Lecture 6

Probabilistic Learning – Classification Using Naive Bayes

Bayesian classifiers

The statement in your textbook that weather forecasts use data from past events to predict the weather is not quite correct. Short-term weather forecasts do not extrapolate past weather histories, instead they use complex computer models that take several inputs, such as the current weather conditions, and produce a probabilistic forecast. The computer models use several ML techniques to improve their performance.

The earliest attempts to use Bayesian classifiers go back to well before World War II. Then, nobody used the term machine learning , but the goal was essentially the same: predict an example's class based on the known attribute values. Here is the essence of Bayesian classifiers: using the Bayesian probabilistic theory, calculate for each class the probability of the given example belonging to it, and then choose the class with the highest value. In general, Bayesian classification follows three steps:

- For each class C_i compute the probability $P(C_i)$ of its occurring (we have a finite set of classes, C_1, C_2, C_3 …). These are called prior probabilities and they do not depend on the data example.

- Compute the conditional probability of observing the data example for each given class, $P(D \mid C_i)$. This is called likelihood, i.e., how likely is to observe the data example D for a given class C_i.

- Choose the class with the maximum $P(D \mid C_i) * P(C_i)$.

Bayesian classifiers *(cont.)*

It is not hard to imagine situations in which the data are not the only available source of information about the population or about the system to be modeled. The Bayesian method provides a principled way to incorporate this external information into the data-analysis process. The process starts with an already given probability distribution for the analyzed data set. As this distribution is given before any data is considered, it is called a *prior distribution.* The new data set updates this prior distribution into a *posterior distribution.* The basic tool for this updating is the Bayes Theorem.

Let D be a data sample whose class label is unknown. Let H be some hypothesis: such that the data sample D belongs to a specific class C for example. We want to determine P(H|D), the probability that the hypothesis H holds given the observed data sample D. P(H|D) is the posterior probability representing our confidence in the hypothesis after D is given. In contrast, P(H) is the prior probability of H for any sample, regardless of how the data in the sample looks. The posterior probability P(H|D) is based on more information then the prior probability P(H). The Bayesian Theorem provides a way of calculating the posterior probability P(H|D) using probabilities P(H), P(D), and P(D|H). The basic relation is P(H|D) = $\frac{P(D|H)*P(H)}{P(D)}$

Bayes' theorem

We know that when two events are dependent, the probability of both occurring is:

(1) $P(A, B) = P(A) * P(B \mid A)$

In this formula, if we replace A with B and B with A, we get:

(3) $P(B, A) = P(B) * P(A \mid B)$

Since P(A, B) = P(B, A), we have:

$P(A) * P(B \mid A) = P(B) * P(A \mid B)$

Which is the same as:

$$P(B \mid A) = \frac{P(A \mid B) * P(B)}{P(A)}$$

This is Bayes' law in its simplest form. The law can be rephrased in terms of ML: we have several possible data classes, C_1, C_2, …, C_n and data D:

$$P(C_i \mid D) = \frac{P(D \mid C_i) * P(C_i)}{P(D)}$$

The interpretation of Bayes' law is as follows. $P(C_i)$ is called **prior probability** and it reflects our belief that class C_i occurs with probability $P(C_i)$. $P(D \mid C_i)$ is called **likelihood** and it is the probability that we will observe data D if the actual class is C_i. P(D) is the general probability that we will observe data D in general. In other words, using the Bayes' theorem, we "revise" our prior belief $P(C_i)$ that class C_i occurs to a new **posterior probability** $P(C_i \mid D)$ that reflects the new information included in data D. This is how we learn from data D.

Computing conditional probability with Bayes‘ theorem

As an example of how to use this rule, consider the following medical diagnosis problem. Suppose you are a woman in your 40s, and you decide to have a medical test for breast cancer called a **mammogram**. If the test is positive, what is the probability you have cancer? That obviously depends on how reliable the test is. Suppose you are told the test has a **sensitivity** of 80%, which means, if you have cancer, the test will be positive with probability 0.8. In other words:

P(pos | cancer) = 0.8
P(neg | cancer) = 1 – 0.8 = 0.2

Many people conclude they are 80% likely to have cancer. But this is false! It ignores the prior probability of having breast cancer, which fortunately is quite low:

P(*cancer*) = 0.004

Ignoring this prior is called the **base rate fallacy**. We also need to take into account the fact that the test may be a **false positive** or **false alarm**. Unfortunately, such false positives are quite likely (with current screening technology):

P(*pos* | ¬*cancer*) = 0.1

Therefore, P(*neg* | ¬*cancer*) = 1- 0.1 = 0.9

Computing conditional probability with Bayes' theorem *(cont.)*

Combining these three terms using Bayes rule, we can compute the correct answer as follows:

$$P(\text{cancer} \mid \text{pos}) = \frac{P(pos \mid cancer) * P(cancer)}{P(pos)}$$

We know that the conditional probability P(pos | cancer) = 0.8 and that the prior probability P(cancer) = 0.004. To use the formula we also need P(pos). How can we find P(pos)? We will use the law of total probability, which says that the probability of any event A is

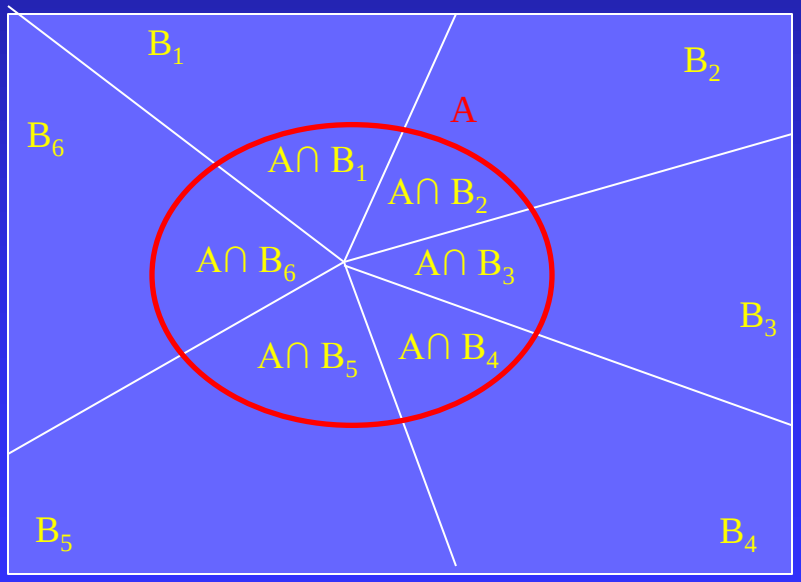
$$P(A) = \sum_{i=1}^{i=n} P(A, Bi)$$

Where B_1, B_2, …, B_n are mutually exclusive events that cover the whole sample space. In other words, B_1, B_2, …, B_n form a partitioning of the sample space into n mutually exclusive events.

Computing conditional probability with Bayes' theorem *(cont.)*

The law of the total probability can be justified using the following Venn diagram. Here, the sample space is partitioned into B_1, B_2, B_3, B_4, B_5 and B_6. The event A is itself the union of its intersections with each B_i event, i.e.,

A = (A ∩ B_1) and (A ∩ B_2) and (A ∩ B_3) and (A ∩ B_4) and (A ∩ B_5) and (A ∩ B_6)



Hence, P(A) = P(A, B_1) + P(A, B_2) + P(A, B_3) + P(A, B_4) + P(A, B_5) + P(A, B_6) =

= $\sum_{i=1}^{i=6} P(A, Bi)$

Computing conditional probability with Bayes' theorem *(cont.)*

Since each P(A, B_i) = P(A | B_i) * P(B_i), the formula of the total probability can be rewritten as:

$P(A) = \sum_{i=1}^{i=n} P(A, Bi) = \sum_{i=1}^{i=n} P(A \mid Bi) * P(Bi)$

Let's go back the cancer example. The last piece we need to compute P(cancer | pos) is P(pos). According to the law of total probability:

P(pos) = P(pos | cancer) * P(cancer) + P(pos | ¬ cancer) * P (¬ cancer)

Here, we use the fact that the events {cancer} and {¬*cancer}* form a partitioning, i.e., a person either has cancer or does not have one. Therefore:

$$P(\text{cancer} \mid \text{pos}) = \frac{P(pos \mid cancer) * P(cancer)}{P(pos)} =$$

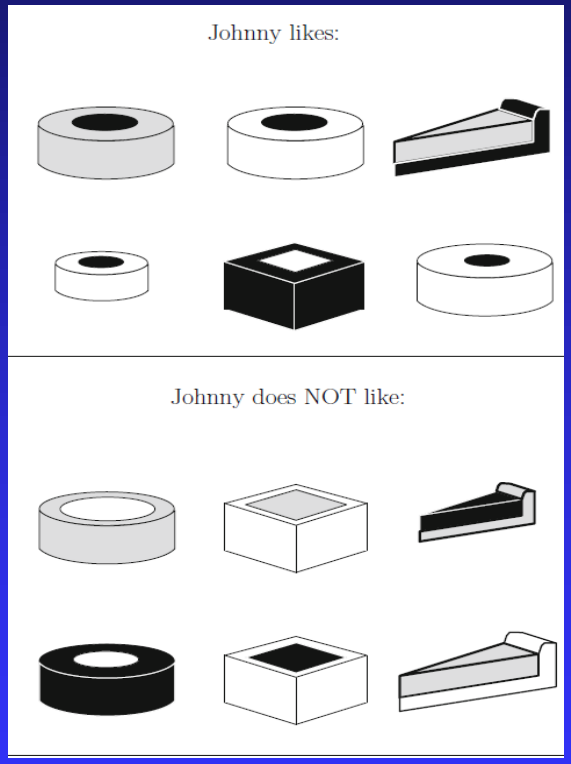
$$= \frac{P(pos \mid cancer) * P(cancer)}{P(pos \mid cancer) * P(cancer) + P(pos \mid \neg cancer) * P(\neg cancer)} =$$

$$= \frac{0.8 * 0.004}{0.8 * 0.004 + 0.1 * 0.996} = 0.031$$

In other words, if you test positive, you only have about a 3% chance of actually having breast cancer! These numbers are from (McGrayne 2011). Based on this analysis, the US government decided not to recommend annual mammogram screening to women in their 40s: the number of false alarms would cause needless worry and stress amongst women, and result in unnecessary, expensive, and potentially harmful followup tests.

Toy example of Bayesian classifier: What Johnny likes

Let's start with a toy example of a single-attribute case. There are six pies that Johnny likes, and six that he does not. These *positive* and *negative examples* of the underlying concept (pies that Johnny likes) constitute a *training set* from which the machine is to induce a *classifier*—an algorithm capable of categorizing any future pie into one of the two *classes*: positive and negative. i.e., examples that Johnny likes and examples he does not like. Therefore, the probability that Johnny likes a randomly chosen pie is:



$$P(\text{positive}) = \frac{Npos_{itive}}{N_{all}} = 6/12 = ½$$

P(positive) is called prior probability and it measures our belief in the absence of any additional data. Let us now take into consideration one of the pie attributes, say, filling-size. The training set contains eight examples with thick filling ($N_{thick} = 8$). Out of these, three are labeled as positive ($N_{pos \text{ and } thick} = 3$). This means that the *conditional probability* of an example being positive given that filling-size=thick is 37.5%:

Toy example of Bayesian classifier: What Johnny likes *(cont.)*

$P(pos \mid thick) = \frac{P(pos\ and\ thick)}{P(thick)} = \frac{3/12}{8/12} = \frac{3}{8}$

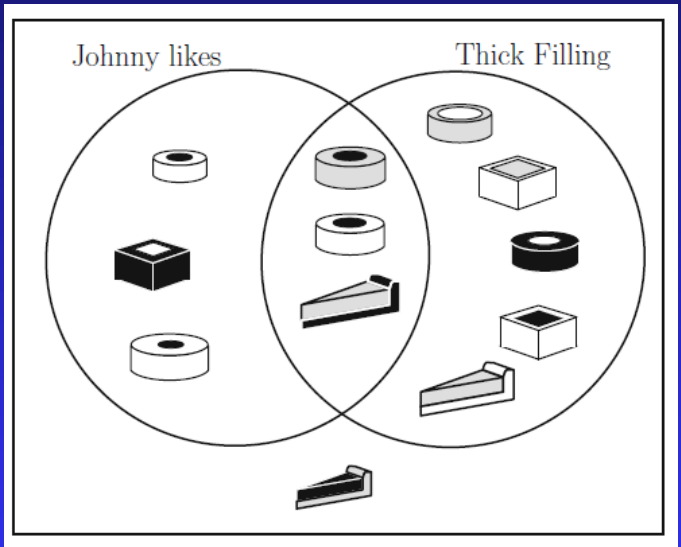
Among the eighth thick pies, five represent the negative class, which means that $P(neg \mid thick) = \frac{5}{8} = 0.625$. Observing that $P(neg \mid thick) > P(pos \mid thick)$, we can conclude that the probability of Johnny disliking a pie with thick filling is greater than the probability of the opposite case. It thus makes sense for the classifier to label all examples with filling-size=thick as negative instances of the concept "pie that Johnny likes." We have shown how to build a simple Bayes' classifier with one attribute.

Note that conditional probability $P(pos \mid thick)$, is more trustworthy than the prior probability $P(pos) = 1/2$, because of the additional information that goes into its calculation.

Conditional probability should not be confused with *joint probability* of two events occurring simultaneously. Be sure to use the right notation. The terms are separated by commas or "and", $P(pos, thick)$ or P(pos and thick) in joint probability, and by a vertical bar, $P(pos \mid thick)$ in conditional probability. For a randomly picked pie, $P(pos, thick)$ denotes the probability that the example is positive *and* its filling is thick; whereas $P(pos \mid thick)$ refers to the occurrence of a positive example among those that have filling-size=thick.

Toy example of Bayesian classifier: What Johnny likes *(cont.)*

The figure on the right illustrates the terms. The rectangle represents all pies. The positive examples are contained in one circle and those with filling-size=thick in the other; the intersection contains three instances that satisfy both conditions; one pie satisfies neither, and is therefore left outside both circles. The conditional probability, *P*(pos | thick) = 3/8, is obtained by dividing the size of the intersection (three) by the size of the circle thick (eight). The joint probability, *P*(pos, thick) = 3/12, is obtained by dividing the size of the intersection (three) by the size of the entire training set (twelve). The prior probability of *P*(pos) = 6/12 is obtained by dividing the size of the circle pos (six) with that of the entire training set (twelve).



Toy example of Bayesian classifier: What Johnny likes *(cont.)*

The joint probability can be obtained from prior probability and conditional probability:

P(pos, thick) = P(pos | thick) * P(thick) = $\frac{3}{8} * \frac{8}{12} = \frac{3}{12}$

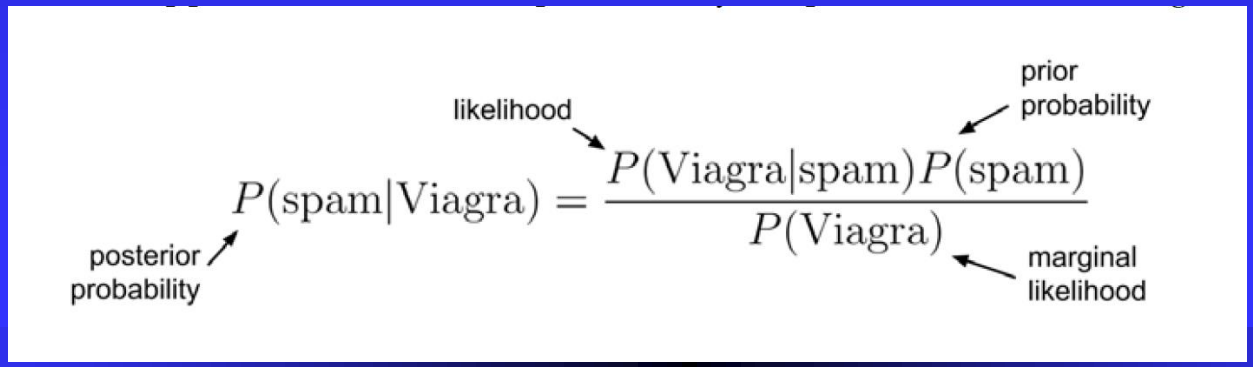
P(thick, pos) = P(thick | pos) * P(pos) = $\frac{3}{6} * \frac{6}{12} = \frac{3}{12}$

Note that that *P*(thick, pos) = *P*(pos, thick) because both represent the same thing: the probability of thick and pos co-occurring.

Note that joint probability can never exceed the value of the corresponding conditional probability: *P*(pos, thick) ≤ *P*(pos | thick). This is because conditional probability is multiplied by prior probability, *P*(thick) or *P*(pos), which can never be greater than 1.

Toy example Bayesian classifier: Spam filter

We want to build a Bayesian classifier with one attribute that filters spam email from non-spam (ham). Based on past experience, we know that spam massages occur with probability 0.2. The attribute we are going to use is the occurrence of the word “Viagra” in email. Obviously, the events {an email contains “Viagra”} and {an email is spam} are dependent. Since the word occurs in both spam and ham, we need to find the probability of the occurrence of “Viagra” in spam and the probability of its occurrence in ham. Suppose, that the probability of the occurrence of “Viagra” in spam is 4/20 and the probability of the occurrence of “Viagra” in ham is 1/80. Now, we are ready to apply the Bayes’ rule to find the posterior probability that a message is spam if it contains the word “Viagra”:



Toy example Bayesian classifier: Spam filter *(cont.)*

Given the data, we can conclude that:

P(spam) = 0.2

P(ham) = 1 – 0.2 = 0.8

P(Viagra | spam) = 4/20

P(Viagra | ham) = 1/80

Since the events {an email is spam} and {email is ham} form a partitioning of the sample space, we can apply the formula for the total probability to find the probability of “Viagra” occurring in an email:

P(Viagra) = P(Viagra | spam) * P (spam) + P(Viagra | ham) * P(ham) =

$$= \frac{4}{20} * \frac{20}{100} + \frac{1}{80} * \frac{80}{100} = 0.05$$

Finally, we can apply the Bayes’ rule to find the posterior probability that a message is spam if it contains the word “Viagra”:

$$P(\text{spam} \mid \text{Viagra}) = \frac{P(Viagra \mid spam) * P(spam)}{P(Viagra)} = \frac{\frac{4}{20} * \frac{20}{100}}{0.05} = 0.8$$

Since the events {an email is spam} and {email is ham} form a partitioning of the sample space, we can conclude that P(ham| Viagra) = 1 - 0.8 = 0.2. Since P(spam | Viagra) > P(ham| Viagra) , we can classify the message as spam.

Maximum a posteriori probability (MAP) estimate

Bayesian classification can be considered as a special case of a **maximum a posteriori probability (MAP) estimation.** MAP is used to estimate an unobserved population parameter θ on the basis of observations. If P(D | θ) is the probability of observing data D when the underlying population parameter is θ (called likelihood), and P(θ) is the prior probability of θ . Then, the value of the estimate is chosen so that to maximize the conditional probability of θ given data, P(θ | D), i.e., the posterior probability of θ given the data.

Example: suppose we want to select one hypothesis about data from a finite set of different competing hypotheses about data: H_1, H_2, .., H_n. Using MAP, we choose the hypothesis that maximizes:

$$P(H_i \mid D) = \frac{P(D \mid Hi) * P(Hi)}{P(D)}$$ for observed data D

Bayesian classifiers are a special case of MAP estimation, in which we have several competing classes and we choose the class that maximizes the conditional probability of observing the class given the data.

Naïve Bayes classifier

The examples of Bayesian classification we have used so far are toy examples because they used only one feature (the presence of “Viagra” or filling size). In addition, many realistic applications have more than two classes, not just the pos and neg; smap and ham; cancer and no cancer. What about data examples with multiple attributes? Suppose we want to apply the Bayes formula in domains where the examples are described by vectors of attributes such as $x=(x_1, x_2,\ldots, x_n)$. Also, suppose that we have multiple classes: $C_1, C_2, \ldots, C_k$. Then, the Bayes formula acquires the following form:

$$P(C_i \mid x_1, x_2,\ldots, x_n) = \frac{P(x_1, x_2,\ldots, x_n \mid C_i) * P(Ci)}{P(x_1, x_2,\ldots, x_n)}$$

We need to compute the value above for each C_i and choose the C_i with maximum value. The problem is, however, that computing $P(x_1, x_2,\ldots, x_n \mid Ci)$ for a large number of features is extremely difficult.

The Naive Bayes classifier is a special case of Bayes classifier that assumes **class-conditional independence**, i.e., all features are independent so long as they are conditioned on the same class value. This is a huge relief, allowing us to use the product rule for independent features:

Naïve Bayes classifier *(cont.)*

$P(x_1, x_2, \ldots, x_n \mid Ci) = P(x_1 \mid Ci) * P(x_2 \mid Ci) * \ldots * P(x_n \mid Ci) = \prod_{k=1}^{n} P(X_k \mid C_i)$

The computation of each $P(X_k \mid C_i)$ resembles the computations used in the toy examples. With the class-conditional independence assumption, the Bayes formula assumes the following form:

$$P(C_i \mid x_1, x_2, \ldots, x_n) = \frac{\prod_{k=1}^{n} P(X_k \mid Ci) * P(Ci)}{P(x_1, x_2, \ldots, xn)}$$

We choose C_i that maximizes the right hand side of the formula. Since the denominator on the right hand side does not depend on C_i, this is equivalent to selecting C_i that maximizes the numerator on the right hand side. In other words, an example will be labeled with C_i if this class maximizes the following version of the Bayes formula's numerator:

$$\prod_{k=1}^{n} P(X_k \mid Ci) * P(Ci)$$

Because of the class-conditional independence assumption, the classifier is called naïve.

Spam filter with Naïve Bayes

Let's extend the spam filter by adding a few additional words to be monitored in addition to the word Viagra: Money, Groceries, and Unsubscribe. The Naive Bayes learner is trained by constructing a likelihood table for the appearance of these four words (labeled *W1*, *W2*, *W3*, and *W4*) in 100 e-mails:

	Viagra (W_1)		Money (W_2)		Groceries (W_3)		Unsubscribe (W_4)		
Likelihood	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Total**
spam	4 / 20	16 / 20	10 / 20	10 / 20	0 / 20	20 / 20	12 / 20	8 / 20	20
ham	1 / 80	79 / 80	14 / 80	66 / 80	8 / 80	71 / 80	23 / 80	57 / 80	80
Total	5 / 100	95 / 100	24 / 100	76 / 100	8 / 100	91 / 100	35 / 100	65 / 100	100

Suppose, we have received an email that contains the words Viagra and Unsubscribe, but does not contain Money and Groceries (i.e., W1, ¬ W2, ¬ W3, W4). What is the posterior probability that the email is spam? In other words, P(spam | W1, ¬ W2, ¬ W3, W4) = ?

According to the Bayes formula:

$$P(\text{spam} \mid W_1, \neg W_2, \neg W_3, W_4) = \frac{P(W_1, \neg W_2, \neg W_3, W_4 \mid spam) * P(spam)}{P(W_1, \neg W_2, \neg W_3, W_4)}$$

Spam filter with Naïve Bayes *(cont.)*

We can apply the class-conditional independence assumption by replacing $P(W_1, \neg W_2, \neg W_3, W_4 \mid spam)$ with $P(W_1 \mid spam) * P(\neg W_2 \mid spam) * P(\neg W_3 \mid spam) * P(W_4 \mid spam)$. That is:

$P(spam \mid W1, \neg W2, \neg W3, W4) = \frac{P(W1 \mid spam) * P(\neg W2 \mid spam) * P(\neg W3 \mid spam) * P(W4 \mid spam) * P(spam)}{P(W_1, \neg W_2, \neg W_3, W_4)} =$

Using the frequency table, we find:

$P(W1 \mid spam) = \frac{4}{20}$

$P(\neg W2 \mid spam) = \frac{10}{20}$

$P(\neg W3 \mid spam) = \frac{20}{20}$

$P(W4 \mid spam) = \frac{12}{20}$

$P(spam) = 0.2$

Therefore, the numerator of $P(spam \mid W1, \neg W2, \neg W3, W4)$ is:

$$\frac{4}{20} * \frac{10}{20} * \frac{20}{20} * \frac{12}{20} * 0.2 = 0.012$$

Spam filter with Naïve Bayes *(cont.)*

Similarly,

P(ham | W1, ¬ W2, ¬ W3, W4) =

$$\frac{P(W1\,|ham)*P(\neg W2\,|ham)*P(\neg W3\,|\,ham)*P(W4\,|ham)*P(ham)}{P(W_1,\neg W_2,\neg W_3,W_4)}$$

Using the frequency table, we get:

P(W1 | ham) = $\frac{1}{80}$

P(¬ W2 | ham) = $\frac{66}{80}$

P(¬ W3 | ham) = $\frac{71}{80}$

P(W4 | ham) = $\frac{23}{80}$

P(ham) = 0.8

Therefore, the numerator of P(ham | W1, ¬ W2, ¬ W3, W4) is:

$$\frac{1}{80} * \frac{66}{80} * \frac{71}{80} * \frac{23}{80} * 0.8 = 0.002$$

Spam filter with Naïve Bayes *(cont.)*

Since the denominators of the posterior probabilities do not depend on the class (spam or ham) we can conclude that the posterior probability of spam is greater than the posterior probability of ham. Therefore, the message can be classified as spam. If you want to complete the computation of the posterior probabilities, you have to compute the common denominator of both probabilities. According to the rule of the total probability, the common denominator is equal to the sum of the two numerators, i.e.,

$P(W_1, \neg W_2, \neg W_3, W_4)$ = 0.012 + 0.002 = 0.014

Therefore, we can conclude that:

P(spam | W1, ¬ W2, ¬ W3, W4) = $\frac{0.012}{0.014} = 0.857$

P(ham | W1, ¬ W2, ¬ W3, W4) = $\frac{0.002}{0.014} = 0.143$

Once again, since P(spam | W1, ¬ W2, ¬ W3, W4) > P(ham | W1, ¬ W2, ¬ W3, W4), the email can be classified as spam.

Laplacian correction

The Laplacian correction (or Laplace estimator) is a way of dealing with zero probability values. Based on the class-conditional independence assumption, we use the estimation:

$P(x_1, x_2, \ldots, x_n \mid Ci) = \prod_{k=1}^{n} P(X_k \mid C_i)$

What if $P(X_k \mid C_i) = 0$ for some k and i, i.e., what if there is a class C_i and an attribute value X_k such that none of the samples in C_i has that attribute value. Then, the whole $P(x_1, x_2, \ldots, x_n \mid Ci)$ is zero even if the other multipliers are large.

Suppose, for example, that we want to classify an email containing all four words: Viagra, Groceries, Money, and Unsubscribe. According to the Naïve Bayes algorithm:

$$P(spam \mid W_1, W_2, W_3, W_4) = \frac{\frac{4}{20}*\frac{10}{20}*\frac{0}{20}*\frac{12}{20}*\frac{20}{100}}{\frac{4}{20}*\frac{10}{20}*\frac{0}{20}*\frac{12}{20}*\frac{20}{100}+\frac{1}{80}*\frac{14}{80}*\frac{8}{80}*\frac{23}{80}*\frac{80}{100}} = \frac{0}{0+0.00005} = 0$$

$$P(ham \mid W_1, W_2, W_3, W_4) = \frac{\frac{1}{80}*\frac{14}{80}*\frac{8}{80}*\frac{23}{80}*\frac{80}{100}}{\frac{4}{20}*\frac{10}{20}*\frac{0}{20}*\frac{12}{20}*\frac{20}{100}+\frac{1}{80}*\frac{14}{80}*\frac{8}{80}*\frac{23}{80}*\frac{80}{100}} = \frac{0.00005}{0+0.00005} = 1$$

Laplacian correction *(cont.)*

Because probabilities in the Naive Bayes formula are multiplied in a chain, the 0 value causes the posterior probability of spam to be zero, allowing the word Groceries to overrule all of the other evidence. This does not seem right! In general, it is statistically a bad idea to estimate the probability of some event to be zero just because you haven't seen it before in your training set.

The solution to having zero values in the chain formula is called the Laplace estimator. The idea of the Laplace estimator rests on the assumption that our training set is so large that adding one to each count would only make a negligible difference in the estimated probabilities, yet would avoid the case of zero probability values.

The formula for the Laplace estimator is:

$$\frac{old\ numerator + 1}{old\ denominator + w}$$

Where w is the number of features (4 in our example of spam filter). For example, the Laplace estimate for

P(Viagra | spam) is $\frac{4+1}{20+4} = \frac{5}{24}$.

In other words, we avoid having zero counts by adding one to the counts. This does not destroy too much probability estimates.

Laplacian correction *(cont.)*

Using the Laplace estimates, we get:

$P(W_1|spam)*P(W_2|spam)*P(W_3|spam)*P(W_4|spam)*P(spam) = \frac{5}{24} * \frac{11}{24} * \frac{1}{24} * \frac{13}{24} * \frac{20}{100} = 0.0004$

$P(W_1|ham)*P(W_2|ham)*P(W_3|ham)*P(W_4|ham)*P(ham) = \frac{2}{84} * \frac{15}{84} * \frac{9}{84} * \frac{24}{84} * \frac{80}{100} = 0.0001$

Therefore,

$P(spam\ |W_1, W_2, W_3, W_4) = \frac{0.0004}{0.0004+0.0001} = 0.80$

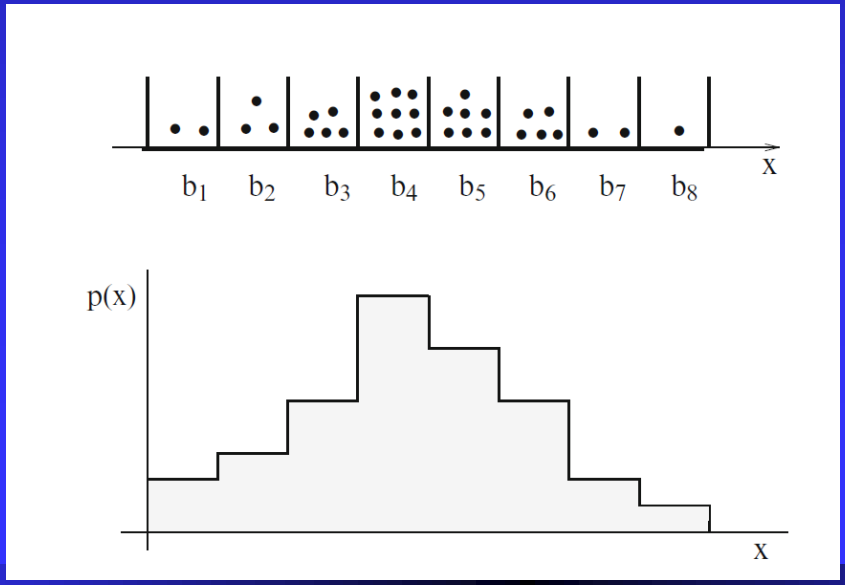
$P(ham\ |W_1, W_2, W_3, W_4) = \frac{0.0001}{0.0004+0.0001} = 0.20$

As a result, the message must be classified as spam.

Using numeric features with Naive Bayes

Because Naive Bayes uses frequency tables to learn the data, each feature must be categorical in order to create the combinations of class and feature values comprising of the matrix. Since numeric features do not have categories of values, Naive Bayes does not work directly with numeric data.

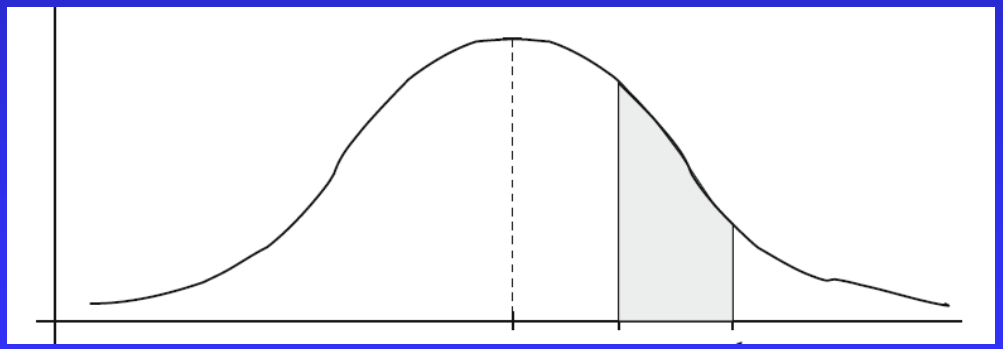
One possibility is to *discretize*, i.e., divide the range of all numeric values onto multiple intervals. Suppose we get ourselves a separate bin for each of these intervals, and place a little black ball into the *i*-th bin for each training example whose value falls into the *i*-th interval. Then, we will reach the following situation:



Using numeric features with Naive Bayes *(cont.)*

The upper part shows the bins, side by side and the *bottom* chart plots the histogram over the individual subintervals.

If the training set is infinitely large, we can, theoretically speaking, keep reducing the lengths of the intervals until they become infinitesimally small. The result of the bin-filling exercise will then no longer be a step-function, but rather a continuous function, such as the one shown below, which could be used as an estimate of the probability density function of the training set data.



Mobile phone spam filter

In the second half of this lecture, we will show how to use Naïve Bayes and R to train and test a mobile phone spam filter. Download the dataset from Blackboard and store it into your working directory. Then, read the data and store it into sms_raw variable as a data frame:

```
> sms_raw <- read.csv("sms_spam.csv", stringsAsFactors = FALSE)
```

We can get a brief look at the dataset using the *str()* function:

```
> str(sms_raw)
'data.frame':   5559 obs. of  2 variables:
 $ type: chr  "ham" "ham" "ham" "spam" ...
 $ text: chr  "Hope you are having a good week. Just checking in"
"K..give back my thanks." "Am also doing in cbe only. But have to pay."
"complimentary 4 STAR Ibiza Holiday or £10,000 cash needs your URGENT
collection. 09066364349 NOW from Landline not to lose out"| truncated
...
```

The dataset contains 5559 messages labeled “ham” or “spam”. The *text* element stores the raw messages. Since the labels are character strings, it is good to convert them into factors:

```
> sms_raw$type <- factor(sms_raw$type)
```

Using the *table()* function we can see that 747 of SMS messages in our data were labeled as spam:

```
> table(sms_raw$type)
 ham spam
4812  747
```

Mobile phone spam filter

Before we can create and run Bayes classifier, we need to preprocess text messages. For example, we need to delete punctuation and irrelevant words, such as “and”, “or”, “the”, etc. We also need to divide messages into words, extract words and build frequency tables. Most of the text processing functionality is provided in the package “tm” created by Ingo Feinerer. In order to use the package we must install and load it:

```
> install.packages("tm")
> library(tm)
```

A short manual of how to use the package written by Ingo Feinerer can be obtained by running:

```
> vignette("tm")
```

NLP stands for Natural Language Processing and the tm package supports many NLP functions. In general, NLP is done on big text collections called corpora. You can create a corpus in tm using the *VCorpus()* function, which creates “a volatile corpus”, i.e., a corpus which resides in working memory. The lifetime of a volatile corpus is the lifetime of the R variable that refers to the corpus. In contrast, the *PCorups()* function creates “ a permanent corpus”, which is stored in permanent memory.

Mobile phone spam filter *(cont.)*

The following function creates a corpus called sms_corpus:

```
> sms_corpus <- VCorpus(VectorSource(sms_raw$text))
```

The function *VectorSource()* creates a source object from the sms_raw$text vector and passes the source object to the *VCorpus()* function. Other possible sources are DataframeSource, DirSource, URISource, "XMLSource" , and ZipSource.

The *print()* function provides a brief summary of the corpus a whole:

```
> print(sms_corpus)
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 5559
```

The *inspect(sms_corpus)* function provides a brief summary of each element of the corpus. Because every corpus is a list, you can apply the *inspect()* function to specific elements:

```
> inspect(sms_corpus[5])
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 1
[[1]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 161
```

Mobile phone spam filter *(cont.)*

To view the content of the 5th message you can run the following function:

```
>  as.character(sms_corpus[[5]])
```

The following call (without the double brackets) yields:

```
>  as.character(sms_corpus[5])
[1] "list(list(content = \"okmail: Dear Dave this is your final
notice to collect your 4* Tenerife Holiday or #5000 CASH award! Call
09061743806 from landline. TCs SAE Box326 CW25WX 150ppm\", meta =
list(author = character(0), datetimestamp = list(sec =
4.5269570350647, min = 4, hour = 11, mday = 2, mon = 11, year = 115,
wday = 3, yday = 335, isdst = 0), description = character(0), heading
= character(0), id = \"5\", language = \"en\", origin =
character(0))))"
[2] "list()"
[3] "list()"
```

This shows that every component of the list is a list. Hence, the double brackets are needed.

To view several messages, we will apply *lapply()* function, which takes two arguments, a list and a function, and applies the function on each component of the list.

Mobile phone spam filter: text cleaning and standardization

```
> lapply(sms_corpus[5:6], as.character)
$`5`
[1] "okmail: Dear Dave this is your final notice to collect
your 4* Tenerife Holiday or #5000 CASH award! Call
09061743806 from landline. TCs SAE Box326 CW25WX 150ppm"
$`6`
[1] "Aiya we discuss later lar... Pick u up at 4 is it?"
```

Before we can extract words from the corpus, you must standardize it. For example, we must convert all uppercase characters to lowercase characters. We will use the function *tm_map()* which takes a corpus and a transformation function as attributes and applies the transformation function to each document in the corpus.

```
> sms_corpus_clean <- tm_map(sms_corpus,
+ content_transformer(tolower))
```

Here, we have applied *tolower()* function to each document in sms_corpus. The function *content_transformer()* is used as a wrapper function to get and set the content of each document. The result is stored in a new corpus called sms_corpus_clean.

Mobile phone spam filter: text cleaning and standardization

Text standardization also includes removing all numbers because they do not count as separate words:

```
> sms_corpus_clean <- tm_map(sms_corpus_clean, removeNumbers)
```

Here, the *removeNumbers()* function does not use a wrapper function to get and set documents because *removeNumbers()* belongs to the tm package.

The next task is to remove the stop words, i.e., words such as “and”, “or”, “to”, “a”, “the”, which are often irrelevant to text mining. Instead of creating a list of stop words, we will use the built-in function *stopwords(),* which returns a list of 174 stop words in English.

The list of stop words will be used as an argument of the built-in function *removeWords()*:

```
> sms_corpus_clean <- tm_map(sms_corpus_clean,
+ removeWords, stopwords())
```

The format of the tm_map function is a bit unusual: all arguments to the transformation (or mapping) function are listed after the transformation function. Here *stopwords()* is an argument of *removeWords()* and is listed right after *removeWords()*.

Mobile phone spam filter: text cleaning and standardization

The next function call removes all punctuation symbols from the corpus:

```
> sms_corpus_clean <- tm_map(sms_corpus_clean,
  + removePunctuation)
```

There is a caveat. After removing punctuation symbols, some words, which were previously separated by punctuation symbols can get concatenated in a single word:

```
> removePunctuation("Hey!John")
[1] "HeyJohn"
```

Instead of using *removePunctuation(),* one can use the following function, which replaces punctuation symbols with blank spaces:

```
> replacePunctuation <- function(x) {
+ gsub("[[:punct:]]+", " ", x)
+ }
```

Another important step in the standardization process is converting all inflected or derived words to their stems. This is called stemming. For example, the words "stemmer", "stemming", "stemmed" will all be reduced to "stem". Stemming allows different modifications of the same word to be counted as appearances of the same word. For example, "stemmer", "stemming", "stemmed" will all be counted as “stem”.

Mobile phone spam filter: text cleaning and standardization

We will use the *stemDocument()* function from SnowballC package to perform stemming:

```
> install.packages("SnowballC")
> library(SnowballC)
> sms_corpus_clean <- tm_map(sms_corpus_clean, stemDocument)
```

The last step in the cleaning and standardization process is removing additional white space using the stripWhitespace() function. The function removes only the additional white space, not the white space at all:

```
> stripWhitespace("Hello     World!     ")
[1] "Hello World! "
```

We will use once again tm_map to apply stripWhitespace to each document in the corpus:

```
> sms_corpus_clean <- tm_map(sms_corpus_clean, stripWhitespace)
```

This completes the cleaning and the standardization phase.

Mobile phone spam filter: splitting documents into words

The next phase is to split each document into words, a process called tokenization. The result is usually stored as a **document-term matrix** or **term-document matrix**. These are matrices that describe the frequency of terms that occur in a collection of documents. Suppose, for example, that we have the following two documents:

D1 = "ML is hard"

D2 = "ML is easy"

In the document-term matrix (DTM), rows represent documents and columns represent words (a.k.a terms):

	ML	is	hard	easy
D1	1	1	1	0
D2	1	1	0	1

The term-document matrix (TDM) is the inverse of the document-term matrix, i.e., rows represent words and columns represent documents.

Mobile phone spam filter: splitting documents into words

A DTM can be created using the *DocumentTermMatrix()* function:

```
> sms_dtm <- DocumentTermMatrix(sms_corpus_clean)
```

The *DocumentTermMatrix()* function provides optional arguments that allow data cleaning, standardization, and tokenization to be done in a single function call:

```
> sms_dtm2 <- DocumentTermMatrix(sms_corpus, control = list(
+ tolower = TRUE,
+ removeNumbers = TRUE,
+ stopwords =  TRUE,
+ removePunctuation = TRUE,
+ stemming = TRUE
+ ))
```

There is a slight difference between sms_stm2 and sms_dtm due to the fact that the default options of the *DocumentTermMatrix()* function (especially the removal of stop words) slightly differ from the way we have done it in sms_dtm.

Mobile phone spam filter: creating training and test datasets

The next phase is to create the training and test datasets. We'll divide the data into two portions: 75 percent for training and 25 percent for testing. This corresponds to 4,169 documents for training and 1,390 documents for testing. Assuming that the documents are stored randomly, we can take the first 4,160 documents for the training set and the remaining documents for the test set. Text corpora are very much like data frames, allowing row and column selection:

```
> sms_dtm_train <- sms_dtm[1:4169, ]
> sms_dtm_test <- sms_dtm[4170:5559, ]
```

The next two commands extract the labels from the raw data:

```
> sms_train_labels <- sms_raw[1:4169, ]$type
> sms_test_labels <- sms_raw[4170:5559, ]$type
```

Visualizing text data: word clouds

A **word or tag cloud** is a visual representation of text data, in which words are scattered around with more important words (more frequent, for example) shown with bigger font size and different color.

Source: peekaboo-vision.blogspot.com

Visualizing text data: word clouds

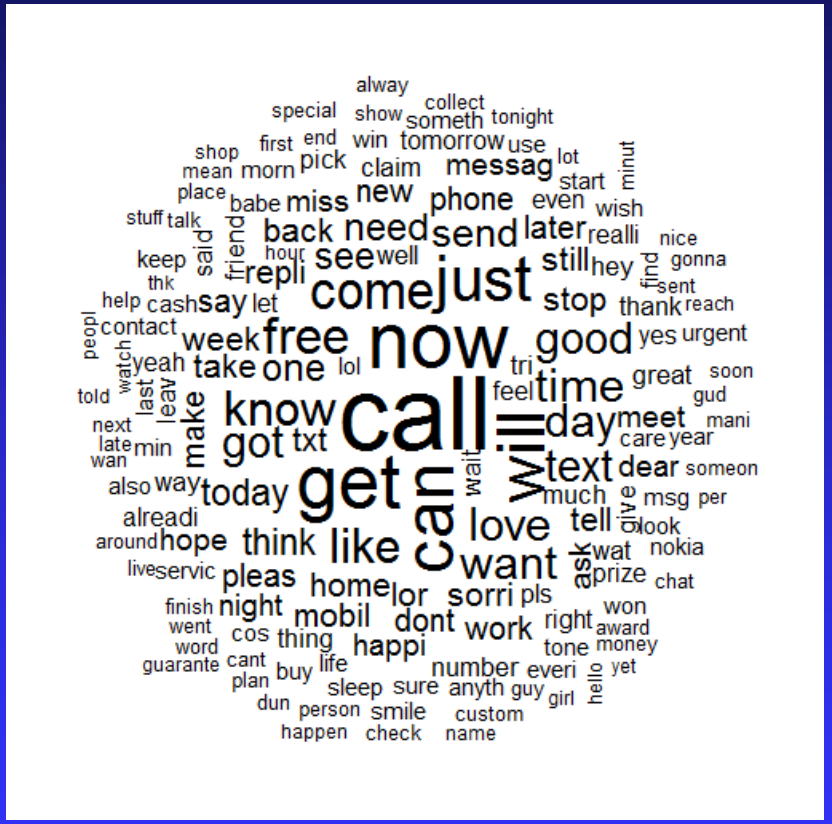
We will use the *wordcloud* package:

```
> install.packages("wordcloud")
> library(wordcloud)
```

A wordcloud can be created with a single function call:

```
> wordcloud(sms_corpus_clean,
+ min.freq = 50, + random.order =
+ FALSE)
```

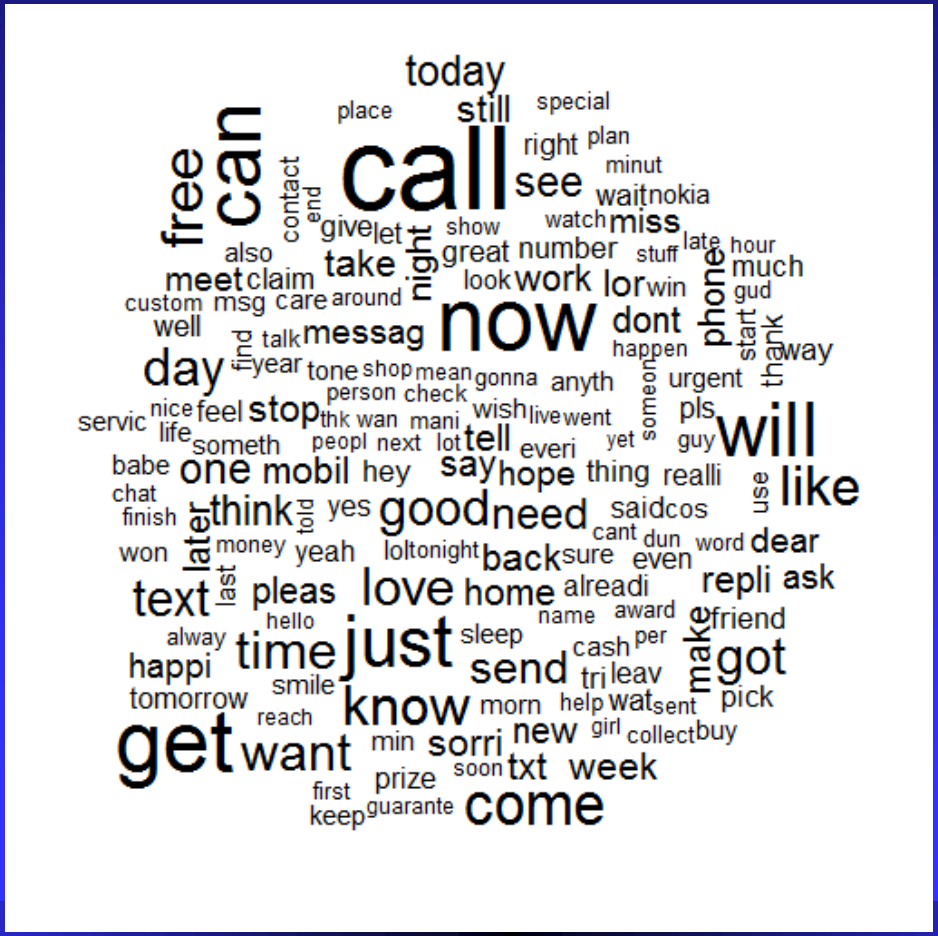
Because the minimum frequency is 50, words with frequency below 50 are not plotted. A random.order = FALSE indicates that the words are plotted in order of decreasing frequency.



Visualizing text data: word clouds *(cont.)*

If you set default random.order=TRUE:

```
> wordcloud(sms_corpus_clean, min.freq = 50)
```



Visualizing text data: word clouds *(cont.)*

Let's create word clouds for spam and ham and compare them. *The subset()* function extracts all records from a data frame (or a vector or matrix) that meet certain condition:

```
> spam <- subset(sms_raw, type == "spam")
> ham <- subset(sms_raw, type == "ham")
```

We have split the original data frame into two data frames, one that contains only spam messages and another that contains only ham messages.

```
> wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
> wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```

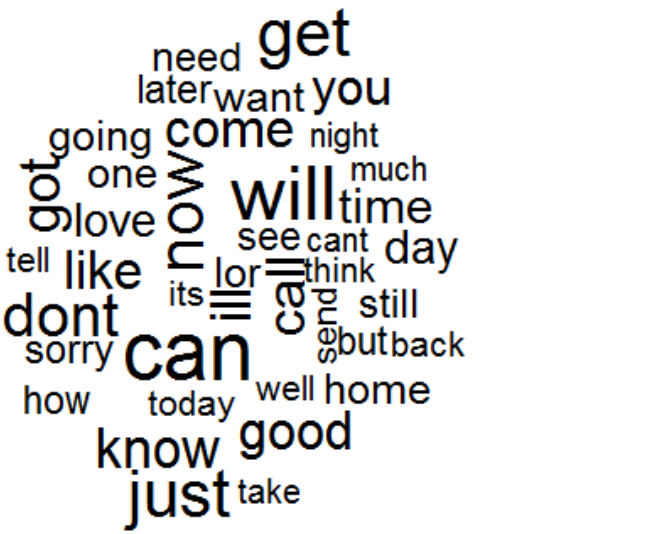
The argument max.words limits the number of plotted words to 40, i.e., only the top 40 most frequent words are plotted. The argument scale defines the range of the font size to be used.

One can see that the most frequent words for spam are "call", "mobile", "txt", "free", etc., whereas the most frequent words for ham are "can", "get", "will", "just", etc.

Visualizing text data: word clouds *(cont.)*

spam

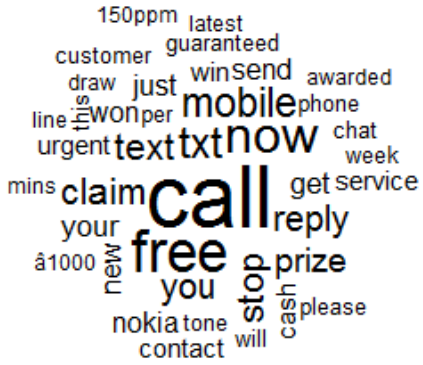
ham



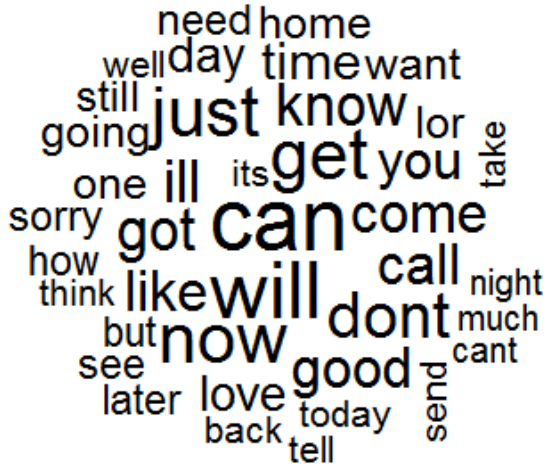
Visualizing text data: word clouds *(cont.)*

The word clouds for spam and ham without randomization:

spam



ham



Mobile phone spam filter: finding frequent words

The DTM we have created is sparse, i.e., most of its cells are zeroes. We will remove the words that appear no more than 5 times. First, we find the words that appear 5 time or more:

```
> sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
```

The sms_freq_words vector includes all words that appear at least 5 times. Then, we apply column selection using the vector sms_freq_words as column index to select only the columns with the most frequent words from the training and the test datasets:

```
> sms_dtm_freq_train<- sms_dtm_train[ , sms_freq_words]
> sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]
```

By doing this we have reduced the number of features (words) from about 7500 to 1180.

Since the Naive Bayes classifier is typically trained on data with categorical features (a feature is present or not), we will convert the numerical counts (of the appearances of a word in a document) in the DTM to categorical “Yes” or “No”. First, we define a function called *convert_counts()* which converts a numerical count to “Yes” if the count is greater than zero. Otherwise, the count is converted to “No”:

Mobile phone spam filter: finding frequent words

```
> convert_counts <- function(x) {
+ x <- ifelse(x > 0, "Yes", "No")
+ }
```

Then, we will apply the function *convert_counts()* to each column on the DTM. This will be done with the *apply()* function, which takes an array or matrix and a function as attributes and applies the function to each row or column:

```
> sms_train <- apply(sms_dtm_freq_train, MARGIN = 2,
+ convert_counts)
> sms_test <- apply(sms_dtm_freq_test, MARGIN = 2,
+ convert_counts)
```

MARGIN = 2 indicates that the function must be applied to each column, whereas MARGIN = 1 indicates that the function is applied to each row.

Mobile phone spam filter: training a model on the data

The Naive Bayes implementation we will use is the *naiveBayes()* function in the e1071 package:

```
> install.packages("e1071")
> library(e1071)
```

Naive Bayes classification syntax

using the `naiveBayes()` function in the `e1071` package

Building the classifier:

```
m <- naiveBayes(train, class, laplace = 0)
```

- `train` is a data frame or matrix containing training data
- `class` is a factor vector with the class for each row in the training data
- `laplace` is a number to control the Laplace estimator (by default, 0)

The function will return a naive Bayes model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "class")
```

- `m` is a model trained by the `naiveBayes()` function
- `test` is a data frame or matrix containing test data with the same features as the training data used to build the classifier
- `type` is either `"class"` or `"raw"` and specifies whether the predictions should be the most likely class value or the raw predicted probabilities

The function will return a vector of predicted class values or raw predicted probabilities depending upon the value of the `type` parameter.

Example:

```
sms_classifier <- naiveBayes(sms_train, sms_type)
sms_predictions <- predict(sms_classifier, sms_test)
```

Mobile phone spam filter: training a model on the data

Unlike the k-NN algorithm we used for classification in the previous chapter, a Naive Bayes learner is trained and used for classification in separate steps. First, we create a naïve Bayes model using the training data in the sms_train matrix and the vector of labels, sms_train_labels:

```
> sms_classifier <- naiveBayes(sms_train, sms_train_labels)
```

Then, we use the model to make predictions on the test data and save the vector of labels in sms_test_pred:

```
> sms_test_pred <- predict(sms_classifier, sms_test)
```

To evaluate the model performance we will create a crosstab on the actual and predicted data:

```
> library(gmodels)
> CrossTable(sms_test_pred, sms_test_labels,
+ prop.chisq = FALSE, prop.t = FALSE,
+ dnn = c('predicted', 'actual'))
```

The arguments *prop.chisq* = *FALSE* and *prop.t* = *FALSE* eliminate the unnecessary proportions in the crosstab. The *dnn* attribute renames the rows and columns.

Mobile phone spam filter: evaluating model performance *(cont.)*

Total Observations in Table: 1390

	actual		
predicted	ham	spam	Row Total
ham	1203	30	1233
	0.976	0.024	0.887
	0.997	0.164	
spam	4	153	157
	0.025	0.975	0.113
	0.003	0.836	
Column Total	1207	183	1390
	0.868	0.132	

Mobile phone spam filter: evaluating model performance *(cont.)*

The crosstab shows that the Bayes classifier made 34 mistakes, corresponding to an error rate of 2,4%. Out of these 34 mistakes, 4 are false positives (a ham message is predicted as spam) and 30 are false negatives (a spam message is predicted as ham). Different domains require a different balance between false positives and false negatives. In the cancer diagnosis domain, it could be more important to reduce false negatives than false positives because false negatives could result in an undiagnosed disease . In the spam filter domain, false positives could be more important than false negatives because false negatives could result in undelivered/lost email.

Mobile phone spam filter: improving model performance

We will try to improve the model by using a Laplacian correction (the model was previously built with laplace = 0 by default):

```
> sms_classifier2 <- naiveBayes(sms_train,
sms_train_labels,
+ laplace = 1)
```

After building the new model, we will use it to make predictions:

```
> sms_test_pred2 <- predict(sms_classifier2, sms_test)
```

Finally, we will compare the actual with predicted classes using a crosstab:

```
> CrossTable(sms_test_pred2, sms_test_labels,
+ prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
+ dnn = c('predicted', 'actual'))
```

The number of false positives remains the same, 4, whereas the number of false negatives is reduced from 30 to 27.

Mobile phone spam filter: improving model performance *(cont.)*

Total Observations in Table: 1390

	actual		
predicted	ham	spam	Row Total
ham	1203	27	1230
	0.997	0.148	
spam	4	156	160
	0.003	0.852	
Column Total	1207	183	1390
	0.868	0.132	

Naïve Bayes strengths and weaknesses

Strengths:

- Simple, easy to implement
- Can deal well with large feature spaces (due to the class-conditional independence assumption)
- Fast learning

Weaknesses:

- Not ideal for datasets with many numeric features
- Estimated probabilities are less reliable than the predicted classes
- Splits phrases into single words. For example, “Chicago bulls” will be split into “Chicago” and “Bulls”, thereby losing the meaning the phrase.
- Relies on an often-faulty assumption of equally important and independent features