each of the classes? Or: how do we know what the probability is that for example the word "soccer" occurs in a document of class Sport? We estimate these probabilities by looking at a corpus, a set of documents that we use to train our classifier.

The $P(w_i|c)$ are estimated by the *smoothed estimates*[2]

[1] The MAP, Maximum A posteriori Probability decision rule says that we decide for that category that has the maximum a posteriori probability.

[2] We give the formula without comments. We refer to the literature for a motivation.

	docId	words of V in doc	spam ?
train	d_1	Uganda, Bank, Uganda	Yes
train	d_2	Uganda, Uganda, Solar	Yes
train	d_3	Uganda, Mexico	Yes
train	d_4	Tokyo, Japan, Uganda	No
test	d_5	Uganda, Uganda, Uganda, Tokyo, Japan	??

Table 1: The classified document set for the Example

$$\hat{P}(w|c) = \frac{M_{w,c} + 1}{M_c + 2}$$

where $M_{w,c}$ is the number of documents in the train set of class c that contains word w and M_c is the total number of documents of class c in the train set.

The classify function of the binary NBC is now:

$$NBC(d) = argmax_C \prod_{i=1}^{i=k} \hat{P}(w_i|C = c_j) \times P(C = c_j)$$

Our classifier has a method *classify* that implements this function, i.e. it will output that class value c_j for which the value of $P(X|C = c_j) \times P(C = c_j)$ is maximal.

The prior probabilities $P(C = c_j)$ are estimated by the relative frequencies $\frac{M_{c_j})}{N}$ where M_{c_j} is the number of documents of class c_j in the train set and N is the total number of documents in the train set.

Why do we have to compute *smoothed* probability estimates? Why can't we use the maximum likelihood probabilities computed from the relative frequencies, i.e., why not use simply:

$$P(w|c) = \frac{M_{w,c}}{M_c}$$

instead of the formula above for the smoothed probabilities? That is because there can be a word w' in the vocabulary that does not occur in one of the train texts that belongs to a certain category c'. In that case the unsmoothed probability for this pair (w', c') will be zero, because $M_{w',c'}$ is zero then. If we would use these zero probabilities in our formula for classifying a text that contains word w' the text will never be classified as of category c'. But, c' could be the correct class for the text. If we can't rule out a possibility on logical grounds, we should not consider a case as having a zero probability on the grounds that we never encountered such a case.

Example binary variant of NBC. Table 1 shows a corpus of 4 train documents, mails that are spam or not spam. We want to classify the test document.

Let V be ordered as $< Ug, Ba, So, Me, To, Ja >$ (using short names for the actual words, e.g., Ug stands for Uganda).

The test document d_5 is then encoded as $d_5 =< 1, 0, 0, 0, 1, 1 >$.

We compute for each word w in V, the estimated probabilities. We denote $\hat{P}(Ug|spam)$ instead of $\hat{P}(x_1 = 1)$, the probability that the word Ug, which is the 1st word in V occurs in a spam mail, etc.:

From these probabilities we can compute $\hat{P}(\overline{Ug}|spam) = \hat{P}(x_1 = 0|spam) = 1 - \hat{P}(x_1 = 1|spam)$.

$\hat{P}(Ug\|spam) = (3+1)/(3+2) = 4/5$	$\hat{P}(Ug\|\overline{spam}) = (1+1)/(1+2) = 2/3$
$\hat{P}(Ba\|spam) = (1+1)/(3+2) = 2/5$	$\hat{P}(Ba\|\overline{spam}) = (0+1)/(1+2) = 1/3$
$\hat{P}(So\|spam) = (1+1)/(3+2) = 2/5$	$\hat{P}(So\|\overline{spam}) = (0+1)/(1+2) = 1/3$
$\hat{P}(Me\|spam) = (1+1)/(3+2) = 2/5$	$\hat{P}(Me\|\overline{spam}) = (0+1)/(1+2) = 1/3$
$\hat{P}(To\|spam) = (0+1)/(3+2) = 1/5$	$\hat{P}(To\|\overline{spam}) = (1+1)/(1+2) = 2/3$
$\hat{P}(Ja\|spam) = (0+1)/(3+2) = 1/5$	$\hat{P}(Ja\|\overline{spam}) = (1+1)/(1+2) = 2/3$

We compute $P(spam|d_5) = P(spam| < Ug, \overline{Ba}, \overline{So}, \overline{Me}, To, Ja >)$

$$P(spam) * P(Ug|spam) * P(\overline{Ba}|spam) * ... * P(Ja|spam)$$

$$= 3/4 * 4/5 * 3/5 * 3/5 * 3/5 * 1/5 * 1/5 \approx 0.0052$$

and

$$P(\overline{spam}|d_5) = P(\overline{spam}) * P(Ug|\overline{spam}) * P(\overline{Ba}|\overline{spam}) * ... * P(Ja|\overline{spam})$$

$$= 1/4 * 2/3 * 2/3 * 2/3 * 2/3 * 2/3 * 2/3 \approx 0.0219$$

4 The Assignment

Implement the binary variant of the Naive Bayesian Classifier (NBC), train and test it on a corpus of e-mails.

The NB classification method is very general. It not only works for e-mail classification, but for any classification problem where texts are classified based on word occurrences into a given number of categories. Your classifier should work for classification problems with any finite number of class values (not just two as in the e-mail data).

In the next sections we explain the different phases of this task.

5 Reading and processing the corpus

For this assignment use train and test sets with e-mails classified as spam or ham[3]. A prepared directory of files with e-mails is available for this assignment. It has 10 parts. Use part 1 up to and including part 9 for training and part 10 for testing. Ask the docent where to get this corpus.

You will have to write a program (method) that reads the text files in the train corpus and stores for each word that occurs in the train corpus for each of the categories (class values: ham and spam) in how many text files of that class the word occurred. For the storage you have to choose a convenient *data structure*. Therefore you have to know what the operations are that you have to define on the data. Read the whole document before you implement the data structure for storing the word counts.

[3]From the Enron e-mail data set, https://www.cs.cmu.edu/ ./enron/

6 Building the vocabulary

The binary variant of the NB classifier that you will implement uses a fixed finite set of Boolean features. Each of these features tell if a specific word occurs in the document that we want to classify. We will select those words that can best be used to distinguish the various categories. For example the word "the" will not have a high discriminative value for telling if an e-mail is spam or not. The *vocabulary* V will contain all those words that we select to be used to classify documents.

In compiling the vocabulary of the classifier we could take all words that occur in the training data. But many words will only occur a small number of times. One way to restrict the number of words is to tokenize the data. Tokenization splits up a character sequence into smaller pieces (tokens). Most programming languages provide text tokenizers or scanners. You also want to normalize your tokens (for example by converting everything to lower case). An example of tokenization is:

> **Original sentence** Hello, everyone. Do you like the new layout?
>
> **Tokens:** [Hello, everyone, Do, you, like, the, new, layout]
>
> **Tokens normalized:** [hello, everyone, do, you, like, the, new, layout]

Make a *simple* tokenizer. Since datasets are small, we recommend to make strong normalizations, for example by removing punctuation. When the program reads a text file it will call the tokenizer and normalizer before it stores the words and the counts.

7 Chi square feature selection

Not all (normalized) words that occur in the training set are good indicators for a specific text category. One way to determine if a word w is a good indicator for a given class is based on the Chi square test: if the Chi square value exceeds a certain threshold value than include the feature for w: i.e. the feature that has value 1 (for true) when w occurs in the text and 0 (for false) otherwise.

7.1 How to apply the χ^2 test?

A table M for word w has 2 rows and n columns, where n is the number of categories. The cell in the i-th row and j-th column of M is denoted $M(i, j)$. The first row $M(1,)$ contains the values M_{w,c_j}, i.e. the number of texts of class c_j in which word w occurs. The second row $M(2,)$ contains the numbers of texts of class c_j in which the word w does not occur. Thus the cell numbers in the j column of M stand for the numbers of times that the word occurs or not occur in the texts of class c_j. The sum of the numbers in column j equals the number of texts in class c_j. This marginal sum we denote by C_j. The sum of the numbers in the first row cells equals the number of texts in which word w occurs. We denote this row total by W_1. The total of the numbers in the second row is denoted by W_2, it is the number of texts that do not contain word w. The total number of texts is denoted by N. The numbers are called the observed values, because they are computed from the corpus.

The Chi square test uses a χ^2 function that computes the distance between the observed values $M(i, j)$ and the *expected* values. The expected value $E(i, j)$ for cell (i, j) is:

$$E(i,j) = \frac{W_i \times C_j}{N}$$

The χ^2 value for table M is computed using:

$$\chi^2(M) = \sum_i \sum_j \frac{(M(i,j) - E(i,j))^2}{E(i,j)}$$

	c_1	c_2	W
w	10	1	11
Not w	50	59	109
C	60	60	120

Table 2: The table with counts for the χ^2 Example

As an example consider the data in Table 2.

The χ^2 value for this table is 8.1 $(3.68 + 3.68 + 0.37 + 0.37)$. The expected values $E(i, j)$ are: top row: 5.5, 5.5, bottom row: 54.5 and 54.5.

Note that according to your statistics book use of the chi-square tests is inappropriate if any expected frequency is below 1 or if the expected frequency is less than 5 in more than 20% of the cells. But we don't care about this here. When a word occurs in all e-mails that are spam and in none of the non-spam mails then it is a good indicator and we should use it as a feature in our classifier. Such a word will have a high chi-square value (as you can easily check).

We refer to the literature for word (feature) selection methods[4].

Implement the χ^2 function, a method that when called with a word w as argument returns the χ^2 value of the table. Use it to select the 300, 200 and 100 words that are most distinguishing. The size of the feature set (i.e. the number of selected words in the Vocabulary) is a parameter of your classifier. It is set a specific value before the classifier is trained.

8 Computing in log space

In order to avoid underflow (the machine cannot represent numbers that are too small), we usually calculate in log space (base 2) (since log(x.y) = log(x)+log(y)):

$$log(P(c)) + \sum_{i=1}^{n} logP(w_i|c)$$

The classifier will output the class value c for which this expression has the maximal value.

Implement the training phase of the NB Classifier. The method train will successively read the texts in the train corpus and store the word counts in the chosen data structure (after word normalization). Then it applies Ci square to select the word features. For each of the selected words it computes the smoothed estimated probabilities.

9 Testing your classifier

For measuring the *performance* of the classifier we apply the classifier to the texts in a test set. The texts in this test set have not been used for building the classifier, but their categories are known. So we can compare for each test file the output of the classifier (the *predicted* category) with the given *correct* category. So we can measure the accuracy of the classifier, that is the percentage correctly predicted texts in the test set. A *baseline classifier* is a classifier that always predicts the most likely class in the train set. If the train set contains 80% spam mail a baseline classifier would always guess "spam". An intelligent classifier, one that takes features of the e-mail into account, should perform better than a baseline classifier.

[4]See for example *A comparative study of feature selection in text categorization* by Y.Yang and J.O.Pedersen

Implement the classify method that applies the classifier to a text and outputs the predicted class using the NBC method with boolean features based on the Vocabulary with the n most indicative words.

Implement a method that computes the accuracy of the classifier for a given test set and a given vocabulary size. Test the classifier with different sizes of the vocabulary (100,200 and 300 words).

Beware: If you normalize words for building your vocabulary (for example by replacing capital letters with small letters) you should also normalize the text before classification.

Beware: Do not test your classifier on data that you already used for training the classifier. It is even better to split data in three parts. A third part (development set) is used for setting additional parameter values, like smoothing constants.

10 To be delivered

Deliver the following in a zip file.

- A document (with your name on it) in which you report about the performance of the classifier on the given train and test sets, for different sizes of vocabularies (100,200,300 words) based on the Chi square feature selection method.
- A text file containing the sorted list of 300 words with the highest Chi square values and their Chi square values.
- The src code of your programs.
- A README file that explains system and software requirements, how and where to install the system and how to run the code.

Do not include the corpus.

Good luck with programming your text classifier!