Lab 6: Log probabilities and assignment 3

Data structures and Algorithms for CL III

Anna Dick, Lea Grüner

December 14, 2020

a2 reminders

- use Github productively, push after every step
- this also makes it easier to share the work with your partner and we can help you better if you have questions related to your code
- don't forget to tag your final submission as final, a comment is not a tag
  git tag final
  git push - -tags
  (or on github "tags", create a new release)

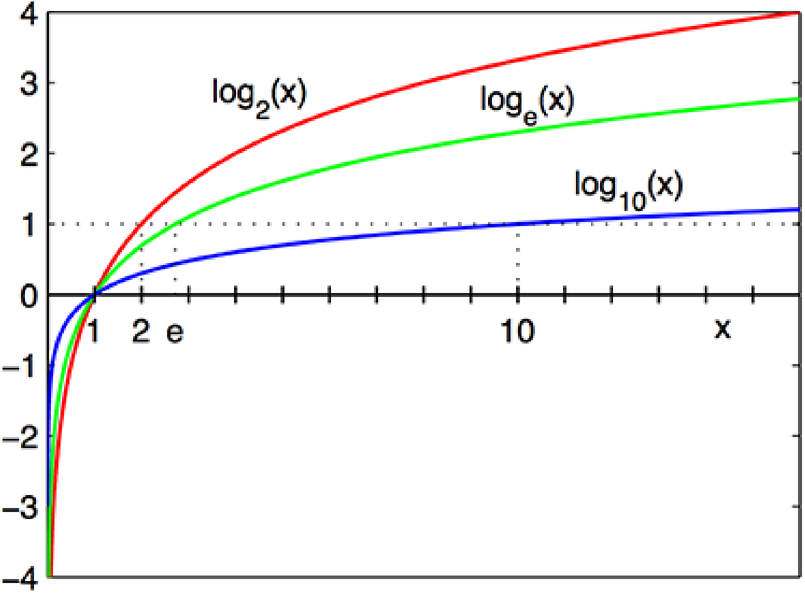
Logarithm refresher

The logarithm is the exponent to which a base has to be raised to produce x (inverse of exponentiation)

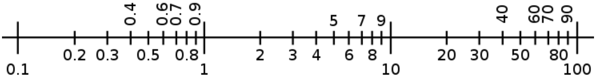
$\log_{10} 1000 = 3$
$\log_{2} \frac{1}{2} = -1$
$\log_{e} 10 = 2.30258509299$

Different mathematical properties like:
The logarithm of a product is the sum of the logarithms of the factors ($\log xy = \log x + \log y$)

Logarithm plots with common bases



Log scale and log probability



- The logarithmic scale makes it much easier to compare and visualize values that cover a wide range like exponential growth
- Log probability is not presented on a standard [0,1] interval but on the logarithmic scale
- the logarithm is undefined for 0, so only non-zero probabilities

Why use log probabilities?

- Numerical stability is improved for very small numbers (probabilities of unlikely words in large corpora)
- Faster runtime because addition is less expensive than multiplication
- standard practice in NLP applications

Calculating log probabilities (in Python)

- Numpy has a natural log function np.log()
- Remember that there is no multiplication in log space: Independent events are not multiplied but added
- use np.exp() to get the regular probability

```
# if w is known
# (1-a)*f(w)
return np.log(1 - a) + np.log(words_counts[word] / nwords)

# if w is not known
# a * product of all f(l) in w
logprob = 0
for l in word:
    logprob += np.log(letter_counts[l] / nletters)
return np.log(self.a) + logprob

# alternative: dealing with unknown letters (not required)
np.log(letter_counts.get(l, 1) /(nletters + len(letter_counts))
```

Assignment 3: Sorting

- implement insertion sort, quicksort and lexicographic sort
- compare runtimes using lists of random words

Quicksort with median-of-3 and cutoff

- quicksort is based on a pivot element, all other values are compared to the pivot
- choosing a better pivot than some arbitrary element decreases run-time
- Median of three: Compare the values of three indices (first, last and middle) and take the median value as pivot
- In a sequence [5, 7, 3, 2, 6, 1, 4] the first value is 5, the middle is 2 and the last is 4 ->pick 4 as pivot
- Cutoff: Once the portion you sort is smaller than a specified cutoff length, sort it with insertion sort

Lexicographic Sorting

- When sorting words, ordering is not as easy as for numbers
- Lexicographic order: like in a lexicon, first letter has highest priority, last letter the least
- For a sequence [people, ball, tree] the order can be specified based on the first letter [ball, people, tree]
- But: For a sequence [baker, baking] the 4th letter is deciding, for a sequence [tutorial, tutoring] the 7th letter is deciding
- For a3, you should implement this letter by letter sort, such that you end up with the word order as it would be in a lexicon (with algorithms from class, not predefined functions/ libraries)

Links

Log rules: https://www.youtube.com/watch?v=o4GWKTr8SVQ&ab_channel=studytimenz

Quicksort with hungarian folk dance:
https://www.youtube.com/watch?v=ywWBy6J5gz8&t=108s&ab_channel=AlgoRythmics