# Homework Writeup

## //--Understanding Wikipedia with Latent Semantic Analysis--

/\*

Most of work for data nerds is assembling data into a searchable format. Use SQL for tables. When not in table, or in nice order, hard for humans

to use. For humans it is not as much assembly but "indexing" or "coercion" when ugly. Standard search index let us find documents based off given

term. However, we may want to find words that relate to the word, but we don't give them the term. These kinda results are rare as most standard

search index fail to capture latent structure.

Latent semantic analysis(LSA) is a way to better understand a corpus of documents and the relationship between the words in those docs. Attempts to

break the corpus into relevant concepts. 3 main attributes:

a level of affinity for each document in the corpus

a level of affinity for each term in the corpus

an importance score reflecting how useful the concept is in describing variance in the data set.

By selecting concepts with high affinity, LSA throw away irrelevant noise and merge "co-occurring strands."

This can provide scores of similarity between words, terms, and docs. Base score deper than occurrence by "encapsulating the patterns of variance

in the corpus." Perfect for finding query terms, grouping docs, finding related words.

LSA does this through lower-dimension representation through technique: SVD - more powerful version of ALS factorization (AAS\_CH3)

1. document-term matrix: generated counting word occurrences

- each doc is a column

- each row is a word

- each element represents word importance

2. SVD factorize matrix into 3 matrices

- terms

- docs

- importance

3. remove columns and rows corresponding to least important concepts: low-rank approximation

-multiple approximation with some matrix and produce matrix close ot original, but approx decreases for each concept removed

uses for svd: detect climate trends, face reg, and image compression

\*/

## //--The Document-Term Matrix--

/\*

Before anaylsis, LSA need rawtext of corpus to become document-term matrix. to an extent, the position of term in text relate to importance.

Most common weighting system: TF-IDF - term frequency \* inverse document frequency

- Tells 2 things

- More often the word is repeated in document, more important

- More important to meet a rare word when looking through entire corpus

- Issues

- Frequency of words in corpus is distributed exponentially

- common word appears 10x more than mildy common

- common appear 100x more than rare

- Due to distriibution, rare words would take all importance

- Use log of inverse document frequency instead

- Treats docs as "bag of words"

- No attention to order, structure, etc

- Problem with polysemy: "radiohead" and "band" vs "broke the band" vs "rubber band"

- 10 million docs in corpus, vs a million terms in english

- Generate the document term matrix as a row matrix (collection of sparse vectors) for each doc

-Turn wiki-formatting to plain text

- Plain text split into tokens into root term: lemmatization

- use token to compute term and document

- all steps ecampsulated in Spark's AssembleDocumentTermMatrix class

\*/

## //--Getting the Data--

/\*

$ curl -s -L https://dumps.wikimedia.org/enwiki/latest/\

$ enwiki-latest-pages-articles-multistream.xml.bz2 \

$ | bzip2 -cd \

$ | hadoop fs -put - wikidump.xml

\*/

## //--Parsing and Preparing the Data--

/\*

Need to work with some dependencies. We need to strip out formtatting and grab content. Start by creating a tuple dataset (title, doc content).

Cloud9 gives a bunch of APIs.

\*/

import edu.umd.cloud9.collection.XMLInputFormat

import org.apache.hadoop.conf.Configuration

import org.apache.hadoop.io.\_

val path = "/proj/cse398-498/course/AAS\_CH6/food\_and\_drink/wikipedia\_category\_food\_and\_drink.xml"

@transient val conf = new Configuration()

//what is this? most of this has no notes, so must guess, configuring how we deparse?

conf.set(XMLInputFormat.START\_TAG\_KEY, "<page>")

//formtatting for start of each doc

conf.set(XMLInputFormat.END\_TAG\_KEY, "</page>")

//formatting for end of each doc

val kvs = spark.sparkContext.newAPIHadoopFile(path, classOf[XMLInputFormat],

classOf[LongWritable], classOf[Text], conf)

val rawXmls = kvs.map(\_.\_2.toString).toDS()

import edu.umd.cloud9.collection.wikipedia.language.\_

import edu.umd.cloud9.collection.wikipedia.\_

def wikiXmlToPlainText(pageXml: String): Option[(String, String)] = {

val hackedPageXml = pageXml.replaceFirst(

"<text xml:space=\"preserve\" bytes=\"\\d+\">",

"<text xml:space=\"preserve\">")

val page = new EnglishWikipediaPage()

WikipediaPage.readPage(page, hackedPageXml)

if (page.isEmpty || !page.isArticle || page.isRedirect ||

page.getTitle.contains("(disambiguation)")) {

None

} else {

Some((page.getTitle, page.getContent))

}

//if (page.isEmpty) None

//else Some((page.getTitle, page.getContent))

}

val docTexts = rawXmls.filter(\_ != null).flatMap(wikiXmlToPlainText)

## //--Lemmatization--

/\*

- Turn into bag of terms

- Remove dead words, that, is, etc

- Words can have diff form, like monkey vs monkeys

-Combining these different forms is called stemming or lemmatization

- Stemming: heuristiscs based tech for chopping of chars

\*/

import scala.collection.JavaConverters.\_

import scala.collection.mutable.ArrayBuffer

import edu.stanford.nlp.pipeline.\_

import edu.stanford.nlp.ling.CoreAnnotations.\_

import java.util.Properties

import org.apache.spark.sql.Dataset

def createNLPPipeline(): StanfordCoreNLP = {

val props = new Properties()

props.put("annotators", "tokenize, ssplit, pos, lemma")

new StanfordCoreNLP(props)

}

def isOnlyLetters(str: String): Boolean = {

//returns true if every char in string is letters

str.forall(c => Character.isLetter(c))

}

def plainTextToLemmas(text: String, stopWords: Set[String], pipeline: StanfordCoreNLP): Seq[String] = {

val doc = new Annotation(text) //the doc we are on

pipeline.annotate(doc)

val lemmas = new ArrayBuffer[String]()

val sentences = doc.get(classOf[SentencesAnnotation]) //convert doc to array of sentences

for (sentence <- sentences.asScala;

token <- sentence.get(classOf[TokensAnnotation]).asScala) {

val lemma = token.get(classOf[LemmaAnnotation])

if (lemma.length > 2 && !stopWords.contains(lemma) && isOnlyLetters(lemma)) {

//Specify min req on lemmas to weed out dead words

//must have length greather than 2, cant be in stop words, must be only letters

lemmas += lemma.toLowerCase

}

}

lemmas

}

val stopWords = scala.io.Source.fromFile("/proj/cse398-498/course/aas/ch06-lsa/src/main/resources/stopwords.txt").getLines().toSet

val bStopWords = spark.sparkContext.broadcast(stopWords)

//broadcast to save memory and not be able to overwrite it

val terms: Dataset[(String, Seq[String])] = docTexts.mapPartitions { iter =>

val pipeline = createNLPPipeline()

iter.map { case(title, contents) =>

(title, plainTextToLemmas(contents, bStopWords.value, pipeline))

}

} //Use mapPartitions so that we only initialize the NLP pipeline object once per partition instead of once per document.

## //--Compute the TF-IDFS--

/\*

As of rn term is a datset of sequences of terms, each relating to a single doc. Next need to count occurance. Use Estimator and Transformer.

\*/

val termsDF = terms.toDF("title", "terms") //add column titles

termsDF.show()

/\*

+--------------------+--------------------+

| title| terms|

+--------------------+--------------------+

|Agricultural science|[agricultural, sc...|

| Agriculture|[agriculture, agr...|

| Annual plant|[annual, plant, a...|

| Arable land|[arable, land, ar...|

| Almond|[almond, the, alm...|

| Agrippina the Elder|[agrippina, elder...|

| Aquaculture|[aquaculture, aqu...|

| Anaxagoras|[anaxagoras, anax...|

| Archery|[archery, archery...|

|Action Against Hu...|[action, hunger, ...|

| Alcoholism|[alcoholism, alco...|

| Beer|[beer, beer, one,...|

| Bubble tea|[bubble, tea, bob...|

| Boomerang|[boomerang, boome...|

| Beagle|[beagle, the, bea...|

| Bubble and squeak|[bubble, squeak, ...|

| Brewing|[brewing, brewing...|

| Bean|[bean, bean, seed...|

| Bacterial vaginosis|[bacterial, vagin...|

| Body mass index|[body, mass, inde...|

+--------------------+--------------------+

\*/

val filtered = termsDF.where(size($"terms") > 1) //remove all docs that have less than 1 word

/\*

CountVectorizer is an Estimator we can use to calculate TF. Scans all data, builds a map, is a Transformer. Then is used to generate

frequency vector for each doc

\*/

import org.apache.spark.ml.feature.CountVectorizer

val numTerms = 2000

val countVectorizer = new CountVectorizer().

setInputCol("terms").

setOutputCol("termFreqs").

setVocabSize(numTerms)

//corpus has shit ton of words, putting a limit will give us only most freq words

val vocabModel = countVectorizer.fit(filtered)

//have to fit the transformer with a dataset then can transform the data

val docTermFreqs = vocabModel.transform(filtered)

docTermFreqs.cache()

//cached since used twice at least, calculate IDFS and final document-term matrix

//counts the number of documents in which each term in the corpus appears and usescounts to compute the IDF scaling factor per term.

import org.apache.spark.ml.feature.IDF

val idf = new IDF().

setInputCol("termFreqs").

setOutputCol("tfidfVec")

val idfModel = idf.fit(docTermFreqs)

val docTermMatrix = idfModel.transform(docTermFreqs).select("title", "tfidfVec")

//as we move from DF to vector matries, we cant search by string. important to save mapping of pos in matrix to terms and docs. pos in

//term vectors are == to cols in document-term matrix

val termIds: Array[String] = vocabModel.vocabulary

/\*

row to document title harder. use zipWithUniqueId(). "it will assign the same unique IDs to the transformed rows as long as the

transformations don’t change the number of rows or their partitioning

\*/

val docIds = docTermFreqs.rdd.map(\_.getString(0)).

zipWithUniqueId().

map(\_.swap).

collect().toMap

## //--Singular Value Decomposition--

/\*

SVD takes a m\*n and return 3 matrixes

- M = U S V ^T

- U = m \* k: columns orthonormal basis for the document space

- S = k \* k: each entries correspond to strength of concept

- V^T = k \* n: columns form orthonormal basis for the term space

- In LSA case, m = #docs & n = #terms & k =#concepts to keep,

- when k = n, product of factor matrixes = original matrix exactly

- k < n, low rank approx: typical case

- as of spark2.2, no SVD for DF but for RDD, so must convert

\*/

import org.apache.spark.mllib.linalg.{Vectors, Vector => MLLibVector}

import org.apache.spark.ml.linalg.{Vector => MLVector}

val vecRdd = docTermMatrix.select("tfidfVec").rdd.map { row =>

Vectors.fromML(row.getAs[MLVector]("tfidfVec"))

}

//to calc SVD, wrarp RDD in rowmatrix

import org.apache.spark.mllib.linalg.distributed.RowMatrix

vecRdd.cache()

//cache to save O(nk) storage (n = storage, k = passes)

val mat = new RowMatrix(vecRdd)

val k = 1000

val svd = mat.computeSVD(k, computeU=true)

//SingularValueDecomposition(org.apache.spark.mllib.linalg.distributed.RowMatrix@a779a2f,[11727.069069516834,4602.546620412862,3224.7304432149404,..

/\*

Review

- term space vector: weight on er term

- document space vector: weight on er doc

- concept space vec: weight on er concept

Each item^ defines an axis, and weight means length along axis

- V: matpping between term space and concept space

- U: row is doc, coluimn is concept, doc and concept space

- S:each diagonal erlement correpsond to a sincle concept (column in V & U)

- Magnitude relate to importance

- Shitty SVD: throw away (n-k) that has least weight until n = k

- Also happenes to be sqroots of eigenvalues of MM^T

\*/

## //--Finding Important Concepts--

/\*

SVD output a shit ton of numbers. V matrix represents concepts through the terms that are important to em.

V col for each concept, row for each term

Position relevate to importance.

\*/

import org.apache.spark.mllib.linalg.{Matrix,

SingularValueDecomposition}

import org.apache.spark.mllib.linalg.distributed.RowMatrix

def topTermsInTopConcepts(

svd: SingularValueDecomposition[RowMatrix, Matrix],

numConcepts: Int,

numTerms: Int, termIds: Array[String]) : Seq[Seq[(String, Double)]] = {

val v = svd.V

val topTerms = new ArrayBuffer[Seq[(String, Double)]]()

val arr = v.toArray

for (i <- 0 until numConcepts) {

val offs = i \* v.numRows

val termWeights = arr.slice(offs, offs + v.numRows).zipWithIndex

val sorted = termWeights.sortBy(-\_.\_1)

topTerms += sorted.take(numTerms).map {

//take as many as requested

case (score, id) => (termIds(id), score)

//remap integer->term using termIDS

}

}

topTerms

}

//using almost same process we can get top docs, but U is in distrubtued space

def topDocsInTopConcepts(

svd: SingularValueDecomposition[RowMatrix, Matrix],

numConcepts: Int, numDocs: Int, docIds: Map[Long, String])

: Seq[Seq[(String, Double)]] = {

val u = svd.U

val topDocs = new ArrayBuffer[Seq[(String, Double)]]()

for (i <- 0 until numConcepts) {

val docWeights = u.rows.map(\_.toArray(i)).zipWithUniqueId()

//trick from last chapter to maintain continuity between rows in matrix

topDocs += docWeights.top(numDocs).map { //rows where DF is derived

case (score, id) => (docIds(id), score)

}

}

topDocs

}

//Inspect top few concepts

val topConceptTerms = topTermsInTopConcepts(svd, 4, 10, termIds)

val topConceptDocs = topDocsInTopConcepts(svd, 4, 10, docIds)

for ((terms, docs) <- topConceptTerms.zip(topConceptDocs)) {

println("Concept terms: " + terms.map(\_.\_1).mkString(", "))

println("Concept docs: " + docs.map(\_.\_1).mkString(", "))

println()

}

/\*

Concept terms: ale, usa, brewing, bronze, silver, gold, brewery, lager, stout, beer

Concept docs: World Beer Cup, GABS Hottest 100 Aussie Craft Beers of the Year,

List of microbreweries, Beer, Champion Beer of Britain, Microbrewery, Brewing, List of breweries in British Columbia,

Alewife (trade), Beer by region

Concept terms: ale, usa, bronze, brewing, silver, lager, gold, stout, pale, brewery

Concept docs: World Beer Cup, Boiling (brewing), Oshi sabo, Endicott Pear, Gallo Salame,

Shi Youzhen, Pre-charged pneumatic (PCP), Bob Taco, Tempe bongkrek, Lentinula edodes

Concept terms: soil, clay, organic, plant, water, mineral, nitrogen, nutrient, matter, material

Concept docs: Soil, Soil compaction (agriculture), Base-cation saturation ratio, Organic farming, Soil pH,

Crop rotation, List of universities with soil science curriculum, Agricultural soil science, No-till farming, Plough

Concept terms: file, gardens, garden, jpg, private, soil, house, park, heritage, rose

Concept docs: Heritage gardens in Australia, Soil, History of gardening, Garden design, Japanese rock garden,

Community gardening in the United States, Garden roses, Gardening, Community gardening, List of New York state parks

\*/

//LOOK THEY ARE ACTUALLY ALL SUPER RELEVANT

//In the textbook, they kept getting image based results, but not an issue here! nonetheless, we fix wikiXMLToPlainText

/\*

def wikiXmlToPlainText(xml: String): Option[(String, String)] = {

...

if (page.isEmpty || !page.isArticle || page.isRedirect ||

page.getTitle.contains("(disambiguation)")) {

None

} else {

Some((page.getTitle, page.getContent))

}

}

\*/

## //--Querying and Scoring with a Low-Dimensional Representation--

/\*

How relevant were our results? Use Cosine Similarity (CS) to mesaure angle btwn 2 vectors.

- Vector in same direction = relevant

- computed as dot product of vectors divided by length

- relevance score between term and doc simplt be element in matrix at intersection!!!

- however this method is a bit shallow, based entirely on frequency.

"For example, if the term “artillery” appears nowhere in a document on the “Normandy landings” article but it

mentions “howitzer” frequently, the LSA representation may be able to recover the relation between “artillery”

and the article based on the co-occurrence of “artillery” and “howitzer” in other documents." <- TOO GOOD EXAMPLE, SORRY FOR QUOTING

\*/

## //--Term-Term Relevance--

/\*

What LSA offer

- Accounting for synonymy by condensing related terms

- Accounting for polysemy by placing less weight on terms that have multiple meanings

- Throwing out noise

Need to discover cosing similarity

- Linear algerbra proves that CS between 2 columns in the reconstructed matrix = CS between columns in SV^T

- Finding CS btwn a term(querry) and all others

- normalizing each row in VS to length 1

- multiply the row corresponding to term

- Each element in result vector rtepresent a similarityh btwn term and querry term

\*/

import breeze.linalg.{DenseMatrix => BDenseMatrix}

import com.cloudera.datascience.lsa.LSAQueryEngine

val termIdfs = idfModel.idf.toArray

val queryEngine = new LSAQueryEngine(svd, termIds, docIds, termIdfs)

queryEngine.printTopTermsForTerm("cheese")

queryEngine.printTopTermsForTerm("rice")

queryEngine.printTopTermsForTerm("yoo-hoo")

/\*

(cheese,0.9999999999999987), (description,0.5087984242519732), (slightly,0.4201983808537367), (hard,0.4156986159998422), (texture,0.3917936414457268), (image,0.36087570804269453), (region,0.33069519644851925), (originate,0.31620751438226624), (similar,0.31370774567535686), (make,0.3113790958611898)

(rice,1.0000000000000009), (coconut,0.4220940870605958), (mix,0.40757177022896), (cook,0.3997445630210344), (dish,0.37193569104631014), (mixed,0.35279727095906804), (sesame,0.35117687090832833), (wrap,0.3476769455099212), (thailand,0.33834983289671694), (consist,0.33721780539766544)

java.util.NoSuchElementException: key not found: yoo-hoo

at scala.collection.MapLike$class.default(MapLike.scala:228)

at scala.collection.AbstractMap.default(Map.scala:59)

at scala.collection.MapLike$class.apply(MapLike.scala:141)

at scala.collection.AbstractMap.apply(Map.scala:59)

at com.cloudera.datascience.lsa.LSAQueryEngine.printTopTermsForTerm(RunLSA.scala:250)

... 83 elided

\*/

//why is there no term called yoo-hoo? its literally one of the best drinks ever... oh wait, it probably got filtered out xD

## //--Document-Document Relevance--

/\*

To find the similarity between 2 docs, comput CS between u1Ts and u2Ts where ui is the row in U corresponding to doc i.

To find one doc vs all, compuite normalize(US)u\_t

\*/

queryEngine.printTopDocsForDoc("Yoo-hoo") //NO YOOHOO WHYYYYYY

queryEngine.printTopDocsForDoc("Cheddar Cheese") //Kinda too specific I guess

queryEngine.printTopDocsForDoc("Wine")

/\*

(Wine,1.0000000000000009), (Winemaker,0.8965201831052058), (Wine auction,0.8903566971796215),

(Wine tasting descriptors,0.8775102691693857), (Wine law,0.8751851408414633), (Wine and food matching,0.8719598374411215),

(Wine for the Confused,0.8668480858989095), (André Jullien,0.865373232160856), (Wine fault,0.8609460307620234),

(Wine tasting,0.8585223710035453)

\*/

## //--Document-Term Relevance--

/\*

- This is equal to udT S vt, where ud is the row in U corresponding to the document, and vt is the row in V corresponding to the term.

- similarity between a term and every document is equivalent to US vt

- similarity between a document and every term comes from udT SV

\*/

queryEngine.printTopDocsForTerm("beer")

/\*

(World Beer Cup,1677.542939316098), (Beer,798.7440445136723), (Beer by region,691.5097257567146),

(Beer festival,439.079028078366), (Low-alcohol beer,346.4795868033664), (Brewing,326.70313247688756),

(Microbrewery,300.20738619575474), (List of drinks,279.2270927188251),

(List of alcohol laws of the United States,242.72705639915338), (Pub,207.7323304673555)

\*/

## 

## //--Multiple-Term Queries--

/\*

Finding doc relevant to single term by selecting row corresponding to query in V

Equivallent to multplying V by term vector (nonzero entry)

Multiple terms: multiplying V by term vector for multiply terms???

\*/

//To maintin weight scheme, set val for each term to inverse

val termIdfs = idfModel.idf.toArray

queryEngine.printTopDocsForTermQuery(Seq("cheese", "beer"))

/\*

(World Beer Cup,4843.644573090004), (List of cheeses,2535.2993404319186), (Beer,2306.257447652742),

(Beer by region,1996.1447443405243), (Cheese,1293.794986106358), (Beer festival,1268.212951393017),

(Low-alcohol beer,1000.6471198980646), (Brewing,943.4903139353289), (Microbrewery,867.230443921393),

(List of drinks,806.3654642144587)

\*/

LSAQueryEngine was not copied in to save room, but all code was explained and all concepts too through the earlier subtitles

# 

# Extension Writeup

## //--Machine Learning Searching for Itself--

To save time and space, I won’t be including anything but the modified code. I created my own wiki dump that included the two following categories: “machine learning” and “artificial intelligence.” I wanted to test if SVD could generate the expected categories given these articles, classification if you so will call it in a sense (and semi clustering as well). Also I updated stop words. It now has over 400-500, so it cleans out a lot more!

val path = "/proj/cse398-498/aas423/AAS\_CH6/machine-learning\_artificial-intelligence.xml"

//query words are concatenated with '-'

//val path = "/proj/cse398-498/aas423/AAS\_CH6/ML\_AI.xml"

//words are unmodified

@transient val conf = new Configuration() //what is this? most of this has no notes, so must guess, configuring how we deparse?

conf.set(XMLInputFormat.START\_TAG\_KEY, "<page>") //formtatting for start of each doc

conf.set(XMLInputFormat.END\_TAG\_KEY, "</page>") //formatting for end of each doc

val kvs = spark.sparkContext.newAPIHadoopFile(path, classOf[XMLInputFormat],

classOf[LongWritable], classOf[Text], conf)

val rawXmls = kvs.map(\_.\_2.toString).toDS()

import edu.umd.cloud9.collection.wikipedia.language.\_

import edu.umd.cloud9.collection.wikipedia.\_

def wikiXmlToPlainText(pageXml: String): Option[(String, String)] = {

// updated parser for updated wiki dumps 2021

val hackedPageXml = pageXml.replaceFirst(

"<text bytes=\"\\d+\" xml:space=\"preserve\">",

"<text xml:space=\"preserve\">")

val page = new EnglishWikipediaPage()

WikipediaPage.readPage(page, hackedPageXml)

if (page.isEmpty || !page.isArticle || page.isRedirect ||

page.getTitle.contains("(disambiguation)")) {

None

} else {

Some((page.getTitle, page.getContent))

}

}

… (Restatement of definitions from above)

val queryEngine = new LSAQueryEngine(svd, termIds, docIds, termIdfs)

println("Machine")

queryEngine.printTopTermsForTerm("machine")

println("Artificial")

queryEngine.printTopTermsForTerm("artificial")

println("Machine learning")

queryEngine.printTopDocsForTermQuery(Seq("machine", "learning"))

println("Artificial intelligence")

queryEngine.printTopDocsForTermQuery(Seq("artificial", "intelligence"))

println("Machine learning Artificial intelligence")

queryEngine.printTopDocsForTermQuery(Seq("artificial", "intelligence", "machine", "learning"))

## //--Results--

/\*

Concept terms: ventures, huawei, icad, samsung, galaxy, bear, instrumental, marvin, teach, situate

Taking a look at this and analyzing this as a human person would, I would say the only two words that has any semblance of resemblance to the category would be teach and icad, which is a company that specializes in detecting cancer, world number one leader when it comes to Breast Cancer detection using AI <https://www.icadmed.com/profoundai.html>

Concept docs: Human Problem Solving, Glossary of artificial-intelligence, Subclass reachability, 0music, Alesis Artificial Intelligence, Query-level feature, Nature Machine Intelligence, Discovery system (AI research), ML Fairness, Thomas Bolander

Looking over this at first glance, the listed documents all have a clear and distinct relationship with the categories, but two documents stand out: 0music and Thomas Bolander. After a quick search with 0music, it seems to be a music creation ai, with no human assistance. Listening to the pieces, I would have never guessed it. <https://en.wikipedia.org/wiki/0music> As for Thomas Bolander, Google Scholar showed confidently that he is someone with big impact in the field. He literally has 105 publications and has been publishing every year since 2002 till now. <https://scholar.google.com/citations?hl=en&user=FxFF9kMAAAAJ&view_op=list_works&sortby=pubdate>

Concept terms: dataset, classification, image, text, none, preprocessing, brief, default, format, creator

This run here generated some better related terms. Nothing really stood out drastically

Concept docs: List of datasets for machine-learning research, Computer vision, Deep learning, Convolutional neural network, Google, Machine learning in bioinformatics, Document classification, GPT-2, Natural language processing, Applications of artificial-intelligence

This here is like the perfect run. We see a bunch of documents that relate to machine learning, while not being classified as directly machine learning, making it a pretty good generator. We have mentions of Computer Vision, Deep Learning, CNN, Document classification, and Natural language processing. What surprised me more was the mention of OpenAi’s algorithm GPt2, which is used for text generation. <https://openai.com/blog/better-language-models/>

Concept terms: game, player, theory, equilibrium, collective, payoff, strategy, nash, google, intelligence

This run seems to be focused entirely on game theory and its application in AI

Concept docs: Game theory, Collective intelligence, Machine learning in video games, General game playing, Artificial intelligence, Google, Computer Arimaa, Applications of artificial-intelligence, Convolutional neural network, AI Dungeon

As such we can see the generation of game based documents

Concept terms: mathbf, mathcal, phi, kernel, dataset, epistemic, embedding, textsf, classification, psus

This chunk here seems to focus primarily on on tex command (all the weird looking ones)

Concept docs: Dynamic epistemic logic, Kernel embedding of distributions, Bayesian interpretation of kernel regularization, List of datasets for machine-learning research, Tensor sketch, Differentiable neural computer, Random forest, Learning with errors, Description logic, Convolutional neural network

The top result actually is something that correlates deeply with machine learning, Dynamics epistemic logic: the study of how agents knowledge changes when an even occurs. This generation of documents seem to be about understanding the core concepts of machine learning and how it applies. <https://en.wikipedia.org/wiki/Dynamic_epistemic_logic>

\*/

/\*

Original Dump

Machine

(machine,1.0000000000000018), (learning,0.7590785618323165), (learn,0.7159204719069167),

(supervise,0.6025095960230853), (algorithm,0.5762248089544058), (svm,0.5497873966433428),

(datum,0.541928582567879), (improve,0.5389858396421375), (research,0.5377524679275919), (perform,0.5353734823679331)

As mentioned above, this run here is an unmodified copy of the xml dump. All words here look good in terms of semblance, with SVM being support vector machines (supervised learning models) and datum means a piece of information or a fixed starting point, which kinda makes sense here)

Artificial

(artificial,0.999999999999999), (aus,0.749740625771433), (research,0.6485758215679959),

(intelligence,0.613955513412785), (human,0.6121735956569119), (superintelligence,0.6064729559611044),

(intelligent,0.5876906850447825), (develop,0.5686512109120677), (technology,0.5566786994439001), (future,0.5545212273181872)

The second term was the only one that stood out in terms of context, but looking into it, it is an artificial urinary sphincter, which controls flow of urine out of bladder, which is as artificial intelligence and machine learning as it can get nowadays in science (other than hacked insulin pumps)

Machine learning

(Machine learning,64.06587924879922), (Outline of machine learning,38.019270405043585), (Deep learning,27.978329098941217

(Quantum machine learning,24.46067021373109), (Artificial intelligence,23.350181318303434),

(Machine learning in bioinformatics,20.158358781214996), (Federated learning,19.446850253852983),

(Machine learning in video games,14.921774612438062), (Supervised learning,14.243512748461708),

(Semi-supervised learning,14.077272629054148)

It is interesting here that most articles didn’t score well at all. Taking a look at the top 5, I wanted to see the mentions of each word here.

Machine learning - 158 mentions,

Outline of machine learning (keep in mind this is only an outline and contains no actual text, but hyperlinks) - 83 mentions,

Deep learning - 20,

Quantum machine learning - 69

Artificial intelligence - 18

Machine learning in bioinformatics - 63

Taking a look at this trend, it seems that even though machine learning was definitely mentioned more in some of the lower percentages, but the documents that had high term frequency but low score tended to be topics that were more specific to a very deep (and complicated) slice of machine learning

Artificial intelligence

(Collective intelligence,70.97626783060878), (Outline of artificial intelligence,49.03109536113169),

(Artificial intelligence,40.44251486098382), (Applications of artificial intelligence,35.308890217846674),

(Regulation of artificial intelligence,24.938955736518853), (Artificial general intelligence,22.904570303826254),

(Artificial intelligence arms race,15.756590575977643), (Computational intelligence,13.722205143285063),

(Distributed artificial intelligence,12.51207201936948), (Artificial consciousness,12.0435363571586)

Collective intelligence - 11

Outline of artificial intelligence - 67

Artificial intelligence - 110

Applications of artificial intelligence - 127

Regulation of artificial intelligence - 88 The scores seen here are clearly higher than the score received in machine learning, which is why the top 5 documents had such high counts. What is interesting is that the first document doesn’t really focus entirely on artificial intelligence that much, but the term artificial intelligence, derives most of its meaning from collective intelligence.

Machine learning Artificial intelligence

(Machine learning,74.45584407810814), (Collective intelligence,73.10747486858763),

(Artificial intelligence,63.79269617928727), (Outline of artificial intelligence,56.61167473702667),

(Outline of machine learning,47.11644726336468), (Applications of artificial intelligence,44.55520293688335),

(Deep learning,33.703204947323876), (Artificial general intelligence,29.75701441942675),

(Quantum machine learning,25.648239534517653), (Regulation of artificial intelligence,25.620542947081656)

These results seem to have just added the two terms score of each document together.

\*/

The only thing of importance here from this dataset was testing to see if making the words a single concatenated term would end up with different and perhaps better results?

/\*

Modified Dump

From machine learning to machine-learning, and same to artificial-intelligence

Machine

(machine,1.0), (artificial,0.5973960374140815), (research,0.5718084938517987), (computer,0.5396223887672426),

(experience,0.511108482933532), (learn,0.5078156801277549), (learning,0.5053879556462173), (researcher,0.497650020553514),

(human,0.4955690896447456), (aus,0.48729287882214994)

Nothing here too different, but it is cool to see that the next term immediately generated was artificial instead of learn, since the two were concatenated. Seems to have generated more terms that relate, instead of follow (I wonder why? xD)

Artificial

(artificial,0.9999999999999979), (aus,0.6546660076178309), (machine,0.5973960374140815),

(research,0.5789052009544935), (superintelligence,0.5686795374831048), (human,0.5503246275692328),

(intelligent,0.5192582229639386), (fiction,0.4987576302767342), (include,0.49171264739018794), (ethic,0.49138284209145766)

Same case here, with the terms relating better, but it is interesting to see ethic and fiction appear. I think fiction in this case may actually refer to massive amount of stories pertaining to ai in science fiction stories.

Machine-learning

(Machine learning,79.10730897555501), (Outline of machine-learning,65.08643299547283),

(Artificial intelligence,55.25041099543391), (Deep learning,50.40735192410311),

(Solomonoff's theory of inductive inference,34.313649107103416), (Artificial consciousness,29.05404357332539),

(Supervised learning,27.726003573567084), (Federated learning,27.523545250934916),

(Semi-supervised learning,27.15696674094668), (Concept learning,25.424872266855903)

One clear difference here was seeing that every document scored increasingly better than before, with even a different order and different text appearing

Machine learning - 158

Outline of machine learning - 83

All articles below here are set higher due to relevance to term, and not frequency

Artificial intelligence -18

Deep learning - 20

Solomonoff's theory of inductive inference - 2

Artificial-intelligence

(Collective intelligence,182.3685222656075), (Outline of artificial-intelligence,84.63336314412014),

(Artificial intelligence,75.80672205260252), (Regulation of artificial-intelligence,53.23833914340918),

(Applications of artificial-intelligence,48.398490130372025), (Artificial general intelligence,39.461439022439656)

(Computational intelligence,33.86136393961341), (Artificial consciousness,29.724068434891308),

(Artificial imagination,26.700743307382464), (Artificial intelligence arms race,21.825735991051364)

Immediately noticed that collective intelligence, which scored around 70 last time scored sigNIFICANTLY higher, by uhhh about 112 more percent…?

Collective intelligence - 11

Outline of artificial intelligence - 67

Artificial intelligence - 110

Regulation of artificial intelligence - 88 these two swapped, for the better? Doesn’t seem so

Applications of artificial intelligence - 127

Machine-learning Artificial-intelligence

(Collective intelligence,185.83271121378914), (Artificial intelligence,131.05713304803652),

(Outline of artificial-intelligence,104.96361223548584), (Machine learning,97.81688897432961),

(Outline of machine-learning,81.7938273791837), (Applications of artificial-intelligence,63.544947787580156),

(Artificial general intelligence,60.36072494642577), (Deep learning,59.622895626338746),

(Artificial consciousness,58.77811200821671), (Regulation of artificial-intelligence,53.959004175270756)

\*/