

# Funktionsweise & Struktur:

1. Ham Mails «lernen»
2. Spam Mails «lernen»
3. HashMaps «fixen»
4. Mails prüfen oder weitere Worte «lernen»

## BAYES-SPAM-Filter

von Sebastian Zimmermann

«Lernen» heisst...

1. Gesamten Ordner als Datei-Array einlesen
2. Pro Array-Element (d.h. Datei)...
3. ... Zeilenweise einlesen
4. ... Zeile teilen
5. ... Wort in Kleinbuchstaben temporär in HashMap mit Wert 1 speichern.

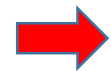
«Fixen» heisst...

1. HashMap mit Ham iterieren, Worte die in Spam nicht drin sind mit kleiner Anzahl hinzufügen
2. Analog für HashMap mit Spam

## P(Mail ist SPAM):

$$P(S|A \cap B) \stackrel{(1)}{=} \frac{P(A|S)P(B|S)}{P(A|S)P(B|S) + P(A|H)P(B|H)}$$

Kürzen!



$$\frac{\frac{\text{spamMap.get(key)}}{\text{spamCounter}}}{\frac{\text{spamMap.get(key)}}{\text{spamCounter}} + \frac{\text{hamMap.get(key)}}{\text{hamCounter}}} = \frac{1}{1 + \frac{P(A|H) * P(B|H)}{P(A|S) * P(B|S)}}$$



Kürzen des Codes auch notwendig!

$$\frac{1}{1 + \frac{\text{hamMap.get(key)} * \text{spamCounter}}{\text{hamCounter} * \text{spamMap.get(key)}}}$$



Pro Wort und Iteration rechnen, Teilfaktoren bilden um NaN zu vermeiden



PRO

- Double Nachkommastellen klein, somit NaN vermieden
- Kalibrieren möglich, durch HAM\_ASSUMPTION und SPAM\_ASSUMPTION

CONTRA

- HashMaps werden häufig iteriert, nicht sehr effizient

```
tmpMap.forEach((key, aDouble1) -> {  
    if(hamMap.containsKey(key) && spamMap.containsKey(key)) {  
        DENOMINATOR[0] = DENOMINATOR[0] * (hamMap.get(key)*spamCounter) / (hamCounter*spamMap.get(key));  
    }  
});  
/*  
the following line make the final calculation,  
considering that we would multiply by HAM_ASSUMPTION (e.g. 1/2=0.5) and  
divide by SPAM_ASSUMPTION (e.g. 1/2=0.5). Note that it's a division because of changing the equation.  
*/  
DENOMINATOR[0] = DENOMINATOR[0]*HAM_ASSUMPTION;  
DENOMINATOR[0] = DENOMINATOR[0] / SPAM_ASSUMPTION;  
return 1/(1 + DENOMINATOR[0]); // final equation
```

<= Finale Formel im Code, wie oben