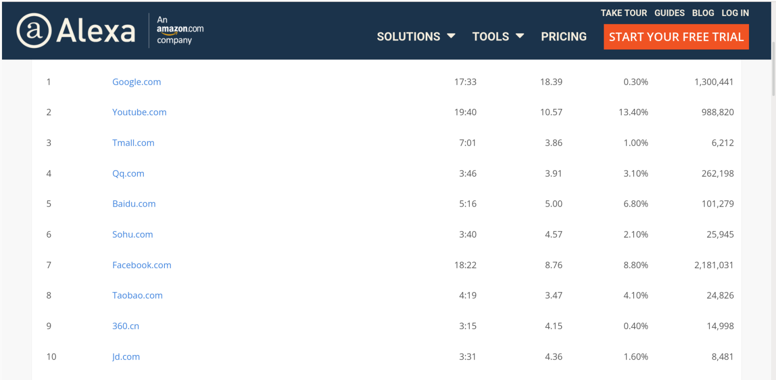
User-facing ML or LTR is everywhere

Senior Machine Learning Programmer at Proxet / Giphy

Goal

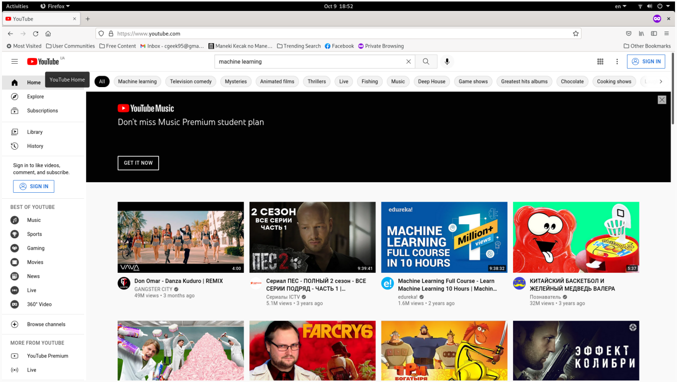
Take a popular service and review Machine Learning use-cases.



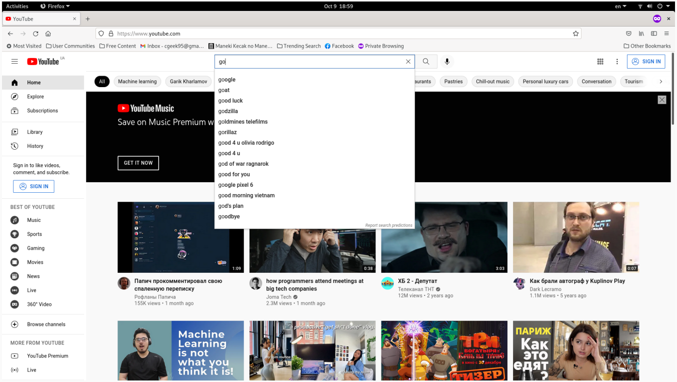
1	Google.com	17:33	18.39	0.30%	1,300,441
2	Youtube.com	19:40	10.57	13.40%	988,820
3	Tmall.com	7:01	3.86	1.00%	6,212
4	Qq.com	3:46	3.91	3.10%	262,198
5	Baidu.com	5:16	5.00	6.80%	101,279
6	Sohu.com	3:40	4.57	2.10%	25,945
7	Facebook.com	18:22	8.76	8.80%	2,181,031
8	Taobao.com	4:19	3.47	4.10%	24,826
9	360.cn	3:15	4.15	0.40%	14,998
10	Jd.com	3:31	4.36	1.60%	8,481

Figure 1: Top websites according to Alex.com

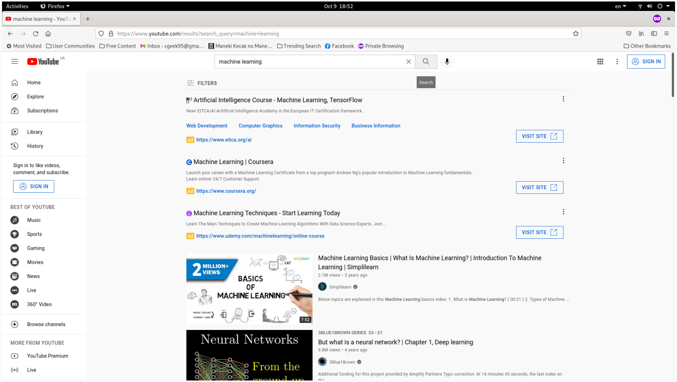
Youtube - Home Page after 1 search query



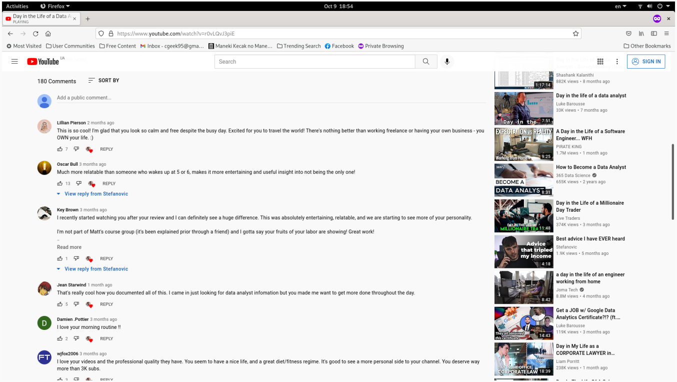
Youtube - Autocomplete



Youtube - Search



Youtube - Comments



Classification of Machine Learning Problems [Part 1]

- Learning Problems
- Hybrid Learning Problems
- Statistical Inference
- Learning Techniques

Classification of Machine Learning Problems [Part 2]

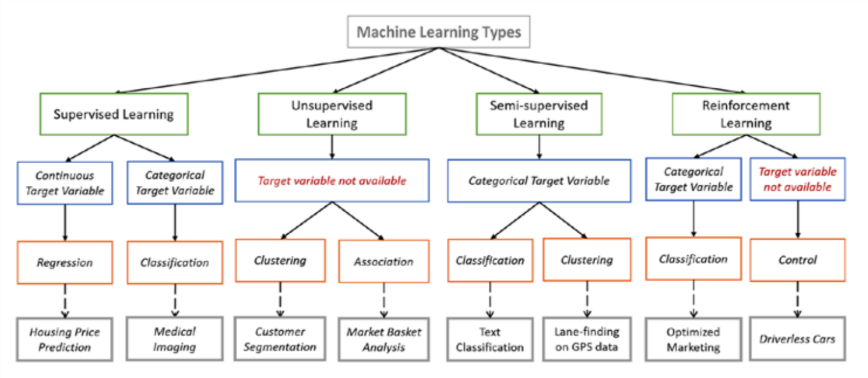


Figure 2: Types of Machine Learning Algorithms

Youtube Services

- [Personalized] Home Page Content Selection
- [Personalized] Global Search
- [Personalized] Channel Search
- [Personalized] Search Autocomplete
- [Personalized] Related Videos
- [Personalized] Playlists
- [Personalized] Trending
- [Personalized] Notifications
- [Personalized] Comments selection
- Copyright Violation
  - By Performer
  - By Music
  - By Text
  - Abusive speech
  - Violance
- [Personalized] ADs
- Security

LTR is everywhere, where we have a list of elements.

Approaches to Learning to Rank

- Pointwise (Regression/Classification)
- Pairwise (LambdaRank IR-SM, Lambda Rank)
- Listwise (Soft Rank, SmoothRank, AdaRank, ListNet, BoltzRank)

LTR-related procedures

- Candidate Generation
- Offline ranking
- Online ranking
- Data Collection
- Data debiasing
- A/B testing

What is LTR

Learning to rank or machine-learned ranking (MLR) is the application of machine learning, typically supervised, semi-supervised or reinforcement learning, in the construction of ranking models for information retrieval systems

- Candidate Generation
- Offline ranking
- Online ranking
- Data Collection
- Data debiasing
- A/B testing

Search Dataset - Click-Based

session_id	query	document_id	relevance
1	machine learning	1	0.0
1	machine learning	2	0.0
1	machine learning	3	0.0
1	machine learning	4	0.0
1	machine learning	5	1.0

Search Dataset - Click-Based

session_id	query	document_id	relevance	position
1	machine learning	1	0.0	1
1	machine learning	2	0.0	2
1	machine learning	3	0.0	3
1	machine learning	4	0.0	4
1	machine learning	5	1.0	5

Search Dataset - Human Relevance

session_id	query	document_id	relevance
1	machine learning	1	3.0
1	machine learning	2	2.0
1	machine learning	3	1.0
1	machine learning	4	4.0
1	machine learning	5	5.0

LTR evaluation

$$PairAccuracy = \sum_{i<j} [rel_i > rel_j] \tag{1}$$

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} \tag{2}$$

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{3}$$

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p} \tag{4}$$

$$\mathrm{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\mathrm{rank}_i}. \tag{5}$$

CatBoost Training

Listing 1: Train CatBoost LTR with YetiRankPairwise loss

```
import catboost
cb_pool_train = catboost.Pool(
    [[0.0], [0.0], [0.0], [1.0]],
    label=[0.0, 0.0, 0.0, 1.0],
    group_id=[1, 1, 1, 1]
)

cb_pool_eval = catboost.Pool(
    [[0.0], [0.0], [0.0], [2.0]],
    label=[0.0, 0.0, 0.0, 1.0],
    group_id=[1, 1, 1, 1]
)
params = {
    "loss_function": "YetiRankPairwise",
    "custom_metric": ["NDCG"]
}
cb_model_ltr = catboost.CatBoost(params=params)
cb_model_ltr.fit(cb_pool_train, eval_set=cb_pool_eval)
```

LTR training protocols

1. split by query
2. backtesting

Clicks vs Explicit feedback

1. Explicit feedback is not vulnerable to spam.
2. Explicit feedback can be outdated.
3. Explicit feedback has biases, which you can control with a user's manual.
4. Explicit feedback has fewer contradictions, but almost all the time, the size of a dataset is much smaller..
5. You must mimic query distribution in your dataset.
6. When you start a new product, you don't have enough clicks.
7. Explicit feedback is expensive, and you have to update the test questions very often.

Search Dataset - Feature Space

1. $f_1(document\_id)$ - knows about document modalities
2. $f_2(session\_id)$ - knows about user id, location, previous actions
3. $f_3(query)$
4. $f_4(query, document\_id)$
5. $f_5(query, session_i d)$
6. $f_6(session\_id, document\_id)$

Search Dataset - Feature Examples

1. textual similarity (BM25, BM25 + Stemming, vector-space models)
2. document-level CTR from search logs
3. document-level CTR from recommendations logs
4. different counter aggregations
5. Word2Vec on clicks (StartSpace implementation)
6. 'PageRank'
7. fraud detection
8. OCR + textual similarity
9. speach recongntion + textual similarity

The gradient boosting with YetiRankPairwise loss function gives the best results. Neural Nets are not as good, but you can build a lot of great features with Siamese neural networks. You can get ideas from Question-Answering models. Pinterest uses a simple Convolutional Neural Net, but they haven't tried the CatBoost with YetiRankPairwise loss.

Search Dataset - Data Colection

You need at least two table:

1. Requests and Responses (timestamp, query, document_id, position, response_id, location, user_id, device_id, session_id)
2. Actions (timestamp, response_id, action_id, document_id)

You can use Column-oriented database and a queue like kafka to collect the logs. Also, you can duplicate the data on S3.

Personalization

- you can train different models for different regions or encode a location in your features
- you can have different search algorithm for different user buckets
- user-based personalization kills caching
- you can cache heavy features from the head of your search log

Search vs Recommentations

- almost identical approach is applicable for Recommendation Systems, but the document id plays the role of a query.
- you should reuse document-level features
- you can recompute the same feature, but on different datasets
- recommentations are almost identical to search, but for relatively small collections you can use a heavy artillery, because you can store predictions in a relatively small matrix
- you can use top search predictions to generate recommendations

LTR for every product

Task	Query	Document	Personalizable	A/B test Target
Search	Query String	Document Id	+	CTR/MRR
Recommentations	Document Id	Document Id	+	CTR/MRR
Home Page Documents	DateTime	Document Id	+	CTR/MRR
Home Page Queries	DateTime	Query String	+	CTR/MRR
Search Autocomplete	Query String	Query String	+	CTR/MRR
Ranking Comments	Document Id	Comment + Metadata	+	Likes

Materials

1. Search Quality Rating Guidelines
2. Introduction to Information Retrieval
3. Information Retrieval: Implementing and Evaluating Search Engines
4. Demystifying Core Ranking in Pinterest Image Search
5. Catboost usage example.
6. StarSpace
7. Personalized Trending Search Suggestions
8. Winning The Transfer Learning Track of Yahoo!'s Learning To Rank Challenge with YetiRank
9. On NDCG Consistency of Listwise Ranking Methods
10. GeoTrend: Spatial Trending Queries on Real-time Microblogs
11. Finding Trending Local Topics in Search Queries
12. Challenges in building large-scale information retrieval systems: invited talk
13. SIMD-Based Decoding of Posting Lists