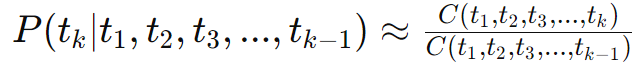
A computer screen shot of a computer code

Description automatically generatedA screenshot of a computer

Description automatically generatedA diagram of a diagram

Description automatically generated**A purple and white symbol

Description automatically generatedA black and white symbolsChapter11: Count-based Language Models**: **Statistical Language Models**: Attempt to capture probabilities • Of observing a term or sequence of terms • Usually given some context • Captures the sequential structure of a language • Grammar/word choice • eg. “There are” more likely than “Their are”. Use: • Predictive text • Language generation • Grammar checking - Evaluating machine-generated text. SLM-has distribution over terms. “term model” provides estimate of P(t). *m* terms in vocabulary, {m-1} numbers to store distribution. **Conditional Models**: “(Conditional bigram model) provides estimate of P(t2|t1)”. Give the last position t1. (what is next word?) {m \* (m-1)}. **More complex models**: p(t1|t2,t3,t4) m\*m\*m\*(m-1). Best-case scenario, 99.9999% of 4-grams never occur. **Sparsity**: when creating new content. Good sparsity: n-grams that don’t make sense have probability 0. (“he you bird now”) • Bad sparsity: Plausible n-grams that happen not to be present in the corpus get probability zero. More data reduces bad sparsity. (Challenge to naïve approaches. – data Sparsity) **Simplifying Assumptions:** p(t1,t2) = p(t1)p(t2). Parameters: 2(m-1). Condition: Independence assumption(egg and Quantum). t1 tell us nothing about what P(t2) will be. P(t1,t2) = p(t2,t1). **(Markov) Bigram Model**: Probability of next term only depends on term immediately before. Assigns probabilities to sequences of arbitrary length • Number of parameters fixed: m×(m – 1). **Trigram Model:** • Probability of next term only depends previous two terms. 4-gram, 5-gram, ... are similar.  Count of times the entire sequence t1​ through tk​ appears together in the corpus. **Smoothing**: In above models, if just one of the probabilities is zero, the whole sequence is given probability zero. Smoothing to avoid zero. **Laplace smoothing:** adds 1 to each count before normalizing appropriately. **Back-off:** simpler model if the complicated ones do not work. Mustard ice-cream -> ice cream. Katz, Lower order gram,Back off and adjust. Stupid ignore the problem (back off only). **Context-based Smoothing**, known words are more likely to appear after an unknown word. **Applications**: **Text-generation**, **Babbling**: is drawing a sequence of words, always conditioning on the most recently generated word(s) to produce the next one. **Filtering(**Evaluation, whether text is plausible**)**: improve Statistical machine translation methods. Speech-to-text. Give more “babble” likely content. **Output from other systems**, **Intelligent “spell checking”,** **Corpus membership classification**(probability of passage belonging to each). **Chapter 12: Neural Network Language Models**: • Instead of storing a number for every n-gram, distribution is computed using the weights in the network. **Ideas In GPT**: • **Softmax output:** P(i) is the probability of the *i*th word in the vocabulary being the next word. e^sj is the exponential of the raw score for the *i*th word (convert output to probabilities) • **Hidden layers**. Output layer: Produces probability distribution over tokens • Output layer after hidden layers: over terms/tokens “**Attention**” to communicate information from word to word (big deal) consider entire sequence of words rather than the immediate previous words. • Word embeddings and **position** information (order of words)• **Byte-pair encodings**(Tokenization): solve out-of vocabulary words. Idea: Fix number of allowed tokens • Allocate tokens to frequently occurring sequences of characters • Still allow any sequence of characters to be mapped to tokens. Start with 1 token per byte • If a pair of bytes occurs frequently, make a new token • If “t” “h” occurs a lot, create a new token “t” “h” and a merge rule. • If “th” “e” occurs a lot, create a new token “the” and a merge rule. • Stop when bored. (For GPT, after 50000 merges.) • To tokenize a new sentence, apply the rules in order. • Common sequences will get their own token; uncommon ones won’t • Any sequence of bytes is tokenizable • In GPT, one token is about ¾ of a word, on average. (Words that are not in vocabulary can be broken down into subword units) eg: “unbreakable” => “un”, “break”, “able”. Allow the model to process and generate txt containing words it has never seen before. **GPT-2** • Vocabulary size: 50,257 • Context length 1024 • 10^9.18 parameters • Count-based: ~10^4814 parameters • **GPT-3** (English part) • Vocabulary size: 50,257 • Context length 2048 • 10^11.25 parameters • Count-based ~10^9628 parameters **Chatgpt:** **step1** collect data and train. Use human-created pairs of prompts and responses. Fine-tune the language production model. **Step 2**, train a reward model that can rank responses. Use human-derived rankings of responses • Training pairs are (model outputs, rankings) • Learn to predict the ranking of a series of outputs (which is best?). **Step3**: Train a more sophisticated babbling model based on the reward model • Learn how to produce good responses by trying to produce high rewards according to the learned reward model • Human not required at this point. **Concerns**: Can models Be too big? Environment: a human average 5tCo2. Train a big model 284t of CO2. Representation: Reddit, men and young people. Wikipedian: 8.8 ~ 15% are girls. GPT use these as training sets. Data updated: BLM movement increase the articles so reduced generation latency. Curation, Documentation, Accountability “Feeding AI with beauty, ugliness, and cruelty, but expecting it to reflect only the beauty is a fantasy. LMs produce text that is so convincing. consider what went into their construction when assessing the credibility of what they produce. **Chapter 13: Big Data and MapReduce** Big Data vs Moore’s Law: The number of transistors on a chip will double about every two years. SLOWING NOW. **Scalable Computing:** Lots of data • Relatively cheap to store • Analyzing data has a lot of benefits • However, for large amounts of data we need many computers and storage units – Need clusters of commodity computers. **Processing Large Datasets**: Centralizing data processing will not work for huge amounts of data. • Data and processing often needs to be distributed • Processing platforms need to enable multiple tasks to be execute on different chunks of data. **Simple, Large-scale computations**: • TF-IDF, Count-based language model, Neural network training. **Large Data Set Analysis**: • Iterate over a large set of records • Extract something of interest from each • Shuffle and sort intermediate results • Aggregate interim results • Generate final output. (can use cloud compute) **MapReduce**: Programming model for processing large data sets • An execution framework that is able to run multiple tasks. From google. Apply at Netfilx, Linkedin, Amazon. Financial: Fraud detection. Search engine: Page Ranking. Google Map. MapReduce is highly scalable and can be used across many computers. **BEFORE:** • Large scale data processing was difficult – Managing hundreds or thousands of processors – Managing parallelization and distribution – I/O scheduling – Status and monitoring – Fault/crash tolerance – Programming models: MPI (Message-passing Interface). Inspired from map and reduce operations commonly used in functional **programming languages** like Lisp • Have multiple workers (processes) on multiple machines run either map or reduce. **Map:** (keyi ,valuei ) -> (keyj ,valuej ) • Input: A key/value pair • Output: A key/value pair. Evaluation – Function defined by user • Might need to parse input and extract relevant data • Produces a new list of key/value pairs – Can be of different type from input pair. **Reduce**: (keyj ,[val] j ) -> [valk ] • All the intermediate values associated with each keyj produced by the mapper are combined together into a list, giving the pair (keyj ,[val]j ) • Reduce function is applied to each of these pairs. The nice thing about the model is that a programmer writes the mapper code and the reducer code • The Shuffle and Sort is handled by an environment like • Hadoop • Elasticsearch/Hadoop • MongoDB (but deprecated) • Riak. **Page Rank**: Each source page has links to target pages • Find (target, list (sources)) for all targets. **Input**: The input are web pages represented by the URL of the web page and the content of the web page. • **Output**: (URL, list-of-URLs) – list-of-URLs represents the URLs of web pages that that hav.e a link to URL. • **Input:** (A,C), (B,C), (B,F), (D,A), (D,B), (E,A), (E,C), (F,C) • Map – Input: a source page S – **Output**: Pairs (T,S) for every link T in S. Output: (A,{D,E}), (B, {D}), (C, {A,B,E,F}), (D,{}), (E,{}), (F,{B}). • Reduce: – Input: Many pairs (T,S) from the mapper – Emits (T, list(S1, S2, …, Sk)). Project was started in 1995 • In1 998 Google was formed. **Random Surfer Model**: The probability that the random surfer clicks on a link is based on the number of links on that page • The probability that the random surfer visits a page is its PageRank. **Ranking**: The goal of ranking web pages is to get a measure of how popular a webpage is – Based on the number of pages that are linked to it. – Or if there are some pages that point to it and have a high PageRank. • A page can have a high PageRank – If there are many pages that point to it – Or if there are some pages that point to it and have a high PageRank. • inlinks[Pi ]: Set of webpages pointing to Pi • |Pj|: Number of outlinks from page Pj • PR(Pj )/|Pi| is the normalized pagerank – The pagerank of webpage, Pj , is shared by all webpages Pj points to. **PageRank Computation:** – Give an initial page rank to every webpage • Example:1/N where N is the total number of webpages. Perform the calculation of pagerank iteratively: 1. Use the pagerank formula to update the pagerank over every webpage. 2.Repeat the above step a number of times until the pagerank values are stable (converge). Assume that initially PR(A), PR(B), PR(C), PR(D)are 0.25 • PR(D)= PR(B)/1 +PR(C)/3. • Let rk (Pi ) be the PageRank of page Pi at iteration k ▪ Starting with r0 (Pi )= 1/n for all pages Pi • At iterationk+1,the page rank of every page Pi is updated using the page ranks at iteration k. (原理). **PageRank and MapReduce:** Multiple mapreduce stages are needed • Output of reducers are fed into the mappers of the next stage. **Mapper:** 1. Initial set up: assign each page rank 1/N. 2. Map op: input: <key, value> = <y, PR(y), [x1..xn]>. y = current page. Xi = outlink page. Output = key-value pairs <xi, PR(y)/#outlink(y)>. **Reduce**: formula above. **Reducer**: – The reducer receives values from mappers – Use the Page Rank formula to aggregate values and calculate new PageRank values. **Chapter 15: NoSQL and Mongo DB Relational Databases (SQL):** Requires predefined data models prior to use i.e., static schemas. Focus on integrity, atomicity. • Distributed implementations an after thought. **MongoDB applications: Personalization-** Data Modelling is a means of meeting a customer’s needs. 2. Customized online experiences for customers based on analysis of behavioral. Customer data is more than names and addresses. Compositional structure of products is challenging to capture. **Blogs(Twitter):** A blog consists of a collection of text entries or posts. A tag is a word or two that reflects the content of the post Main data entity is a post. Relational structure may be hard to manage at scale. **Problems With Relational Databases**: Overhead for complex select, update, delete operations. Schemas may be difficult to define and may evolve over time. **Properties:** Atomicity: if one part fails then the transaction fails, Consistency: Every read receives the most recent write or an error. Were designed to run on a single server to maintain the integrity of the tables. **Application**: Banking, finance because “Truth” and consistency are necessary requirements. Scaling these kinds of databases is hard. **NoSQL**: better solution when Strict schema won’t work, Relational structures are complicated, Atomicity and consistency maybe not crucial, Distribution/efficiency is crucial, Fault-tolerance.  **NoSQL**(1998): A form of database management system that is non-relational. Systems are often schema-less, avoid joins and are easy to scale, by – Adding computers – Sharding and replication. **Why?** The data we store is more complex and dynamic. Data can’t fit on a single server. Easy Distribution(high availability). **Types:** • Document Store • Key-value store • Graph • BigTable. **MongoDB:** • An open source and document-oriented database. • Documents are in BSON format, consisting of field-value pairs – Binary JSON; like JSON but less space • Designed for scalability and developer agility • Dynamic schemas (more like no schema at all.). **Json**: JavaScript Object Notation – Built on • Name and value pairs – Objects can be nested. **BSON**: Binary JSON – Binary encoded serialization of JSON-like documents. **SQL:** database, tables, rows and columns. **MongoDB:** Database, collections, documents, fields. **Benefits:** When doing a query: Embedded objects/documents retrieved in the same query as parent object/document – Only 1 trip to the DB server required • Objects in the same collection are stored contiguously on disk –Faster access • Easier than specifying joins. **Chapter 17 MongoDB and the Aggregation Pipeline** [myjson.EXCERPT.SPEECH.1.LINE.0] $ can define custom var. $obj. Other programs may not understand. Dates are stored internally as number of milliseconds since 1 January 1970. **Mongoimport:**  • JSON, CSV, TSV, file from mongoexport • can specify types for CSV/TSV, but not for JSON. **Commands:** show dbs; use tweetdb; db.tweets.findOne(); Create(db.users.insert({obj1: value})), Read(db.users.find({age: {$gt: 18}}).sort({age:1})), Update, Delete. **Read: MongoDB Atlas**: Atlas Search is an embedded full-text search in MongoDB. Built on Apache Lucene. **Self-hosted**: Must create a "text index" for any fields to be searched. **Mongo Functions:** **Projecting**: db.users.find({ age: { $gt: 25 } }, { name: 1, email: 1, \_id: 0 }) **Update**: db.users.updateOne({ name: "John Doe" }, { $set: { email: "john.doe@newdomain.com" } }); **Updates to fix datatypes:** $set: { createdDate: { $dateFromString: { dateString: "$created\_at" } } }. **Delete:** db.users.deleteMany({ age: 25 }); **Aggregation:** db.users.aggregate([{ $match:{status: “A}}$group: { \_id: $cust\_id, total: { $sum: "$amount" } } }]); **$match** works basically like find**. $group** works by combining documents together into new documents. When we want an operator to use a field, prefix that field with a dollar sign. **$project** • Choose which fields to send along the pipeline. If directly examining output, can be helpful to add as a last step. • **$bucket** • Group documents according to some criterion. **$sort** • Sorts by given field • **$lookup** • LEFT JOIN **$unwind** • Replicate document for each value in specified array. (拆分list&count). **Syntax:** **Dollar signs**: • field path • Path to a field in the document. To specify a field path, use a string that prefixes the field name with a dollar sign ($). “$user.name” • Also used for operators, built-in functions, parameters… • Except, when creating a new field? • **Quotation mark**s • { $user.name : “Arshin” } // Error • { “$user.name” : “Arshin” } // OK **Chapter 19-Cluster Architectures:** Large-scale clusters consist of several racks Racks consist of nodes Communication within racks is faster than between racks. A **shard** is a partition of data – Each shard is on a different computer (or node) – An index or file may be partitioned into shards. **Why?** Files may not fit on one disk • Multiple users may want to access the same file • Shards enable the use of multiple computers to handle requests • The load is balanced better if shard access is uniform • Elasticsearch, Mon goDB split indices into shards; Hadoop File System splits files into "blocks" **Why not shard**? Sharding issues: Hadoop FS shards (blocks) at the byte level might not divided properly. **Cluster Failures**: **Replicas.** What if a node goes down? • This means that a shard (and hence part of an index is lost). • We may want to create at least one replica of a shard to ensure availability. There are multiple file systems that can shard, replicate, and maintain files so that hardware failures can be tolerated • Google GFS • Hadoop HDFS • Kosmix KFS. Cluster underlying Linux. – Linux is open source, – You can modify Linux to suit your needs. **MongoDB Features and Design Decisions:** • Differences from "traditional" DBMS • Data consistency is "optional" • User control over how much consistency is required. **mongod** is the main daemon process for the MongoDB system. It handles data requests, manages data access, and performs background management operations. • mongod can be a: • "regular" server • shard server • config server. The **mongosh** shell connects to a **mongod** process (for non-sharded deployments) or a **mongos** process (for sharded deployments). **Replica set** is a group of mongod processes that maintain the same data: • Primary: only one - receives all write operations • Secondaries: normally at least 2 - replicate operations from the primary to maintain an identical data set. • Minimum recommended config is 3 in a replica set. (Primary + 2 secondaries.). **Primary** maintains oplog – log of all operations on its data. **Secondaries** replicate this and apply to their own data so everything is synced. • **Secondaries** can dynamically choose their own sync from source based on ping, availability. **Replica Sets**: The required number of secondary replicas depends on the read volume – Can dynamically be added – You don’t have to stop operations All **replicas** have a number, 0, 1, 2, …, N • Replicas know about each other • heartbeats are exchanged. **Automatic Failover:** When a secondary replica does not receive a response from the primary within the specified amount of time, it invokes the bully algorithm to “elect” a new primary. **Bully Algorithm:** When a secondary replica, P, notices that the primary is no longer responding, it initiates an election. – P **sends** an ELECTION message to all secondary replicas with **higher numbers**. – If no one responds, P wins the election and becomes the primary. – If one of the higher-ups answers, it takes over. P’s job is done. – This algorithm is used in many distributed systems. When a replica **gets** an ELECTION message from one of its lower-numbered colleagues: – Receiver sends an **OK** message back to the sender to indicate that it is alive and will take over. – Receiver holds an election, unless it is already holding one. – Eventually, all replicas give up but one, and that one is the new primary. – The new primary announces its victory by sending all processes a message telling them that starting immediately it is the new primary. **Network Partition:** • If the Primary cannot see a majority of the nodes in its replica set, it steps down and becomes a Secondary. • Secondary nodes that can't see a majority won't start an election. **Why?** Because 3 nodes are not enough to be a primary replica • B) Because of the network problem • C) Because “Write” ops. goes to two primary replicas. **Why not two primaries?** Two primaries could lead to a split-brain scenario where both primaries accept write operations and create conflicting versions of the data. **Rollbacks**: Suppose primary accepts a write, but crashes (or network fails) before replication. • Primary is removed from the Replica Set • Later on, it comes back as a Secondary. The node "rolls back" (undoes) the write operation. **Availability During Elections**: • replica cannot write until the election completes• The replica set can continue to serve read queries if such queries are configured to run on secondaries while the primary is offline. **Read Preference**: • primary • primaryPreferred • secondary • secondaryPreferred • nearest. **Consistency of Data**: read operations issued to the primary are consistent with the last write. Reads to a secondary have eventual consistency. It is possible that a client reads stale data. **Data "durability**" is just how resistant a system is to data loss. W1 = write at least one. W:m = majority nodes. **Read concern:** • "local" • No guarantees; might be rolled back • "available" • No guarantees; might be rolled back, might return "orphaned documents" • "majority" • Only returns majority-acknowledged writes • "linearizable" • Returns \*all\* majority-acknowledged writes • "snapshot" • Returns \*all\* majority-acknowledged writes at specified timestamp. **Hadoop does not have these concer**ns because: Hadoop has one-copy-update semantics • Roughly, an HDFS file must behave like a regular POSIX file (Linux, etc.) • Once a create/update/delete completes, everybody sees the same thing. • A file doesn't become visible to anybody until it is completely written. • This is ensured by the **NameNode.** **NameNode** • HDFS cluster has a single NameNode – A master server that manages the file system and regulates access to files by clients. • A physical server is dedicated to NameNode • The NameNode keeps track of which blocks make up each file and where they are stored. **Sharding** helps MongoDB horizontally scale by distributing data across multiple servers. It's necessary when server capacity maxes out, offering an alternative to costly vertical scaling. Data distribution relies on a shard key; the balancer ensures even distribution across shards. **Lecture 21: Executing and Optimizing Hadoop MapReduce: Terminology:** A **job** is a “full program” with mapper and reducer code, input dataset and a location for the output • A **task** is an execution of a mapper or a reducer on part of the data. Example: Run “Word Count” across a file with 20 blocks is one job – Assume a file is assigned to one map task – 20 blocks typically result A purple background with text and a black rectangle

Description automatically generatedin 20 map tasks – These may send data to many reduce tasks. **Job Processing** • JobTracker assigns work to TaskTrackers • After map, the TaskTrackers exchange mapoutput information to build the reduce key space • JobTracker partitions reduce() keyspace into m chunks, where m is set by user – Assigns work to reducers. **Assigning to Reducers:** Use a hash(ki ) – Assumes equal distribution of data over hashes. **How many Mappers and Reducers**: • **Mappers** – By default this is the number of HDFS blocks being processed – The number of maps can also be controlled by specifying the minimum split size • **Reducers** – This is a function of the number of nodes and the amount of data – for n nodes; two suggestions: – 0.95n (all reduces start immediately) – 1.75n (fast nodes can end up doing 2 reduce jobs)。**Optimizing** using Local Aggregation: "**Combiners**": **"Plain" MapReduce**: • Map • One input pair -> one output pair. • Reduce • Many input pairs with same key -> one output pair. **Word Count: Issue**: Large pairs. Mapper output is written to disk and read from the disk by reducer • Writing and reading to disks is relatively slow • Potential for network congestion. **Use Combiners (benefit)** • Perform local aggregation on the output of the map function but before the shuffle/sort phase – Word Count: Count occurrence of each word locally resulting in a pair (w,n) representing n occurrences of the word w – The number of intermediate pairs A screenshot of a computer code

Description automatically generatedthat go to the shuffle/sort is the number of unique words from that map • This works when the operation on the pairs is associative and commutative. **Hadoop Combiner** is an optional class in the MapReduce framework – The combiner receives intermediate pairs from mappers • You can save networking time by doing local aggregation. **When not good?** consider an operation where the processing of values depends on all the data being present. when the aggregation operation is not commutative and associative, or when the aggregation logic is complex and might lead to incorrect intermediate results. **Hadoop Ecosystem**: Hive – SQL Database on Hadoop • HBase – NoSQL Database on Hadoop • Flume – Log processing and storage on Hadoop. **Lec 01:** “**structured**” data to be things like • Relational databases. Tasks Using Unstructured Data: Information retrieval, Labeling, Structure-finding. **Unstructured Data are often BIG** • Volume (size) • Variety (representation) • Velocity (stored vs. streaming) • Veracity – Accuracy and precision – Errors, completeness, and integrity • Validity – Data governance and management **Big Data Challenges** • Storage • Computation. Storage – E.g., Hadoop Distributed File System, relies on replication, sharding • Computation – E.g., MapReduce programming model, implemented by e.g., MongoDB, Hadoop. **Adapting to Big and Unstructured Data** • Enterprises are looking to a new generation of databases referred to as NoSQL • Document-centric • Flexible schema • Often, no traditional relations. **Lec 02: Unstructured Data**: Structure, while not formally defined, can still be implied. Data with some form of structure may still be characterized as unstructured if its structure is not helpful for the processing task at hand. Unstructured information might have some structure (semi-structured) or even be highly structured but in ways that are unanticipated or unannounced. **Compositional structure:** array, structure. **Relational structure:** . Data are organized into tables with rows and cols. i.e. SQL database, linked with key and ref. **choosing a data standard** • Flexibility • Interoperability • Efficiency. **XML:** Legal Unicode, No syntax chars “<,&”Use entities,”&lt”. Tags are case-sensitive. Tags cannot use special chars or start with digit. Tags cannot overlap. HTML5 is valid XML. **Json:** Developed for browser <--> web server communication of JavaScript objects. JSON's simplicity offers faster parsing and less overhead, beneficial for web applications, syslog, config, while XML's structured format is better for document-centric applications requiring validation. **UIMA** – Unstructured Information Management Architecture: • artifact – a segment of unstructured content (e.g., a document, a video etc.) • analysis – act of assigning semantics to a region of an artifact • analytic software component or service that performs the analysis • artifact metadata – results of the analysis of an artifact by an analytic. : in-memory for processing, XML-based for communicating. **UIMA Implementation** • Common Analysis Structure (CAS): • Annotator (analysis engine), Aggregate Analysis Engine. UIMA **Standoff** Annotations. Keep original text • Add annotations with associated offsets in the original text. **Lec 03: Text Tasks** • Information retrieval: return info given a query. • Summarization • Question answering • “Labelling” tasks: sentiment analysis. • “Structure-finding” tasks: topic in collected doc • Annotation tasks: building blocks. **Tokens**: Imposing structure, A lexical token (or term) is a substring in a corpus • Tokens typically selected to carry semantic meaning • A collection of unique tokens is a dictionary. **Inverted Indices** • An inverted index is a data structure, created from a corpus • Given a token, tells where that token occurs in the corpus. **Tokenization**: Roughly: identifies the words in documents • Form of annotation. Once complete, a document is represented only by the sequence of tokens it generates • Searches are done using tokens rather than original document, so useful to map different inputs to the same token if they "mean the same thing." Tokenization is a way to explicitly ignore differences we don't care about. **Tokenizing Problems**: meaning lost: CAD, cad. Small words can be important in some queries. **Tokenizing steps** must remain consistent across documents and queries (if applicable) for results to make sense. **Tokenizing Process**: identify parts of a document that are “text”. Perform text segmentation… lower case conversion, additional rule.(problem above). **Lec 04: Text Pre-processing**: **Stopwords, Lucene** is an open-source text processing library Part of the Apache family, written in Java. . StandardAnalyzer – most basic analyzer • Tokenize according to Unicode Standard Annex • Convert to lower case • Remove stopwords. **Stemming**: attempt to simplify morphological variations of words to a common stem. Different variations often have related meaning. Can be done at indexing time and/or as part of query processing. Two basic types: Dictionary-based: uses lists of related words (lemmatizers). Or Algorithmic: use program to det. **Lemmatization**: Stemming sometimes reduces different forms to same lexeme, sometimes not • Often not possible to lemmatize tokens independently of one another. **Lemmatization vs. Stemming** • The word "better" has "good" as its lemma • Stemming does not produce the lemma • The word "walking" has "walk" as its lemma • Stemming produces the lemma. Not for Sentiment analysis. **Tools for Text Cleaning** • **iconv, sed**, **tr, awk, shell, python, OpenRefine.** **Lec 05: Doc repr & retrieved**: term based vs **Boolean search**. Term-based: search with given terms only. Bollean search: term-based + Boolean operation. (AND, OR, NOT). **Boolean seach: Pros:** Easy to understand, Comprehensive results, Efficient. **Cons**: Feast or famine, Coarse(粗糙): no keyword ranking. Can be cumbersome: May lead to queries trying desperately to filter out irrelevant documents. **Vector Representations**: consistent way of mapping object to a vector. One-hot encoding. Vectors: Multiplication by a matrix can produce different length. • Dot Product gives scalar (single number. v · w = v1w1 + v2w2 + … + vpwp. **Bag of word Representations**: BOW discard order but keep multiplicity. Dense repr vs sparse repr. **Query Representations**: A query is a (tiny) doc. **BoW dot product**: d[term1]\*q[term1] + d[term2]\*q[term2]. query {term:n}. d[term] be count of term in document d. q[term] be count of term in query q. So it is 0 iff none of the word in query shown in document, and BoW dot product never be negative. **Similarity**: • Mimics boolean “OR” • If at least one term matches, similarity > 0 • If no terms match, similarity == 0. **Pro:** ranking, easy to understand, fast to compute. **Cons**: sensitive to doc length: a larger doc could reach better grade even less revelent. **Cosine**: Normalizing for document length, improve the drawback above. Divide by. Define similarity to be dot product of normalized document vectors. Eg; query : {apple:1, banana:2}, document: {apple:3, banana:4}. Calculation: L1 norm for ||query|| = (1^2 + 2^2) = , for ||doc|| = 3^2 + 4^2) = . Normalization for query = q’ = {apple = 1/, banana = 2/}. For doc d’ = {apple = 3/}. Cos sim(q’,d’) = 1/x 3/2/ x /5 = 0.984. Sim 属于[0,1] this is the cosine angle between the vectors that represent the doc. Higher if sim value smaller angle (diff). **pro:** Fast to compute, easy, rank, Invariant to document length. **con**: Treats all words equally. **TF-IDF Representation: Term Frequency – Inverse Document Frequency**: a Different vector representation for documents, Replaces BoW counts to reflect term “importance” relative to the corpus, that is less widespread[不常见] word in the corpus get more weight. Many variants[变体]. IDF: For term t, • Let 𝑁 be number of documents • Let 𝑁𝑡 be number of documents containing term t. Term Frequency: Higher frequency implies higher importance • Empirically, a diminishing return[收益递减] is helpful. A white board with blue writing

Description automatically generated **BM25**: Lucene default similarity function, related to TF-IDF. • The numbers 𝑘 and 𝑏 are “tuning parameters” • qt is frequency of term t in the query • 𝑑 is document length; 𝑎𝑣𝑔 𝑑𝑜𝑐 𝑙𝑒𝑛𝑔𝑡ℎ is average document length. **Lec 07: Latent Semantic Analysis** • SVD • Interpretation as compression • Decomposition • Documents as sums of term-collections • Terms as sums of document-collections • Representations • Term-term similarity • Document-document similarity • Term-document similarity. **Compressing the TDM** • Saves space • Reveals patterns • Improves retrieval. **Singular Value Decomposition**: A purple rectangular sign with white text

Description automatically generated A column of Ṁ is a weighted sum of the columns of U. A row of Ṁ is a weighted sum of the rows of V^T. **Lec 08:** **Representation “Accuracy”.** If p = min(m,n), then Mm×n = Ṁm×n no error, but there is no compression. Usually, we set p <= min(m,n), and compute only p columns of U and p rows of V. SVD computes the “best” p vectors. The square of the matrix Σ shows how much each column of U (row of VT ) contributes to the approximation. if p = 0, best compression, best error -> nothing kept, it is the worst possible error. **Choosing p**: “Elbow”, “Share”, Application-driven. **E\_pxp >=0** on the diagonal, 0 elsewhere. It also tells us the importance of the topic i. its entries σ1, σ2, σp. σ^2 tells us how much column i of U and row i of VT improve the approximation (reduce the error).The "**share**" mentioned in the context of choosing p for Singular Value Decomposition (SVD) refers to the cumulative sum of squared singular values normalized by the total sum of squares (SS) of the singular values in the diagonal matrix Σ. Precision = proportion of documents returned that are relevant. Recall = proportion of relevant documents in the corpus that are returned. **Latent Semantic Analysis**: Rows of V^T identify documents that contain the same words. Columns of U identify terms that co-occur in the same document. Consider 1M documents, 1M terms. Original matrix M is a terabyte if 1 byte/entry if dense. (It isn’t dense, but consider.) • If p = 5, then we have 5M + 5 + 5M ≈ 10M • Smaller by a factor of 100,000. **The SVD algorithm** \*has\* to find structure in the corpus to represent it using a small number of columns of U and rows of VT . • Has to find words that tend to co-occur • Has to find documents that tend to have similar words • This procedure is **called Latent Semantic Analysis** • “Latent” because structure is hidden but discovered through the analysis. “Semantic” because the structure that is found often corresponds to groups of terms that describe similar concepts. **SVD as dimensionality reduction**: • LSA/SVD gives us • Much more compact representations – length p • A representation of all terms and documents in the same space. Can use Cosine similarity, dot product, or other techniques like Euclidean distance, etc. • **Can compare document to document and document to term! Adding documents**: Representation depends on entire corpus • To allow a new document to modify the representation (of all words and documents), must “re-compute” the SVD • (There are algorithms for updating SVDs without a total re-do.). **Non-negative Matrix Factorization**: Similar (sometimes easier) interpretation because no negative weights. • Cannot approximate the original matrix any better than SVD does. (Why?) • Unlike SVD, not guaranteed to find global minimum. • Topics not “ordered” by importance. **Probabilistic Latent Semantic Analysis**: gives probabilistic interpretation of topics. **Language Structure** • What does it mean to recover structure? • Word Embeddings • Neural Networks • Autoencoders • General networks • Thinking about General Representations. **Word Representations** • Map each word to a vector • Relational structure captured by vector similarity (cosine, Euclidean distance, whatever.) • One “relational structure” may be “similar meaning” • Does the word “boat” mean something similar to the word “ship?” If so, representations should be close. • Representation for “boat” and representation for “mountaineering” should probably be far apart. “**Word Embeddings**” • Word representations where similarity (cosine) is high between words if they are used similarly • “Used Similarly” is captured by “used nearby” in text. • Corpus is not divided into documents. **Topic Models** • Similarity: Occur within same documents • Capture information about documents in the corpus • Representations supposed to be useful within the given corpus • Can work even on small-ish datasets. **Word Embeddings** • Similarity: Occur nearby similar words within sentences • Capture information about language use in general • Representations supposed to be useful more generally • Work best when learned from very large datasets. **(Simple) Word Embedding Uses** • Building representations of sentences/paragraphs/documents by summing the vectors of their words • Used as input representations for sequences of words going into neural networks • Building up more complex embeddings that take context into account. **word2vec**: NN-based word representation learner • m input units (one per word) • p hidden units (dimension of representation) • m output units (one per word) • Output not same as input. **Purpose of word2vec** was never to “predict an output” • What matters is the representation created as a “side-effect” of training. **Intermediate Representations** • Same intermediate representation → same output • Advantageous for NN to map different inputs that should give same output to similar representation. • Train a NN that has at least one layer that “compresses” the input. • Output of that hidden layer, or the hidden layer weights, become your representation. **GloVe** is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. **GloVe**: • b is for “bias” – accounts for some terms being overall more common than other terms • There is no “closed form” solution to find W and b • (Can’t use SVD.) • Rows of W are the word representations **Lec 10:** Rows of V represent documents, In NN terms: • one-layer network • m inputs, p outputs • fully-connected • linear transfer function (no ReLU or anything) • weights are (Σ -1 p×p UT p×m). **SVD/LSA Versus Autoencoder:** auto”-encoder meaning “self”-encoder • Learns smaller representation for each input (column, document) vector. Learned weights Ũm×p give row (word) representations • The entries in Ũ will not exactly match the ones you would get from SVD depending on how you train, but they span the same space. • They’re not “ordered” in terms of importance. Suppose you have the matrix V resulting from applying latent semantic analysis to a termdocument matrix M. Consider a document d in the corpus that was used to create M. **The matrix V obtained from LSA has m rows and p columns. Each column of the V transpose is a p-dimensional vector that represents a document from the corpus. We can start by identifying the column that corresponds to document d. We then use cosine similarity to determine how similar that column is to all of the other columns. The documents corresponding to the five most similar columns can be considered the most similar to d, with the exception of d itself (which will have a similarity of 1.0). We could do the same by comparing columns of M, each of which represents a document. However, in this case, documents will only be considered similar if they contain identical words. By using the representation, documents are considered similar if they contain words from the same topic, even if those words differ from one document to the other. This may retrieve more relevant documents.**

**Guest-lec: Game Analytics** is the application of data analytics to games to better understand players and play. Why? - Better understanding of players and play, Better games, Better player experience, Profit! **Pipeline:** Ingest-Store-Process-analyze-present. **Guest Lec: Big data** refers to the creation, collection, and utilization of data to create value in businesses. It is not the amount of data that matters, but what businesses do (how to create value) with the data that counts. ● Businesses leverage big data to propel growth. **Data** – Structured, Semi structured, Unstructured. Semi-structured data: Semi-structured data has some organizational properties such as semantic tags or metadata to make it possible to search and manage, but there is still fluidity in the data. **IoT**: **Data representation** refers to data structure and encoding for storage, retrieval, and analysis. **Query representation** refers to the users’ queries formulation to retrieve specific information from data. **Indexing** is the process of organizing data to allow for efficient retrieval of data based on certain criteria. **Representation models:** Sensor model, Observation model, Heterogeneity-enabled sensor model, Query model. **(HAR) system:** Central deep learning, Federated learning **Guest Lec: AI Adversaries, 1.** Resilience-against **physical natural corruptions**. Naturalistic Support Artifacts increasing prediction Score. 2. **Multi-Patch Attack**. Harnessing true potential of patch attacks. **Type:** single patch, split patch, mono multi, poly multi. **Single patch based Defence:** Saliency Map, Adversarial Training, Small Receptive Fields., Certified, Others. The **total variation** loss/score is a metric to determine the complexity of an image to its spatial variation in pixel values. Outliers in the TV loss across the image landscape are removed to resurface the image. Model Agnostic, Task Agnostic, # patch Agnostic, Patch Location Agnostic, Patch Location Detector. **Framework:** Channel-wise Blocks, Block wise Total variation, Channel-wise outlier Detection, Pixel Cropping and Masking, Cropped, Inpainting. **Guest Lec: Archetype-based Modeling and Search (ABMS) Motivation**: Exploratory search needs to go beyond the way people typically search. Utility from finding things that people would miss otherwise. Vector representations of authors exist in high dimensional space, and are trained to capture similarities and put them near each other. **ABMS Limitations** • ABMS requires direct computation of a representation for every author in the corpus of interest. • Many document types or platforms are not well suited to the focus on author representations • Some predictions by the model can be difficult to explain. • Strong correlations may or may not be related to the archetype we wished to pursue**. Archetype-based Information Retrieval (AIR) Motivation:** • ABMS offers exploratory utility, but is computationally expensive and difficult to interpret. AIR – BM25 and Elasticsearch. **Searching with AIR** • Indexing and querying documents is very fast compared to training author representations • Keywords extracted can be manually reviewed; this facilitates human-in-the-loop oversight. **AIR & Query Expansion**: There is overlap in AIR and query expansion results. – AIR may find more that describe personal experience – Top 10 results of AIR vs Query Expansion on “Opioid” contain more describing personal experience**. AIR Limitation**s • The words that AIR produces may not be interpretable as to why they are relevant. - Some of the use case applications for AIR have a temporal component (i.e., binge drinking episodes). **Archetype-based Temporal Language Adaptive Stratification (ATLAS): Motivation**: • ATLAS provides a way to look for change in a single author’s archetypal behaviours • Some behaviours of interest do display periodic or transitive behaviour. **ATLAS conclusion** is able to, even in the low-data environment with our synthetic author documents, identify words associated with the change in archetypal behaviour. • ATLAS shows promise in being able to monitor changes online and at scale. **Limitations** • All of the algorithms described find correlations, not causations. • People lie on the internet. • Complexity of language precludes complete accuracy (i.e., coin collectors and cryptocurrency).