Bayes Theorem

Bayes’ Theorem is based on the assumption of independence among predictors and can be used as classification technique. A Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

For example, a fruit may be considered to be an apple if it is green, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as ‘Naive’.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating the so called “posterior probability” *P(c|x)* from *P(c)*, *P(x)* and *P(x|c)*:

$$\underbrace{P(c \mid x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x \mid c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

- *P(c|x)* is the posterior probability of class (c, target) given predictor (x, attributes).
  Probability of the class after seeing the data
- *P(c)* is the prior probability of class.
  Probability of the class before seeing anything
- *P(x|c)* is the likelihood which is the probability of predictor given class.
  Conditional likelihood of the data given the class
- *P(x)* is the prior probability of predictor.
  Unconditional probability of the data

Pros and Cons of Naive Bayes

Pros

- Easy and fast
- Possible for a multi class prediction problem
- For independant features Naive Bayes classifier performs better compared to other models like logistic regression
- Less training data necessary
- Performs well in case of categorical input variables compared to numerical variable(s)

Cons

- “Zero Frequency”: If a categorical variable has a category (in the test data set), which was not observed in the training data set, then the model will assign a 0 (zero) probability and will be unable to make a prediction.
  Solution: Use smoothing technique (e.g. Laplace estimation)
- The predicted probability output could be completely wrong (“bad estimator”)
- Assumption of independent predictors is a limitation and quite unreal

Applications of Naive Bayes Algorithms

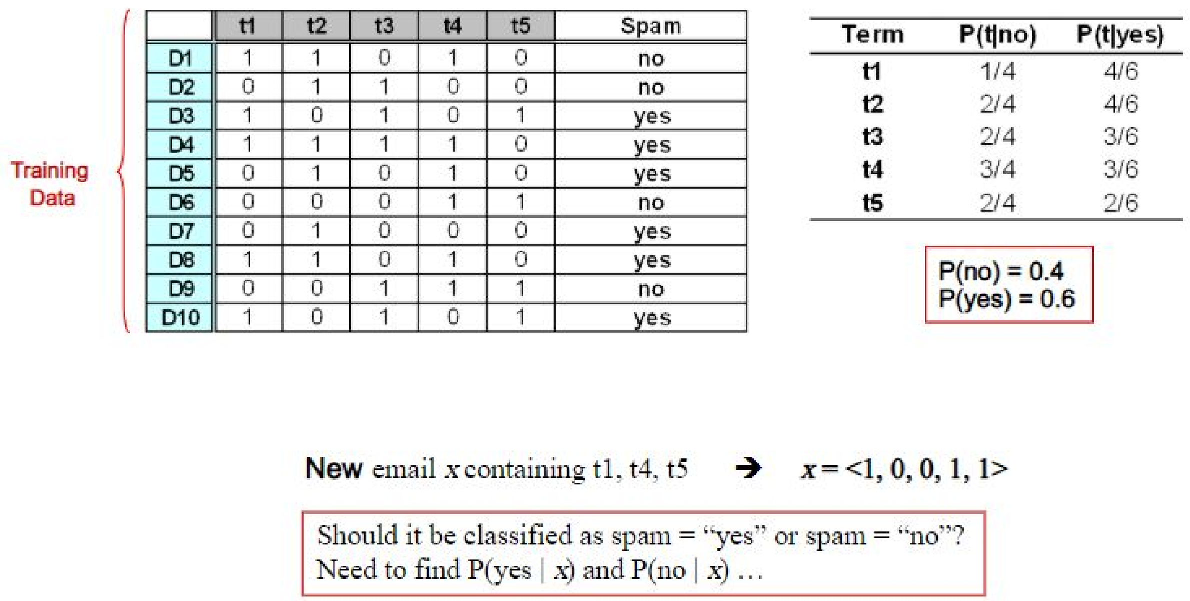
- Real time Prediction
- Multi class Prediction
- Text classification/ Spam Filtering/ Sentiment Analysis
- Recommendation System

Types of Naive Bayes Algorithms

There are different Naive Bayes algorithms:

- Multinomial Naive Bayes: Suitable for classification with discrete features
- Gaussian Naive Bayes: Assumes that the input data has a Gaussian (normal) distribution
- Bernoulli Naive Bayes: Assumes that the input data has a multivariate Bernoulli distribution

Example Spam detection



Training Data

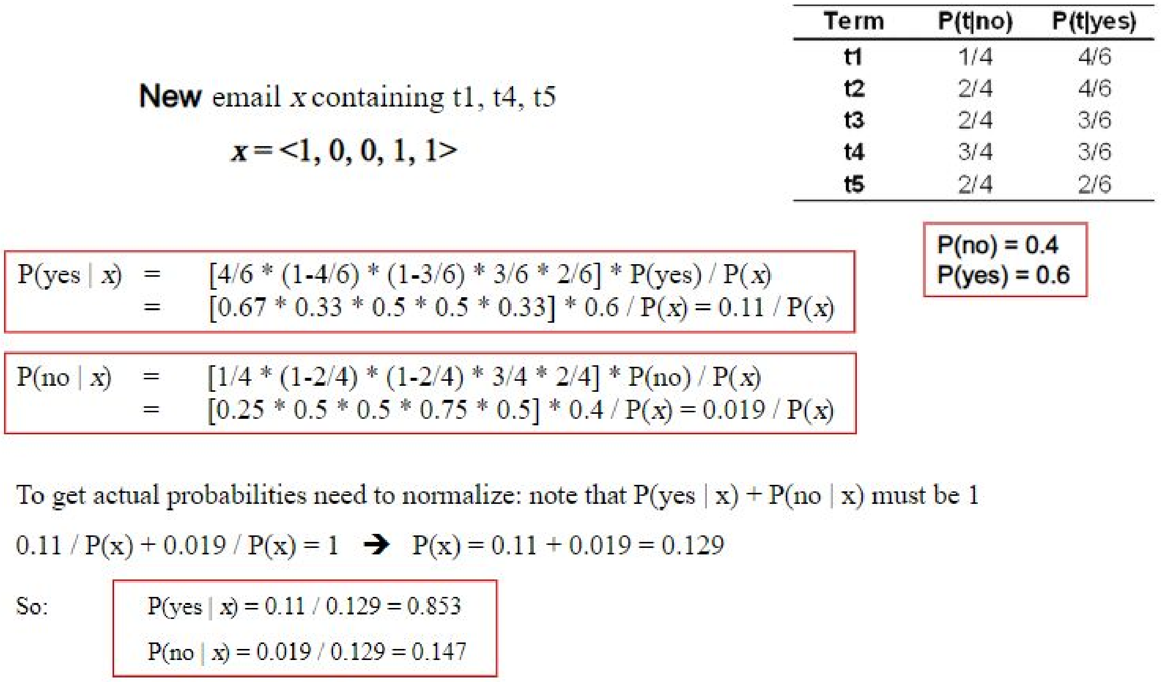
	t1	t2	t3	t4	t5	Spam
D1	1	1	0	1	0	no
D2	0	1	1	0	0	no
D3	1	0	1	0	1	yes
D4	1	1	1	1	0	yes
D5	0	1	0	1	0	yes
D6	0	0	0	1	1	no
D7	0	1	0	0	0	yes
D8	1	1	0	1	0	yes
D9	0	0	1	1	1	no
D10	1	0	1	0	1	yes

Term	P(t|no)	P(t|yes)
t1	1/4	4/6
t2	2/4	4/6
t3	2/4	3/6
t4	3/4	3/6
t5	2/4	2/6

P(no) = 0.4
P(yes) = 0.6

New email *x* containing t1, t4, t5 ➔ ***x*** **= <1, 0, 0, 1, 1>**

Should it be classified as spam = "yes" or spam = "no"?
Need to find P(yes | *x*) and P(no | *x*) …



New email *x* containing t1, t4, t5

x **= <1, 0, 0, 1, 1>**

Term	P(t|no)	P(t|yes)
t1	1/4	4/6
t2	2/4	4/6
t3	2/4	3/6
t4	3/4	3/6
t5	2/4	2/6

P(no) = 0.4
P(yes) = 0.6

P(yes | *x*) = [4/6 * (1-4/6) * (1-3/6) * 3/6 * 2/6] * P(yes) / P(*x*)
= [0.67 * 0.33 * 0.5 * 0.5 * 0.33] * 0.6 / P(*x*) = 0.11 / P(*x*)

P(no | *x*) = [1/4 * (1-2/4) * (1-2/4) * 3/4 * 2/4] * P(no) / P(*x*)
= [0.25 * 0.5 * 0.5 * 0.75 * 0.5] * 0.4 / P(*x*) = 0.019 / P(*x*)

To get actual probabilities need to normalize: note that P(yes | x) + P(no | x) must be 1

0.11 / P(x) + 0.019 / P(x) = 1 ➔ P(x) = 0.11 + 0.019 = 0.129

So:

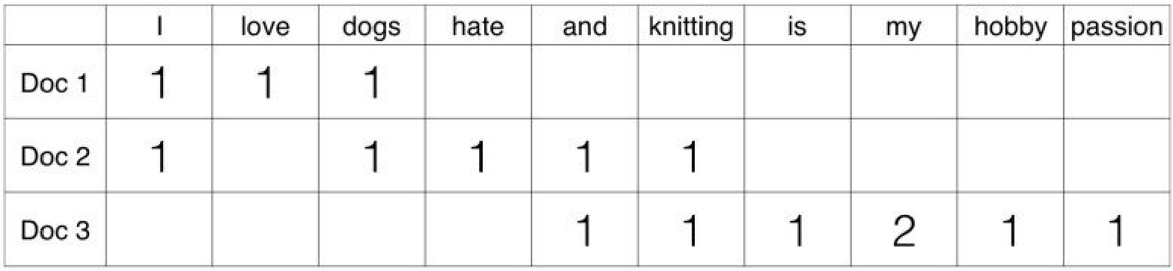
P(yes | *x*) = 0.11 / 0.129 = 0.853

P(no | *x*) = 0.019 / 0.129 = 0.147

Bag of Words (BoW)

The Bag of Words (BoW) concept works with a collection of text data. The basic idea is to take a piece of text and count the frequency of the words in that text. BoW concept treats each word individually and the order in which the words occur does not matter.

We do convert a collection of documents to a matrix, with each document being a row and each word (token) being the column, and the corresponding (row,column) values being the frequency of occurrence of each word or token in that document.



	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	2	1	1

The Scikit-learn function *CountVectorizer* converts a collection of text documents to a matrix of token counts:

1. It tokenizes the string (separates the string into individual words) and gives an integer ID to each token.
2. It counts the occurrence of each of those tokens.

CountVectorizer method automatically converts all tokenized words to their lower case, ignores all punctuation, and ignores all words that are in the so-called stop_words list (e.g. 'am', 'an', 'and', 'the').