IN[34]120 SØKETEKNOLOGI – WEB SEARCH, CLASSIFICATION AND DATA STRUCTURES

TA: OLIVER RUSTE JAHREN
(OLIVERRJ@IFI.UIO.NO)

GROUP 1: 2023-11-03 10:15 @ CHILL
GROUP 2: 2023-11-07 12:15 @ FORTRESS

THÈMES:

- CLASSIFICATION
  - Concept
  - Naïve Bayes for ass. E-1
  - Decision boundaries
- WEB SEARCH
  - PageRank
  - Crawling
- DATA STRUCTURES
  - Bloom filters
  - Cuckoo filters
- ASSIGNMENT D SOLUTION REVIEW
- BOOK OF THE WEEK

SAME SAME *BUT DIFFERENT*

Sergey Jakobsen 11:15

Jeg har pleid å kjøre

- En historie (bare for å varme litt opp, få folk litt engasjert. Noe jeg har vært på, feks fjelltur i nord-norge, fotballkamp jeg var på etc)
- Nøyere forklaring av oblig med hint, eksempler, osv. Prøve å gjøre dem så forståelig som mulig (feks vise funksjonssignaturer, «samle» hintene til Øhrn, for assB viste jeg hvordan self.__haystack og suffixes skulle se ut)
- Repetisjon av forrige ukes pensum (konsentrere på det som er viktig for eksamen/virkeligheten. unngår de eksemplene i boka der man viser «naive» løsninger osv.
- Ukas bok («har noen lest den? ja, hva synes dere?» og hvorfor jeg mener de bør lese den)

A STORY! THE TALE OF THE PARIS HACKATHON

The faculty of legal studies contacted FUI about a hackathon in Paris

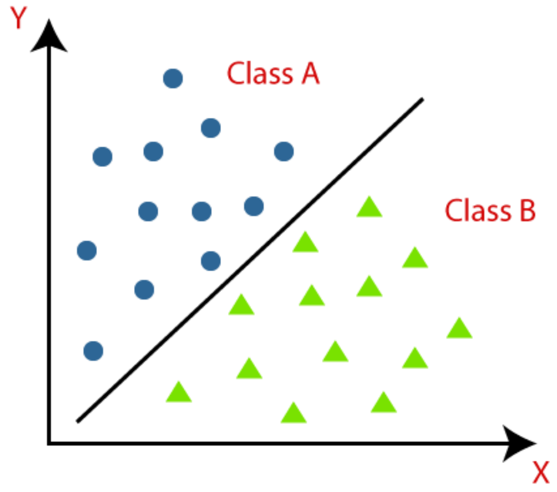
I signed up and promoted it

The hackathon concerns classification of legal documents

I also shared this in our mattermost: https://mattermost.uio.no/ifi-in3120/pl/n3dzcx3ddib6pmccohq1qhyqxy

(The signup deadline was 2023-11-02)

ASSIGNMENT HINTS! FOR E-1

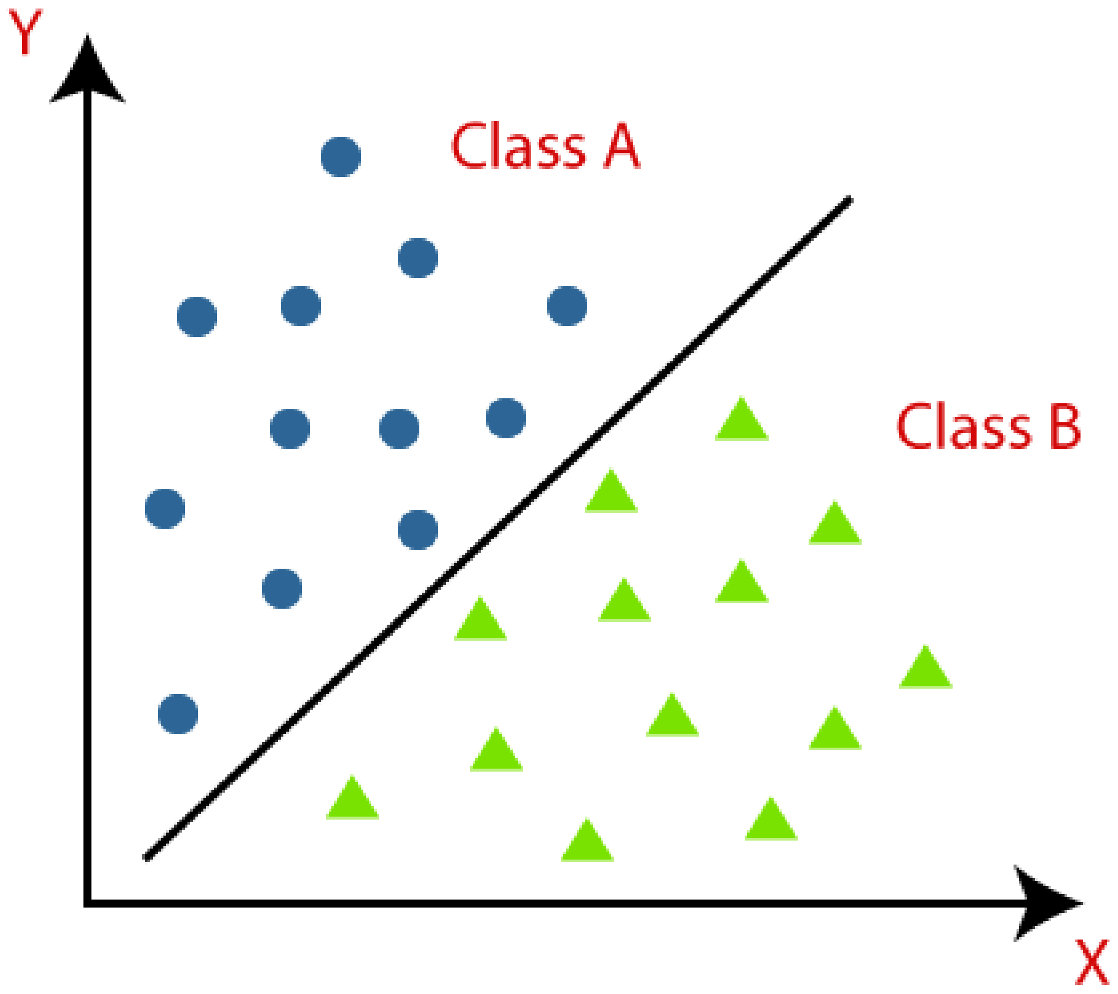


- Naïve Bayes classification
  - What is classification?
    - Determine what language a text is*
  - We have classes
    - Our classes are the languages.
      - In E-1: NO, EN, DA, GER
      - *How could we "learn" more categories? What determines what classes we can use?*
  - Test what class fits the best
    - Try all classes. Best result is our prediction.
    - Report the findings

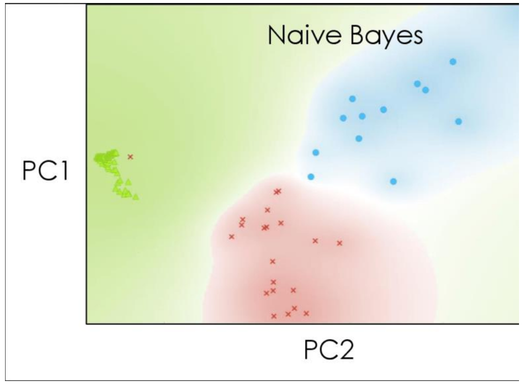
*Oversimplification. Other forms of classification exist also. They are not relevant for E-1.

CLASSIFICATION

- "Given an input, what class does it belong to?"
  - "*Ég kann ekki ríta íslensku*" -> what language is this?
    - The language is our class or category
- In this context, "class" and "category" mean the same
- Machine learning / AI
  - Naïve Bayes
  - Support Vector Machines (SVMs)
  - Supervised / unsupervised learning
    - Relates to how the models are trained
    - Supervised learning
      - We have labeld training data
    - Unsupervised learning
      - Unlabelled data
      - The model must find the categories by itself



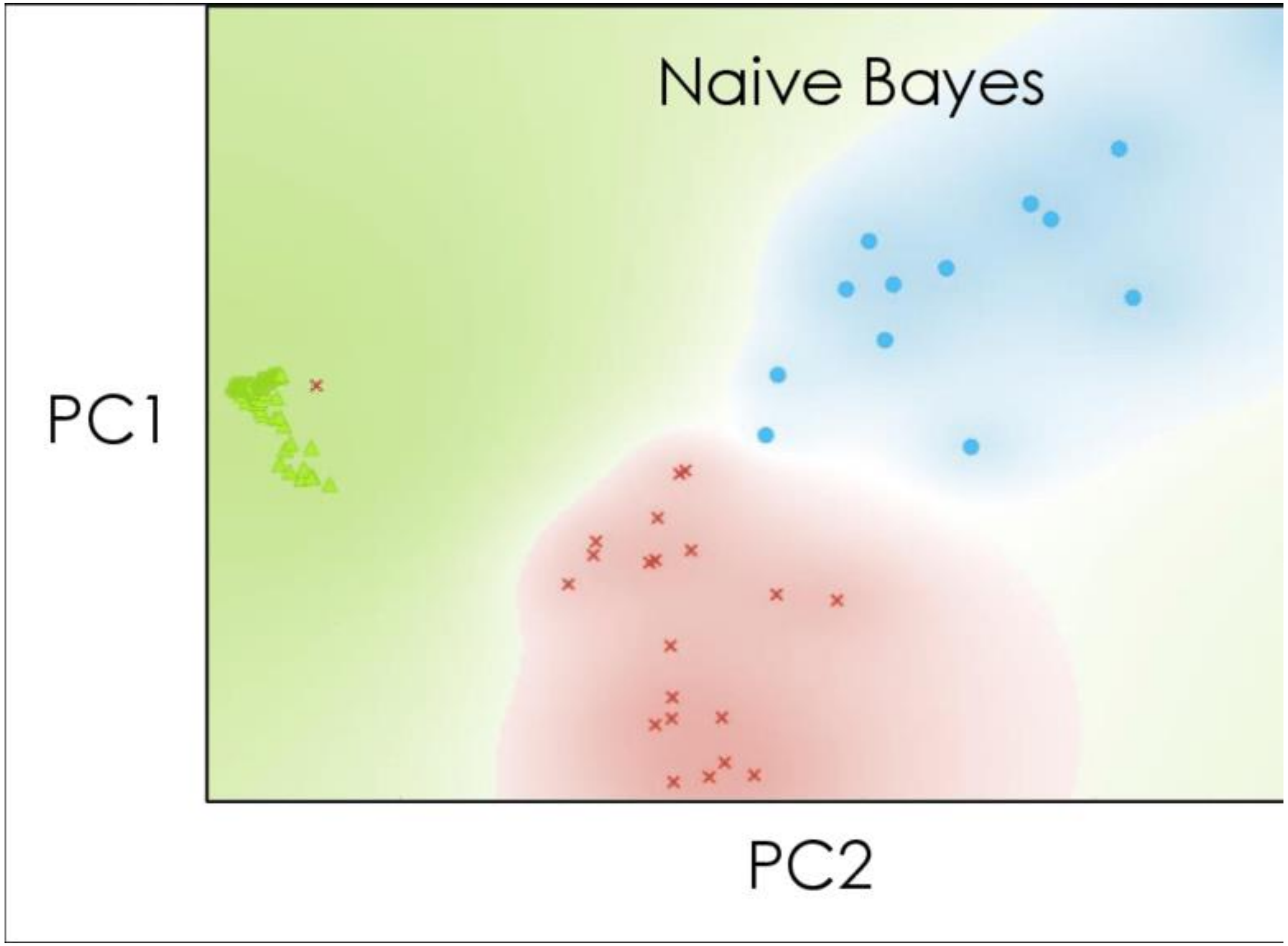
NAÏVE BAYES



- Why naïve?
  - Uses Bag-of-words model
    - I.e. the order of the tokens doesn't matter
- Why Bayes?
  - Named after Thomas Bayes (1701-1761), English statistician

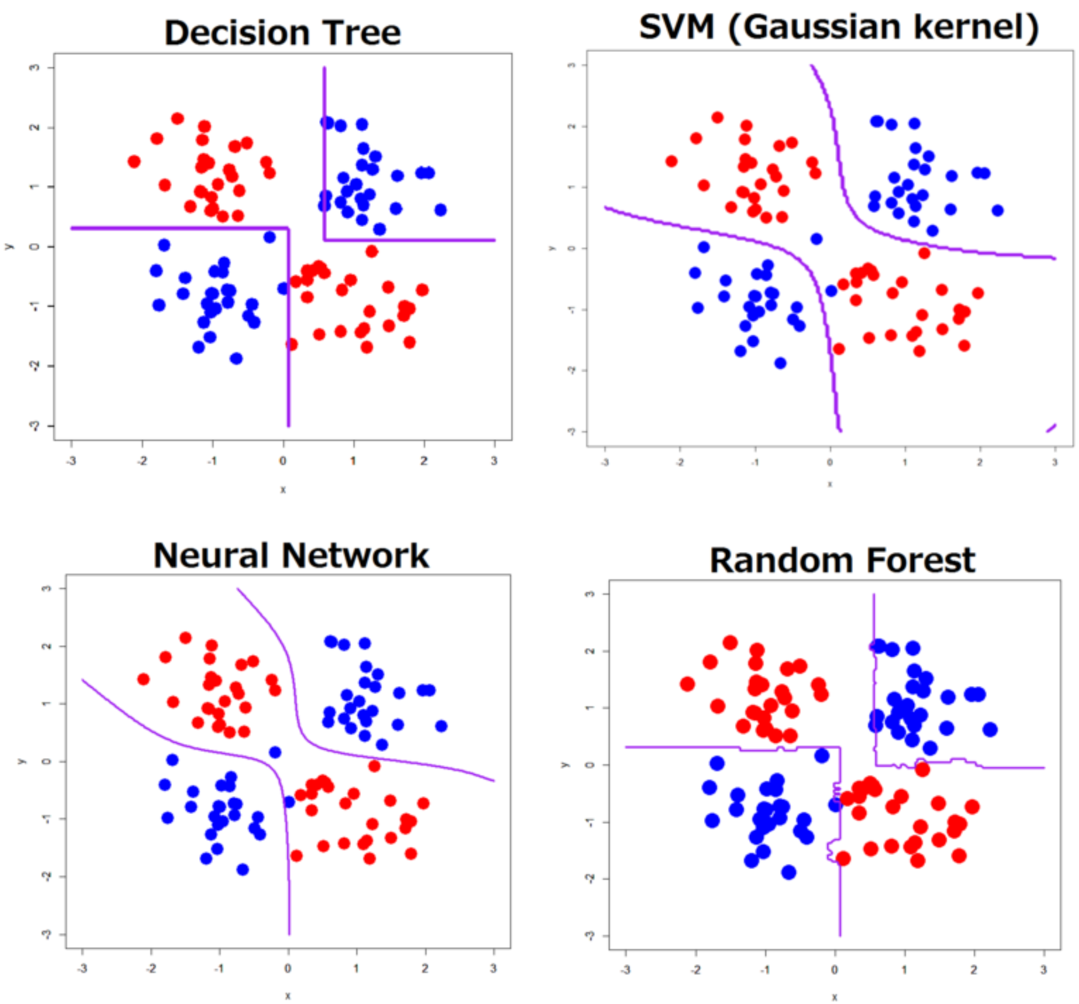
DECISION BOUNDRIES

- Note how the color blobs for the most part match our samples
  - The marks are our samples
- But there are some outliers
  - **Note the red cross in the sea of green**
  - This means that our model would probably make some mistakes sometimes
- Naïve Bayes is very basic, other methods can achieve better results



THE IMPACT OF DECISION BOUNDRIES

- Looking at the decision boundries allows us to visually evaluate the effectiveness of our model
- Further reading: see “overfitting” and other models in other språktek-courses like IN2110, IN3050/4050 and IN4080



CODE FOR ASSIGNMENT E-1!

- self.__compute_priors(training_set)
- self.__compute_vocabulary(training_set, fields)
- self.__compute_posteriors(training_set, fields)
- self.__classify(self, buffer: str)

PRIORS! (LONG FORM: ’PRIOR PROBABILITY’)

```python
    def __compute_priors(self, training_set):
        """
        Estimates all prior probabilities needed for the naive Bayes classifier.
        """
        raise NotImplementedError("You need to implement this as part of the assignment.")
```

- What is a prior?
  - The probability of seeing a class regardless of the term
  - Intuition: if 85% of the animals in your farm are pigs, if you select a random animal it will probably be a pig.
- How can it be calculated?
  - You need the counts of the categories
    - How big are the categories
    - How many terms do you have with all categories added up
  - Big categories are more probable

VOCABULARY!

```python
    def __compute_vocabulary(self, training_set, fields):
        """
        Builds up the overall vocabulary as seen in the training set.
        """
        raise NotImplementedError("You need to implement this as part of the assignment.")
```

- What terms do we have in our corpora?
  - We need to add all the terms to our vocabulary (self.__vocabulary)
- Iterate over the training set corpora
  - Iterate over the documents in the corpus
    - Iterate over the fields (fields is given as arg)
      - Iterate over the terms in the field
        - Add the terms (self.__vocabulary.add_if_absent(term))

POSTERIORS!

```
    def __compute_posteriors(self, training_set, fields):
        """
        Estimates all conditional probabilities needed for the naive Bayes classifier.
        """
        raise NotImplementedError("You need to implement this as part of the assignment.")
```

- “All conditional probabilities”….?
  - We must:
    - Principally: set conditionals
    - But how?
      - Set and use the denominators

POSTERIORS CONT.

1. We must deal with all categories and all corpora. => Iterate over the training set
2. We need all terms for all fields for all documents in the corpus
3. Count the terms of step #2
4. The denominator for the current category will be the sum of the counting from step #3 added with the vocabulary size
    1. This provides a relation between corpus size and probability
5. The conditional for the category will be set for each term in the category: The conditional for term t given class x will be the term frequency of t +1* divided by the denominator for x.
    1. *We use "'laplace-add-one'-smoothing", which entails adding 1.

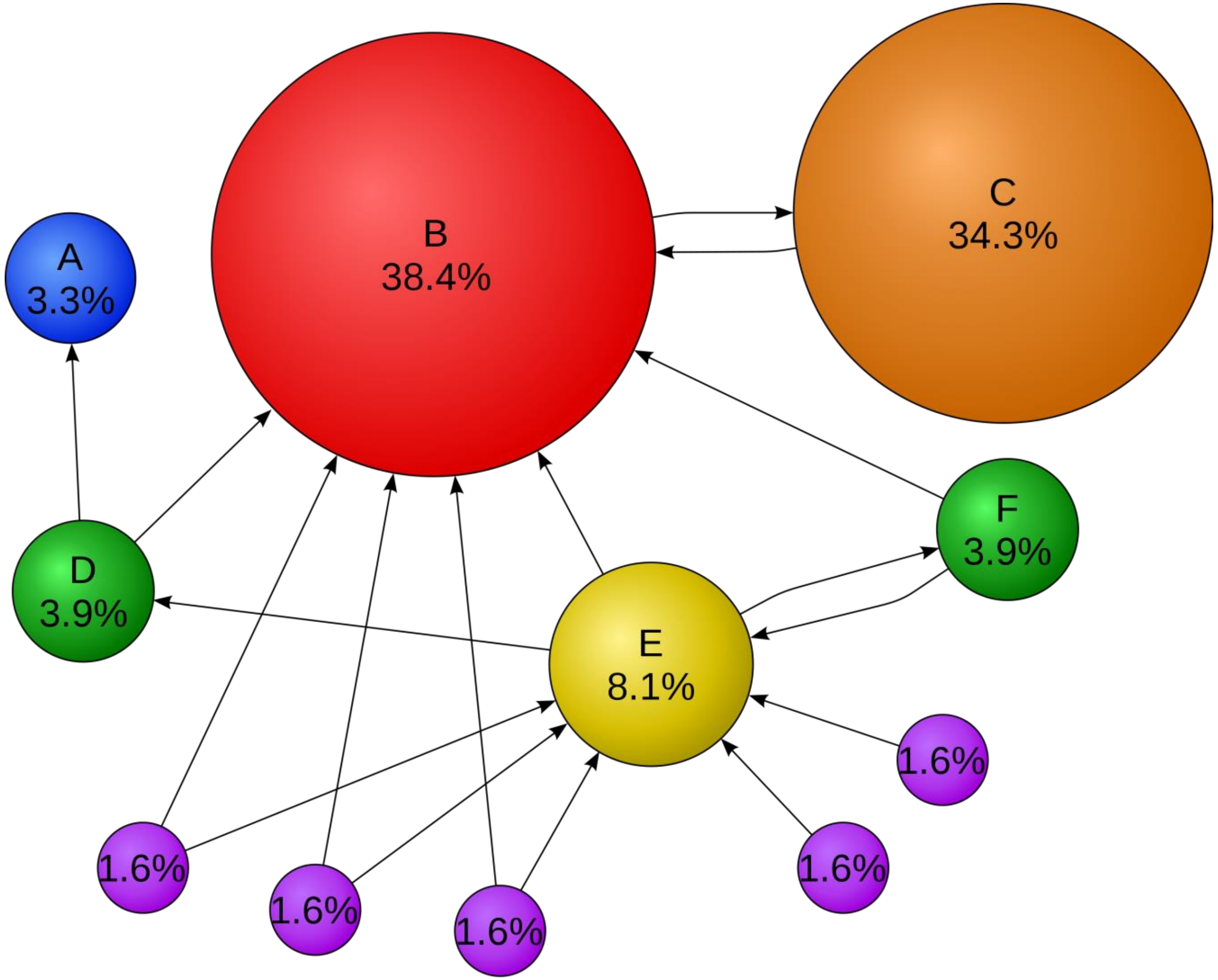
CLASSIFY

```python
def classify(self, buffer: str) -> Iterator[Dict[str, Any]]:
    """
    Classifies the given buffer according to the multinomial naive Bayes rule. The computed (score, category) pairs
    are emitted back to the client via the supplied callback sorted according to the scores. The reported scores
    are log-probabilities, to minimize numerical underflow issues. Logarithms are base e.

    The results yielded back to the client are dictionaries having the keys "score" (float) and
    "category" (str).
    """
    raise NotImplementedError("You need to implement this as part of the assignment.")
```

- We have all the probabilities from
- Given a text (“buffer”), classify it.
  - Find the terms
  - Init a hashmap of scores. Each category starts with its prior.
  - Iterate over the categories. Increase the score by the conditional for every term given the category.
    - Use 1/denominator for the cases when the term t never appears in the category (0 is a bad value)
  - Finally yield all values for all categories in sorted order

WEB SEARCH

- Basic principle: Query -> sorted list of ranked web pages
- The web is a directed graph of sites. Links are edges and pages are nodes.
  - Graphs allow us to use maths, logic and algorithms!



PAGERANK

- Ranking algorithm for web pages
  - Invented by and named for Larry Page, Google co-founder. Photo on the left.
    - Ofc also a good pun. *See if you can figure out why.*
  - Main principle:
    - If several sites {a, b, c} are linking to site x, site x is probably a good site and should be ranked highly(? Grammar).
      - (…Obviously?)
      - PageRank is a big deal because of finding a way of caluclating this.

Larry Page

90 languages

Article Talk Read View source View history Tools

From Wikipedia, the free encyclopedia

For the singer, see Larry Page (singer).

Lawrence Edward Page[2][3][4] (born March 26, 1973) is an American businessperson, computer scientist and internet entrepreneur best known for co-founding Google with Sergey Brin.[2][5]

Page was chief executive officer of Google from 1997 until August 2001 when he stepped down in favor of Eric Schmidt and then again from April 2011 until July 2015 when he became CEO of its newly formed parent organisation Alphabet Inc. which was created to deliver "major advancements" as Google's parent company,[6] a post he held until December 4, 2019 when he along with his co-founder Brin stepped down from all executive positions and day-to-day roles within the company. He remains an Alphabet board member, employee, and controlling shareholder.[7]

As of October 2023, Page has an estimated net worth of $118 billion according to the Bloomberg Billionaires Index, making him the sixth-richest person in the world.[8] He has also invested in flying car startups Kitty Hawk and Opener.[9]

Page is the co-creator and namesake of PageRank, a search ranking algorithm for Google[17] for which he received the Marconi Prize in 2004 along with co-writer Brin.[18]

Early life

Larry Page

Page speaking at the European Parliament in 2009

Born	Lawrence Edward Page March 26, 1973 (age 50)

PAGERANK A BIG DEAL?

GETS GOOD SCREEN REAL ESTATE ON WIKIPEDIA.

(USE THIS INFORMATION AS YOU WANT)

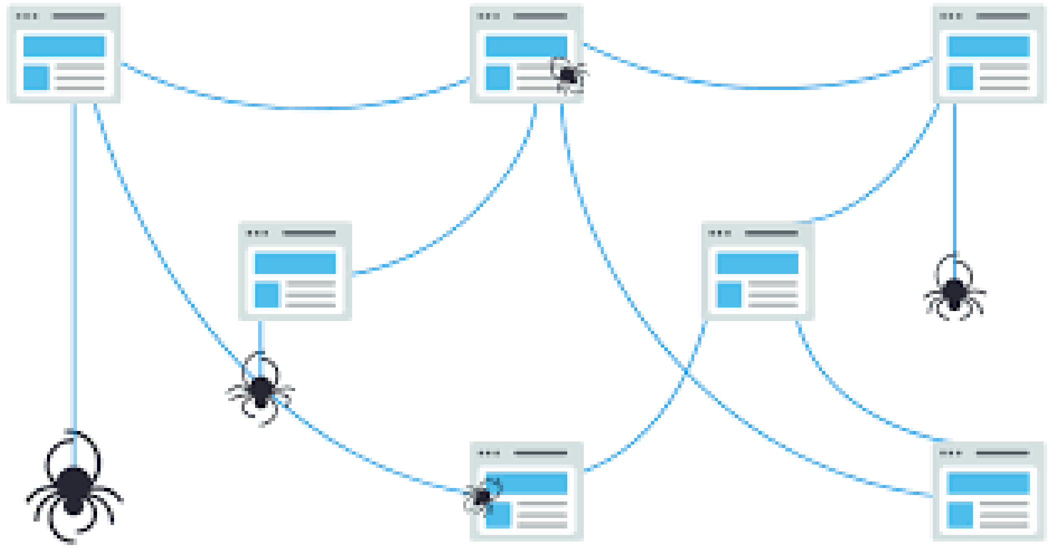
TANGET: THE IMPORTANCE OF ALGORITHMS IN THE SEARCH INDUSTRY

GOOGLE IS AN ADVERTISMENT COMPANY

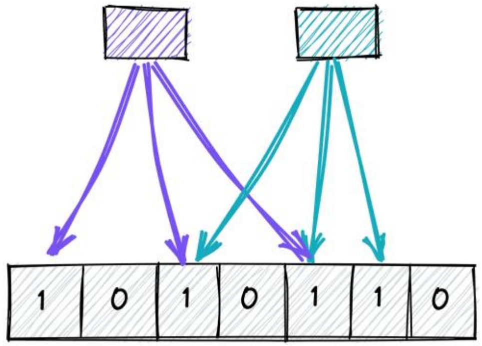
HAVING A GOOD ALGORITHM IS IRRELEVANT IF YOU CANNOT KEEP AFLOAT FINANCIALLY

CRAWLING

- What is crawling?
  - ”Finding” all the pages on the internet and learning what they contain
- Do you want to crawl?
  - No, You want to avoid it
    - Why?
      - Painful
      - Expensive
      - Possibly illegal



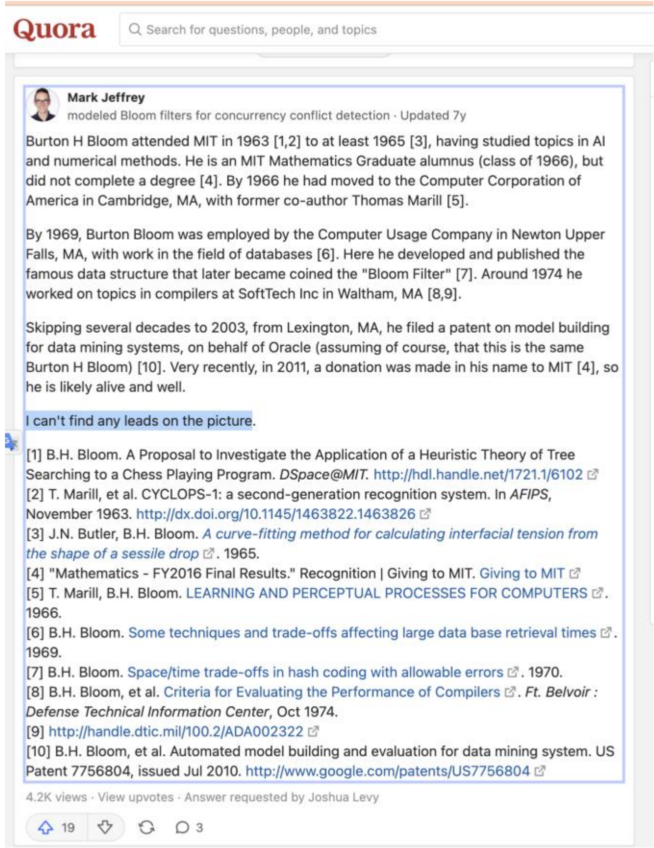
BLOOM FILTERS



- Use case: "Is this element a part of my set?"
  - Main claim to fame: uses minimal memory
- Can give false positives
  - I.e. it can say "yes you have this" without it being true
- But always true negatives
  - I.e. if it says "no, you don't have this", you can be sure of it being correct

- NB: You cannot delete items
  - Cuckoo filters address this. Covered in a few slides.
  - Also: Counting Bloom filters exist. Not so relevant.
  - For understanding, think: *Why is it impossible to delete items?*

BLOOM FILTERS CONT.



Quora

Search for questions, people, and topics

Mark Jeffrey
modeled Bloom filters for concurrency conflict detection · Updated 7y

Burton H Bloom attended MIT in 1963 [1,2] to at least 1965 [3], having studied topics in AI and numerical methods. He is an MIT Mathematics Graduate alumnus (class of 1966), but did not complete a degree [4]. By 1966 he had moved to the Computer Corporation of America in Cambridge, MA, with former co-author Thomas Marill [5].

By 1969, Burton Bloom was employed by the Computer Usage Company in Newton Upper Falls, MA, with work in the field of databases [6]. Here he developed and published the famous data structure that later became coined the "Bloom Filter" [7]. Around 1974 he worked on topics in compilers at SoftTech Inc in Waltham, MA [8,9].

Skipping several decades to 2003, from Lexington, MA, he filed a patent on model building for data mining systems, on behalf of Oracle (assuming of course, that this is the same Burton H Bloom) [10]. Very recently, in 2011, a donation was made in his name to MIT [4], so he is likely alive and well.

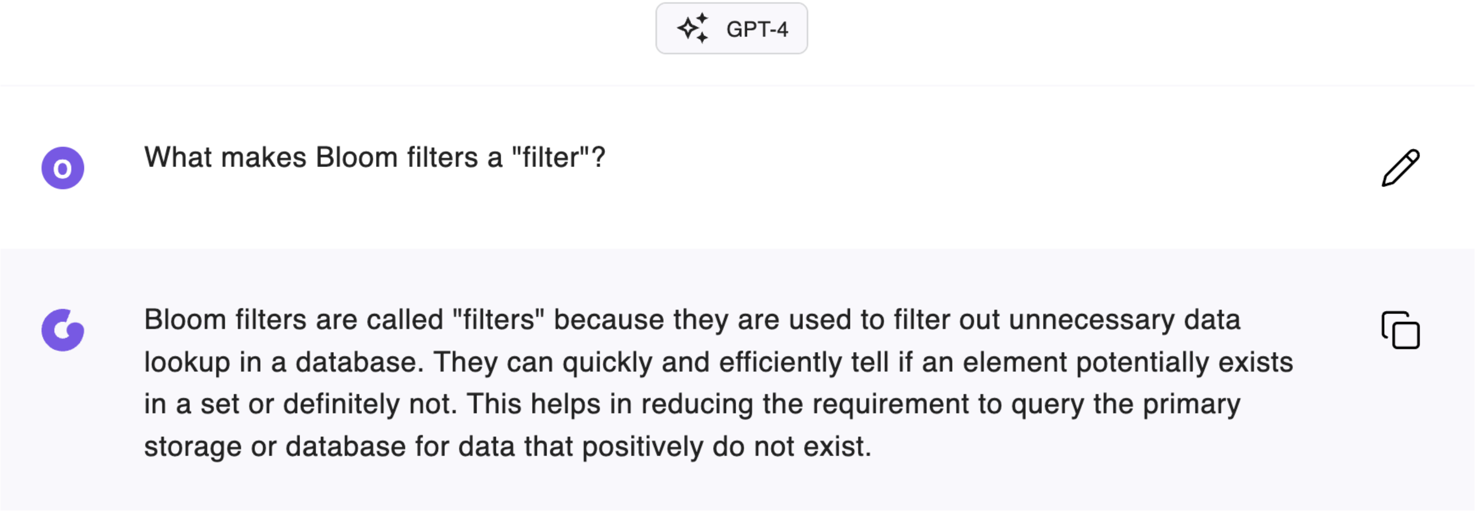
I can't find any leads on the picture.

[1] B.H. Bloom. A Proposal to Investigate the Application of a Heuristic Theory of Tree Searching to a Chess Playing Program. *DSpace@MIT*. http://hdl.handle.net/1721.1/6102
[2] T. Marill, et al. CYCLOPS-1: a second-generation recognition system. In *AFIPS*, November 1963. http://dx.doi.org/10.1145/1463822.1463826
[3] J.N. Butler, B.H. Bloom. *A curve-fitting method for calculating interfacial tension from the shape of a sessile drop*. 1965.
[4] "Mathematics - FY2016 Final Results." Recognition | Giving to MIT. Giving to MIT
[5] T. Marill, B.H. Bloom. LEARNING AND PERCEPTUAL PROCESSES FOR COMPUTERS. 1966.
[6] B.H. Bloom. Some techniques and trade-offs affecting large data base retrieval times. 1969.
[7] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. 1970.
[8] B.H. Bloom, et al. Criteria for Evaluating the Performance of Compilers. *Ft. Belvoir : Defense Technical Information Center*, Oct 1974.
[9] http://handle.dtic.mil/100.2/ADA002322
[10] B.H. Bloom, et al. Automated model building and evaluation for data mining system. US Patent 7756804, issued Jul 2010. http://www.google.com/patents/US7756804

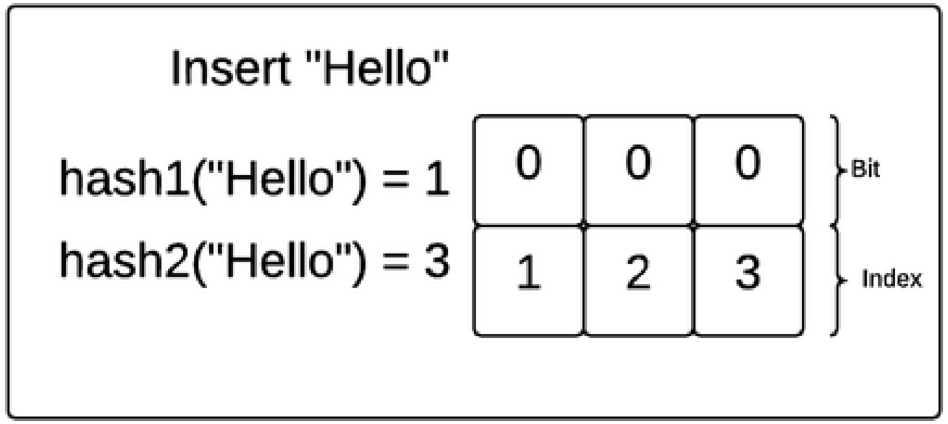
4.2K views · View upvotes · Answer requested by Joshua Levy

19 3

- Why Bloom?
  - Named for Burton Howard Bloom
    - He has no Wikipedia page and no images online.
    - His seminal paper is in the repo.
- Why filter?
  - Not 100% sure. GPT-4 claims it is to filter out unnecessary operations. May be true.

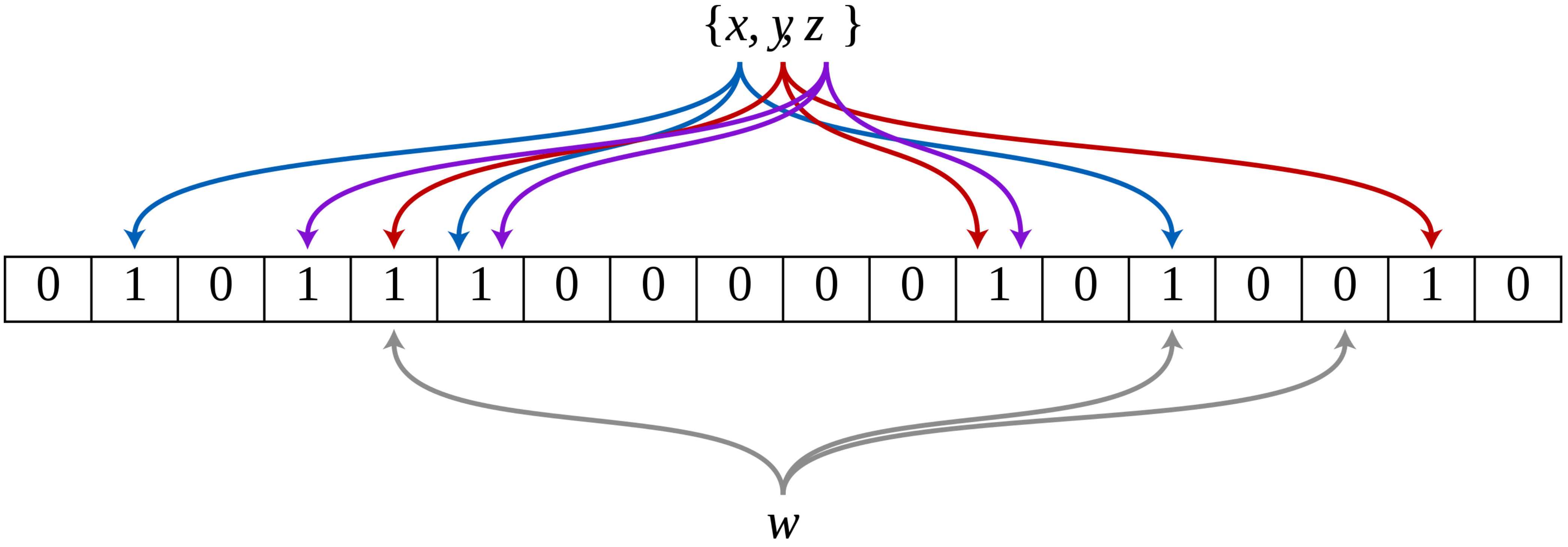


HOW DOES A BLOOM FILTER WORK?



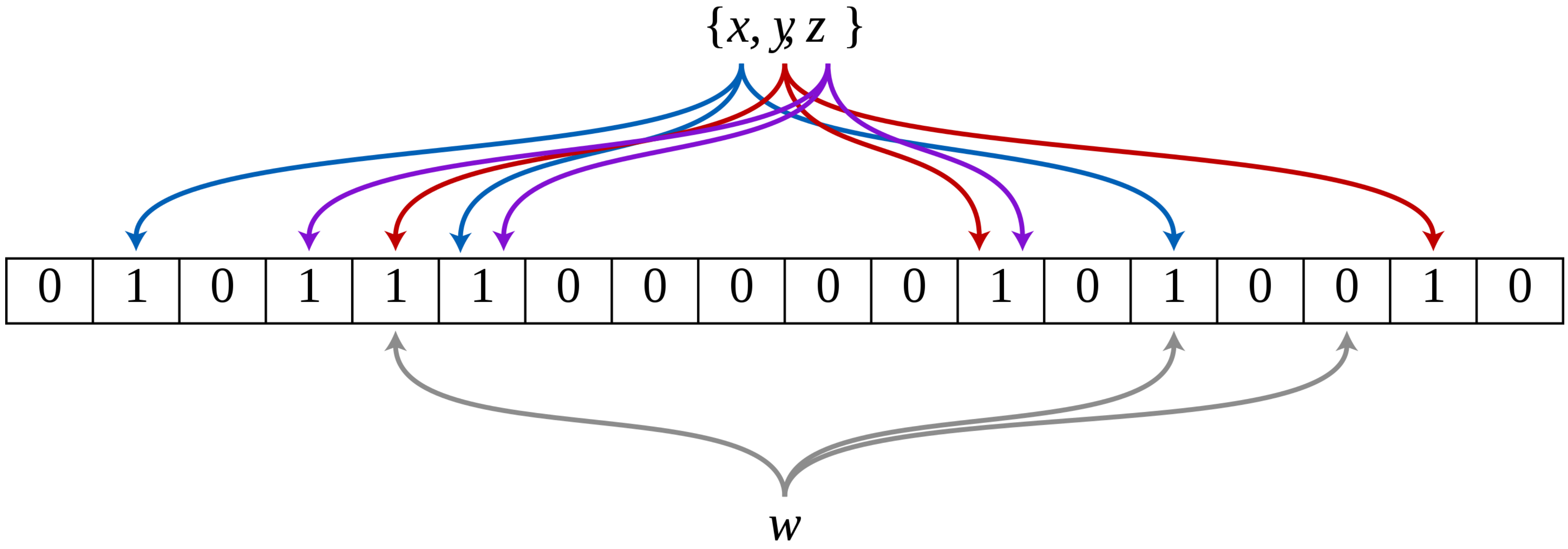
- Initizlize an empty array of size m and set all the values to 0
- **Adding elements**: Assume you have k hash functions. When you add an item, hash the value k times with the k functions and flag the resulting indeces as present.
- **Checking for presence**: Use the same k hash functions as when you added elements. Hash the input value k times. If all the k indeces are flagged, you probably have the value. If 1 or more are not flagged, you 100% don't have the value.

<- example explenation: m is 3 (because we have 3 bits) and k is 2 (because there are 2 hash functions)



WHY CAN BLOOM FILTERS GIVE FALSE POSITIVES?

WHY CAN YOU NOT DELETE ITEMS FROM BLOOM FILTERS?



CUCKOO FILTERS

- "Bloom filters but better"
- Allows deleting items
- Use even less memory
- New datastructure, first described in 2014

Cuckoo Filter: Practically Better Than Bloom

Bin Fan, David G. Andersen, Michael Kaminsky[†], Michael D. Mitzenmacher[‡]
Carnegie Mellon University, [†]Intel Labs, [‡]Harvard University
{binfan,dga}@cs.cmu.edu, michael.e.kaminsky@intel.com, michaelm@eecs.harvard.edu

ABSTRACT
In many networking systems, Bloom filters are used for high-speed set membership tests. They permit a small fraction of false positive answers with very good space efficiency. However, they do not permit deletion of items from the set, and previous attempts to extend "standard" Bloom filters to support deletion all degrade either space or performance.

We propose a new data structure called the *cuckoo filter* that can replace Bloom filters for approximate set membership tests. Cuckoo filters support adding and removing items dynamically while achieving even higher performance than Bloom filters. For applications that store many items and target moderately low false positive rates, cuckoo filters have lower space overhead than space-optimized Bloom filters. Our experimental results also show that cuckoo filters outperform previous data structures that extend Bloom filters to support deletions substantially in both time and space.

Categories and Subject Descriptors
E.1 [**Data**]: Data Structures; E.4 [**Data**]: Data Compaction and Compression

Keywords
Cuckoo hashing; Bloom filters; compression

1. INTRODUCTION
Many databases, caches, routers, and storage systems use *approximate set membership* tests to decide if a given item is in a (usually large) set, with some small false positive probability. The most widely-used data structure for this test is the Bloom filter [3], which has been studied extensively due to its memory efficiency. Bloom filters have been used to: reduce the space required in probabilistic routing tables [25]; speed longest-prefix matching for IP addresses [9]; improve network state management and monitoring [24, 4]; and encode multicast forwarding information in packets [15], among many other applications [6].

CoNEXT'14, Dec 02-05 2014, Sydney, Australia
ACM 978-1-4503-3279-8/14/12.
http://dx.doi.org/10.1145/2674005.2674994

A limitation of standard Bloom filters is that one cannot remove existing items without rebuilding the entire filter (or possibly introducing generally less desirable false negatives). Several approaches extend standard Bloom filters to support deletion, but with significant space or performance overhead. *Counting Bloom filters* [12] have been suggested for multiple applications [24, 25, 9], but they generally use 3–4× space to retain the same false positive rate as a space-optimized Bloom filter. Other variants include *d-left counting Bloom filters* [5], which are still 1.5× larger, and *quotient filters* [2], which provide significantly degraded lookup performance to yield comparable space overhead to Bloom filters.

This paper shows that supporting deletion in approximate set membership tests need not impose higher overhead in space or performance compared to standard Bloom filters. We propose the *cuckoo filter*, a practical data structure that provides four major advantages.

1. It supports adding and removing items dynamically;
2. It provides higher lookup performance than traditional Bloom filters, even when close to full (e.g., 95% space utilized);
3. It is easier to implement than alternatives such as the quotient filter; and
4. It uses less space than Bloom filters in many practical applications, if the target false positive rate ϵ is less than 3%.

A cuckoo filter is a compact variant of a cuckoo hash table [21] that stores only *fingerprints*—a bit string derived from the item using a hash function—for each item inserted, instead of key-value pairs. The filter is densely filled with fingerprints (e.g., 95% entries occupied), which confers high space efficiency. A set membership query for item x simply searches the hash table for the fingerprint of x, and returns true if an identical fingerprint is found.

When constructing a cuckoo filter, its fingerprint size is determined by the target false positive rate ϵ. Smaller values of ϵ require longer fingerprints to reject more false queries. Interestingly, while we show that cuckoo filters are practically better than Bloom filters for many real workloads, they are asymptotically worse: the minimum fingerprint size used in the cuckoo filter grows logarithmically with the number of entries in the table (as we explain in Section 4). As a consequence, the per-item space overhead is higher for larger tables, but this use of extra space confers a lower false positive rate. For practical problems with a few billion items or fewer,

CUCKOO FILTERS

Cuckoo hashing

Article Talk

From Wikipedia, the free encyclopedia

Cuckoo hashing is a scheme in computer programming for resolving hash collisions of values of hash functions in a table, with worst-case constant lookup time. The name derives from the behavior of some species of cuckoo, where the cuckoo chick pushes the other eggs or young out of the nest when it hatches in a variation of the behavior referred to as brood parasitism; analogously, inserting a new key into a cuckoo hashing table may push an older key to a different location in the table.

- Why cuckoo?
  - Because of the underlying cuckoo hashing
- Why filter?
  - Probably to filter out unneeded expensive work like wigh Bloom filters.

HOW CUCKOO FILTERS WORK DIFFERENTLY FROM BLOOM FILTERS?

- Main difference lies in the hashing algorithm
  - Described in 2001.
    - Recall that the cuckoo filters appeard in 2014.
  - Cuckoo hashing is not relevant for the course - not covered in detail. See the paper if interested.

They outline 4 advantages in the paper:

1. It supports adding and removing items dynamically
2. It provides higher lookup performance than traditional Bloom filters, even when close to full (e.g., 95% space utilized);
3. It is easier to implement than alternatives such as the quotient filter
4. It uses less space than Bloom filters in many practical applications, if the target false positive rate is less than 3%.

ASSIGNMENT D-1 SOLUTION SKETCH REVIEW

NOTE TO SELF: SHOW IN A CODE EDITOR AND EXPLAIN WHAT OCCURS.

Code Blame 57 lines (47 loc) · 2.42 KB

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

import math
from .ranker import Ranker
from .corpus import Corpus
from .posting import Posting
from .invertedindex import InvertedIndex


class BetterRanker(Ranker):
    """
    A ranker that does traditional TF-IDF ranking, possibly combining it with
    a static document score (if present).

    The static document score is assumed accessible in a document field named
    "static_quality_score". If the field is missing or doesn't have a value, a
    default value of 0.0 is assumed for the static document score.

    See Section 7.1.4 in https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf.
    """

    # These values could be made configurable. Hardcode them for now.
    _dynamic_score_weight = 1.0
    _static_score_weight = 1.0
    _static_score_field_name = "static_quality_score"
    _static_score_default_value = 0.0

    def __init__(self, corpus: Corpus, inverted_index: InvertedIndex):
        self._score = 0.0
        self._document_id = None
        self._corpus = corpus
        self._inverted_index = inverted_index

    def reset(self, document_id: int) -> None:
        self._score = 0.0
        self._document_id = document_id

    def update(self, term: str, multiplicity: int, posting: Posting) -> None:
        assert term is not None
        assert multiplicity > 0
        assert posting is not None
        assert posting.term_frequency > 0
        assert posting.document_id == self._document_id
        tf_score = 1.0 + math.log10(posting.term_frequency)
        idf_score = math.log10(self._corpus.size() / float(self._inverted_index.get_document_frequency(term)))
        self._score += (1.0 + math.log10(multiplicity)) * tf_score * idf_score

    def evaluate(self) -> float:
        # Now that the dynamic (query-dependent) score is fully updated, combine it
        # with the static (query-independent) score using a simple weighted sum. Other
        # ways of combining the two are plausible. In a large real-world search system,
        # weights would be machine-learnt offline and it'd be up to the chosen ML model
        # how to best combine the many features to yield a compound relevance score.
        document = self._corpus[self._document_id]
        static_quality_score = float(document[self._static_score_field_name] or self._static_score_default_value)
        return (self._dynamic_score_weight * self._score) + (self._static_score_weight * static_quality_score)
```

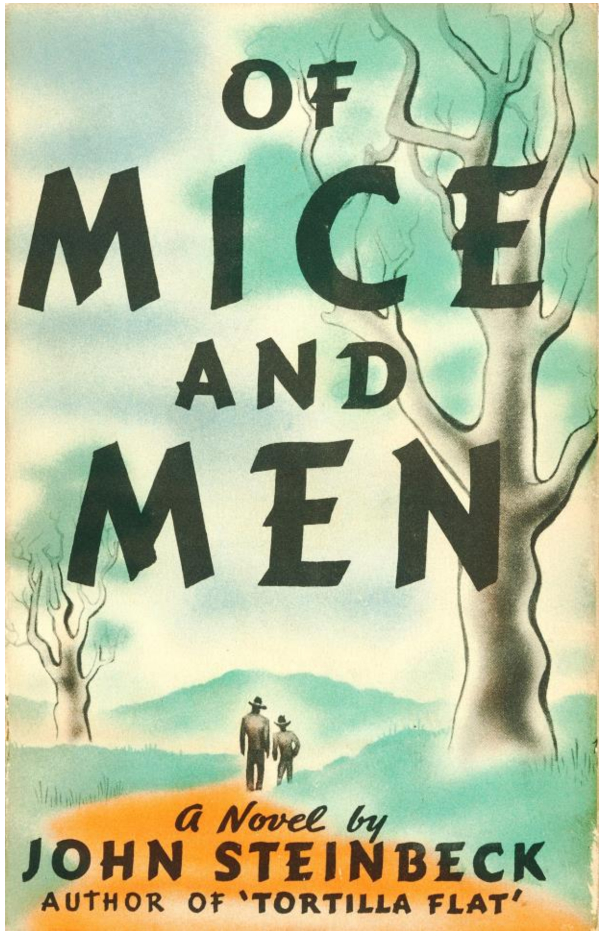
Code Blame 33 lines (26 loc) · 1.02 KB Raw

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

from typing import Iterator, Tuple
from .tokenizer import Tokenizer


class ShingleGenerator(Tokenizer):
    """
    Tokenizes a buffer into overlapping shingles having a specified width. For example, the
    3-shingles for "mouse" are {"mou", "ous", "use"}.

    If the buffer is shorter than the shingle width then this produces a single shorter-than-usual
    shingle.

    The current implementation is simplistic and not whitespace- or punctuation-aware,
    and doesn't treat the beginning or end of the buffer in a special way.
    """

    def __init__(self, width: int):
        assert width > 0
        self.__width = width

    def ranges(self, buffer: str) -> Iterator[Tuple[int, int]]:
        """
        Locates where the shingles begin and end.
        """
        if buffer == "":
            return
        elif len(buffer) <= self.__width:
            yield (0, len(buffer))
        else:
            yield from ((i, i + self.__width) for i in range(0, len(buffer) - self.__width + 1))
```

BOOK OF THE WEEK

Themes

> In every bit of honest writing in the world there is a base theme. Try to understand men, if you understand each other you will be kind to each other. Knowing a man well never leads to hate and nearly always leads to love. There are shorter means, many of them. There is writing promoting social change, writing punishing injustice, writing in celebration of heroism, but always that base theme. Try to understand each other.
>
> — John Steinbeck in his 1938 journal entry[7]



Of Mice and Men, John Steinbeck. USA, 1937

[...] Of Mice and Men has been a frequent target of censors for vulgarity, and what some consider offensive and racist language; consequently, it appears on the American Library Association's list of the Most Challenged Books of the 21st Century.

~ Wikipedia https://en.wikipedia.org/wiki/Of_Mice_and_Men

OR PROPOSE OTHER TOPICS TO REVIEW OR DISCUSS.

THAT'S ALL. ASSIGNMENT AID?