TAKING AI/ML TO THE NEXT LEVEL - TEXT!

NICK BURCH

Berlin Buzzwords 2021

NICK BURCH

Director of Engineering

FLEC

@Gagravarr

CODE, SAMPLE DATA, SLIDES

github.com/Gagravarr/BBuzz21-ML_Text

All code in slides is taken from here!

OUR TALK TODAY

- Not your typical AI talk...
- The problem to solve
- AI and text - how?
- "Simple" AI for text
- Neural Networks for text
- Making it better
- Further resources

WHAT IS AI?

AND ML?

AND WHY NOW?

AI - ARTIFICIAL INTELLIGENCE

ML - MACHINE LEARNING

(Larry) Tesler's Theorem - "AI is whatever hasn't been done yet."

Which makes... ML what we can do today!

First big "AI Bubble" was mid-1980s, over $1 billion in AI startups that were generally touted as "expert systems"

Two problems - not enough training data available, computers much more expensive than the experts they were trying to replace

More expensive than the old experts?

Moore's Law to the rescue! One million dollars worth of 1985 computing power costs under one dollar today.

Amazon will rent you a machine with 1 TB of memory for roughly the cost of a latte per hour.

A 4 TB memory machine is about $25 / hour, less if reserved!

A 24 TB memory machine is available, pricing not disclosed, but it exists...

No data or software?

We have a lot more data available to train our ML models on.

You may not have Google's 3.5 billion searches per day, or Facebook's 65 billion messages today, but your company is certainly generating more than a few punchcards of data...

Open Source libraries and frameworks for AI / ML make it easy to get started, mean you can focus on your problem, not the system.

https://xkcd.com/1838/

YOUR TYPICAL AI / ML DEMO

YOUR TYPICAL AI / ML DEMO

IMAGE CLASSIFICATION!

YOUR TYPICAL AI / ML DEMO

IMAGE CLASSIFICATION!

It's fun, and it's easy!

DOGS OR CATS?

DOGS OR CATS? DOGS

DOGS OR CATS?

DOGS OR CATS? KOALA

WHAT ANIMAL?

WHAT ANIMAL? KANGAROO

WHAT ANIMAL?

WHAT ANIMAL? WOMBAT

WHAT ANIMAL?

WHAT ANIMAL? ECHIDNA

IMAGE CLASSIFICATION - COOL!

(Assuming your humans know what everything is, and can correctly give the right labels to your training data, and you're using the right kind of algorithm for your problem)

BUT MY DATA DOESN'T LOOK LIKE THAT....

	A	B	C	
1	**5**	**Quality Processes and Standards**		
2	**Please answer by YES or NO if requested and provide further information if necessary :**			
3			**YES/NO**	
4	**5.1**	**Has your organization ever been audited/inspected by any regulatory agency (e.g. FDA, EMA, etc)? If so, please provide dates of audit/inspection, scope of services audited, and key findings.**		
5				
6	**5.2**	**Has your organization ever had a critical finding or warning letter issued from a regulatory inspection? If so, please describe circumstances and findings of all audits/inspections, including corrective and preventative actions and current status.**		
7				
8	**5.3**	**Has your organization ever been audited by our company? If so, when and where? Please provide a summary of the audit scope, any critical or major findings, and status of findings**		

	A	B	C	D	E	F	G	H	I
1	**Include**	**Variable Name**	**Variable Label**	**Type**	**Format**	**Code List Name**	**VLM for Reviewer**	**VLM for Programmer**	**Origin**
2	Yes	STUDYID	Study Identifier	Char					
3	Yes	DOMAIN	Domain Abbreviation	Char		DOMAIN			
4	Yes	USUBJID	Unique Subject Identifier	Char					
5	Yes	TRSEQ	Sequence Number	Num					
6	No	TRGRPID	Group ID	Char					
7	No	TRREFID	Reference ID	Char					
8	No	TRSPID	Sponsor-Defined Identifier	Char					
9	Yes	TRLNKID	Link ID	Char					
10	No	TRLNKGRP	Link Group	Char					
11	Yes	TRTESTCD	Tumor Assessment Short Name	Char			TRTESTCD		
12	Yes	TRTEST	Tumor Assessment Test Name	Char			TRTEST		
13	Yes	TRORRES	Result or Finding in Original Units	Char					
14	Yes	TRORRESU	Original Units	Char		UNIT			

IF YOU WANT AN IMAGE CLASSIFICATION TALK

Large Scale Landuse Classification of Satellite Imagery

https://berlinbuzzwords.de/18/session/large-scale-landuse-classification-satellite-imagery

STILL HERE? LET'S TALK TEXT!

MY PROBLEM - I HAVE LOTS OF DOCUMENTS

From Big Companies, using lots of non-standard words and terms.

In Specialist Areas, using unusual terms and words in atypical ways.

Including Policies, Procedures, Training Guides, Help Information, Questionnaires and Proposals

(I have data too, but that's out-of-scope for today!)

I NEED TO DO INEXACT SEARCHING OVER THE CONTENTS OF ALL THESE DOCUMENTS

I NEED TO DO INEXACT SEARCHING OVER THE CONTENTS OF ALL THESE DOCUMENTS

Project training / help

Someone new to a project - won't know the right teminology, won't know all the abreviations, can't do an exact search on training material

Current workaround - just message busy team lead and ask them

I NEED TO DO INEXACT SEARCHING OVER THE CONTENTS OF ALL THESE DOCUMENTS

RFP questionnaires

Different clients use different wording when asking about common things, using different response templates.

Current workaround - lots of text gets re-written by busy people

I NEED TO DO INEXACT SEARCHING OVER THE CONTENTS OF ALL THESE DOCUMENTS

Job Titles in Job Postings

Different clients use different wording when describing common types of jobs, confusing people looking for them, and especially people with English as 2nd+ language.

Current workaround - have to give a "translation guide" to people posting jobs, asking them to call it different things

But I'm not allowed to show you most of those documents....

BERLIN BUZZWORDS TALKS TO THE RESCUE!

Let's use past Talk Titles and Abstracts as our test data

Partly clustering - what talks are similar to what other talks? What words are similar to other words?

Partly recommending - people using these search terms also found these talks relevant

It isn't - exact matching

It doesn't have - classification labels, certain answers

It can cope with - people who gave their talks fun / cool titles!

AI / ML FRAMEWORKS DON'T LIKE WORD DOCUMENTS / SPREADSHEETS / RAW HTML

Apache Tika lets us turn all of these documents into clean, semantically meaningful HTML

Then split the HTML into chunks (eg document sections, sheets etc), and finally generate something like JSON

For BBuzz talks, used BeautifulSoup to process talk pages into JSON

SAMPLE DATA


```
{
  "level": "Beginner",
  "track": "Scale",
  "abstract": "Whether we're in the position of a team lead.....",
  "title": "Building and Scaling a High Performing Development Team by Finding the Facts
  "url": "https://2019.berlinbuzzwords.de/19/session/building-and-scaling-high-performin
  "speaker": "Will Hayes"
},
{
  "level": "Intermediate",
  "track": "Search",
  "abstract": "Search is fundamental feature of mobile.de platform and we as Data Team w
  "title": "Architecture of relevancy search at mobile.de",
  "url": "https://2019.berlinbuzzwords.de/19/session/architecture-relevancy-search-mobil
  "speaker": "Richard Knox"
```

WE HAVE OUR TALKS AS JSON

So, we're set right? Just feed the JSON into the AI, and magic happens?

Sadly not... ML frameworks don't generally work on Text-in-JSON

ML NEEDS A MATRIX OF NUMERIC VALUES

Ideally mostly -1.0 to 1.0 or 0.0 to 1.0

One value for each *feature* of the thing to learn / predict on

Can be sparse (only a few non-zero values) or dense (mostly non-zero)

More features requires more memory and more CPU, so a tradeoff!

How can we turn our text into something like that?

AN ASIDE - SOME TERMINOLOGY

Feature - An input variable, a value for one aspect of the thing to predict. eg height / weight / 1st RGB channel

Label - The value to be predicted / trained for, eg Dog / Cat / price of house the features describe

Training - Creating / learning a model to map from the features to the label

Inference - Applying the model to something new to get a prediction

https://developers.google.com/machine-learning/crash-course/framing/ml-terminology

AN ASIDE - SOME TERMINOLOGY

Regression - A model to predict continuous values. eg "Given the location and number of bedrooms, what's the likely price of a house"

Classification - A model to predict discrete values. eg "Is this an image of a Dog, a Cat or a Wombat?"

Clustering - A model to group similar things together. If those are labelled, it's classification. If those aren't labelled, it's clustering, and relies on unsupervised machine learning

https://developers.google.com/machine-learning/clustering/overview

SIMPLE VECTOR SPACE FOR TEXT

First up, we need to break our text down into smaller units. We call this *tokenisation*.

The simplest way to do that is to split on whitespace and punctuation, so we get one token per word.

(We'll cover more advanced things, eg stopwords and stemming, a bit later on)

See any Lucene introduction talk for more!

Lucene in Action or Taming Text have good stuff on this in early chapters too.

SIMPLE VECTOR SPACE FOR TEXT

Next, build up a *term dictionary* for all the different tokens in our text, assigning each a unique index

```
The mouse ran up the clock

The mouse ran down

{ 'the':1, 'mouse':2, 'ran':3, 'up':4, 'clock':5,
                                'down':6 }

The mouse ran up the clock = [ 1, 2, 3, 4, 5 ]

The mouse ran down = [ 1, 2, 3, 6 ]
```

SIMPLE VECTOR SPACE FOR TEXT

When one word occurs multiple times, what then? Two easy options

One-Hot encoding - `1` if present, otherwise `0`

CBOW Count - how many times across the *continuous bag of words* each term occurs. (This looses position information though)

`The mouse ran up the clock = [ 2, 1, 1, 1, 1, 0 ]`

`The mouse ran down = [ 1, 1, 1, 0, 0, 1 ]`

TF-IDF

If a document contains a given term a lot, we want to rate that document higher for that term.

If most documents have a term, it's probably a less interesting thing to look for, and not that specific. If only a few do, that term is probably more interesting.

If a document is very long, its counts will naturally be higher than a very short document's counts, as it'll have more text.

Weigh rare terms higher, common terms lower, across all documents. Within a document, rate the term higher the more it is used. Weight shorter documents higher than longer ones.

TF-IDF = TERM FREQUENCY – INVERSE DOCUMENT FREQUENCY

TF-IDF is the simplest common way to do this, implemented in most ML libraries you'll come across.

For each talk, tokenise then do CBOW count, then apply TF-IDF. (Generally 1-2 lines of python code!). Gives us, for each talk, a value between `0` and `1` for each term. A big ML friendly matrix!

Other relevancy techniques exist, for more details see https://2016.berlinbuzzwords.de/session/bm25-demystified.html and https://2017.berlinbuzzwords.de/17/session/bm25-so-yesterday-modern-techniques-better-search-relevance.html

TOP TF-IDF TERMS FROM OUR TALKS

data search apache talk scale use learning time stream using ll processing real like applications new open machine solr systems source flink streaming open source based spark machine learning model kafka elasticsearch

OUR NEXT CHALLENGE - HOW TO TRAIN THE AI?

We don't know what the right answer is for which talk(s) go with which query. We don't have the training data, the labels or the scoring function.

For many techniques, we would need to gather that data first! Perhaps by watching what users did, or sitting down and manually classifying loads of things.

However, if we just did things on text similarity, would that get us close enough for now?

FIRST APPROACH - PREDICT ONE TALK, SUGGEST SIMILAR

1. Build a classification model based on talks
2. Train model on text from talk (title, abstract), using TF-IDF to make feature vector for each talk
3. Ask model to classify our query
4. Result is talk "most like" our query
5. Return other "similar" talks too, based on text similarity

IN CODE - BUILD THE MODEL

```
# How to do the TF-IDF conversion
tf_settings = dict( analyzer="word", ngram_range=(1,2),
     sublinear_tf=True, min_df=0, stop_words='english' )

# Build the TF-IDF over all the talks
# Use title + category + abstract for our text
tfidf = TfidfVectorizer(**tfs)
tfidf_matrix = tfidf.fit_transform(talks["learn"])
print(tfidf_matrix.shape)    # eg 294 x 30076

# Build the similarities of each talk against every other talk
# We'll use this for scoring
tfidf_similarities = linear_kernel(tfidf_matrix, tfidf_matrix)

# Build a model, using Multinomial Naive Bayes
```

IN CODE - USE THE MODEL

```
query = "apache tika"

# Ask the model to compare our query against every talk,
#  then pick the talk it thinks is the most similar
pred_idx = model.predict([query])

# The prediction should be the index of that talk
print("Best match - talk %d" % pred_idx)

# Get the pairwise similarity scores of all other talks with that one
# Filter for ones high enough, and sort so highest scores come first
similarities = tfidf_similarities.[pred_idx]
sim_scores = list( [idx,s] for idx,s in enumerate(similarities[0]) if s > 0.01 )
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
```

HOW WELL DOES IT WORK?

Moderately well?

(But we don't have a set of known-good examples, so we can't calculate an exact score)

LET'S TRY A LIVE DEMO!

`BuildAndPredict-SciKitLearn.py` lines 160-200

NEXT APPROACH - CLUSTER TALKS FIRST

Clustering - Grouping similar, but unlabelled things together

Rather than matching then finding similar things, as we did with the Naive Bayes approach, what if we clustered first?

K-Means, originally from signal processing, will let us group *n* things into *k* groupings, based on minimising the sum of the squared errors.

But how many clusters should we make?

CLUSTER SIZING

If we have *k* terms in our TF-IDF, then *k* clusters with fit with zero error. *1* cluster will be maximum error. Neither helps much...

We need something in-between, where most things are grouped together, and not too many things are a long way from their neighbours.

Various measures can be used to test the effectiveness of the clustering, *Average Silhouette* and *Gap Statistic* methods seem quite popular.

Need to run a few times at each k value (with different init seeds to avoid local-minima), then compute effectiveness measure, then pick best.

IN CODE - BUILD DIFFERENT CLUSTERS

```
# Try a range of k-sizes
krange = range(25, talks.shape[0]-25)

# Try each one with a few different inits, to avoid getting stuck in local-minima
kms = []
for k in krange:
    tfidf = tfidf_word

    print("Building k-means cluster of size %d" % k)
    km = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=15, verbose=False)
    km.fit(tfidf.matrix)

    scoff = metrics.silhouette_score(tfidf.matrix, km.labels_, sample_size=1000)
    print(" - Silhouette Coefficient: %0.4f" % scoff)
```

CAN WE SEE HOW IT WORKED?

Our input data had 20-30k dimensions, our cluster has ~50

The human brain struggles with much more than about 4...

A 3D + colour plot is about the limit for most people

Techniques like t-SNE and PCA let us throw away a load of the dimensionality, whilst still keeping some (but not all) of the info

NEXT APPROACH - CLUSTER TALKS FIRST

First, identify our "optimal" cluster size

Next, build a k-means clustering of our talks

Match the text query to a cluster, and find the cluster centre

Now match talks based on similarity to that cluster centre (rather than one specific talk), possibly boosting slightly talks from our cluster

IN CODE - USING CLUSTERS

```
# Identify which cluster each talk belongs to
talk_clusters = km.predict(tfidf.matrix)
cluster_talks = defaultdict(list)
for talk_id, cluster_id in enumerate(talk_clusters):
    cluster_talks[cluster_id].append(talk_id)

# Find best cluster for our query, cluster centre and talks
query_tf = tfidf.transform([query])
cluster_id = int(km.predict(query_tf))

c_centre_tfidf = km.cluster_centers_[cluster_id]
c_talk_ids = cluster_talks[cluster_id]

# Compare all talks with this cluster centre, then order
similarities = linear_kernel(c_centre_tfidf, tfidf.matrix)
```

HOW DOES K-MEANS CLUSTERING DO?

Fairly similar? Maybe a tiny bit worse?

(Still don't have a set of known-good examples, so we still can't calculate an exact score!)

Clusters look to have sensible terms in them, results seem mostly what we'd expect

LET'S TRY ANOTHER LIVE DEMO!

`BuildAndPredict-SciKitLearn.py` lines 300-337

ADDING TALK YEAR TO SCORING?

Can we prefer newer talks to older ones?

Can't easily add it as a feature, since we don't have a year at query time, and we can't teach the model directly what to prefer

We can add it at scoring time, by multiplying scores by year factor. However, that only applies once we have matched to a talk / cluster.

Ideally need to feed in before we build the model, not after.

If we reduce TF-IDF weights slightly for older talks, that will de-boost them before model is built, but may affect how we cluster.

OUR DATA ISN'T LONG-TERM STATIC

Next year, there'll be another Berlin Buzzwords! That means more talks

(Likewise we're also adding new training data to our onboarding chatbot, answering more RFPs, building more complex SDTM mapping rules etc)

Let's say we took the time to manually identify the best talks for some query terms, either explicitly, or via user feedback in a webapp

We boost / prioritise / train based on these "known correct" answers, and it all looks good!

Then we add more talks, quite possibly even more relevant ones for our queries, but we our ML won't prioritise them as not on the "good list"

TOKENISATION REVISISTED

We need to turn our text into a stream of chunks to feed into the TF-IDF or Embeddings

Stopwords - Very common words that don't help much with the query, and can bloat the TF-IDF.

Stemming - Bringing different forms of a word to a common root, eg `talk talks talked talking` so they can be matched interchangably

n-grams - word-based means treating several words as one token, eg word bigrams means word pairs. char-based means splitting word into overlapping chunks, eg trigrams of `hello` are `hel ell llo`

Need to use the same for learning *and* for querying!

https://developers.google.com/machine-learning/guides/text-classification/step-3

EMBEDDINGS AND FEATURE EXTRACTION

Even from just a few hundred talks, our TF-IDF had a lot different terms in it. However, most talks only use a subset of those terms, so lots of TF-IDF matrix values are `0`. Our TF-IDF is *sparse*

Bayse and K-Means, amongst others, are fine with sparse matricies. Other techniques, eg Neural Networks, need *dense* matricies, where most values are non-zero

Using stop-words and stemming helps a little bit here, but we need a few more orders of magnitude change!

https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction https://developers.google.com/machine-learning/crash-course/embeddings/obtaining-embeddings

EMBEDDINGS AND FEATURE EXTRACTION

TF-IDF has no semantic information

Word order matters - information dependsing on the context

The place of the word in the sentence makes a big difference!

My kindle is easy to use, I do not need help

I do need help, my kindle is not easy to use

EMBEDDINGS AND FEATURE EXTRACTION

We need to extract some new features from our existing data, which have much lower dimensionality.

Instead of 20k-30k terms, we want more like 50-250

Our per-talk matrix of features will then be smaller and denser (fewer features/dimensions, more non-zero values)

This new dense set of features is known as an *embedding*

As well as being needed before we can use Neural Networks on our kind of input, they also help with visualisations of you data

Our k-means clusters are a form of term-embedding, but not a great one!

SOME EMBEDDING APPROACHES

- Word Embeddings - word2vec, GloVe
- Contextualized word embeddings - ELMo
- Bidirectional Encoder Representation from Transformers - BERT
- Generative Pre-Training 2 - GPT-2

WORD2VEC

Word2Vec was originally developed by Google. It's a series of models that produce *word embeddings*.

Based on shallow, 2-layer neural networks, trained to reconstruct linguistic contexts of words

Produces a vector space, typically reduced down to a few hundred dimensions, where words with similar contexts are located nearby.

With enough data, model can be used to make guesses on a word's meaning and association, e.g. “man” is to “boy” what “woman” is to “girl”

https://developers.google.com/machine-learning/guides/text-classification/step-3

We will return to Word Embeddings in a little bit...

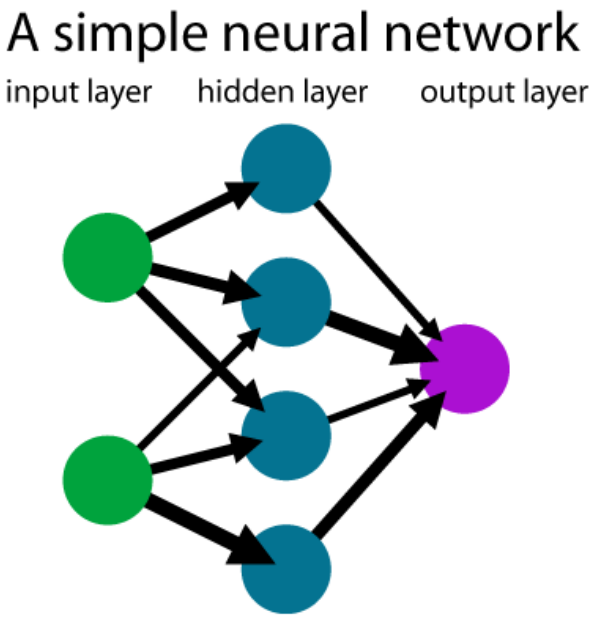
NEURAL NETWORKS

System built up with a series of layers, with data flowing one way

Input layer accepts your features / embeddings

Output layer constrained to give answer you expect, get "cat or dog"

Potentially many hidden layers between, which do the processing



Training done to build the nodes, the weights of the links between them, if you should add or remove nodes in a layer etc

TESTING, CROSS VALIDATION

When we know what our model should be predicting for a set of input data (eg what the human-provided labels are for a bunch of input features), we need to measure how well the model does!

Train on one set of known-data, then predict against another set, and see how close the predictions came to what we know are the answers

Our model might do *too well*, if it ends up learning some specific bits of our training data, this is *over-fitting*

Generally we want a train / test split, and maybe validate too, which should all be representative of the data we'll predict with

If we don't have much data available, we can train the model several times with different segments as train/test, and ensure always similar accuracy

HYPER-PARAMETERS AND TUNING

A hyper-parameter is anything we tune / select / change in our ML, that's independant of the data and of the features selected

eg k-means, that includes the range of clusters to try fitting to

Often relate to seeds, steps, number of clusters / layers, how much to change between iterations, and how much to leave alone

Pick the wrong parameters, and your model might get stuck in a local minima a long way off the best answer, or might take ages to converge, or may never even complete!

Along with identifying appropriate features, and cleaning your data, selecting appropriate hyper-parameters is a tough bit of data science

ERRORS

For a binary classification, there are 4 possible states: True Positive, True Negative, False Positive and False Negative

What to aim for depends on your problem, and the distribution of your data

eg if detecting cancer, is it better to give someone the all-clear when they actually have cancer (false negative)? Or to send them for treatment that they don't need (false positive)

Precision is ratio of correct values in our results, **recall** is ratio correct-found to correct-all. **Confidence** is how well the model thinks it did.

BIASES

If you input data is biased, the model will be biased

If you try to hide some biases from your model, it might still find them from other features.

eg hide Gender, but leave in Name. eg hide Race, but leave in postal code / zipcode, or first / elementary school

Be aware of your data biases, be aware of how people will use your model, try to re-weight your models to counteract biases

Be aware of implicit biases in your data, and impact if model is fed data from somewhere else / something else

APACHE MXNET

" *A Flexible and Efficient Library for Deep Learning* "

8 language bindings - Deep integration into Python and support for Scala, Julia, Clojure, Java, C++, R and Perl.

Tools and libraries - enable use-cases in computer vision, NLP, time series etc

GluonNLP - libraries and tools for NLP and Text

APACHE MXNET AND TEXT

- `mxnet.contrib.text`
- https://mxnet.apache.org/api/python/docs/api/contrib/text/index.html
- GluonNLP - https://gluon-nlp.mxnet.io/
- NLP Model Zoo - https://gluon-nlp.mxnet.io/model_zoo/index.html
- Model Zoos for Word Embedding, Text Classification etc
- Plus all the primatives you need to build your own!

See also https://d2l.ai/chapter_natural-language-processing/index.html

MXNET AND LOADING A WORD EMBEDDING

```
from mxnet import nd
from mxnet.contrib.text import embedding

# Fetch if needed, then load GloVe embeddings
glove = embedding.GloVe(pretrained_file_name='glove.6B.50d.txt')
print("GloVe loaded, contains %d terms" % len(glove))

# For finding cosine-similar embeddings
def find_nearest(vectors, wanted, num):
    cos = nd.dot(vectors, wanted.reshape((-1,))) / (
            (nd.sum(vectors * vectors, axis=1) + 1e-9).sqrt() * nd.sum(wanted * wanted).
    top_n = nd.topk(cos, k=num, ret_typ='indices').asnumpy().astype('int32')
    return top_n, [cos[i].asscalar() for i in top_n]

# Looking up some similar words
```

MXNET TRYING OUT WORD EMBEDDING

```
Similar tokens to: search
 - Cosine sim=0.890: searching
 - Cosine sim=0.805: searches
 - Cosine sim=0.786: information
Similar tokens to: linux
 - Cosine sim=0.849: unix
 - Cosine sim=0.793: open-source
 - Cosine sim=0.778: kernel
Similar tokens to: gpl
 - Cosine sim=0.865: lgpl
 - Cosine sim=0.860: gnu
 - Cosine sim=0.761: gplv2

The analogy for berlin = germany of paris is france
```

HAS MY MODEL / PIPELINE IMPROVED?

- How will you tell?
- How can you measure it?
- Do you have known-good tests to run?
- Will your users give direct / indirect feedback?
- Don't test with training data only!

APACHE TIKA EVAL

- Apache Tika regression corpus, 2+ TB
- When I change one library, does the overall system get better or worse?
- Can't manually check millions of files...
- Identify key changes, and some examples
- Track how error counts alter with time
- Provide examples for manual checkin

HOW DO I RECREATE MY MODEL?

- Don't just tinker with it until you're happy...
- Make sure you have the training data saved somewhere
- Record how you pre-process your training data
- Record what parameters you used along with the model
- Take care with random seeds
- Think about "Data Engineering", how you'll "productionize" it, how to build a data pipeline

DOES LUCENE / ELASTIC HAVE DO BETTER?

Out of the box, and with our data sizes - mostly no

If we'd spent some time configuring the similarity and clustering features - probably!

If we had a lot more data, Lucene / SOLR / Elastic would win - most of the AI techniques need the whole model in memory (Lucene can efficiently pages bits in from disk), and Lucene has lower CPU needs at index/predict

Initially we indexed to both, and gave people the option. Most use was on ML solution, but sometimes Elastic was needed to find the best results.

But as an AI / ML learning exercise, a chance to put that into action on a few production projects, and to aid planning of our next data-driven ML projects, it was invaluable!

TRYING OUT AI / ML YOURSELF

Basically everything is open source! Installation and setup isn't always the easiest though...

Jupyter - an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. (There's also *Apache Zeppelin*!)

There are several *free* hosted Jupyter instances, which come "batteries included" with all the libraries you need to get started!

Azure Machine Learning - Notebooks - https://ml.azure.com/

Google Colaboratory - https://colab.research.google.com/

GitHub Codespaces beta - https://github.com/features/codespaces

ABOUT THE CODE

github.com/Gagravarr/BBuzz21-ML_Text

BuildAndPredict-SciKitLearn.py - Guides you through all the beginners text for ML, using Text and a TF-IDF matrix. Start here if you're new to ML!

BuildAndPredict-mxnet.py - Advanced text reasoning with Apache MXNet and GloVe. Shows off Word Embeddings for Word Similarity and Analogy. Next steps - ELMo and BERT!

FURTHER LEARNING RESOURCES - COURSES

Stanford University

- https://www.coursera.org/learn/machine-learning

Google

- https://developers.google.com/machine-learning/crash-course/
- https://developers.google.com/machine-learning/guides/rules-of-ml/
- https://developers.google.com/machine-learning/guides/text-classification/

Amazon

- https://aws.amazon.com/training/learning-paths/machine-learning/

FURTHER LEARNING RESOURCES - COURSES

Microsoft

- https://docs.microsoft.com/en-us/azure/machine-learning/

Jeremy Howard / Fast.AI

- http://course18.fast.ai/ml
- http://course18.fast.ai/part2.html

O'Reilly Safari has loads!

Others

- https://scikit-learn.org/stable/tutorial/basic/tutorial.html
- https://scipy-lectures.org/

FURTHER LEARNING RESOURCES - OTHERS

- http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
- https://www.ben-evans.com/benedictevans/2019/4/15/notes-on-ai-bias
- https://www.graphcore.ai/posts/removing-bias-from-machine-learning
- https://arxiv.org/abs/1704.00051
- https://github.com/facebookresearch/DrQA

FURTHER LEARNING RESOURCES - BOOKS

Taming Text - https://www.manning.com/books/taming-text

Introduction to Information Retrieval - https://nlp.stanford.edu/IR-book/

Loads available from Manning

And from O'Reilly

ANY QUESTIONS?