

lecture_14.py



```
1 from dataclasses import dataclass
2 import math
3 import torch
4 import torch.nn as nn
5 from torch.nn.functional import softmax
6 import numpy as np
7 import kenlm
8 import fasttext
9 import itertools
10 import mmh3
11 from bitarray import bitarray
12 from basic_util import count, repeat
13 from file_util import download_file
14 from execute_util import text, image, link
15 from lecture_util import article_link, named_link
16 from references import dolma
17
18 def main():
19     Last lecture: overview of datasets used for training language models
20     • Live service (GitHub) → dump/crawl (GH Archive) → processed data (The Stack)
21     • Processing: HTML to text, language/quality/toxicity filtering, deduplication
22
23     This lecture: deep dive into the mechanics
24     • Algorithms for filtering (e.g., classifiers)
25     • Applications of filtering (e.g., language, quality, toxicity)
26     • Deduplication (e.g., Bloom filters, MinHash, LSH)
27
28     filtering_algorithms()
29     filtering_applications()
30     deduplication()
31
32
```

Summary

```
33     • Algorithmic tools: n-gram models (KenLM), classifiers (fastText), importance resampling (DSIR)
34     • Applications: language identification, quality filtering, toxicity filtering
35     • Deduplication: hashing scales to large datasets for fuzzy matching
36     • Now you have the tools (mechanics), just have to spend time with data (intuitions)
37
38
39 def filtering_algorithms():
40     Algorithmic building block:
41     • Given some target data T and lots of raw data R, find subset T' of R similar to T.
```

42



43

44 Desiderata for filtering algorithm:

- 45 • Generalize from the target data (want T and T' to be different)
- 46 • Extremely fast (have to run it on R, which is huge)

47

```

48 kenlm_main()      # Train n-gram model
49 fasttext_main()   # Train a classifier
50 dsir_main()       # Train bag of n-grams model, do importance resampling
51 filtering_summary()

```

52

53 Survey paper on data selection [\[Albalak+ 2024\]](#)

54

55

56 `def kenlm_main():`57 **n-gram model with Kneser-Ney smoothing** [\[article\]](#)

- 58 • KenLM: fast implementation originally for machine translation [\[code\]](#)
- 59 • Common language model used for data filtering
- 60 • Extremely simple / fast - just count and normalize

61

62

Concepts

63 Maximum likelihood estimation of n-gram language model:

- 64 • $n = 3$: $p(\text{in} \mid \text{the cat}) = \text{count}(\text{the cat in}) / \text{count}(\text{the cat})$

65 Problem: sparse counts (count of many n-grams is 0 for large n)

66 Solution: Use Kneser-Ney smoothing to handle unseen n-grams [\[article\]](#)

- 67 • $p(\text{in} \mid \text{the cat})$ depends on $p(\text{in} \mid \text{cat})$ too

68

69 # Download a KenLM language model

70 `model_url = "https://huggingface.co/edugp/kenlm/resolve/main/wikipedia/en.arpa.bin"`71 `model_path = "var/en.arpa.bin"`72 `download_file(model_url, model_path)`73 `model = kenlm.Model(model_path)`

74

75 # Use the language model

76 `def compute(content: str):`77 `# Hacky preprocessing`78 `content = "<s> " + content.replace(",", " ").replace(".", ". ") + "</s>"`

79

80 `# log p(content)`81 `score = model.score(content)`

82

83 `# Perplexity normalizes by number of tokens to avoid favoring short documents`84 `num_tokens = len(list(model.full_scores(content)))`85 `perplexity = math.exp(-score / num_tokens)`

86

```

87         return score, perplexity
88
89     score, perplexity = compute("Stanford University was founded in 1885 by Leland and Jane Stanford as a tribute to the
memory of their only child, Leland Stanford Jr.") # @inspect score, @inspect perplexity
90     score, perplexity = compute("If you believe that the course staff made an objective error in grading, you may submit a
regrade request on Gradescope within 3 days after the grades are released.") # @inspect score, @inspect perplexity
91     score, perplexity = compute("asdf asdf asdf asdf asdf") # @inspect score, @inspect perplexity
92     score, perplexity = compute("the the the the the the the the the the the the the the the the the the") # @inspect score,
@inspect perplexity
93
94

```

CCNet

```

95 [Wenzek+ 2019]
96 • Items are paragraphs of text
97 • Sort paragraphs by increasing perplexity
98 • Keep the top 1/3
99 • Was used in LLaMA
100
101 Summary: Kneser-Ney n-gram language models (with KenLM implementation) is fast but crude
102
103

```

```

104 def fasttext_main():
105     fastText classifier [Joulin+ 2016]
106     • Task: text classification (e.g., sentiment classification)
107     • Goal was to train a fast classifier for text classification
108     • They found it was as good as much slower neural network classifiers
109
110

```

Baseline: bag of words (not what they did)

```

111 L = 32 # Length of input
112 V = 8192 # Vocabulary size
113 K = 64 # Number of classes
114 W = nn.Embedding(V, K) # Embedding parameters (V x K)
115 x = torch.randint(V, (L,)) # Input tokens (L) - e.g., ["the", "cat", "in", "the", "hat"]
116 y = softmax(W(x).mean(dim=0)) # Output probabilities (K)
117 Problem: V*K parameters (could be huge)
118
119

```

fastText classifier: bag of word embeddings

```

120 H = 16 # Hidden dimension
121 W = nn.Embedding(V, H) # Embedding parameters (V x H)
122 U = nn.Linear(H, K) # Head parameters (H x K)
123 y = softmax(U(W(x).mean(dim=0))) # Output probabilities (K)
124 Only H*(V + K) parameters
125
126 Implementation:
127 • Parallelized, asynchronous SGD
128 • Learning rate: linear interpolation from [some number] to 0 [article]
129
130

```

Bag of n-grams

```

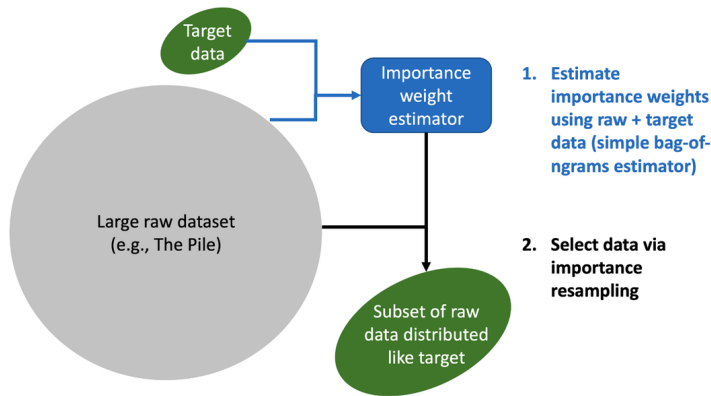
131 x = ["the cat", "cat in", "in the", "the hat"] # @inspect x
132 Problem: number of bigrams can get large (and also be unbounded)
133 Solution: hashing trick
134 num_bins = 8 # In practice, 10M bins
135 hashed_x = [hash(bigram) % num_bins for bigram in x] # @inspect hashed_x

```

- For quality filtering, we have $K = 2$ classes (good versus bad)
 - In that case, fastText is just a linear classifier ($H = K = 2$)
- In general, can use any classifier (e.g., BERT, Llama), it's just slower

```
def dsir_main():
```

Data Selection for Language Models via Importance Resampling (DSIR) [Xie+ 2023]



```
importance_sampling()
```

Setup:

- Target dataset D_p (small)
- Proposal (raw) dataset D_q (large)

Take 1:

- Fit target distribution p to D_p
- Fit proposal distribution q to D_q
- Do importance resampling with p , q , and raw samples D_q

Problem: target data D_p is too small to estimate a good model

Take 2: use hashed n-grams

```
training_text = "the cat in the hat"
```

```
# Hash the n-grams
```

```
num_bins = 4
```

```
def get_hashed_ngrams(text: str):
```

```
    ngrams = text.split(" ") # Unigram for now
```

```
    return [hash(ngram) % num_bins for ngram in ngrams]
```

```
training_hashed_ngrams = get_hashed_ngrams(training_text) # @inspect training_hashed_ngrams
```

```
# Learn unigram model
```

```
probs = [count(training_hashed_ngrams, x) / len(training_hashed_ngrams) for x in range(num_bins)] # @inspect probs
```

```
# Evaluate probability of any sentence
```

```
hashed_ngrams = get_hashed_ngrams("the text") # @inspect hashed_ngrams
```

```
prob = np.prod([probs[x] for x in hashed_ngrams]) # @inspect prob
```

Result: DSIR slightly better than heuristic classification (fastText) on the GLUE benchmark

	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	Avg
Random selection	82.63 _{0.41}	86.90 _{0.28}	89.57 _{0.30}	67.37 _{1.69}	90.05 _{0.41}	87.40 _{1.08}	49.41 _{3.67}	88.63 _{0.11}	80.25
Heuristic classification	82.69 _{0.17}	85.95 _{0.79}	89.77 _{0.32}	68.59 _{1.75}	88.94 _{0.98}	86.03 _{0.93}	48.17 _{3.19}	88.62 _{0.22}	79.85
Top- k Heuristic classification	83.34 _{0.22}	88.62 _{0.24}	89.89 _{0.19}	70.04 _{0.99}	91.15 _{0.76}	86.37 _{1.00}	53.02 _{3.56}	89.30 _{0.11}	81.47
DSIR	83.07 _{0.29}	89.11 _{0.14}	89.80 _{0.37}	75.09 _{2.76}	90.48 _{0.57}	87.70 _{0.68}	54.00 _{1.34}	89.17 _{0.13}	82.30
Top- k DSIR	83.39 _{0.06}	88.63 _{0.38}	89.94 _{0.17}	72.49 _{1.29}	91.01 _{0.79}	86.18 _{1.12}	49.90 _{1.10}	89.52 _{0.21}	81.38

```

178
179 Comparison with fastText:
180 • Modeling distributions is a more principled approach capturing diversity
181 • Similar computation complexity
182 • Both can be improved by better modeling
183
184
185 def importance_sampling():
186     Setup:
187     • Target distribution p (want samples from here)
188     • Proposal distribution q (have samples from here)
189
190     vocabulary = [0, 1, 2, 3]
191     p = [0.1, 0.2, 0.3, 0.4]
192     q = [0.4, 0.3, 0.2, 0.1]
193
194     # 1. Sample from q
195     n = 100
196     samples = np.random.choice(vocabulary, p=q, size = n) # @inspect samples
197     Samples (q): [0 2 0 2 1 1 0 1 0 1 0 0 0 0 1 1 0 3 1 1 1 1 1 1 3 1 0 2 0 1 3 0 2 1 1 2 0 0 0 3 2 1 1 0 1 1 0 3 2 0 2 0 1
198     0 1 2 2 2 0 0 0 0 2 1 0 2 0 1 3 0 0 0 0 0 0 0 1 1 2 2 2 1 0 1 1 0 1 0 1 0 1 1 3 3 0 1 0 0 2 0]
199
200     # 2. Compute weights over samples (w \propto p/q)
201     w = [p[x] / q[x] for x in samples] # @inspect w
202     z = sum(w) # @inspect z
203     w = [w_i / z for w_i in w] # @inspect w
204
205     # 3. Resample
206     samples = np.random.choice(samples, p=w, size=n) # @inspect samples
207     Resampled (p): [2 2 1 3 3 2 0 3 3 2 1 0 3 2 0 3 3 1 3 1 3 2 3 2 3 2 3 1 0 3 2 2 2 0 2 1 2 0 3 1 1 1 3 1 3 3 3 1 0 2
208     3 1 2 1 2 2 2 2 1 1 0 2 1 1 0 2 3 2 1 3 2 3 1 2 3 2 3 2 1 1 3 3 3 1 1 3 2 1 3 1 3 1 0 3 1 1 2 3 2 1]
209
210 def filtering_summary():
211     Implementations: KenLM, fastText, DSIR
212

```

General framework

```

213 Given target T and raw R, find subset of R similar to T
214 1. Estimate some model based on R and T and derive a scoring function
215 2. Keep examples in R based on their score
216
217

```

Instantiations of the framework

```

218
219 Generative model of T (KenLM):
220 1.  $\text{score}(x) = p_T(x)$ 
221 2. Keep examples x with  $\text{score}(x) \geq \text{threshold}$  (stochastically)
222
223 Discriminative classifier (fastText):
224 1.  $\text{score}(x) = p(T | x)$ 
225 2. Keep examples x with  $\text{score}(x) \geq \text{threshold}$  (stochastically)
226
227 Importance resampling (DSIR):
228 1.  $\text{score}(x) = p_T(x) / p_R(x)$ 
229 2. Resample examples x with probability proportional to  $\text{score}(x)$ 
230
231

```

```

232 def filtering_applications():
233     The same data filtering machinery can be used for different filtering tasks.
234     language_identification()
235     quality_filtering()
236     toxicity_filtering()
237
238
239 def language_identification():
240     Language identification: find text of a specific language (e.g., English)
241
242     Why not just go multilingual?
243     • Data: difficult to do curation / processing of high-quality data in any given language
244     • Compute: in compute-limited regime, less compute/tokens dedicated to any given language
245     Models differ on multilinguality:
246     • English was only 30% of BLOOM (was undertrained), English performance suffered [Laurençon+ 2023]
247     • Most frontier models (GPT-4, Claude, Gemini, Llama, Qwen) are heavily multilingual (sufficiently trained)
248
249     fastText language identification [article]
250     • Off-the-shelf classifier
251     • Supports 176 languages
252     • Trained on multilingual sites: Wikipedia, Tatoeba (translation site) and SETimes (Southeast European news)
253
254     Example: Dolma keeps pages with p(English) >= 0.5 [Soldaini+ 2024]
255
256     # Download the model
257     model_url = "https://dl.fbaipublicfiles.com/fasttext/supervised-models/lid.176.bin"
258     model_path = "var/lid.176.bin"
259     download_file(model_url, model_path)
260     model = fasttext.load_model(model_path)
261
262     # Make predictions
263     predictions = model.predict(["The quick brown fox jumps over the lazy dog."]) # English @inspect predictions
264     predictions = model.predict(["The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy
dog."]) # Duplicate @inspect predictions
265     predictions = model.predict(["OMG that movie was 🔥! So dope 😎!"]) # Informal English @inspect predictions
266     predictions = model.predict(["Auf dem Wasser zu singen"]) # German @inspect predictions
267     predictions = model.predict(["The quadratic formula is  $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ "]) # Latex @inspect predictions
268     predictions = model.predict(["for (int i = 0; i < 10; i++)"]) # C++ @inspect predictions
269     predictions = model.predict(["Hello!"]) # English @inspect predictions
270     predictions = model.predict(["Bonjour!"]) # French @inspect predictions
271     predictions = model.predict(["Feliz Navidad / Próspero año y felicidad / I wanna wish you a Merry Christmas"]) #
Spanish + English @inspect predictions
272
273     Caveats:
274     • Difficult for short sequences
275     • Difficult for low-resource languages
276     • Could accidentally filter out dialects of English
277     • Hard for similar languages (Malay and Indonesian)
278     • Ill-defined for code-switching (e.g., Spanish + English)
279
280     OpenMathText [Paster+ 2023]
281     • Goal: curate large corpus of mathematical text from CommonCrawl
282     • Use rules to filter (e.g., contains latex commands)
283     • KenLM trained on ProofPile, keep if perplexity < 15000
284     • Trained fastText classifier to predict mathematical writing, threshold is 0.17 if math, 0.8 if no math
285     Result: produced 14.7B tokens, used to train 1.4B models that do better than models trained on 20x data
286
287
288 def quality_filtering():

```

```

289     • Some deliberately do not use model-based filtering (C4, Gopher, RefinedWeb, FineWeb, Dolma)
290     • Some use model-based filtering (GPT-3, LLaMA, DCLM) [becoming the norm]
291
292 GPT-3 [Brown+ 2020]
293     • Positives: samples from {Wikipedia, WebText2, Books1, Books2}
294     • Negatives: samples from CommonCrawl
295
296 Train linear classifier based on word features \[article\]
297 Keep documents stochastically based on score
298 def keep_document(score: float) -> bool:
299     return np.random.pareto(9) > 1 - score
300
301 ** LLaMA/RedPajama** [Touvron+ 2023]
302     • Positives: samples from pages referenced by Wikipedia
303     • Negatives: samples from CommonCrawl
304     • Keep documents that are classified positive
305
306 phi-1 [Gunasekar+ 2023]
307 Philosophy: really high quality data (textbooks) to train a small model (1.5B)
308 Includes synthetic data from GPT 3.5 (later: GPT-4) and filtered data
309
310 R = "Python subset of the Stack" # Raw data
311 prompt = "determine its educational value for a student whose goal is to learn basic coding concepts"
312 T = "Use GPT-4 with this prompt to classify 100K subset of R to get positive examples"
313 Train random forest classifier on T using output embedding from pretrained codegen model
314 Select data from R that is classified positive by the classifier
315
316 Result on HumanEval:
317     • Train 1.3B LM on Python subset of The Stack (performance: 12.19% after 96K steps)
318     • Train 1.3B LM on new filtered subset (performance: 17.68% after 36K steps) - better!
319
320
321 @dataclass
322 class Example:
323     text: str
324     label: int
325
326
327 def toxicity_filtering():
328     # WARNING: potentially offensive content below
329     Toxicity filtering in Dolma \[Soldaini+ 2024\]
330
331 Dataset: Jigsaw Toxic Comments dataset (2018) \[dataset\]
332     • Project goal: help people have better discussions online \[article\]
333     • Data: comments on Wikipedia talk page annotated with {toxic, severe_toxic, obscene, threat, insult,
identity_hate}
334
335 Trained 2 fastText classifiers
336     • hate: positive = {unlabeled, obscene}, negative = all else
337     • NSFW: positive = {obscene}, negative = all else
338
339 # Examples from the dataset: (obscene, text)
340 train_examples = [
341     Example(label=0, text="Are you threatening me for disputing neutrality? I know in your country it's quite common
to bully your way through a discussion and push outcomes you want. But this is not Russia."),
342     Example(label=1, text="Stupid peace of shit stop deleting my stuff asshole go die and fall in a hole go to
hell!"),
343 ]
344

```

```

345 # Download model
346 model_url = "https://dolma-artifacts.org/fasttext_models/jigsaw_fasttext_bigrams_20230515/
jigsaw_fasttext_bigrams_nsfw_final.bin"
347 model_path = "var/jigsaw_fasttext_bigrams_nsfw_final.bin"
348 download_file(model_url, model_path)
349 model = fasttext.load_model(model_path)
350
351 # Make predictions
352 predictions = model.predict([train_examples[0].text]) # @inspect predictions
353 predictions = model.predict([train_examples[1].text]) # @inspect predictions
354 predictions = model.predict(["I love strawberries"]) # @inspect predictions
355 predictions = model.predict(["I hate strawberries"]) # @inspect predictions
356
357
358 def print_predict(model, content):
359     """Run classifier `model` on `content` and print out the results."""
360     predictions = model.predict([content])
361     print(predictions)
362     #labels, prob =
363     #labels = ", ".join(labels)
364     #text(f"{content} => {labels} {prob}")
365
366
367 def deduplication():
368     Two types of duplicates:
369     • Exact duplicates (mirror sites, GitHub forks) [Gutenberg mirrors]
370     • Near duplicates: same text differing by a few tokens
371
372     Examples of near duplicates:
373     • Terms of service and licenses [MIT license]
374     • Formulaic writing (copy/pasted or generated from a template)

```

Dataset	Example	Near-Duplicate Example
Wiki-40B	<code>\n_START_ARTICLE.\nHum Award for Most Impact-ful Character \n_START_SECTION.\nWinners and nominees\n_START_PARAGRAPH.\nIn the list below, winners are listed first in the colored row, followed by the other nominees. [...]</code>	<code>\n_START_ARTICLE.\nHum Award for Best Actor in a Negative Role \n_START_SECTION.\nWinners and nominees\n_START_PARAGRAPH.\nIn the list below, winners are listed first in the colored row, followed by the other nominees. [...]</code>
LM1B	I left for California in 1979 and tracked Cleveland 's changes on trips back to visit my sisters .	I left for California in 1979 , and tracked Cleveland 's changes on trips back to visit my sisters .
C4	Affordable and convenient holiday flights take off from your departure country, "Canada". From May 2019 to October 2019, Condor flights to your dream destination will be roughly 6 a week! Book your Halifax (YHZ) - Basel (BSL) flight now, and look forward to your "Switzerland" destination!	Affordable and convenient holiday flights take off from your departure country, "USA". From April 2019 to October 2019, Condor flights to your dream destination will be roughly 7 a week! Book your Maui Kahului (OGG) - Dubrovnik (DBV) flight now, and look forward to your "Croatia" destination!

- Minor formatting differences in copy/pasting

Product description repeated 61,036 times in C4

"by combining fantastic ideas, interesting arrangements, and follow the current trends in the field of that make you more inspired and give artistic touches. We'd be honored if you can apply some or all of these design in your wedding. believe me, brilliant ideas would be perfect if it can be applied in real and make the people around you amazed!

[example page]

Deduplication training data makes language models better [Lee+ 2021]

- Train more efficiently (because have fewer tokens)
- Avoid memorization (can mitigate copyright, privacy concerns)

Design space:

1. What is an item (sentence, paragraph, document)?
2. How to match (exact match, existence of common subitem, fraction of common subitems)?
3. What action to take (remove all, remove all but one)?

Key challenge:

- Deduplication is fundamentally about comparing items to other items

```

392     • Need linear time algorithms to scale
393
394     hash_functions()
395
396     exact_deduplication()
397     bloom_filter()
398
399     jaccard_minhash()
400     locality_sensitive_hashing()
401
402
403 def hash_functions():
404     • Hash function h maps item to a hash value (integer or string)
405     • Hash value much smaller than item
406     • Hash collision:  $h(x) = h(y)$  for  $x \neq y$ 
407
408     Tradeoff between efficiency and collision resistance \[article\]
409     • Cryptographic hash functions (SHA-256): collision resistant, slow (used in bitcoin)
410     • DJB2, MurmurHash, CityHash: not collision resistant, fast (used for hash tables)
411
412     We will use MurmurHash:
413     h = mmh3.hash("hello") # @inspect h
414
415
416 def exact_deduplication():
417     Simple example
418     1. Item: string
419     2. How to match: exact match
420     3. Action: remove all but one
421
422     # Original items
423     items = ["Hello!", "hello", "hello there", "hello", "hi", "bye"] # @inspect items
424
425     # Compute hash -> list of items with that hash
426     hash_items = itertools.groupby(sorted(items, key=mmh3.hash), key=mmh3.hash)
427
428     # Keep one item from each group
429     deduped_items = [next(group) for h, group in hash_items] # @inspect deduped_items
430
431     • Pro: simple, clear semantics, high precision
432     • Con: does not deduplicate near duplicates
433     • This code is written in a MapReduce way, can easily parallelize and scale
434
435     C4 [Raffel+ 2019]
436     1. Item: 3-sentence spans
437     2. How to match: use exact match
438     3. Action: remove all but one
439     Warning: when a 3-sentence span is removed from the middle of a document, the resulting document might
         not be coherent
440
441
442 def bloom_filter():
443     Goal: efficient, approximate data structure for testing set membership
444
445     Features of Bloom filters
446     • Memory efficient
447     • Can update, but can't delete
448     • If return 'no', definitely 'no'
449     • If return 'yes', most likely 'yes', but small probability of 'no'

```

```

450 • Can drive the false positive rate down exponentially with more time/compute
451
452 items = ["the", "cat", "in", "the", "hat"]
453 non_items = ["what", "who", "why", "when", "where", "which", "how"]
454
455 First, make the range of hash function small (small number of bins).
456 m = 8 # Number of bins
457 table = build_table(items, m)
458 for item in items:
459     assert query_table(table, item, m) == 1
460 result = {item: query_table(table, item, m) for item in non_items} # @inspect result
461 num_mistakes = count(result.values(), True) # @inspect num_mistakes
462 false_positive_rate = num_mistakes / (len(items) + num_mistakes) # @inspect false_positive_rate
463 Problem: false positives for small bins
464
465 Naive solution: increase the number of bins
466 Error probability is  $O(1/\text{num\_bins})$ , decreases polynomially with memory
467
468 Better solution: use more hash functions
469 k = 2 # Number of hash functions
470 table = build_table_k(items, m, k)
471 for item in items:
472     assert query_table_k(table, item, m, k) == 1
473 result = {item: query_table_k(table, item, m, k) for item in non_items} # @inspect result
474 num_mistakes = count(result.values(), 1) # @inspect num_mistakes
475 false_positive_rate = num_mistakes / (len(items) + num_mistakes) # @inspect false_positive_rate
476 Reduced the false positive rate!
477
478 false_positive_rate_analysis()
479
480
481 def false_positive_rate_analysis():
482     Assume independence of hash functions and items [article]
483     m = 1000 # Number of bins
484     k = 10 # Number of hash functions
485     n = 100 # Number of items we're inserting
486
487     Consider a test input (not in the set) that would hash into a given test bin (say, i).
488     Now consider putting items into the Bloom filter and seeing if it hits i.
489
490     # Insert one item, ask if the test bin  $B(i) = 1$ ?
491     # B: [0 0 1 0 0 0 0 0 0 0] - have to miss 1 time
492     f = 1 / m # P[B(i) = 1 after 1 insertion with 1 hash function] # @inspect f
493     # B: [0 0 1 0 0 1 0 1 0 0] - have to miss k times
494     f = 1 - (1 - 1 / m) ** k # P[B(i) = 1 after 1 insertion with k hash functions] # @inspect f
495
496     # Insert n items, ask if the test bin  $B(i) = 1$ ?
497     # Have to miss  $k \cdot n$  times
498     f = 1 - (1 - 1 / m) ** (k * n) # P[B(i) = 1 after n insertions for 1 hash function] # @inspect f
499     # Get k chances to miss (since test input is hashed k times too)
500     f = f ** k # P[B(i) = 1 after n insertions for k hash functions] # @inspect f
501
502     Optimal value of k (given fixed m / n ratio) [results in  $f \sim 0.5$ ]
503     k = math.log(2) * m / n # @inspect k
504     Resulting false positive rate (improved)
505     f = 0.5 ** k # @inspect f
506
507     Tradeoff between compute (k), memory (m), and false positive rate (f) [lecture notes]
508

```

```

509 Example: Dolma
510 • Set false positive rate to 1e-15
511 • Perform on items = paragraphs
512
513
514 def build_table(items: list[str], num_bins: int):
515     """Build a Bloom filter table of size `num_bins`, inserting `items` into it."""
516     table = bitarray(num_bins) # @inspect table
517     for item in items:
518         h = mmh3.hash(item) % num_bins # @inspect item, @inspect h
519         table[h] = 1 # @inspect table
520     return table
521
522
523 def build_table_k(items: list[str], num_bins: int, k: int):
524     """Build a Bloom filter table of size `num_bins`, inserting `items` into it.
525     Use `k` hash functions."""
526     table = bitarray(num_bins) # @inspect table
527     for item in items:
528         # For each of the k functions
529         for seed in range(k):
530             h = mmh3.hash(item, seed) % num_bins # @inspect item, @inspect h, @inspect seed
531             table[h] = 1 # @inspect table
532     return table
533
534
535 def query_table(table: bitarray, item: str, num_bins: int, seed: int = 0):
536     """Return whether `item` is in the `table`."""
537     h = mmh3.hash(item, seed) % num_bins
538     return table[h]
539
540
541 def query_table_k(table: bitarray, item: str, num_bins: int, k: int):
542     """Return 1 if table set to 1 for all `k` hash functions."""
543     return int(all(
544         query_table(table, item, num_bins, seed)
545         for seed in range(k)
546     ))
547
548
549 def jaccard_minhash():
550     Let's now look at approximate set membership.
551     First we need a similarity measure.
552
553

```

Jaccard similarity

```

554 Definition: Jaccard(A, B) = |A intersect B| / |A union B|
555 A = {"1", "2", "3", "4"}
556 B = {"1", "2", "3", "5"}
557
558 def compute_jaccard(A, B):
559     intersection = len(A & B) # @inspect intersection
560     union = len(A | B) # @inspect union
561     return intersection / union
562 jaccard = compute_jaccard(A, B) # @inspect jaccard
563
564 Definition: two documents are near duplicates if their Jaccard similarity >= threshold
565

```

566 Algorithmic challenge: find near duplicates in linear time

567

568

MinHash

569 MinHash: a random hash function h so that $\Pr[h(A) = h(B)] = \text{Jaccard}(A, B)$

570

571 Normally, you want different items to hash to different hashes

572 ...but here, you want collision probability to depend on similarity

573

```
574 def minhash(S: set[str], seed: int):
575     return min(mmh3.hash(x, seed) for x in S)
```

576

577 Characteristic matrix representation:

578 item | A | B

579 1 | 1 | 1

580 2 | 1 | 1

581 3 | 1 | 1

582 4 | 1 | 0

583 5 | 0 | 1

584

585 Random hash function induces a permutation over items

586 Look at which item is first in A and which item is first in B.

587 Each item has the same probability as being first (min)

588 • If 1, 2, 3 is first, then first in A = first in B.

589 • If 4, 5 is first, then first in A \neq first in B.

590

591 # Verify MinHash approximates Jaccard as advertised

592 n = 100 # Generate this many random hash functions

593 matches = [minhash(A, seed) == minhash(B, seed) for seed in range(n)]

594 estimated_jaccard = count(matches, True) / len(matches) # @inspect estimated_jaccard

595 assert abs(estimated_jaccard - jaccard) < 0.01

596

597 Now we can hash our items, but a collision doesn't tell us $\text{Jaccard}(A, B) > \text{threshold}$.

598

599

```
600 def locality_sensitive_hashing():
```

601 Locality sensitive hashing (LSH) [\[book chapter\]](#)

602

603 Suppose we hash examples just one MinHash function

604 $P[A \text{ and } B \text{ collide}] = \text{Jaccard}(A, B)$

605 On average, more similar items will collide, but very stochastic...

606

607 Goal: have A and B collide if $\text{Jaccard}(A, B) > \text{threshold}$

608 We have to somehow sharpen the probabilities...

609

610 Solution: use n hash functions

611 Break up into b bands of r hash functions each ($n = b * r$)

612

613 n = 12 # Number of hash functions

614 b = 3 # Number of bands

615 r = 4 # Number of hash functions per band

616 Hash functions:

617 h1 h2 h3 h4 | h5 h6 h7 h8 | h9 h10 h11 h12

618

619 Key: A and B collide if for *some* band, *all* its hash functions return same value

620 As we will see, the and-or structure of the bands sharpens the threshold

621

622 Given $\text{Jaccard}(A, B)$, what is the probability that A and B collide?

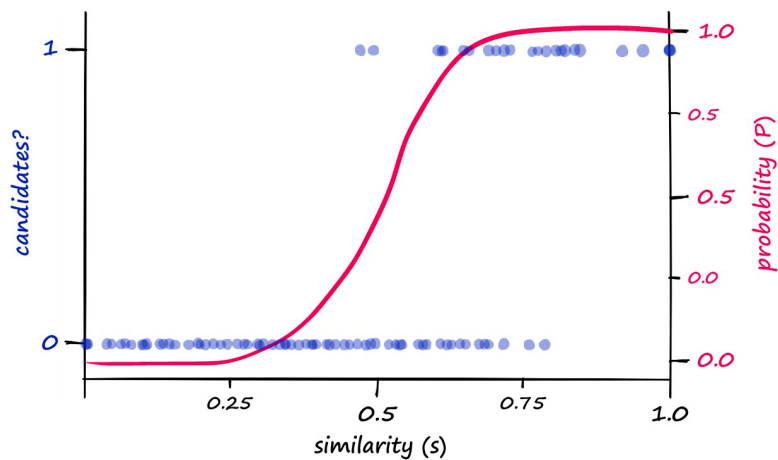
```

def get_prob_collision(sim, b, r): # @inspect sim, @inspect b, @inspect r
    prob_match = sim ** r          # Probability that a fixed band matches @inspect prob_match
    prob_collision = 1 - (1 - prob_match) ** b # Probability that some band matches @inspect prob_collision
    return prob_collision

```

Example

```
prob_collision = get_prob_collision(sim=0.8, b=5, r=10) # @inspect prob_collision
```



```

sims = [0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.98]
probs = {sim: get_prob_collision(sim=sim, b=10, r=10) for sim in sims} # @inspect probs

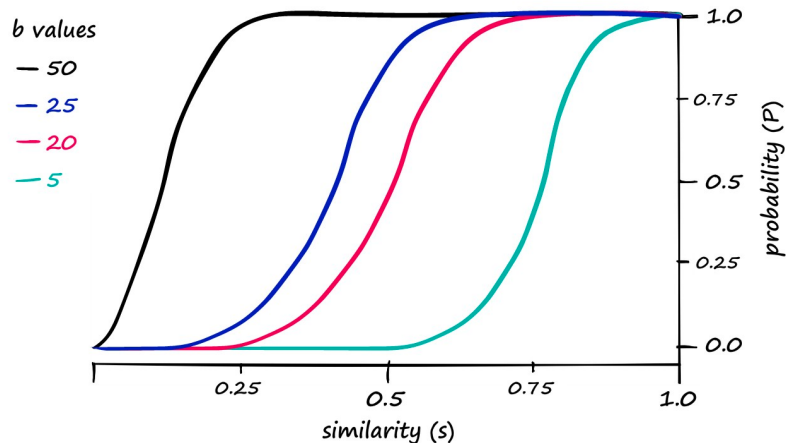
```

Increasing r sharpens the threshold and moves the curve to the right (harder to match)

```
probs = {sim: get_prob_collision(sim=sim, b=10, r=20) for sim in sims} # @inspect probs
```

Increasing b moves the curve to the left (easier to match)

```
probs = {sim: get_prob_collision(sim=sim, b=20, r=20) for sim in sims} # @inspect probs
```



Example setting [Lee+ 2021]: $n = 9000$, $b = 20$, $r = 450$

```
b = 20
```

```
r = 450
```

What is the threshold (where the phase transition happens)?

```
threshold = (1 / b) ** (1 / r) # @inspect threshold
```

Probability that a fixed band matches:

```
prob_match = (1 / b) # @inspect prob_match
```

Probability that A and B collide ($\approx 1-1/e$):

```
prob_collision = 1 - (1 - 1 / b) ** b # @inspect prob_collision
```

```
655 if __name__ == "__main__":  
656     main()
```