Data Science ≠ Machine Learning: Some Thoughts on the Role of Data Management in the new AI-Tsunami

Jens Dittrich
Saarland Informatics Campus

Talk Manuscript for my Keynote at DEEM Workshop at SIGMOD 2018 (accompanied by slides, scroll down for slides) will post it on http://twitter.com/jensdittrich

Prelude: Buzzword Bullshit Bingo

Buzzword [People not understanding each other]

<something>


The political game [parliament]

Data Science vs Machine Learning

clarify term "data science" [slides from intro lecture]

data science pipeline

where is data management in that pipeline?

similarities with DWH

show different topics

mention projects,

DAWN

HoloClean

Spark, Flink

tons of exciting work going on at this workshop and aiDM, thumbs up, guys!
1.) opportunities for doing research at the intersection of <something> and data management,

Software 2.0, Andrey Karpathy

Where this leads to is:

\[ f(X) = Y \]

software is many of those functions

RNNs are turing complete

In principle any function could be learned by some model

In principle...

which are suitable functions to investigate?

\[ f(X) = Y \]

\[ f("SELECT * FROM ...") = [table] \]

\[ f("SELECT * FROM ...") = [cost estimate] \]

\[ f("SELECT * FROM ...") = \{suitable indexes\} \]

\[ f("SELECT * FROM A", "SELECT * FROM B", "SELECT * FROM ...") = \{suitable indexes\} \]

\[ f("SELECT * FROM ...") = [LLVM bit code] \]

basically, any language translation

Software 2.0: finding out which functions to replace with learned functions

I briefly spoke about this in my VLDB 2017 keynote [title slide]

suitable candidates for functions are in general:

everything that is currently done by humans:

- data cleaning [Stanford, HoloClean]

- data integration

- schema design
- physical design including index selection [we did that using reinforcement learning, still fiddling around with that]

- knob tuning, a fully automatic DBA, "self-driving DBA" [see Pavlo work, our own ongoing stuff]

- DBMS installation and setup, parallel setups

- DBMS maintenance, bug fixing, trouble shooting

- DBMS security, threat prevention, mitigation, and counter measures

- DBMS performance analysis and debugging

- DBMS calibration to hardware

- again: everything currently done by humans: DBAs and consultants

trying to automate these things is not a new idea (some of this was researched 30 years ago already)

but these things should be revisited in the light of deep learning

--

another option is to look at functions like

\[ f(dataset, attribute) = \text{Index structure} \]

So, given a dataset, and an attribute, learn a suitable index.

Indexes are currently implemented and run by machines super well already.

indexes are lightning fast, tons of research, insane performance: >10 million operations per second and thread

---

** "The Case for Learned Indexes [Kraska et al]"**

** Idea of LI:

- observation: in an index, a KEY predicts the physical position of a VALUE

- so why not treat indexing as a regression problem, e.g.
- the index IS the model
- we want to have a function f(KEY) = physical location of the VALUE
- the index predicts the physical location of a VALUE
- so we train a model

- f(dataset, attribute) = Index structure
- learn cumulative density function (CDF)
- learn maximum error during training to guarantee correctness for testing

in more detail:
- train a model (deep learning in this case) to learn the data distribution of a particular attribute
- author's observation: predictions (e.g. index lookups) too slow in tensorflow
- therefore generate C++-index code based on the trained model
- eventually use a hybrid structure, e.g. a model that is recursively refined
- so far read-only structures only, no inserts/updates
- authors discuss B-trees, hash tables, and bitmap indexes
- extensions to multiple dimensions planned

** result summary:

show something from the paper

- performance of the learned indexes in the same ballpark as main-memory optimized indexes
- but: memory footprint much better (~2 orders of magnitude)

** A Criticism of "Learned Indexes" **
Let’s be precise here:

** A Positive Criticism of "Learned Indexes" **

nice high-level observation:

- "an index predicts the position of a value"
- "an index is a model"

- a paper that stirs discussion (we need much of this at SIGMOD/VLDB) rather than super-polished (possibly) incremental papers

- Food for thought

we need more of these kind of controversial papers, much better than some super-polished paper!

** Some Constructive Criticism of "Learned Indexes" **

I am not the first to criticize this work, see blog articles by Thomas Neumann, TUM and the Stanford DAWN group. The LI work has pros and cons (as any work).

some thoughts...

**Trees vs Models**

[show B-tree]

every node in a tree/trie is a model of the data underneath anyways,

not only the entire B- or whatever tree is a model, every node is already a model

this is a somewhat old observation

B-trees are already regression trees (as mentioned in the talk)

But also: each node in a tree is a model!

** coarse-granular/sparse indexes/hybrid indexes **

[show one-level tree]

[show two-level tree]
this is recursive modeling of the data!

hybrid figure from paper

a b-tree is NOT a black-box model (as claimed in the talk), it is white-box

a b-tree is a model of a model of a model, each node and even the leaves in a sparse tree are models!

has been known and exploited for long

e.g. bulkloading, (co-)partitioning, any technique using data distribution for tuning, statistics, data partitioning for joins [e.g. Mirek Riedewald's join papers for instance]

This hybridness is also prominent in index structures that adapt their node types to the particular distribution in a range, e.g. ARTful index, or any other multi-level hybrid index

** Work on modeling the data domain **

histograms, density estimation, any other classical method to describe the data distribution, entropy-based encoding, index compression, succinct trees, minimal perfect hashing, etc. etc. some overlap here, not clear to me what LI improves here, basically these are also statistical learning methods

[show train/test graph]

# Example: decision trees!

[decision tree]

trade-off between generalization and overfitting very visible

more layers: more fitting, eventually overfitting [show]

less layers: less fitting to the data, more generalization [show]

sweet spot

# So, let's talk about A.I.
I mean Adaptive Indexing of course.

** briefly introduce [from Immanuel’s Damon 2018 paper]

[show cracking graphic]

adaptive indexing partitions the data according to the search predicates found in the queries

** a classification of tree-based partitioning strategies:

** tree/trie

tree: according to data inserts -> data distribution
trie: according to the domain of the data inserts

** tree/trie

trai: according to the query predicates -> query predicate distribution
(tria: according to the domain of the query predicates)

trie and tria are typically more or less the same

adaptive indexes LEARN the query predicates and partition the column according like that

standard cracking is a tria!

stochastic cracking is a blend of a tria and a trie! A triaie? (just kidding)

many methods in-between, but all about LEARNING the search predicates

the family of adaptive indexing LEARNS the search predicates

in contrast, LI LEARNs the data distribution

this relationship is unfortunately not discussed at all in the LI paper

note that there is also: "Adaptive Adaptive indexing [Schuhknecht et al ICDE 2018]"

** Code generation **

LI uses code-generation, baselines however are off-the-shelf, not a fully fair comparison

baselines should at least be calibrated to the data distributions/domains

system and code calibration correlates heavily with code generation, very vague boundary between the two
**Repeatability (in general)**

In general (from my humble experience): index structures in main memory are very sensitive to minor programming tricks.

Also see our previous work on indexing:

- tree-indexing [ICDE 2015] and hashing: [PVLDB 2016]
- partitioning [VLDB 2015] and cracking [VLDB 2014, bpa]

**The insanity of evaluating indexes in main memory** [scream]

Change a single code line, and all of a sudden the CPU is enabled to perform completely different optimizations.

- e.g. dependent vs independent statements, memory stalls, parallelization
- there are tons of weird CPU-tricks/optimizations that change everything, e.g. SIMD, pipelining, branch mis-predictions, etc.

So, you assume that your index is fast, because you played some clever algorithmic trick.

Then you find out that the CPU just played some dirty trick you were not aware off.

see: basically, every database on new hardware-paper every published.

the theorists are kind of right here (unfortunately, and it hurts to say this ;-) ): the complexity of your index often won't change, it is just some constant that changes (which, however, may be huge and plays a role in real life...)

so, tiny little code changes may lead to factors in performance differences [visual support, maybe from DAMON 2018 paper?]

as LI are in the same ballpark as state-of-the-art

for LI, as for any other index, I believe it is important to investigate whether these are principal differences or just due to implementation/code generation variations

looking forward to repetitions of LI ;-)
LI summary:

The glass is half full: a nice fresh view on indexing, nice observations, food for thought

The glass is half empty: (still) some doubts on practicality of this approach, and diff to related work.

2.) experiences from teaching <something>

undergrad seminar on "data science", teaching award from student's council for the best seminar

intro lecture "Intro to Databases" - "Intro to Data Science" (ongoing this semester)

syllabus

screenshots from the system

slides

problems

3.) experiences from solving problems in the <something>-domain together with domain experts.

** project with German weather service (DWD)

[some screen shot]

idea: the bug is the feature

error in short term weather forecasts (nowcasting) -> lightning prediction

nowcasting: 2D models

background: thunderstorm: wind upward movement in the cloud (I mean a real cloud) not modelled by nowcasting

hypothesis: error in nowcasting correlates to a cloud where we will see lightning
Executive Summary:

1. remind people that we are one third of what people mean when they say “data science”

2. We should investigate Software 2.0, f(X) = Y

   In particular, when f(X) is a function currently implemented by humans.

3. We need to support and push programs and lectures in data science

4. get out of your LAB and solve some real problems
Data Science ≠ Machine Learning

Some Thoughts on the Role of Data Management in the new AI-Tsunami

Jens Dittrich
Saarland Informatics Campus
twitter.com/jensdittrich

Note: presentation = slides ≠ manuscript
Prelude: Buzzword Bullshit Bingo
Big Data!

Data Science!

ML!

AI!
Data Science vs Machine Learning
Application Domain
Machine Learning
A.I.
Big Data Management
Data Mining
Statistics
Data Science
Artificial Intelligence - Machine Learning

Data Mining

Data Management

Application Domain

Statistics
Math
Programming
Visualization
The Data Science Cake

Ingredients:
- 50g statistics
- 120g linear algebra
- 200g programming
- 1kg visualisation
- 300g software engineering

Additional skills:
- creativity
- out of the box thinking
- grit
- team spirit
Datenbanken im Wasserfallmodell?

- data collection
- data acquisition
- data profiling, exploration & visualization
- data cleaning
- feature engineering
- modeling
- model training
- model testing
- result interpretation

Achtung: Data Transformation kann zu beliebigen Zeitpunkten stattfinden und ist deshalb hier nicht als extra Schritt eingezeichnet.
Achtung: Data Transformation kann zu beliebigen Zeitpunkten stattfinden und ist deshalb hier nicht als extra Schritt eingezeichnet.
Datenbanken im Wasserfallmodell?

- data collection
- data acquisition
- data profiling, exploration & visualization
- data cleaning

Achtung: Data Transformation kann zu beliebigen Zeitpunkten stattfinden und ist deshalb hier nicht als extra Schritt eingezeichnet.

Prof. Dr. Jens Dittrich
Datenbanken 2 / 9

Database Management System (DBMS)
Datenbanken im Wasserfallmodell?

- Datenkollektiv
- Datenakquisition
- Datenprofiling, Exploration & Visualisierung
- Datenreinigung
- Feature Engineering
- Modellierung
- Modelltraining
- Modelltestung
- Ergebnisinterpretation

Achtung: Data Transformation kann zu beliebigen Zeitpunkten stattfinden und ist deshalb hier nicht als extra Schritt eingezeichnet.

Prof. Dr. Jens Dittrich
Datenbanken 2 / 9
Datenbanken im Wasserfallmodell?

Data collection

Data acquisition

Data profiling, exploration & visualization

Data cleaning

Feature engineering

Relational model, relationale algebra, databases, MapReduce, Spark, Workflows

Modeling

Model training

Model testing

Result interpretation

Achtung: Data Transformation kann zu beliebigen Zeitpunkten stattfinden und ist deshalb hier nicht als extra Schritt eingezeichnet.
Datenbanken im Wasserfallmodell?

Data collection

Data acquisition

Data profiling, exploration & visualization

Data cleaning

Feature engineering

Modeling

Model training

Model testing

Result interpretation

Achtung: Data Transformation kann zu beliebigen Zeitpunkten stattfinden und ist deshalb hier nicht als extra Schritt eingezeichnet.

Database Management System (DBMS)

Relational model, relationale algebra, databases, Data Warehousing

Data materialized in a DBMS

Next step (data profiling) outside the DBMS again
Data collection, data acquisition, data profiling, exploration & visualization, data cleaning, feature engineering, modeling, model training, model testing, result interpretation.

Achtung: Data Transformation kann zu beliebigen Zeitpunkten stattfinden und ist deshalb hier nicht als extra Schritt eingezeichnet.

Prof. Dr. Jens Dittrich
Datenbanken 2 / 9

relational model, relationale algebra, databases, Data Warehousing

Schema Design

Relational Model

Database Management System (DBMS)
or: Data Warehouse (DWH)
Datenbanken im Wasserfallmodell?

Data collection → Data acquisition → Data profiling, exploration & visualization → Data cleaning → Feature engineering → Modeling → Model training → Model testing → Result interpretation

Achtung: Data Transformation kann zu beliebigen Zeitpunkten stattfinden und ist deshalb hier nicht als extra Schritt eingezeichnet.

linear regression
classification
decision trees
Deep Learning/RNNs
timeseries
HMM
2 Opportunities

<something> ∩ databases
Software 2.0
f(X) = Y
f( ) = 21
SELECT A FROM...

\[ f( \quad ) = \]
SELECT A FROM...

\[ f(\quad ) = \text{LLVM code} \]
SELECT A FROM...

\[ f() = \sim 42 \text{ ms} \]
SELECT A FROM...

\[ f() = \text{CREATE index ON A} \]
CREATE index ON A
CREATE index ON D

SELECT A FROM...
SELECT B FROM...
SELECT C FROM...
SELECT D FROM...

$f() = CREATE \text{ index ON A}
CREATE \text{ index ON D}$
Deep Learning (m)eats Databases

Jens Dittrich

Saarland University
Saarland Informatics Campus
d:AI:mond.ai
\(? (X) = Y\)
CREATE index ON A

\[ f(\quad) = \quad \]
3 The Case for Learned Indexes

[Kraska et al]
CREATE index ON A

\[ f(\quad) = \quad \]
CREATE index ON A

\[
f(\{\text{tensorflow... code generation...}\}) =
\]

```sql
CREATE index ON A
```
With neural networks, databases and other systems can benefit from evaluating our approach on synthetic and real-world datasets. The potential power of this approach comes from the fact that it replaces traditional index structures with learned indexes. Rather, the key is not the look-up key. The data has to be sorted to allow for efficient range requests. This same general concept also applies to replace B-Trees with learned indexes on specialized hardware, like TPUs.

commonly used in (database) systems. If successful, the core idea of deeply embedding learned models into algorithms and data structures could lead to a radical departure from the way we design systems. While we see the future of compute and storage will be driven by a combination of new hardware and software. The remainder of this paper is outlined as follows: In the next two sections we introduce the general idea of learned indexes, which complements existing work and, arguably, opens up an entirely new research direction for decades-old data structures could lead to a radical departure from the way we design systems. If successful, the core idea of deeply embedding learned models into algorithms and data structures could lead to a radical departure from the way we design systems.

It is important to note that we do not argue to completely replace B-Trees with learned indexes. The B-Tree is a model, the data are the variables. All filters. All possible keys. For new data, B-Trees need to be re-balanced, more importantly, the strong error bounds are not even needed. For e...
Another advantage of the recursive model index is, that we might be thousands of simple linear regression models as they are usually able to learn a wide-range of complex data. (3) It serves as picking an expert which has a better knowledge about certain keys (see also [62]).

Note, that we currently train stage-wise and not fully end-to-end. End-to-end training for hybrid indexes is done as shown in Algorithm 1.

Algorithm 1:

Input:
- threshold, int stages[], NN_complexity

Output:
- new index

1. For each stage, do:
   a. For each model, do:
      i. Calculate the prediction error on the training data.
      ii. If the error is less than the threshold, then:
         a. Create a new B-Tree trained on the data in that model.
      iii. Otherwise, create a new neural network trained on the data in that model.

2. For each key within the range, do:
   a. Calculate the prediction error for the key.
   b. Select the model with the smallest error.
   c. Add the position of the key to the index.

3. Return the index.

Figure 3: Staged models
Research ways easy to save space at the expense of lookup performance which is actually larger. As the main metrics we show the size we used 64-bit keys and 64-bit payload/value.

This data is of course highly non-linear, making the CDF more speedup and size columns indicates how much faster or slower by simply having no index at all). The color-encoding in the a page size of 128 as the space savings compared to a B-Tree with page size of 128 in time in paranthesis. Furthermore, we show the speedup and results are shown in Figure 4. Note, that the page size for B-Trees models, and linear models can be learned optimally.

ear models, had the best performance. This is not surprising as to two hidden layers and layer-width ranging from 4 to 32 cache-line optimization, dense pages (i.e.,

dataset. Finally, to test how the index works on heavy-tail dis-
tively linear and has fewer irregularities than the Weblogs

The main re-

200M user-

−

\[ \sigma = 2. \] The values are scaled up to be integers up to 1B.

Learned Index vs B-Tree performance:

As our baseline, we used a production quality B-Tree imple-
cary) lookup-tables. Often lookup-tables have a solutions to this issues would yield a B-Tree, and histograms

the maps dataset we indexed the longitude of

semester breaks, etc., which are notoriously hard to learn. For

settings.

Figure 4: Learned Index vs B-Tree

<table>
<thead>
<tr>
<th>Type</th>
<th>Config</th>
<th>Map Data</th>
<th></th>
<th>Web Data</th>
<th></th>
<th>Log-Normal Data</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Size (MB)</td>
<td>Lookup (ns)</td>
<td>Model (ns)</td>
<td>Size (MB)</td>
<td>Lookup (ns)</td>
<td>Model (ns)</td>
</tr>
<tr>
<td>Btree</td>
<td>page size: 32</td>
<td>52.45 (4.00x)</td>
<td>274 (0.97x)</td>
<td>198 (72.3%)</td>
<td>51.93 (4.00x)</td>
<td>276 (0.94x)</td>
<td>201 (72.7%)</td>
</tr>
<tr>
<td></td>
<td>page size: 64</td>
<td>26.23 (2.00x)</td>
<td>277 (0.96x)</td>
<td>172 (62.0%)</td>
<td>25.97 (2.00x)</td>
<td>274 (0.95x)</td>
<td>171 (62.4%)</td>
</tr>
<tr>
<td></td>
<td>page size: 128</td>
<td>13.11 (1.00x)</td>
<td>265 (1.00x)</td>
<td>134 (50.8%)</td>
<td>12.98 (1.00x)</td>
<td>260 (1.00x)</td>
<td>132 (50.8%)</td>
</tr>
<tr>
<td></td>
<td>page size: 256</td>
<td>6.56 (0.50x)</td>
<td>267 (0.99x)</td>
<td>114 (42.7%)</td>
<td>6.49 (0.50x)</td>
<td>266 (0.98x)</td>
<td>114 (42.9%)</td>
</tr>
<tr>
<td></td>
<td>page size: 512</td>
<td>3.28 (0.25x)</td>
<td>286 (0.93x)</td>
<td>101 (35.3%)</td>
<td>3.25 (0.25x)</td>
<td>291 (0.89x)</td>
<td>100 (34.3%)</td>
</tr>
<tr>
<td>Learnt Index</td>
<td>2nd stage models: 10k</td>
<td>0.15 (0.01x)</td>
<td>98 (2.70x)</td>
<td>31 (31.6%)</td>
<td>0.15 (0.01x)</td>
<td>222 (1.17x)</td>
<td>29 (13.1%)</td>
</tr>
<tr>
<td></td>
<td>2nd stage models: 50k</td>
<td>0.76 (0.06x)</td>
<td>85 (3.11x)</td>
<td>39 (45.9%)</td>
<td>0.76 (0.06x)</td>
<td>162 (1.60x)</td>
<td>36 (22.2%)</td>
</tr>
<tr>
<td></td>
<td>2nd stage models: 100k</td>
<td>1.53 (0.12x)</td>
<td>82 (3.21x)</td>
<td>41 (50.2%)</td>
<td>1.53 (0.12x)</td>
<td>144 (1.81x)</td>
<td>39 (26.9%)</td>
</tr>
<tr>
<td></td>
<td>2nd stage models: 200k</td>
<td>3.05 (0.23x)</td>
<td>86 (3.08x)</td>
<td>50 (58.1%)</td>
<td>3.05 (0.24x)</td>
<td>126 (2.07x)</td>
<td>41 (32.5%)</td>
</tr>
</tbody>
</table>
A Criticism of The Case for Learned Indexes
A positive Criticism of The Case for Learned Indexes
CREATE index ON A

\[ f(\quad ) = \quad \]
A constructive Criticism of The Case for Learned Indexes
Coarse-granular and Sparse B-Trees

\[ k = 1 \]

\[ k^* = 1 \]
Coarse-granular and Sparse B-Trees

\[ k = 1 \]
\[ k^* = 1 \]
Coarse-granular and Sparse B-Trees

\[ k = 1 \]
\[ k^* = 1 \]
Coarse-granular and Sparse B-Trees

\[ k = 1 \]
\[ k^* = 1 \]
The diagram illustrates the concept of early stopping in the context of model training. The error (L) decreases as the number of iterations increases for both train and test datasets. However, overfitting occurs when the error on the test set starts to increase, indicating that the model is learning the noise in the training data. Early stopping is a technique to prevent overfitting by stopping the training process before the error on the training set starts to increase significantly. This is typically done by monitoring the error on a validation set and stopping training when the validation error stops decreasing.
Adaptive Indexing
Q1: ... where $A < 67$

Q2: ... where $A > 35$

Column A after Q1

Column A after Q2

crack-in-two kernel

A < 67

67 <= A

35 < A < 67

67 <= A

67 <= A
partition the dataspace

partition the data

partition the predicates

partition the predicate space

~same as trie
Adaptive Indexing

learns the search predicates
The difficulty of measuring stuff in main memory...
(still) some doubts on practicality of this approach, and diff to related work.

a nice fresh view on indexing, nice observations, food for thought
experience from teaching Data Science
# Materialien

## 1. Vorlesungen

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Titel</th>
<th>Größe</th>
</tr>
</thead>
<tbody>
<tr>
<td>01a</td>
<td>Einführung</td>
<td>1.9 MB</td>
</tr>
<tr>
<td>01b</td>
<td>Einführung</td>
<td>464 KB</td>
</tr>
<tr>
<td>02</td>
<td>Big Data Analytics</td>
<td>1.4 MB</td>
</tr>
<tr>
<td>03a</td>
<td>Relationales Modell und Algebra</td>
<td>1.9 MB</td>
</tr>
<tr>
<td>03b</td>
<td>Zusätzliche Beispiele für abgeleitete Operatoren</td>
<td>159 KB</td>
</tr>
<tr>
<td>04</td>
<td>Datenbanken</td>
<td>276 KB</td>
</tr>
<tr>
<td>05</td>
<td>Data Exploration</td>
<td>2.3 MB</td>
</tr>
<tr>
<td>06</td>
<td>Statistik</td>
<td>28 MB</td>
</tr>
<tr>
<td>07</td>
<td>MapReduce &amp; Apache Spark</td>
<td>1.1 MB</td>
</tr>
<tr>
<td>08</td>
<td>Lineare Regression</td>
<td>2.7 MB</td>
</tr>
<tr>
<td>09</td>
<td>Klassifizierung</td>
<td>4.9 MB</td>
</tr>
<tr>
<td>10</td>
<td>Entscheidungsbäume</td>
<td>672 KB</td>
</tr>
<tr>
<td>11</td>
<td>Neural Networks</td>
<td>5.2 MB</td>
</tr>
<tr>
<td>12</td>
<td>Clustering</td>
<td>4.5 MB</td>
</tr>
</tbody>
</table>

## 2. Notebooks

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Titel</th>
<th>Größe</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>Data Cleaning</td>
<td>137 KB</td>
</tr>
<tr>
<td>03</td>
<td>Relationale Algebra</td>
<td>12 KB</td>
</tr>
<tr>
<td>04</td>
<td>SQL</td>
<td>19 KB</td>
</tr>
<tr>
<td>07</td>
<td>Spark</td>
<td>13 KB</td>
</tr>
<tr>
<td>10</td>
<td>Entscheidungsbäume</td>
<td>6.0 KB</td>
</tr>
<tr>
<td>11</td>
<td>Neural Networks</td>
<td>2.1 MB</td>
</tr>
</tbody>
</table>
In [30]: # bedingte Einfärbung der Zellen des DataFrames:
   
   linksPivot.style.apply(lambda x: "background: orange" if v > 42 else "background: lightgreen" if v > 0 else "") for v

Out[30]:

```
<table>
<thead>
<tr>
<th>source</th>
<th>target</th>
<th>Action</th>
<th>Adventure</th>
<th>Animation</th>
<th>Biography</th>
<th>Comedy</th>
<th>Crime</th>
<th>Drama</th>
<th>Family</th>
<th>Fantasy</th>
<th>History</th>
<th>Horror</th>
<th>Music</th>
<th>Musical</th>
<th>Mystery</th>
<th>Romance</th>
<th>Sci-Fi</th>
<th>Sport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>2</td>
<td>155</td>
<td>10</td>
<td>11</td>
<td>45</td>
<td>54</td>
<td>72</td>
<td>7</td>
<td>40</td>
<td>4</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adventure</td>
<td>0</td>
<td>1</td>
<td>38</td>
<td>8</td>
<td>59</td>
<td>8</td>
<td>52</td>
<td>27</td>
<td>52</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>6</td>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Animation</td>
<td>0</td>
<td>0</td>
<td>34</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Biography</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>12</td>
<td>74</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Comedy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32</td>
<td>26</td>
<td>100</td>
<td>19</td>
<td>13</td>
<td>0</td>
<td>14</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>71</td>
<td>52</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Crime</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>97</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drama</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>48</td>
<td>13</td>
<td>32</td>
<td>28</td>
<td>35</td>
<td>13</td>
<td>4</td>
<td>52</td>
<td>98</td>
<td>28</td>
<td>15</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Family</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Fantasy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>History</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Horror</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Music</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>22</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Musical</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Mystery</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Romance</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Sci-Fi</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Thriller</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
```

In [31]: # die obige Visualisierung führt auf natürliche Weise zu einer Heatmap:
   
   plt.subplots(figsize=(15,15))
   sns.heatmap(linksPivot, cmap=sns.color_palette("Greens", 42), annot=True)

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1128dd3b38>
5 experiences from solving real data science problems with domain experts
WEATHER PREDICTION USING DEEP LEARNING
idea: the bug is the feature
2D nowcasting error
~
lightning prediction

**AdaBoost Confusion matrix**

- **True label**
  - no strike: 27589 (0.86) / 4558 (0.14)
  - strike: 4301 (0.13) / 27846 (0.87)

**Predicted label**

- no strike
- strike
Alle reden über Data Science. Aber was bedeutet das für mein Unternehmen?

BRINGEN SIE DATA SCIENCE IN IHR UNTERNEHMEN

Als Data Science Consulting helfen wir Ihnen Mehrwerte aus Ihren Daten zu schöpfen und dadurch mehr Effizienz, Transparenz und Struktur in Ihr Unternehmen zu bringen. Dadurch können Sie sich Wettbewerbsvorteile sichern. Produktionsketten effizienter gestalten, neue Geschäftsideen aus Ihren Daten entwickeln und sozusagen die Zukunft in den Griff bekommen.
Σ executive summary
1. remind people that we are one third of what people mean when they say “data science”

2. We should investigate Software 2.0. In particular, when f(X) is a function currently implemented by humans.

3. We need to support and push programs and lectures in data science

4. get out of your LAB and solve some real problems