Naïve Bayes algorithm

Dr. Johan Hagelbäck

johan.hagelback@lnu.se

http://aiguy.org

Linnæus University

Algorithms for classification

- There are several algorithms that can be for classification tasks:
  - Artificial Neural Networks
  - Support Vector Machines
  - k-Nearest Neighbor
  - Decision Trees
  - … and many more
- We will focus on the basic but common algorithm *Naïve Bayes*
- It has several benefits such as high speed, and is often very effective for text classification

Bayes' theorem

- First, we need to learn about Bayes' theorem
- It describes the probability of an event, based on prior knowledge of conditions that might be related to the event
- Bayes' theorem is stated using the following formula:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

- … where P(A|B) shall be interpreted as "probability that A occurs given B"
- It is best explained using an example:

Example

- We are interested in knowing if
  ”a stiff neck is a good sign of being a good FIFA player?”
- To answer this using Bayes’ theorem we need to know the prior probabilities:
  - 50% of the good FIFA players have a stiff neck:
    P(stiff | good) = 0.5
  - One in 50000 players is good at FIFA:
    P(good) = 1/50000
  - One in 20 players suffer from a stiff neck:
    P(stiff) = 1/20

Example

- We can now use the prior probabilities:
  - P(stiff | good) = 0.5   P(good) = 1/50000   P(stiff) = 1/20
- … to calculate the probability of being a good FIFA player if you have a stiff neck:

$$P(good|stiff) = \frac{P(stiff|good) \cdot P(good)}{P(stiff)}$$

$$P(good|stiff) = \frac{0.5 \cdot 1/50000}{1/20} = 0.0002$$

Example

- Given the prior probabilities:
  - P(stiff | good) = 0.5   P(good) = 1/50000   P(stiff) = 1/20
- … we can use Bayes' theorem to say that the probability that a player is good at FIFA if he has a stiff neck is 0.0002, or one in 5000 players

- So even if 50% of the good FIFA players have a stiff neck, it is not a good indicator of being good or bad at FIFA

Naïve Bayes

- Bayes' theorem only takes one attribute into consideration (stiff neck) when calculating the probability of belonging to a specific category (good FIFA player)
- In most real-world applications we have more than one attribute:
  - stiff neck
  - good gamepad
  - high resolution TV with a large screen
  - …
- We need a way of combining several inputs to get a probability of belonging to a specific category
- This is handled by the Naïve Bayes classifier

Naïve Bayes

- The classifier is called *naïve* because it assumes that the attributes are *independent* of each other
- It means that the probability of one attribute belonging to a specific category is completely unrelated to the probability of other attributes belonging to that category
- There are no relations between attributes!
- This is actually a false assumption for most tasks
- Example: ”money” is a better spam indicator if in combination with ”casino” than with ”programming”

Naïve Bayes

- The independence between attributes means that the actual probability calculated by the Naïve Bayes classifier is inaccurate
- You cannot say that the resulting probability is the *actual* probability that an example belongs to a category
- We can however *compare* the results of the example belonging to different categories, and see which category has the highest probability
- This works surprisingly well for many real-world classification problems

Multinomial Naïve Bayes

- We will first take a look at the *Multinomial Naïve Bayes* algorithm
- It works best when the inputs are categorical or text

- Let's start with an example:

Example dataset

Game pad?	Stiff neck?	Player skill
Great	Yes	Good
Average	Yes	Good
Junk	Yes	Good
Average	No	Good
Junk	No	Bad
Average	No	Bad
Great	Yes	Bad
Average	No	Bad
Average	No	Bad

Frequency table

- First step is to generate a frequency table:

Game Pad?			Stiff neck?			Player skill?	
	Good	Bad		Good	Bad	Good	Bad
Great	1	1	Yes	3	1	4	5
Average	2	3	No	1	4		
Junk	1	1					

Game pad?	Stiff neck?	Player skill
Great	Yes	Good
Average	Yes	Good
Junk	Yes	Good
Average	No	Good
Junk	No	Bad
Average	No	Bad
Great	Yes	Bad
Average	No	Bad
Average	No	Bad

Prior probabilities

- We continue filling the table with prior probabilities:

Game Pad?			Stiff neck?			Player skill?	
	Good	Bad		Good	Bad	Good	Bad
Great	1	1	Yes	3	1	4	5
Average	2	3	No	1	4		
Junk	1	1					
P(Great | x)	1/4	1/5	P(Yes | x)	3/4	1/5	4/9	5/9
P(Avg | x)	2/4	3/5	P(No | x)	1/4	4/5		
P(Junk | x)	1/4	1/5					

Classification

- The table is all we need for classification
- Now we can answer questions like:
  - A player has an average game pad and a stiff neck. Is he a good or bad player?
- We have two possible categories, *Good* or *Bad* player
- Let's calculate the probabilities of the above mentioned player belonging to the two categories:

Classification

- Classify the player:
  - {game pad = average, stiff neck = yes}
- Probability that the player is *Good*:

  P(Good) * P(average | Good) * P(yes | Good) = 4/9 * 2/4 * 3/4 = **0.1667**

- Probability that the player is *Bad*:

  P(Bad) * P(average | Bad) * P(yes | Bad) = 5/9 * 3/5 * 1/5 = **0.0667**

- Good has higher probability than Bad, so we classify the player as Good!

Another example

- Classify the player:
  - {game pad = great, stiff neck = no}
- Probability that the player is Good:

  P(Good) * P(great | Good) * P(no | Good) = 4/9 * 1/4 * 1/4 = **0.0278**

- Probability that the player is Bad:

  P(Bad) * P(great | Bad) * P(no | Bad) = 5/9 * 1/5 * 4/5 = **0.0889**

- Bad has higher probability than Good, so we classify the player as <u>Bad</u>!

Threshold

- In many applications it is better to return a ”don’t know” than a misclassified example
- For example in spam filtering, it is often desirable to avoid having non-spam emails end up in the spam folder than to catch every single spam message
- This can be solved by using a *threshold*
- A threshold of 3 means that the probability for the highest category must be at least 3 times higher than the probability of the other category, otherwise the classifier is unsure
- In our examples we used a threshold of 1, meaning that we always classify an example as the highest category regardless of the difference in probabilities

The examples using threshold

{game pad = average, stiff neck = yes}				
P(Good)	P(Bad)	Ratio	Threshold	Classified as
0.1667	0.0667	2.499	1	Good
0.1667	0.0667	2.499	3	Don't know

{game pad = great, stiff neck = no}				
P(Good)	P(Bad)	Ratio	Threshold	Classified as
0.0278	0.0889	3.198	1	Bad
0.0278	0.0889	3.198	3	Bad

Text classification

- In the examples we have seen so far we have had two nominal attributes:
  - Game pad: {great, average, junk}
  - Stiff neck: {yes, no}
- In text classification, we have to classify texts of different length
- To do this we first have to convert the text contents of each document to a *bag-of-words* (number of times each unique word is found in a text)
- Then we have to count the frequency and calculate the probability for each word belonging to each category
- Let's look at an example:

Example dataset

Text	Spam?
Buy cheap Rolex?	Yes
You want cheap Vicodin?	Yes
Can you buy milk?	No
Want candy tonight?	No
Gym tonight?	No

- The unique words are (special characters removed):
  - buy, cheap, rolex, you, want, vicodin, can, milk, candy, tonight, gym
- First step is to create a frequency matrix

Frequency table

						Spam?	
	Yes	No		Yes	No	Yes	No
buy	1	1	can	0	1	2	3
cheap	2	0	milk	0	1		
rolex	1	0	candy	0	1		
you	1	1	tonight	0	2		
want	1	1	gym	0	1		
vicodin	1	0					

Let's continue with the probabilities…

Prior probabilities

						Spam?	
	Yes	No		Yes	No	Yes	No
buy	1	1	can	0	1	2	3
cheap	2	0	milk	0	1		
rolex	1	0	candy	0	1		
you	1	1	tonight	0	2		
want	1	1	gym	0	1		
vicodin	1	0				2/5	3/5
P(buy | x)	1/2	1/3	P(can | x)	0/2	1/3		
P(cheap | x)	2/2	0/3	P(milk | x)	0/2	1/3		
P(rolex | x)	1/2	0/3	P(candy | x)	0/2	1/3		
P(you | x)	1/2	1/3	P(tonight | x)	0/2	2/3		
P(want | x)	1/2	1/3	P(gym | x)	0/2	1/3		
P(vicodin | x)	1/2	0/3					

Classification

- Now you want to classify the text
  ”buy cheap candy”
- As in the previous examples, we calculate the probability for being spam or not being spam:

P(yes) * P(buy | yes) * P(cheap | yes) * P(candy | yes) = 2/5 * 1/2 * 2/2 * 0/2 = 0

- Here we can see a problem: if a word has never showed up in a category, we multiply with a 0 and the result will always be 0…
- To solve this we can apply Laplace correction:

Laplace correction

- In Laplace correction we always add some constant value to each probability to avoid 0 probabilities
- If we use 1/3 as Laplace correction the probability for being spam looks like:

  P(yes) * P(buy | yes) * P(cheap | yes) * P(candy | yes) =
  = 2/5 * (1/2+1/3) * (2/2+1/3) * (0/2+1/3) =
  = 0.4 * 0.833 * 1.333 * 0.333 = **2.9**

- And for not being spam:

  P(no) * P(buy | no) * P(cheap | no) * P(candy | no) =
  = 3/5 * (1/3+1/3) * (0/3+1/3) * (1/3+1/3) =
  = 0.6 * 0.667 * 0.333 * 0.667 = **0.089**

- This message is classified as <u>spam</u>!

Spam or not?

- The text ”buy cheap candy” was clearly classified as spam
- Is this correct?
- The word that is most prominent in the result is ”cheap”, which exists in 2 of 2 spam and 0 of 3 non-spam messages
- If ”cheap” is not a good indicator for spam, we need more training data where cheap appears in non-spam messages
- We need quite large amounts of data for text classification to be accurate

Other variants of Naïve Bayes

- The approach described here is called *Multinomial Naïve Bayes*
- There are a number of other variants of Naïve Bayes, mainly *Gaussian* and *Bernoulli*
- In *Bernoulli*, we don't count the actual frequency of an attribute in a category
- Instead we use 1 if the attribute appears in any document belonging to the category, and 0 otherwise
- In *Gaussian*, we assume that attributes are numeric and follow a normal distribution
- It is used when inputs are numerical
- Let's take a look at how it works:

Example: reduced Iris dataset

Petal Length	Petal Width	Class
1.4	0.2	Iris-setosa
1.3	0.2	Iris-setosa
1.5	0.2	Iris-setosa
1.4	0.2	Iris-setosa
1.7	0.4	Iris-setosa
1.4	0.3	Iris-setosa
3.7	1.0	Iris-versicolor
3.9	1.2	Iris-versicolor
5.1	1.6	Iris-versicolor
4.5	1.5	Iris-versicolor
4.5	1.6	Iris-versicolor
4.7	1.5	Iris-versicolor

Training

- First, we divide the dataset into each category
- Then we calculate the mean value of each attribute for each category:

$$\bar{x} = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right) = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

- And the standard deviation (how much each value differs from the mean) of each attribute for each category:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2},$$

Training

Iris-setosa

	Petal Length	Petal Width
	1.4	0.2
	1.3	0.2
	1.5	0.2
	1.4	0.2
	1.7	0.4
	1.4	0.3
Mean	1.45	0.25
Stdev	0.14	0.08

Iris-versicolor

	Petal Length	Petal Width
	3.7	1.0
	3.9	1.2
	5.1	1.6
	4.5	1.5
	4.5	1.6
	4.7	1.5
Mean	4.40	1.40
Stdev	0.52	0.24

Classification

- To classify new examples, we need to calculate the probabilities of the input attributes belonging to each category using the *Gaussian Probability Density Function* (PDF):

  pdf(x_i, $mean_i$, std_i) = (1 / (sqrt(2 * PI) * std_i)) * e^(-((x_i - $mean_i$)^2)/(2 * std_i^2)))

- Next, we multiply the probabilities for all attributes for each category
- Each probability is then normalized by dividing with the sum of the probabilities for all categories
- We classify the example as the category with the highest probability

Note! The Gaussian (normal) distribution is available in many API's, Excel etc.

Classification

Iris-setosa

	Petal Length	Petal Width
Mean	1.45	0.25
Stdev	0.14	0.08
x	1.6	0.8
PDF	1.601	1.970e-09
P	3.154e-09	
P_{norm}	0.102	

Iris-versicolor

	Petal Length	Petal Width
Mean	4.40	1.40
Stdev	0.52	0.24
x	1.6	0.8
PDF	3.424e-07	0.081
P	2.776e-08	
P_{norm}	0.898	

The probability of the example belonging to Iris-versicolor is 0.898, so we classify this flower as an <u>Iris-versicolor</u>

Log probabilities

- The probability for each attribute belonging to a category is typically very small
- Multiplying lots of small values can lead to numerical underflow
- This is fixed by combining the log of the probabilities together
- This is done by:
    1. Transform attributes *x* and *y* as $ln(x)$ and $ln(y)$
       - natural logarithm, often called Math.log(x) or Math.log(x,Math.E)
    2. Perform the log equivalent to multiplication, which is addition:
       $ln(xy) = ln(x) + ln(y)$
    3. Transform the equivalent product $ln(xy)$ back into the original form:
       $e^{ln(xy)}$
       - often called Math.exp(value)

Log probabilities

Iris-setosa

	Petal Length	Petal Width
Mean	1.45	0.25
Stdev	0.14	0.08
x	1.6	0.8
PDF	1.601	1.970e-09
ln(PDF)	0.471	-20.045
∑ln(PDF)	-19.575	
$e^{\sum \ln(PDF)}$	3.154e-09	
P_{norm}	0.102	

Iris-versicolor

	Petal Length	Petal Width
Mean	4.40	1.40
Stdev	0.52	0.24
x	1.6	0.8
PDF	3.424e-07	0.081
ln(PDF)	-14.887	-2.512
∑ln(PDF)	-17.400	
$e^{\sum \ln(PDF)}$	2.776e-08	
P_{norm}	0.898	

Same result, but we avoid possible numerical underflow

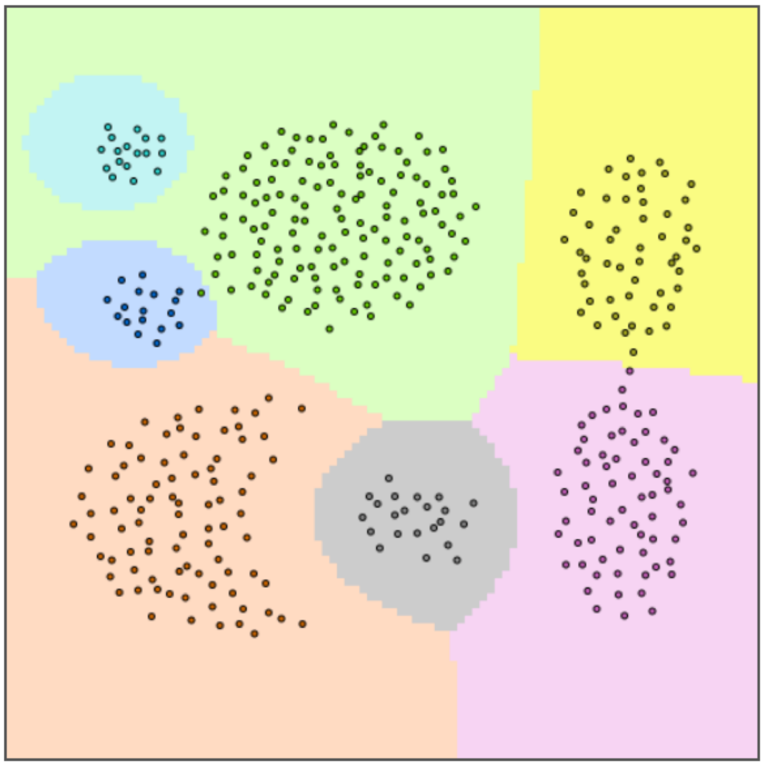
Summary

- Despite its simplicity, Naïve Bayes is often effective for text classification and numerical datasets that are not too complex
- Since it is very fast, you can use it as a baseline to compare performance of other, more complex algorithms

Test it on some datasets

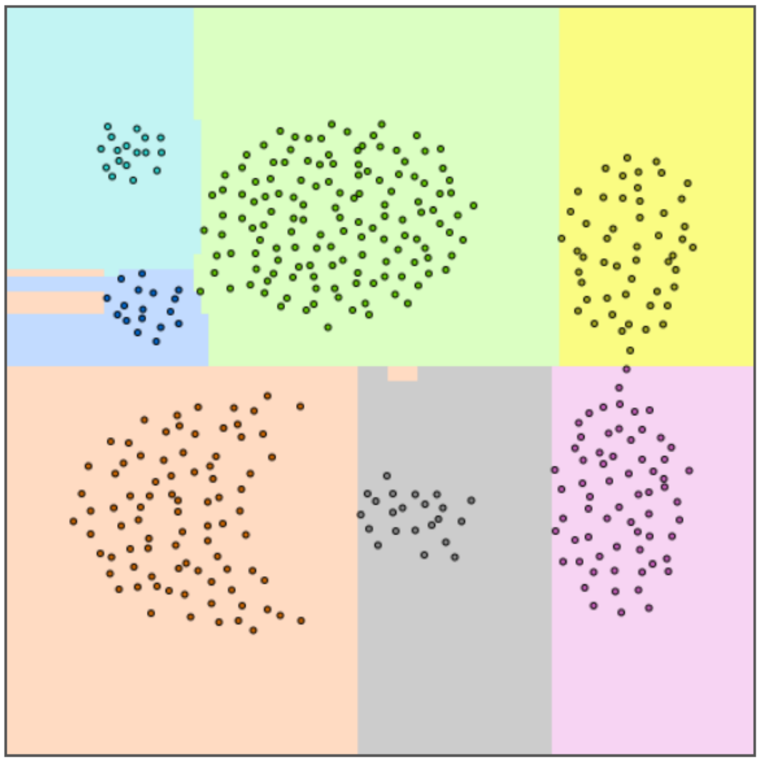
Naïve Bayes

Accuracy: 99.75% (393/394 correctly classified)



Random Forest

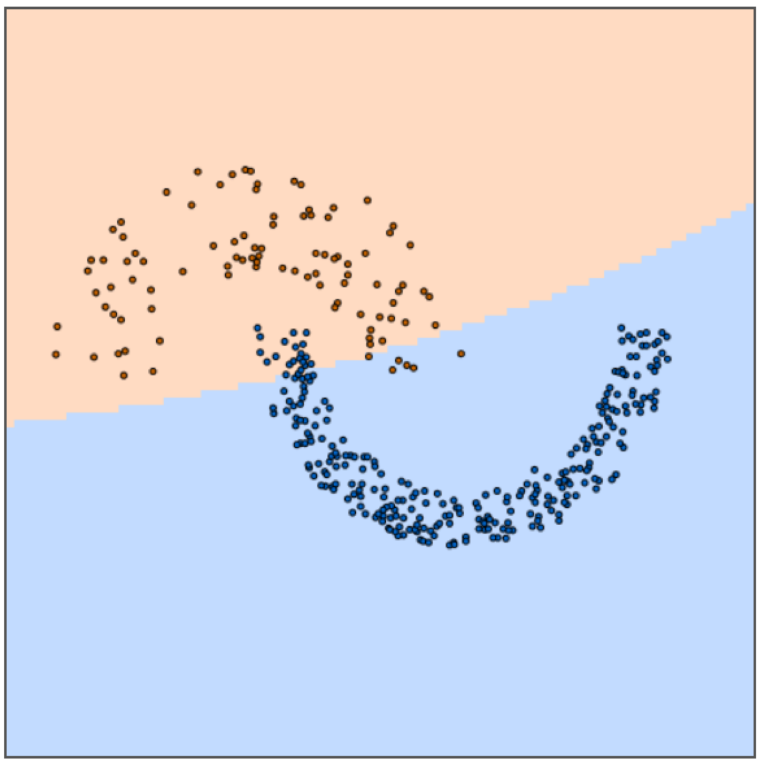
Accuracy: 100.00% (394/394 correctly classified)



Test it on some datasets

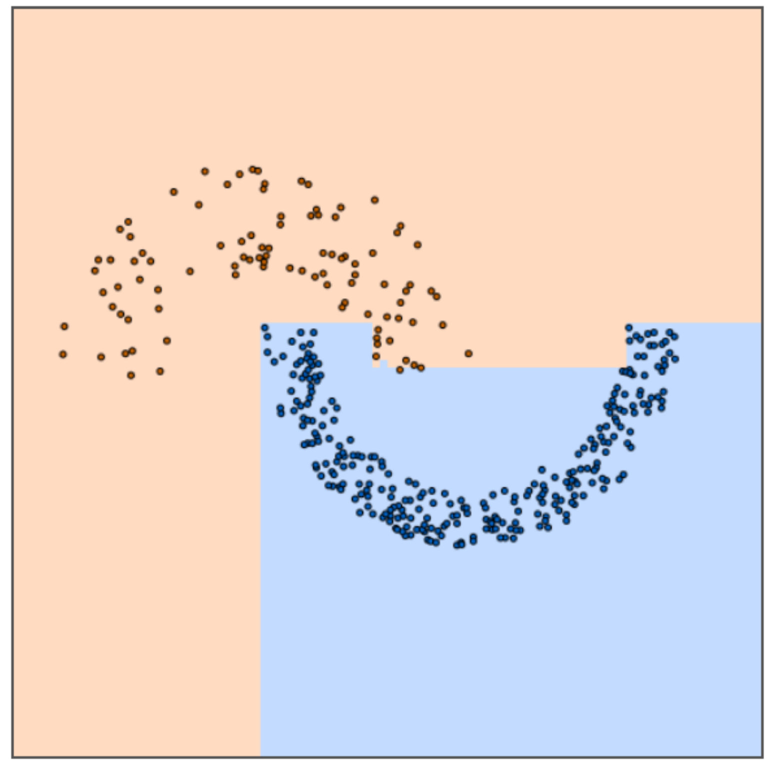
Naïve Bayes

Accuracy: 93.03% (347/373 correctly classified)



Random Forest

Accuracy: 100.00% (373/373 correctly classified)



Naïve Bayes algorithm

Dr. Johan Hagelbäck

johan.hagelback@lnu.se

http://aiguy.org