Build your dataset for LTR

THE BASICS

1) A typical dataset.

implicit or explicit.

- A set of labelled <query, document> pairs
- Each <query, document> example is composed by:
  - relevancy rating
  - query Id
  - feature vector

3	qid:1	**0**:3.4 **1**:0.7 **2**:1.5 **3**:0
2	qid:1	**0**:5.0 **1**:0.4 **2**:1.3 **3**:0
0	qid:1	**0**:2.4 **1**:0.7 **2**:1.5 **3**:1
1	qid:2	**0**:5.7 **1**:0.2 **2**:1.1 **3**:0
3	qid:2	**0**:0.0 **1**:0.5 **2**:4.0 **3**:0
0	qid:3	**0**:1.0 **1**:0.7 **2**:1.5 **3**:1

Interactions

this json is what you aim to collect and then decide which one should be used to build the training set


```
[{
    "productId": 206,
    "interactionRelevance": 0,
    "interactionType": "impression",
    "timestamp": "2019-03-15T18:19:34Z",
    "userId": "id4",
    "query": "free text query",
    "userDevice": "item9",
    "querySelectedBrands": [209, 201, 204],
    "cartCategories": [200, 206, 204],
    "userFavouriteColours": [208, 208, 202, 202],
    "userAvgPrice": 43,
    "productBrand": 207,
    "productPrice": 22.0,
    "productDiscount": 0.7,
    "productReviewAvg": 4.5,
    "productReviews": 200,
    "productSales": 207,
    "productSalesLastWeek": 203
},{...},{...},{...},{...},{...},{...},{...}]
```

FEATURE ENGINEERING

Each sample is a **<query,document>** pair, the feature vector is its numerical representation

Document level

This feature describes a
property of the DOCUMENT.
The value of the feature depends only on
the **document** instance.

e.g.
Document Type = E-commerce Product

<Product price> is a Document Level feature.
<Product colour> is a Document Level feature.
<Product size> is a Document Level feature.

Document Type = Hotel Stay

<Hotel star rating> is a Document Level feature.
<Hotel price> is a Document Level feature.
<Hotel food rating> is a Document Level feature.

Query level

This feature describes a
property of the QUERY.
The value of the feature depends only on
the **query** instance.

e.g.
Query Type = E-commerce Search

<Query length> is a Query Level feature.
<User device> is a Query Level feature.
<User budget> is a Query Level feature.

Query Dependent

This feature describes a
property of the QUERY in correlation with the DOCUMENT.
The value of the feature depends on
the **query and document** instance.

e.g.
Query Type = E-commerce Search
Document Type = E-commerce Product

<first Query Term TF in Product title>
is a Query dependent feature.
<first Query Term DF in Product title>
is a Query dependent feature.
<query selected categories intersecting the product categories>
is a Query dependent feature.

Quantitative

A *quantitative* feature describes a property
for which the possible values are
a measurable quantity.

e.g.
Document Type = E-commerce Product

<Product price> is a quantity

e.g.
Document Type = Hotel Stay

<Hotel distance from city center>
is a quantity

Ordinal

An *ordinal* feature describes a property
for which the possible values are **ordered**.
Ordinal variables can be considered
"in between" categorical and quantitative variables.

e.g.
Educational level might be categorized as
1: Elementary school education
2: High school graduate
3: Some college
4: College graduate
5: Graduate degree

1<2<3<4<5

Categorical

A categorical feature represents an attribute of an
object that have a set of distinct possible values.
In computer science it is common to call the possible
values of a categorical features Enumerations.

e.g.
Document Type = E-commerce Product

<Product colour> is a categorical feature
<Product brand> is a categorical feature

N.B. It is easy to observe that to give an order
to the values of a categorical feature
does not bring any benefit.
For the Colour feature :
red < blue < black has no general meaning.

Categorical Features

e.g.
Document Type = E-commerce Product

<Product colour> is a categorical feature

Values: Red, Green, Blue, Other

Encoded Features:
Given a cardinality of N, we build N-1 encoded
binary features

product_colour_red = 0/1
product_colour_green = 0/1
product_colour_blue = 0/1

product_colour_other = 0/1

⚠ **High Cardinality Categoricals**
you may need to encode only the top frequent
-> information loss

⚠ **Dummy Variable Trap**
when you have highly correlated features ->
predict feature value from others features
gender_male
gender_female

Categorical Features

e.g.
Document Type = E-commerce Product

<Product colour> is a categorical feature

Values: Red, Green, Blue, Other

Encoded Features:
1) Ordinal Encoding
Red=0, Green=1 Blue=2 Other=3

2) Binary Encoding
product_colour_bit1 = 0/1
product_colour_bit2 = 0/1

you can represent numbers with bits and columns.

Better for high cardinality categorically

⚠ **Multi Valued?**
you may have collisions and not able to use binary features

Categorical Features

e.g.
Document Type = E-commerce Product

<Product colour> is a categorical feature

Values: Red, Green, Blue, Yellow, Purple, Violet

Encoded Features:
Size=3
Hashing function will take in input each category and allocate to a bucket represented by a vector of the size specified

2) Binary Encoding
product_colour_hash1 = +2
product_colour_hash2 = -1
product_colour_hash3 = +1

- Choose a hash function
- Specify a size for the hash (dedicated number output features) -> we are going to add just this number of features.
  We are basically defining the fine grain of the representation
- In total/ per feature
- Good for dealing with large scale cardinality features

N.B. be careful as the explainability of your model goes down

- Doing One Hot encoding properly => no information loss
- … but it is expensive in time and memory
- So the recommendation is to experiment :
  - One Hot Encoding
  - One Hot Encoding with freq threshold
  - Binary (if it fits your use case)
  - Hash per feature
  - Hash per feature set
  - Hash of different sizes

You want to observe: time, memory consumption, offline model evaluation impact at least (and potentially run an online experiment)

When?

It is very likely you have defined **a pipeline of steps** executed to build your training set from some sort of implicit/explicit feedback.

When should you encode your features?

Ideally on the smallest data-set as possible.
Encoding is expensive in time/memory.

So if you do any reduction, collapse, manipulation of data that reduce in size your training data, do feature encoding as late as possible, as it is quite an expensive procedure, especially for high cardinality features.

N.B. doing it later on in the pipeline may open the possibility of using encodings minimising information loss (such as full One Hot Encoding)

Missing values

- Some times a missing value is equivalent to a **0 value** semantic
  e.g.
  Domain: e-commerce products
  Feature: Discount Percentage - [quantitative, document level feature]
  a missing discount percentage could model a 0 discount percentage,
  missing values can be filled with 0 values
- Some times a missing feature value can have a completely different semantic
  e.g.
  Domain: Hotel Stay
  Feature: Star Rating - [quantitative, document level feature]
  a missing star rating it's not equivalent to a 0 star rating, so an additional feature should be added to distinguish

- **Take Away** :
  discuss with the business layer and try to understand your specific use case requirements
  check nulls and zeros count across your features, you may discover interesting anomalies

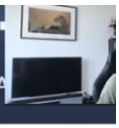
Relevance Label : Signal Intensity

- Each sample is a **user interaction** (click, add to cart, sale, ect)
- Some sample are **impressions** (we have showed the document to the user)
- A rank is attributed to the user interaction types
  e.g.
  0-Impression < 1-click < 2-add to cart < 3-sale
- The rank becomes the relevance label for the sample

↳ what if a user does multiple things?

Discordant training samples

Relevance Label : Simple Click Model



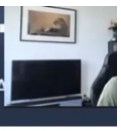
- Each sample is a **user interaction** (click, add to cart, sale, ect)
- Some sample are **impressions** (we have showed the document to the user)
- One interaction type is set as the target of optimisation
- Identical Samples are aggregated, the new sample generated will have a new feature:

  [Interaction Type Count/ Impressions]

  e.g. CTR (Click Through Rate) = for the sample, number of clicks/ number of impressions
- We then get the resulting score (in CTR 0<x<1) and normalise it to get the relevance label
  The relevance label scale will depend on the Training algorithm chosen

This collapses relevance to 1 result for each query.

Relevance Label : Advanced Click Model



Given a sample:

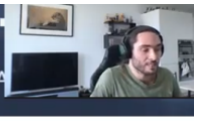
- We have the CTR from previous model
- We compare it with the avg CTR of all samples
- We take into account the statistical significance of each sample (how likely is we got that result by chance on that size of observations)
- The relevance label will be CTR/avg CTR (and we drop samples deemed to be statistically irrelevant)

seems like AB testing.

Yet to test it online in production, stay tuned!

More info in John Berryman blog:
http://blog.jnbrymn.com/2018/04/16/better-click-tracking-1/

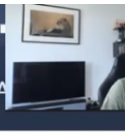
Relevance Label : Normalisation



- Min/Max normalisation based on local values (local to your data-set)
  e.g.
  your max CTR is 0.8, that is the max relevance you can expect
  your min CTR is 0.2 that is the 0 relevance
- Min/Max normalisation based on **absolute** values
  e.g.
  max CTR is 1.0, that is the max relevance you can expect
  min CTR is 0, that is the 0 relevance
- Scale of relevance?
  0-1
  0-4
  0-10 ?
  Experiment! Risk of overfitting - not simple to compare offline

Using absolute values,
you have consistent data-sets over time
But you are flattening the relevance labels
in your current data set

Learning To Rank : List-wise and NDCG



- **The list is the result set of documents for a given query Id**
- In lambdaMart is often used **NDCG@K** per list
- The Evaluation Metric is **avg** over the query Ids when evaluating a training iteration

What happens if you have very small ranked lists per query?
What happens if all the documents in a ranked list have the same relevance label?

It is extremely important to assess the distribution of training samples per query Id

↓

Under sampled QueryId can potentially sky rocket your NDCG avg

Query1

Model1	Model2	Model3	Ideal
1	1	1	1
1	1	1	1

Query2

Model1	Model2	Model3	Ideal
3	3	3	3
7	7	7	7

careful with searches that provide very small number of results.

Build the Lists: QueryId Hashing 1/3

- Carefully design how you calculate your query Id,
  it represents the Q of <**Q**,D> in each query-document sample).
  the same query across your data set must have the same Id

- No free text query? Group query level features (simple hashing, clustering)
- The free text query is an additional info to build a proper query Id where available (can be used as Id on its own)

- **Target**: reaching a uniform distribution of samples per query Id
- Drop training samples if queryIds are under-sampled
- Experiment!

How to Generate the Query Id

- Concatenate the most prominent query level selected features (from business)
- Concatenate **all** query level selected features
- Clustering query id generation using clustering algorithms of your choices.
  In particular, for k-means, with many different number of clusters
  (The number of clusters is a required k-means input parameter.
  In all the clustering approaches a cluster corresponds to a query id group).
- A lot of **hyper-parameter**! -> a lot of **experiments**

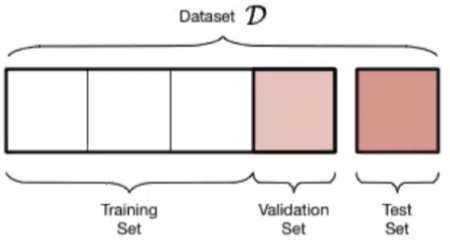
Future Works

- what happens if we **keep the undersampled query ids** instead of dropping them (assigning a new queryId to all of them for example).

How to Compare the Results?

- The average number of samples per query id in the data set
- The standard deviation of the query id distribution
  (which tell us if the number of rows per query id is equally distributed or not)
- The number of distinct query ids
- The number of query id with a low number of rows (below a threshold)
- The number of query id with 1 sample
- The number of query id and the percentage inside some ranges of values
- For example:
- (0 to 2 samples] - 3 query ids - 30%
- (2 to threshold samples] - 2 query ids - 20%
- (threshold to 99 samples] - 5 query ids - 50%
- The drop rate (the percentage of dropped rows during the removal of the under-sampled query ids)

Split the Set: Training/Test



- Each training set iteration will assess the evaluation metric on the training and validation set
- At the end of the iterations the final model will be evaluated on an unknown Test Set
- This split could be random
- This split could depend on the time the interactions were collected