Experiments in data mining, entity disambiguation and how to think data-structures for designing beautiful algorithms

Ekta Grover

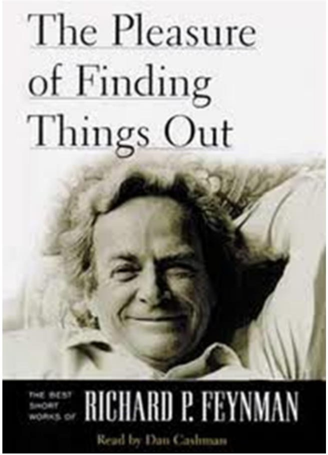
1st September, 2013

Structure of this Talk

- How to think through a data mining problem
- 3 problems in data-mining around disambiguation, Natural language processing & Information Retrieval
- Measuring your Model performance - developing a near-real time performance aggregation platform
- Putting it all together

The only algorithm you will ever need to know

1. Write down the problem
2. Think real hard
3. Write down the solution



Thinking through Data-Mining problem

- What is the data and how should you represent it ?
- What are the relationships you are interested in ?
- How to define your Objective Function ?
- How will you know that the score represents the direction you want to capture ?
- How will you measure the performance after you are done ?

Problem # 1 : Entity Disambiguation

Problem

So, How much is your network really worth ?

What is Disambiguation ?

Information = Related + Unrelated
Extract significant attributes/features of an entity
Relationships between entities & features
Objective function is a just a way to measure - rank/score the features for an entity wrt to other entities

Disambiguation is not a deterministic problem

Problem # 1 : Entity Disambiguation

Guiding question #1

What form of disambiguation to use for the entities ?

Algorithm & approach

1. Preprocessing - lower casing & removing punctuation
2. Exploratory Analysis - frequency plots (with NLTK freqdist & Pretty table)
3. Tokenization to find "keywords" to disambiguate
4. Handling stop words & Internationalization
5. Frequent item-set mining with support = #of buckets
6. Post processing, frequency plots & visualization

Extensions: Handling Typo graphical errors and augmenting this data-set with scraping

Problem # 1 : Talking Specifics & answering the "**Why**"

Frequent item-set mining & concept of support

How do we know the number of significant words in an entity name?

1. For every company name - strip the 1^{st} word and store it as the key of "local" dictionary, with values as list
2. On the local dict created in step 1, do frequent item-set mining with support = # baskets - if a word is significant(and uniquely describes an entity), it should be present in all baskets
3. Treat all the values in a local dict as a set - If all elements of a local dict are same - no need for item-set mining
4. For every key - find the smallest list of tokens(by length) and search the presence of all tokens iteratively till the support appears in less than #baskets
5. Replace the words fetched post support to equal all values in that dictionary
6. Fetch all values from all keys in the dict of list - flatten this list
7. Regular frequency counts on this flattened list of now disambiguated elements

Problem # 1 : Limitations of this approach

1. **Abbreviations** : Boston Consulting Group & BCG, State Bank of India & SBI - build a referencing domain specific Word Sense Disambiguation (WSD) database
2. **Differently referenced entities** such as "Innovation labs 247 customer private ltd" and "247 innovation labs" : The referencing key will be different for these
3. **Sub-entities** of a company with punctuation "Ebay/Paypal" - the "key" depends on how we choose to treat the punctuation for {/,:,-}
4. Need a module to handle **inadvertent typographical errors** in entity names, names are proper nouns, yet no referencing from Wordnet database - plug in Problem #2

Extensions

Cross document and Co-reference resolution,Multi-pass learning, smoothing & annealed learning algorithms

Problem # 1 : Representation

List of orignal values in the csv file (37 in total)

```
['SAP Labs India', 'SAP Labs India', 'SAP', 'SAP Labs India Pvt. Ltd', 'SAP Labs Bulgaria', 'SAP Labs India Pvt Ltd', 'SAP', 'SAP Singapore'
Inc', 'SAP SD', 'SAP AG', 'SAP Labs India Pvt. Ltd.', 'SAP Labs India Pvt. Ltd.', 'SAP Labs India Pvt. Ltd.', 'SAP AG', 'SAP AG', 'SAP Labs
'SAP Labs', 'SAP Labs', 'SAP LABS INDIA PVT LTD', 'SAP Labs', 'SAP', 'SAP Germany', 'SAP', 'SAP', 'SAP AG', 'SAP', 'SAP', 'SAP', 'SAP Labs',
', 'SAP', 'SAP Labs']
```

unique values in the orignal csv file

```
set(['SAP Labs Bulgaria', 'SAP Labs India', 'SAP Labs', 'SAP America Inc', 'SAP Labs India Pvt. Ltd', 'SAP AG', 'SAP Labs India Pvt. Ltd.',
AP Labs India Pvt Ltd', 'SAP', 'SAP LABS INDIA PVT LTD', 'SAP Singapore'])
```

Post removing punctuation and lowercasing

```
['sap labs india', 'sap labs india', 'sap', 'sap labs india pvt ltd', 'sap labs bulgaria', 'sap labs india pvt ltd', 'sap', 'sap singapore',
nc', 'sap sd', 'sap ag', 'sap labs india pvt ltd', 'sap labs india pvt ltd', 'sap labs india pvt ltd', 'sap ag', 'sap ag', 'sap labs india pv
s', 'sap labs', 'sap labs india pvt ltd', 'sap labs', 'sap', 'sap germany', 'sap', 'sap', 'sap ag', 'sap', 'sap', 'sap', 'sap labs', 'sap lab
, 'sap labs']
```

Post tokenization and keyin-in to define dicts with "key" as the 1st word in company name and value as the list of values with the same 1st word or "key"

```
{'sap': ['sap labs india', 'sap labs india', 'sap', 'sap labs india pvt ltd', 'sap labs bulgaria', 'sap labs india pvt ltd', 'sap', 'sap sing
merica inc', 'sap sd', 'sap ag', 'sap labs india pvt ltd', 'sap labs india pvt ltd', 'sap labs india pvt ltd', 'sap ag', 'sap ag', 'sap labs
'sap labs', 'sap labs', 'sap labs india pvt ltd', 'sap labs', 'sap', 'sap germany', 'sap', 'sap', 'sap ag', 'sap', 'sap', 'sap', 'sap labs',
', 'sap', 'sap labs']}
```

Post frequent item set mining . In this simple case with singular key, there is no need to flatten the list - and we can directly take FreqDist/counts on this entity

```
['sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap',
, 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap', 'sap']
```

Problem # 1 : Representation, a more extreme example & Codewalk

List of orignal values in the csv file (37 in total)

```
['24/7 Customer', '24/7 Customer', '[24]7 Inc. (iLabs)', '[24]7 Inc', '[24]7 Innovation Labs', '24/7 Customer', '24/7, Inc', '[24]7 Innovation Labs', '[24]7-inc', '24/
7, Inc', '[24]7,Innovation Labs', '24/7, Inc', '24/7, Inc', '24/7, Inc', '24/7, Inc', '[24]7, Inc', '24/7, Inc', '[24]7 Inc', '24/7, Inc', '24/7, Inc', '[24]7 Inc - In
novation Labs', '[24]7 Inc', 'Innovation Labs, 24/7 Customer Pvt Ltd', '24/7 Customer', '24/7, Inc', '24/7, Inc', '24/7, Inc', '24/7, Inc', '24/7, Inc', '24/7, Inc', '
24/7 Customer', '[24] 7  innovations labs', 'Innovation Labs @ [24]7, Inc', '24/7, Inc', '24/7, Inc', '24/7, Inc', '24/7, Inc']
```

Unique values in the orignal csv file

```
set(['[24]7 Inc - Innovation Labs', 'Innovation Labs, 24/7 Customer Pvt Ltd', 'Innovation Labs @ [24]7, Inc', '[24]7,Innovation Labs', '[24] 7  innovations labs', '[24
]7-inc', '24/7, Inc', '[24]7, Inc', '[24]7 Innovation Labs', '[24]7 Inc', '24/7 Customer', '[24]7 Inc. (iLabs)'])
```

Post removing punctuation & lowercasing

```
['247 customer', '247 customer', '247 inc ilabs', '247 inc', '247 innovation labs', '247 customer', '247 inc', '247 innovation labs', '247inc', '247 inc', '247innovati
on labs', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc  innovation labs', '247 inc', 'innovation labs 24
7 customer pvt ltd', '247 customer', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 customer', '24 7  innovations labs', 'innovation labs  247
inc', '247 inc', '247 inc', '247 inc', '247 inc']
```

We have 5 dicts for this entity, which is a more extreme case need a multi-pass here/ pre-processing prior to frequent item-set mining

Post tokenization & keying-in to define dicts with "key" as the 1st word in the company name and value as the list of values with the same 1st word(or key)

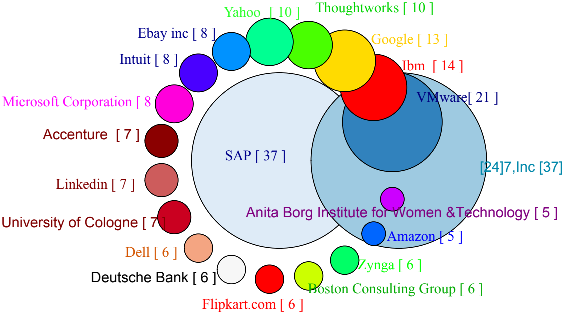
```
{'24': ['24 7  innovations labs'], '247': ['247 customer', '247 customer', '247 inc ilabs', '247 inc', '247 innovation labs', '247 customer', '247 inc', '247 innovatio
n labs', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc  innovation labs', '247 inc', '247 cust
omer', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 inc', '247 customer', '247 inc', '247 inc', '247 inc', '247 inc'], 'innovation': ['innovation labs 2
47 customer pvt ltd', 'innovation labs  247 inc'], '247inc': ['247inc'], '247innovation': ['247innovation labs']}
```

Post frequent item-set mining without the multipass/pre-processing

```
[['24 7  innovations labs'], ['247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247
', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247', '247'], ['innovation labs  247', 'innovation labs  247'], ['247inc'], ['247innovation l
abs']]
```

Codewalk from Github

https://github.com/ekta1007/Experiments-in-Data-mining



Top companies by Frequency distribution, post item-set mining

*I used R(ggplot2 and igraph) to plot the nodes with ColorBrewer palette and Inkspace for "cosmetics"

Problem # 2 : Custom Distance metric for handling Typographical errors optimized for a QWERTY keyboard

Usecase

Disambiguating the typographical errors with a contrasting (noun)item

Three types of misspellings[1]

1. **Typographic errors** - know the correct spelling, but motor coordination slip when typing
2. **Cognitive errors** - caused by lack of knowledge of the person
3. **Phonetic errors**[2] - sound correct, but are orthographically incorrect

Distance measures: Levenshtein, Manhattan, Euclidean, Cosine - inappropriate for distance in our context. Need a **hybrid contextual** approach

[1] Karen Kukich - Techniques for automatically correcting words in text, ACM Computing Surveys Volume 24 Issue 4, Dec. 1992 ; Creating a Spellchecker with Solr `http://emmaespina.wordpress.com/2011/01/31/20/`

[2] Soundex algorithm; example : Chebyshev & Tchebycheff

Problem # 2 : QWERTY distance metric

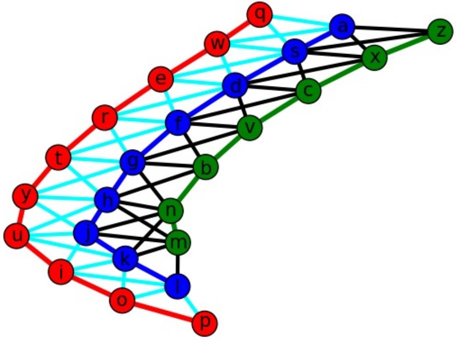
Guiding Assumption

Since typographical errors are inadvertently introduced, they should not count towards the overall distance

Algorithm & approach?

1. **Data structures** - QWERTY keyboard as a graph with connected adjacent nodes, neighborhood nodes with appropriate distances
2. **Define a typographical error & it's thresholds**- what can be the difference in length of words, how many possible "swaps" can occur in a typo
3. **Implement** the custom distance metric - fall back to Levenshtein when non-typographical - Hybrid approach
4. **Output** whether the words are identical, or the distance taking typographical errors into account
5. **Contrast and blend in** with other fuzzy logic methods, Apache Solr, Spell checker & Auto-suggest

Problem # 2 : Representation & Codewalk



QWERTY Keyboard representation

Codewalk from Github

https://github.com/ekta1007/Custom-Distance-function-for-typos-in-hand-generated-datasets-with-QWERY-Keyboard/

Problem # 3 : Designing your custom job feeds at Linkedin

Part 1

Create filtering criteria to **build your custom data-set**
Linkedin REST API's, scraping(Scrapy, Beautiful soup), raw html with urllib & NLTK

Part 2

Build **relevance score** of the filtered results
TF-IDF(with cosine similarity), Structured Relevance Models (SRM), Proximity search, classical Information Retrieval & pattern matching algorithms

Problem # 3 : Part 1

Algorithm & approach

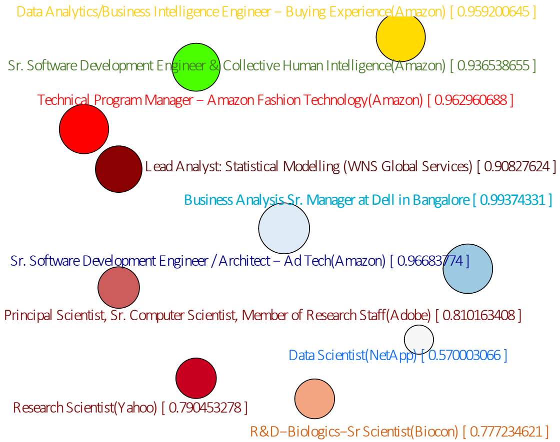
1. **Exploit the commonality** in url pattern of the Jobs page, looping constructs for **job_index** `http://www.linkedin.com/jobs?viewJob=&jobId=job_index&trk=rj_jshp`
2. Post this **only 4 options** exist per Job page
    1. The job you're looking for is no longer active
    2. We can't find the job you're looking for
    3. The job passes your filtering criteria (location, skill-specific keywords etc)
    4. The job is active, but does not pass your filtering criteria
3. For urls in 2.3, **build a simple csv file** with all elements of interest - Company Name, Title of Position, Job Posting Url, Posted on, Functions, Industry, similar Jobs
4. Can **pre-filter** by company name - by heuristics, or a quick scoring function for each company name

Problem # 3 : Part 2

Algorithm & Approach

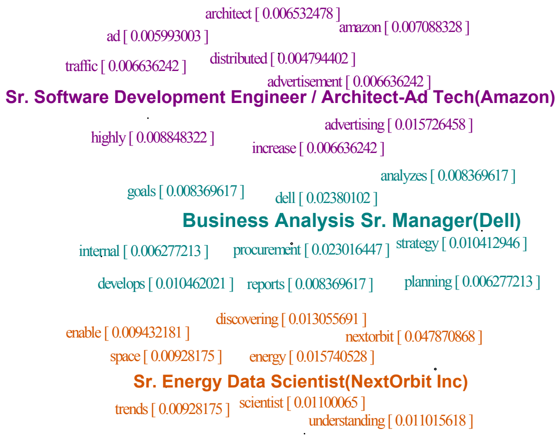
1. A document is a **bag-of-words** - Tokenize, standard preprocessing(punctuation, lower case)
2. **Exploratory analysis** with NLTK's Dispersion plot, FreqDist, collocations, common_contexts, count, concordance
3. **Lemmatization & stemming** with NLTK's derivationally related forms with Morphy(Wordnet), Porter, Wordnet_app
4. **TF-IDF** - with stop word list from NLTK & additional weighage for words of your specific interest
5. **Inverted index** - for each word in vocabulary, store the doc-id with it's TF-IDF, making up the whole TF-IDF vector
6. Given the TF-IDF vector of the document corpus & query, compute the **cosine similarity** of each document with the base query
7. **Output top-k** relevant Job listings, with the top-k' words in each job feed
8. **Augment** this with a classical structured relevance model, Proximity search or pattern matching algorithm

Problem # 3 : Representation & Visualization



Relevant Job feeds ranked by Cosine Similarity & TF-IDF to the base query

Problem # 3 : Representation & Codewalk



Top ranking words in the respective job-feeds[3]

▸ Codewalk from Github

https://github.com/ekta1007/Creating-Custom-job-feeds-for-Linkedin

[3] Note that some of the words are relatively uninformative like "understanding", "Amazon" - could use any amongst Proximity search, Phase search, Positional index, Next word index

Problem # 4 : Designing a near-real time Performance Aggregation Platform

Concept

Once you roll your competing candidate models into production, they should self-decide based on performance

Competing Goals can be amongst

1. **Minimizing entropy** of the complete system(more centered distribution)
2. **Optimizing precision & recall**[4] - tradeoff between type I & type II
3. **Accounting for trends** - 1st order(velocity) and 2nd (acceleration)
4. **Minimize bias** - Low mean square error, moving towards a more global representation of a problem

[4] precision - fraction of retrieved instances that are relevant, recall - fraction of relevant instances that are retrieved

Problem # 4 : Defining the aggregated proxy for Performance

Algorithm & approach

1. **Apriori:** Initialize the candidate population, all else equal
2. **Define model** equations, which generate the scores, per model
3. **Define the metrics** for weighted performance and what is considered a "success event"
4. **Define threshold** conditions and measure for success for each metric, wherever applicable
5. **Scale the aggregated performance** to have the next period candidate population
6. Roll out the new candidate population to next period, per model (Probabilistic model)

Scores are **centrality scores**(Welford's Algorithm) - not to be over-enthusiastic(pessimistic) for state-outcomes and weights for **recency of performance**, over long time learning cycles

Problem # 4 : Designing the performance benchmark

		Proxy for bias	Type I & Type II errors		Measure of entropy		Velocity Metric		Acceleration Metric	
Time period bench-marked against	Model #	Mean Square Error	Precision	Recall	Mean Score	Mean Variance Score	Velocity by Metric 1	Velocity by Metric 2	Acceleration by Metric 1	Acceleration by Metric 2
t1	M1									
	M2									
	M3									
	M4									
	M5									
t2	M1									
	M2									
	M3									
	M4									
	M5									

Problem # 4 : Representation & Codewalk

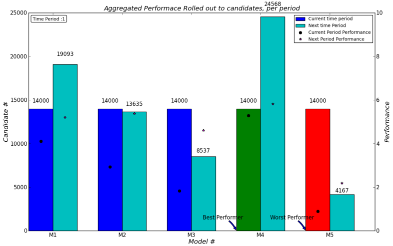
```
Prototype of the Real Time Batch Aggregator
*******************************************************************************************************************************
*******************************************************************************************************************************
                                              #####################Time 1 #####################
 Model# | Time | #Candidates |Mean Sq.| Mean | Variance | Precision| Accuracy| Velocity(V) |Velocity(C) |Acceleration(V) |Acceleration(C)
                                Error  (Score)  (Score)                        (visit)       (chats)       (Visit)            (chats)

Model# 1|   1 |      14000   | 68.431776 | 0.501543 | 0.082948 | 0.148243 | 0.611786 | 0.075000 | 0.750000 | 0.000000 | 0.000000

Model# 2|   1 |      14000   | 68.477386 | 0.501891 | 0.083045 | 0.089988 | 0.602214 | 0.045000 | 0.281250 | 0.000000 | 0.000000

Model# 3|   1 |      14000   | 68.481001 | 0.500558 | 0.084417 | 0.042317 | 0.404500 | 0.030000 | 0.300000 | 0.000000 | 0.000000

Model# 4|   1 |      14000   | 68.762492 | 0.503686 | 0.084035 | 0.099326 | 0.398286 | 0.075000 | 3.750000 | 0.000000 | 0.000000

Model# 5|   1 |      14000   | 68.190888 | 0.499248 | 0.082894 | 0.055354 | 0.550000 | 0.030000 | 0.500000 | 0.000000 | 0.000000

                                               ####################PERFORMANCE AGGREGATED METRIC ###########################

  Model# |    Time    | #Candidates |   Performance Metric
                                    (Aggregated & Normalized)

Model# 1 |     1     |     14000     | 4.10874748831

Model# 2 |     1     |     14000     | 2.93421049334

Model# 3 |     1     |     14000     | 1.83716273545

Model# 4 |     1     |     14000     | 5.28702053689

Model# 5 |     1     |     14000     | 0.89674366591

 ***********************************************************************
 delta t - Waiting to Aggregate,Process & Roll the next period decisions
 ***********************************************************************
```

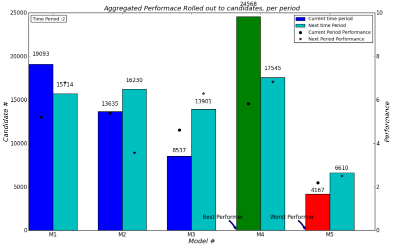
Aggregated performance in Period 1

Codewalk from Github

https://github.com/ekta1007/Performance-aggregation-platform—learning-in-near-real-time

Problem # 4 : Representation cont..

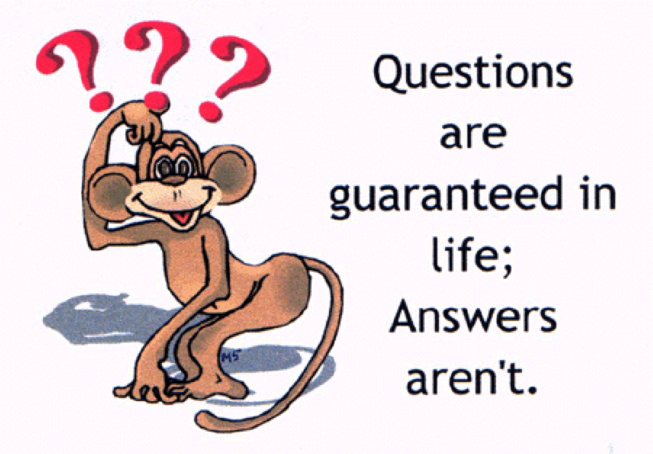




Aggregated performance in Period 1 & 2 respectively

Resources

- Link to the problems I talked today(I keep updating this)
- Understanding data structures in Python
- A Programmer's Guide to Data Mining - Ron Zacharski
- Creating a Spell Checker with Solr
- An Intuitive example to TF-IDF, Iraq War logs - Jonathan Stray
- An Introduction to Information Retrieval - Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze
- Natural Language Processing with Python - Steven Bird, Ewan Klein, Edward Loper
- Stemming & lemmatization
- Programming Collective Intelligence: Building Smart Web 2.0 Applications - Toby Segaran
- Mining the Social Web: Analyzing Data from Facebook, Twitter, LinkedIn - Matthew A. Russell


Questions
are
guaranteed in
life;
Answers
aren't.