ML basics cheat sheet

Common models

Model name	Details	Pros	Cons	Applications
Logistic regression	h(thetaTx) = 1/(1+e^-thetaTx) Produces probability Cost function: cross-entropy	Makes no assumption on data. With large datasets, work well.	Needs lots of training data.	
Softmax regression	A generalized form of logistic regression, models multinomial distribution $p(y=i\|x;\theta) = \phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^{k} e^{\eta_j}} = \frac{e^{\theta_i^T x}}{\sum_{j=1}^{k} e^{\theta_j^T x}}$			
Gaussian discriminant analysis	Assume p(x|y) is multivariate normal distribution.	Data efficient when modelling assumption is correct.		
Naive bayes	Assume x_i s are conditionally independent given y. $p(y=1\|x) = \frac{p(x\|y=1)p(y=1)}{p(x)} = \frac{(\prod_{i=1}^{n} p(x_i\|y=1))\,p(y=1)}{(\prod_{i=1}^{n} p(x_i\|y=1))\,p(y=1) + (\prod_{i=1}^{n} p(x_i\|y=0))\,p(y$ and pick whichever class has the higher posterior probability.	Requires less training data than eg logistic regression.		Sentiment analysis at Facebook. Document categorization for PageRank at Google. Email spam filtering at gmail.
Support	Maximize margin.			

vector machine	Kernel trick: fast to compute inner product, efficient for high dimensional feature space. Regularization: outlier. Solve with quadratic programming such as SMO. $\min_{\gamma,w,b} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i$ $\text{s.t.} \ \ y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \ \ i = 1, \ldots, m$ $\xi_i \geq 0, \ \ i = 1, \ldots, m.$			
Bayesian MAP	Treat model parameter theta as a random variable with unknown value.	Common choice of prior is N(0,r^2I). Thus parameters have smaller norm and reduce overfitting. (ie., Bayesian logistic regression)		
K-means clustering	Unsupervised clustering.	Beats hierarchical clustering in: tighter cluster; faster with small K.	Non-convex, may converge to local minimal.	Cluster webpages for search engines. (‘relevance rate’)
Mixture of Gaussians	Assume data comes from K Gaussian distributions with unknown values. p(x,z) = p(x|z)p(z). K zs, z comes from Gaussians. Solve using EM.			
Principle component analysis	Normalize data to 0 mean and 1 std. Find major axis of variation.	Reduce redundancy in data (correlated feature)		Compression; reduce dimension to input to learning algorithms; noise reduction (Eigenface)

Independent component analysis	Assume data is produced by 'mixing' of n independent sources, how to find the 'unmixing' matrix. Solve using stochastic gradient ascent (maximize log likelihood)		Ambiguous (permutation of matrix). Only if data is not Gaussian and enough data.	
Reinforcement learning	When there is no 'correct answer' to direct learning, provide 'reward function'. Markov decision process: state/transitions.	Positive reward as soon as possible and postpone negative reward as long as possible.		
Boosting	Intuition: Take a set of *weak classifiers* (slightly better than random), and learn to weight them to produce a *strong classifier*. Learning: assume have access to a weak classifier. Begins by assigning training samples with equal weight. Iteratively update the weights according to classification result s.t. Incorrect samples have larger weights. Then learn classifier using the updated weights. Often used together with decision tree.	Guaranteed to converge. Lower bias while not increasing variance.		
Bayesian logistic regression	Assume model parameters follow certain distributions (ie normal distributions). Treat logistic regression outputs as probabilities.			
Gradient boosting	builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.			

Decision trees	Supervised. Top-down split is computed using local optimal according to certain metric, such as information gain.	Does not require data processing; can be visualized and interpreted; inference is fast (tree height)	overfit (can create complex model); sensitive to outlier; global optimal is NP-hard thus can only create local optimal through greedy; concepts such as XOR cannot be learned; create biased tree if one class dominates	Finance.
K- nearest neighbors	Can be supervised or unsupervised. Classification: voting of K neighbors; regression: average of K neighbors. Weight neighbors using distance so neighbors close-by contribute more.			
Random forests	Learn an ensemble of trees. During learning of each tree: randomly select a subset of instances; randomly select a subset of features. Inference is down through taking majority vote for classification or average for regression. ExtraTrees: a variation where in learning, do not compute local optimal split but choose random value.	No data processing; does feature selection; usually performs well; fast training; lots of excellent, free and open-source implementation.	Can be very large (hundreds of megabytes to store); difficult to interpret.	Predict avg number of social media shares and performance scores. Predict patterns in speech recognition.

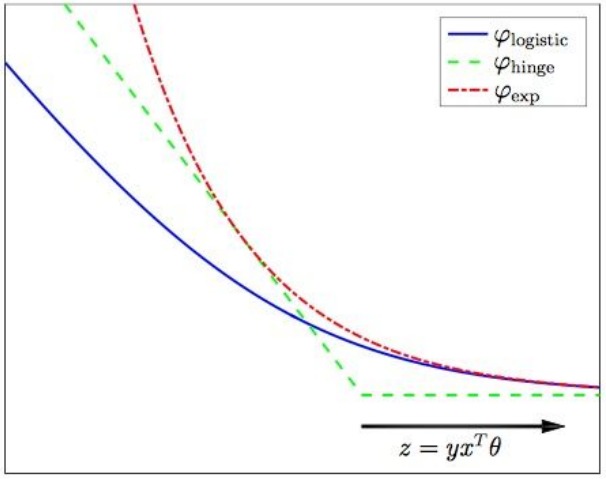
Hidden Markov Model	Related to Bayesian network. Models the markov process of hidden states. Markov process: state at St is dependent on St-1 and independent of other states before t-1. States do not depend on time. States and transitions are associated with probability. Used in reinforcement learning, speech recognition, etc. Learning task: aims at learning the parameters given the observed sequence. Intractable, but can learn using local optimal approach EM.			
Lasso regression	Force sum of the abstract value of coefficients to be small, thus set some coefficients to 0, i.e., L1 norm. L1 can be interpreted as Laplacian prior.	Improves prediction performance while reducing overfitting and perform feature selection.		
Ridge regression	L2 norm. L2 can be interpreted as Gaussian prior.		In practice you can think of it as that median is less sensitive to outliers than mean, and the same, **using fatter-tailed Laplace distribution as a prior makes your model less prone to outliers, than**	

			using Normal distribution.	
Bootstrapping	Sampling training data with replacement, creating “phantom” samples.			
Bagging	Meta-learning method using bootstrapping. Use ensemble of independently trained models.			
Boosting	An iterative process that attempts to mitigate prediction errors of earlier models by predicting them with later models.			

Loss functions

Background: NP-hard to minimize loss if it is average number of misclassification.
Thus, desired loss function should be 1) convex and 2) -> 0 when z -> infinity.

$$\varphi(z) \to 0 \quad \text{as } z \to \infty, \quad \text{while} \quad \varphi(z) \to \infty \quad \text{as } z \to -\infty.$$



Loss function	Function	Details
Logistic loss	$\varphi_{\text{logistic}}(z) = \log(1 + e^{-z})$	functions that correctly classify points with high confidence are penalized less,leads the logistic loss function to be sensitive to outliers in the data.
Hinge loss (SVM)	$\varphi_{\text{hinge}}(z) = [1 - z]_+ = \max\{1 - z, 0\}$	convex & continuous but not differentiable (thus cannot be solved using GD). Hinge loss matches indicator function, can be solved using QP when used in SVM.
Exponential loss (boosting)	$\varphi_{\text{exp}}(z) = e^{-z}.$	
Square loss		Sensitive to outliers, slow convergence.

Cross entropy	Compute average cross-entropy between true probability & predicted probability over all samples.	Convex and can be minimized using SGD, widely used in deep neural nets.

Evaluation metrics

AUC: the area under the curve (often referred to as simply the AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one

Log loss: cross-entropy

F-score:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

F-measures do not take the true negatives into account

Machine learning concepts

logistic regression,
Cost函数是什么
什么是overfitting, 如何detect，如何避免
tree 的overfitting
有哪些distance metrics
Pvalue是什么
k-means 算法具体怎么做的
如何找outlier
recommendation system
similarity metrics

What's the simplest classification model?

What are your favorite machine learning models and why?

Why is feature selection an important step in modeling and what is your favorite method to do it?

How do you go about tuning algorithm specific hyperparameters?

How do you know if your model is over-fitting and what do you do about it?

You inherited a patch of land from your uncle. The first year under your management, land yield goes down to half what it was the prior year, you investigate and find out that you uncle had a secret recipe that he didn't pass on. There are three possible types of seeds, four types of fertilizers, and two types of pesticide. How would you go about re-discovering you late uncle's formula?
A: Randomized patch experiment with varying parameters. Evaluate results using performance and evaluate confidence intervals.

What kind of metrics would you track for you music streaming website?

*metrics have blind spots, and it's good to change metrics now and then.
If you were training a classifier, which metrics would you use for model selection and why?
A: AUC

You get a weekly spam message predicting the outcome of one football game each week, the spammer claims he has insider information and will let you in on it for a significant fee. You ignore it of course, but you keep getting the weekly message and it keeps guessing the game outcome correctly for 10 weeks in a row, should you pay him? What's going on here?

On some online stores, you notice reviews for multiple-installment novels follow a peculiar trend that goes slightly up for each installment, even though the number of reviews goes down. What do you think is happening here?
A: Selection bias.

OK, that makes sense. Now what can we do to de-bias these ratings and get a score on which we can compare novels on an equal footing.
A: stratified sampling or a score combining reviews and numbers.

You are chief scientist for the air forces in WW II and you are tasked with making air strikes safer for fighter pilots (i.e. you want more of them to come back). You personally inspect damaged planes after coming back from battle (say 70% of the planes make it back on average, and 20% are damaged). You find that bullet damage is distributed in a highly non-uniform way (e.g. way more bullets in the wings region than is merited by their area). What could be the reason for this? What would you do to make planes less prone.
The United States Chess Federation (USCF) invites you to devise their new ranking system that will replace Elo. You are free to devise enhancements to the current system or propose a completely new ranking algorithm.

How would you go about building an ensemble of hundreds of highly diverse models? (resulting from different algorithms and different parameters)
A: Bagging, boosting are good. Use stacking here.

How would you sample uniformly from a continuous stream of data?
A: Reservoir sampling.

Assume you already have a classification model with great ROC curve, but the model produces arbitrary scores that do not map to probability estimates, how would you go about calibrating the scores into probabilities?

Explain the bootstrap sampling method and when it can be useful.

What type of problem does the model try to solve?
Is it prone to over-fitting? If so – what can be done about this?
Does the model make any important assumptions about the data? When might these be unrealistic? How do we examine the data to test whether these assumptions are satisfied?
Does the model have convergence problems? Does it have a random component or will the same training data always generate the same model? How do we deal with random effects in training?
What types of data (numerical, categorical etc…) can the model handle?
Can the model handle missing data? What could we do if we find missing fields in our data?
How interpretable is the model?
What alternative models might we use for the same type of problem that this one attempts to solve, and how does it compare to those?
Can we update the model without retraining it from the beginning?
How fast is prediction compared to other models? How fast is training compared to other models?
Does the model have any meta-parameters and thus require tuning? How do we do this?
(Deeper machine learning questions)
What is the EM algorithm? Give a couple of applications
What is deep learning and what are some of the main characteristics that distinguish it from traditional machine learning
What is linear in a generalized linear model?
What is a probabilistic graphical model? What is the difference between Markov networks and Bayesian networks?
Give an example of an application of non-negative matrix factorization
On what type of ensemble technique is a random forest based? What particular limitation does it try to address?
What methods for dimensionality reduction do you know and how do they compare with each other?
What are some good ways for performing feature selection that do not involve exhaustive search?
How would you evaluate the quality of the clusters that are generated by a run of K-means?
(Tools and research)
Do you have any research experience in machine learning or a related field? Do you have any publications?

What tools and environments have you used to train and assess models?
Do you have experience with Spark ML or another platform for building machine learning models using very large datasets?

A very complete question sets: Link

1 — Statistics

In order to understand Machine Learning, a solid knowledge of statistics fundamentals is essential. This involves understanding the following:

- Different ways to measure model success (precision, recall, area under ROC curve, etc.). How your choice of loss function and evaluation metric biases the outputs of your model.
- How to understand overfitting and underfitting, and the bias/variance tradeoff.
- What confidence can you attribute to the results of your model.

2 — Machine Learning Theory

When you are training a neural network, what is actually happening? What makes some tasks doable and others not? A good approach to this might be to first try to understand Machine Learning through graphics and examples, before diving deeper into the theory.

Concepts to understand range from how different loss functions work, why back propagation is useful, or what a computational graph is. A deep understanding is crucial both for building a functional model, and to communicate about it efficiently to the rest of the organization. Following are a few resources, starting with high level overviews, and diving deeper.

- Google's Deep Learning course is a solid general introduction

- Fei-Fei Li's Computer Vision course, Richard Socher's NLP course for more specialized approaches
- Goodfellow's excellent Deep Learning book for a more comprehensive overview of fundamentals.

Another fundamental skill is the ability to read, understand and implement research papers. It can seem like a daunting task at first, so a good way to start is to look up a paper that already has code attached to it (on GitXiv for example) and try to understand the implementation in depth.

3 — Data Wrangling

Ask any Data Scientist and they'll tell you 90% of the work they do is data munging. This is just as important for Applied AI, as the success of your model correlates hugely with the quality (and quantity) of your data. Data work comes in many aspects, and falls within a few categories:

- Data acquisition (finding good data sources, accurately gauging the quality and taxonomy of the data, acquiring and inferring labels)
- Data pre-processing (missing data imputation, feature engineering, data augmentation, data normalization, cross validation split)
- Data post-processing (making the outputs of the model usable, cleaning out artifacts, handling special cases and outliers)

The best way to get familiar with data wrangling is to grab a dataset in the wild and try to use it. There are many datasets online and many social mediaand news outlets sites have great APIs. Following the steps above, a good way to learn is to:

- Grab an open dataset and examine it. How big is it (number of observations and features)? How is the data distributed? Are there any missing values or clear outliers?

- Start building a pipeline of transformations to go from raw data to usable data. How will you backfill missing values? What is a proper way to deal with outliers? How will you normalize data? Can you create more expressive features?
- Examine your transformed dataset. If everything looks good, move on to the next part!

4 — Debugging/Tuning models

Debugging Machine Learning algorithms that fail to converge or to give sensible results involves a very different process from debugging code. In the same vein, finding the right architecture and hyperparameters requires solid theoretical fundamentals, but also good infrastructure work to be able to test different configurations out.

Because of the pace at which the fields evolve, the methods to debug models are constantly evolving. Here are a few “sanity checks” from our discussions and experience deploying models that mirror in some ways the principles of KISS familiar to many Software Engineers.

- Start with a simple model that has been proven to work on similar datasets to get a baseline as soon as possible. Classical statistical learning models (linear regression, Nearest-neighbors, etc.) or simple heuristics or rules will often get you 80% of the way and be much faster to implement. Start with solving the problem in the simplest way (See the first few points of Google’s rules of ML).
- If you decide to train a more complex to improve upon that baseline, start by training it to overfit on a very small sub-section of your dataset. This assures that your model at least has the capacity to learn. Iterate on your model until you can overfit on 5% of your data.
- Once you start training on more data, hyperparameters start playing a bigger role. Understand the theory behind those parameters to understand what are reasonable values to explore.

- Take a principled approach to tuning your model. At the bare minimum write down the configurations you've used and a summary of their result. Ideally, use an automated hyperparameter search strategy. Random search can be sufficient at first. Feel free to explore more principledapproaches.

A lot of those steps can be accelerated significantly by your development skills, which brings us to our last skill.

5 — Software Engineering

A lot of Applied Machine Learning will allow you to leverage Software Engineering skills, sometimes with a little twist. These skills include:

- Testing various aspects of the pipeline (data pre-processing and augmentation, input and output sanitization, model inference time).
- Building code in a modular and reusable way to accelerate the speed at which you can experiment.
- Backing up (checkpointing) models at different points in training.
- Setting up a distributed infrastructure to run training, hyperparameter search, or inference more efficiently.

For more details on some of the software skills we recommend acquiring to become a quality Machine Learning Engineer, check out our post dedicated to transitioning to Applied AI from Academia.

Putting the tools to work

The resources above will help you approach and tackle actual Machine Learning problems. But the field of Applied AI changes extremely quickly, and the best way to learn, is to get your hands dirty and actually try to build out an end-to-end solution to solve a real problem.

Action Items:

- Find a product you could build that would be interesting. What would make your life more efficient? What is a tool that could improve the way something is done using data? What is a data driven way to solve an interesting problem?
- Search for datasets related to the question. For most tractable problems today, labelled data is what you are looking for here. If no labelled datasets exist for exactly your problem, it is time to get creative. What are ways you can find similar data, or label it efficiently, or bootstrap it some other way?
- Start by exploring the data and see if the task you are trying to accomplish is possible with the amount and quality of data you have. Before you bring out TensorFlow, it is a good idea to look online for ways that people have solved similar problems. What are some relevant blog posts, and papersyou could read to get up to speed on good avenues to explore?

Machine learning system design

Recipe:

Move fast for scale: start with the most simple model (i.e., logistic regression). **Try** different things quickly.

1. Problem formulation: regression; classification; clustering; supervised/unsupervised

Tool: analyze scenario. Goal.(Metric, what to optimize). QPS.

2. Data: where/how to get what data. How to store/retrieve the data. What features to use.

Tool: think of feature engineering. Transfer intuitive into concrete feature (For example: number of likes is a good idea but a better feature would involve normalization, smoothing and bucketing.) Feature selection (lasso(L1) regularization; decision tree; direct feature selection) Skewed-sampling (most updates have no events/clicks...too many negative samples).

3. Model: what model to choose.

Tool: common models (logistic regression, decision tree, boosted decision tree, random forest, svm, neural network, HMM, bayesian network, bayesian logistic regression, mixture of gaussians, K-means, PCA…) Trade-offs.

4. Train/evaluate: offline/online. What metrics to use.

Tool: cross-validation. Implicit, explicit metric. (Explicit: survey)

4.5 Scale up

Tool: distributed system. Hadoop, MapReduce.

5. Re-train.
6. Debug model. Keep fresh(new data; new model). Compartmentalize each part of the model.
7. Deploy: A/B testing.
8. How to add new requirements/features to the model. Or, how to combine with other products.

Think about the problem end to end. What will you do after you train the model and the model does not perform well? How do you go about debugging an ML model? How do you evaluate and continuously deploy an ML model?

How would you build, train, and deploy a system to detect if multimedia and/or ads contents being posted violate terms or contains offensive materials?

设计一个系统来检测fake information（从数据收集一直到train和eval）

设计一个CV系统来帮助在线卖家自动分类所卖物品的类别

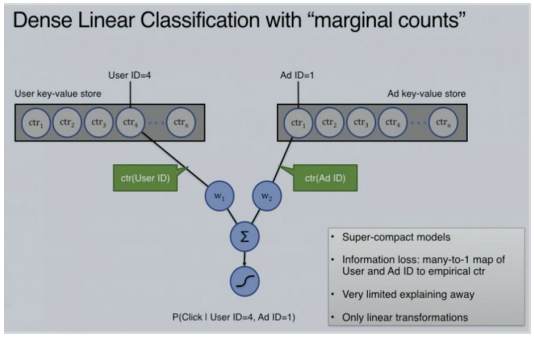
Design a pages recommendation system (FB pages)

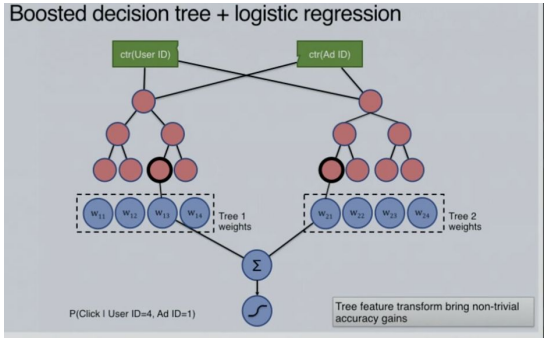
Design FB ad targeting, how likely a user will click an ad.

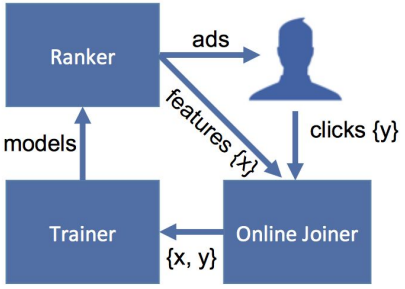
* geo-graphically distributed set of fast web servers, low-latency data storage, a pipeline for getting data back, some form of data storage system, analytic platform on top of that storage layer.
* Cassandra not necessary, keep things in server cache or use HBase or other in-memory store.
2014 data: 750 million daily active users and over 1 million active advertisers.
2016: 1 billion daily users
10s of millions ads to choose from => many trillion ads are ranked every day.

A paper introduced a model including decision trees with logistic regression.
Feature is important: those capturing historical information about the user or ad are dominant. Feature can have two types: contextual features (exclusively depends on current information regarding the context in which the ad is to be shown) and historical features (depends on previous interaction for the ad or the user, e.g., CTR of the ad in the last week, or avg CTR of the user).

Early work proposed the bid and pay per click auction. The efficiency of an ads auction depends on the accuracy and calibration of click prediction. The click prediction system needs to be robust and adaptive, and capable of learning from massive volumes of data. In sponsored search advertising, the user query is used to retrieve candidate ads, which explicitly or implicitly are matched to the query. At Facebook, ads are not associated with a query, but instead specify demographic and interest targeting. As a consequence of this, the volume of ads that are eligible to be displayed when a user visits Facebook can be larger than for sponsored search. In order to tackle a very large number of candidate ads per request, where a request for ads is triggered whenever a user visits Facebook, we would first build a cascade of classifiers of increasing computational cost.
Normalized entropy (normalized cross-entropy): predictive log loss normalized by the entropy of the background click through rate (CTR). Background CTR is the average empirical CTR of the training dataset.
Calibration is the ratio of the average estimated CTR and empirical CTR: it is the ratio of the number of expected clicks to the number of actually observed clicks. Calibration is a very important metric since accurate and well-calibrated prediction of CTR is essential to the success of online bidding and auction. The less the calibration differs from 1, the better the model is.
Data freshness is important: data distribution changes over time.
Training model more frequently. Featuring generating part can be trained less frequently, while the linear classifier can be trained much faster, almost real-time.

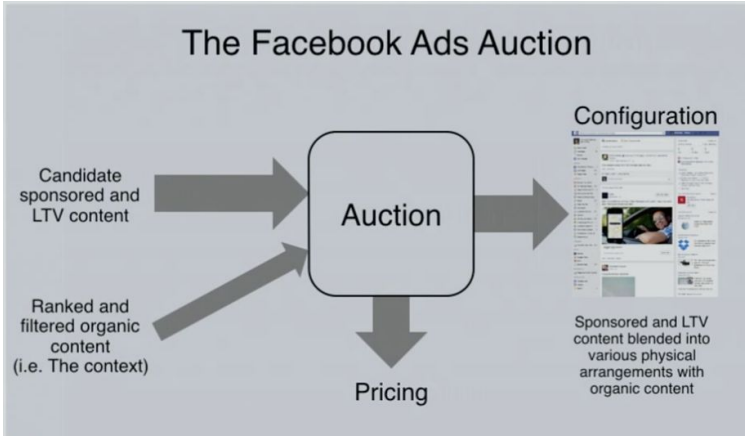






online learning data/model flows

Using heterogeneous network with users and ads: based on user-ad pairs (user clicked ad), predict clicks for other user-ad pairs.
Ads auction:
There are three parts in the UI: real-time organic content (users), real-time sponsored content (ads sponsors), long-term value content (promoted by FB, a lot recommendations, friends, groups, activities).



from ads side, do not interfere or insert into organic section.

Design an Ads statistic system

Answer the following types of queries
1. Given a user, return his CTR for all types of Ads
2. Given an Ad type, return its average CTR over all the users.
3. Given an Ad type, return the top-X users with highest CTR, 1<X<100.
Key: how to design relational database (DB). This system is write-heavy, caching is useful (delayed write)
So design a table, which is a relation. The row (a tuple, or a record) is for a user, and column (attribute or field) is for an Ads type. Store clicks and impressions separately

	Ads type 001 clicks	Ads type 001 impressions		Ads type 298…
User 2315	# clicks	# impressions		

User 7098				

Because write-heavy, i.e., updating the clicks and impressions constantly, generally the main bottleneck of performance is disk I/O. Delayed writes improve performance in three ways (benefits): by allowing the file cache to absorb some writes without ever propagating them to disk (write cancellation), by improving the efficiency of disk writes, and by spreading bursts out over time.
MySQL command to find out the sum of a column:

```
   A B C
1. 2 2 2
2. 4 4 4
3. 6 6 6
```

```
select sum(A),sum(B),sum(C) from mytable where id in (1,2,3);
```

for query 3, perhaps do denormalization in cache while adding one column in database, counting for ranking of CTR.

Design FB event reminder, allow user to set time period to send out the reminder.
How to extend database, storage scalability

FB news feeds ranking

Goal: to show users the stories users care about, matters the most. Every time you visit, only new content, put it on top. As a result, frequent users will almost get chronological news feed, more casual users will get content ranked by score (importance or interests). The **score or importance is a weighted sum over several events (like, click, share, hide, etc.)**, and model for each event can be trained separately. (related to the feature selection later on)
Stats:

- Over 1 billion users per day
- Hundreds of billions of stories seen per day
- Trillions of stories ranked per day
- Publish in feed <1s
- Retrieval + rank time < 200ms

Average person has 1500+available stories per day
Without ranking: people miss important ones; people have different preferences. Cannot measure user interests directly.
Feed metrics: clicks, likes, comments, shares, view time. And then the expected feed value: assign each event some number of points (score for different actions, like is 4 pts, share is 40 pts, hide is -100, etc.); for each story, predict the probability the user will take that action. And then can compute expected value.
Predicting likes: features?

I. Target user's historical like-through-rate on stories from certain user.

II. Target user's historical like-through-rate on certain relationship (or other types) stories (a distribution over different types of events)
III. overall historical like-through-rate on this kind of story.

Proxies to knowing that a user is interested in this story and predict the actions based on:
1) who posted it. 2) type of content (prefer to comment or click on photo, or other types) 3) interactions with the post (the more likes, more comments, etc., although similar content from the same place, the more likely to click it) 4) when it posted (recency is relevant) relevant score
How to determine if making any changes works (evaluate the news feed ranking effectiveness): are people liking more, sharing more, commenting more with friends, spending more time on FB (more interactions). Feed quality program: feed quality panel (have people who spend time organizing their stories from the most interesting to the least, and then compare their order to FB order (ranking algo)) + online surveys (tens of thousands of these get filled out a day, asking people how interested they are in a specific story, in 30 languages; and try to get better at predicting this over time)
Furthermore, controls for people (give users tools to manually manage what they want to see): follow/friend, unfollow, hide, see first.

What to do to get higher in ranking: 1) compelling headlines 2) avoid overly promotion 3) try things!!!! 4) publisher tools (audience optimization)

* dealing with baiting and hoaxes content (fake news); reducing overly promotional posts

Feature selection:
Start by training with smaller amount of data and select the most meaningful features. train model for each type of event separately (like model, click model, share model, etc.). Train boost decision tree (BDT) and find which of the features are important out of over 100k potential (dense) features. Feed in the number of training samples that can fit in the memory, and remove the least important features, and keep adding more data. How to cut the data size: train only on the events that are important for each specific model. For example, don't have to use the click data when training a like model.
*Observation: most events are rare, so most examples are negative => 1) under sampling the negative data, 2) remove most of the impressions where nobody is interacting with, keep all over sampling the events that we care about. This means **skewed sampling** helps.
*Model: needs to be really quick (BDT is not fast enough). Logistic regression is simple fast and easy to distribute. Stacking BDT (as feature transform) and logistic regression together. Can feed the output of BDT as categorical features into a logistic regression model. Also can include neural networks (featurizers), and use the last layer as categorical feature. Also can sue sparse features directly.
Label Generating:
What are the labels. For most events, such as clicks or likes, labels can be binary: click or no-click, etc. {-1, +1}.

Scoring based Ranking

- Given a potential feed story, how good is it?
  - Express as probability of click, like, comment, etc.
  - Assign different weights to different events, according to significance
- Example: close coworker feels earthquake
  - Highest chance of click
  - Decent chance of like/comment

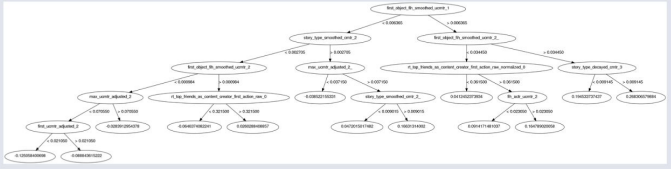
Event	Probability	Value*
Click	5.1%	1
Like	2.9%	5
Comment	0.55%	20
Share	0.00005%	40
Friend	0.00003%	50
Hide	0.00002%	-100
Total		0.306

*Example, not real values

feature selection, just run the algorithm and look at which variables are included in the resulting tree

Feature Selection (BDTs)

- Start with over >100K potential (dense) features and all historical activity
- First, prune these to top ~2K
  - Training time is proportional to number of examples * number of features
  - Under-sample negative examples (impressions, no action) to help with # of examples
  - Start with 100K features, max rows, keep most important 10K, train 10x rows
- Do this for each feed event type: train many forests
- Historical counts and propensity are some of the strongest features



Refer to the practical lessons paper of FB

$error = bias + variance$

- Boosting is based on **weak** learners (high bias, low variance). In terms of decision trees, weak learners are shallow trees, sometimes even as small as decision stumps (trees with two leaves). Boosting reduces error mainly by reducing bias (and also to some extent variance, by aggregating the output from many models).
- On the other hand, Random Forest uses as you said **fully grown decision trees** (low bias, high variance). It tackles the error reduction task in the opposite way: by reducing variance. The trees are made uncorrelated to maximize the decrease in variance, but the algorithm cannot reduce bias (which is slightly higher than the bias of an individual tree in the forest). Hence the need for large, unprunned trees, so that the bias is initially as low as possible.

Please note that unlike Boosting (which is sequential), RF grows trees in **parallel**. The term `iterative` that you used in thus inappropriate.

side note: random forest vs. boosted decision tree

Logistic regression is linear in the sense that the predictions can be written as

$$\hat{p} = \frac{1}{1 + e^{-\hat{\mu}}}, \text{ where } \hat{\mu} = \hat{\theta} \cdot x.$$

Thus, the prediction can be written in terms of $\hat{\mu}$, which is a linear function of x. (More precisely, the predicted log-odds is a linear function of x.)

Conversely, there is no way to summarize the output of a neural network in terms of a linear function of x, and that is why neural networks are called non-linear.

Also, for logistic regression, the decision boundary $\{x : \hat{p} = 0.5\}$ is linear: it's the solution to $\hat{\theta} \cdot x = 0$. The decision boundary of a neural network is in general not linear.

(a)

The logistic regression model is of the form

$$\operatorname{logit}(p_i) = \ln\left(\frac{p_i}{1 - p_i}\right) = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \cdots + \beta_p x_{p,i}.$$

It is called a *generalized* linear model not because the estimated probability of the response event is linear, but because the logit of the estimated probability response is a linear function of the ~~predictors~~ parameters.

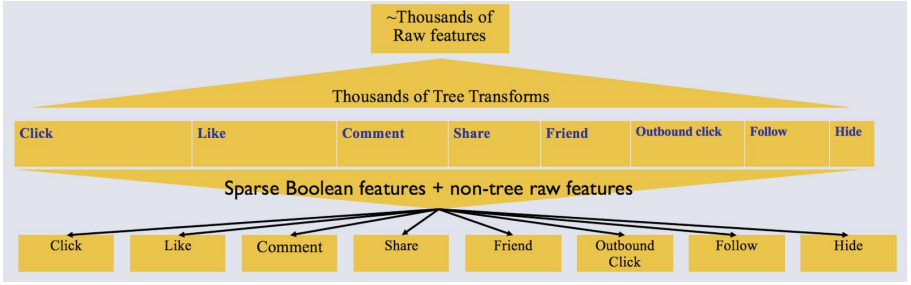
More generally, the Generalized Linear Model is of the form

$$g(\mu_i) = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \cdots + \beta_p x_{p,i},$$

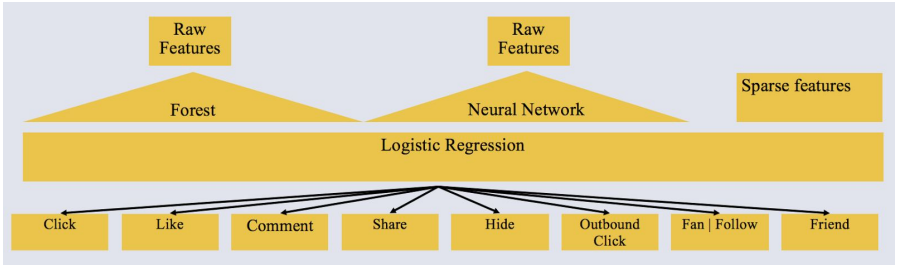
where μ is the expected value of the response given the covariates.

(b)

(a) and (b) are talking about: logistic regression can be considered as a linear classifier.



BDT + LR



BDT+NN+sf+LR

On using neural networks: use multitasking neural nets:
Raw features => shared deep nets => [click, like, …] transfer learning between labels
* most important features for feed ranking are based on user affinity: simple things like previous likes/comments do well. But we can do more using graph structure. Embededness: tends to find large cliques, might miss important bridging nodes. (consider dispersion of graph)

Data freshness matters - simple models allows for online learning and twitch response
Historical counters as features provides highly predictive features, easy to update online

Measures: define metrics:
- Implicit (limited):
- Explicit: survey.
- Compare two different stories side by side.
- Want this feed or not

Rating (how much do you want to see the story in your feed), in context survey. Pros: in context; cons: can distract, lead to abandonment.

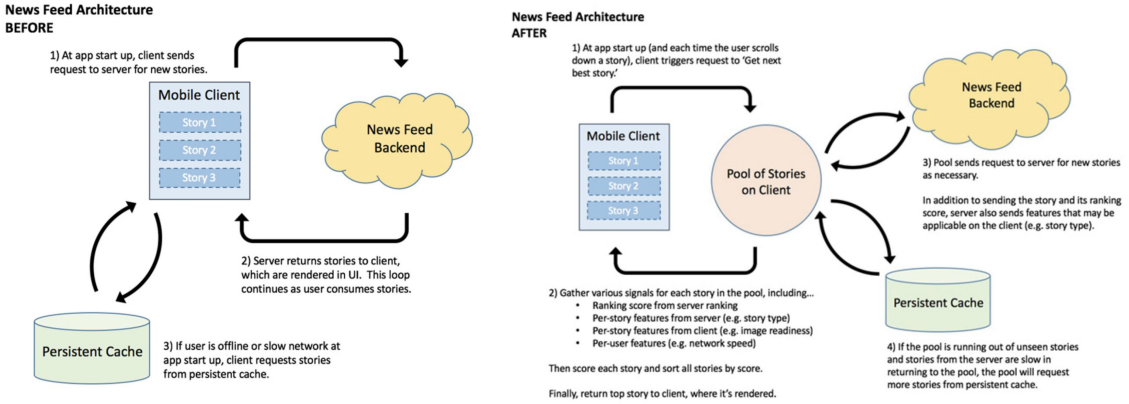
Absolute vs. relative ratings. (use both)

Pull or push news/stories to user? Need to keep a queue for every user? High traffic such as when it is new year, what will happen to traffic? How to solve potential problem?

Facebook expects over 1 billion photos shared on New Year's Eve. Different events generate different loads: holidays more photos and videos, sports more status updates. Try to predict surge in traffic. Prepare to bring additional (cache) capacity.

* on decision tree: leaves represent labels and branches represent conjunctions of features that lead to those class labels. Decision trees are very powerful, but a small change in the training data can produce a big change in the tree. This is remedied by the use of a technique called gradient boosting. Namely, the training examples that were misclassified have their weights boosted, and a new tree is formed. This procedure is then repeated consecutively for the new trees. The final score is taken as a weighted sum of the scores of the individual leaves from all trees. Like other boosting methods, **gradient boosting combines weak "learners" into a single strong learner in an iterative fashion.**

* news feed re-ranking on the client side:



Design privacy setting: users can post things, how to design friends only can see. Follow up: how to design friends of friends only can see.

Services:

User service: register(), Login()

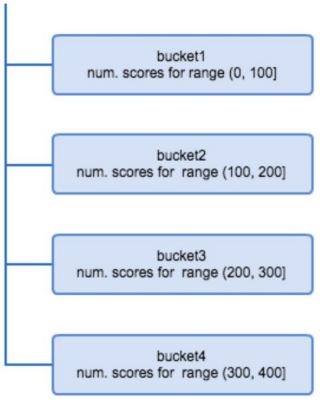
Post service: createPost()

Considering the TAO paper by FB. “Facebook’s application servers would query this event’s underlying nodes and edges every time it is rendered. Fine-grained privacy controls mean that each user may see a different view of the checkin: the individual nodes and edges that encode the activity can be reused for all of these views, but the aggregated content and the results of privacy checks cannot.”

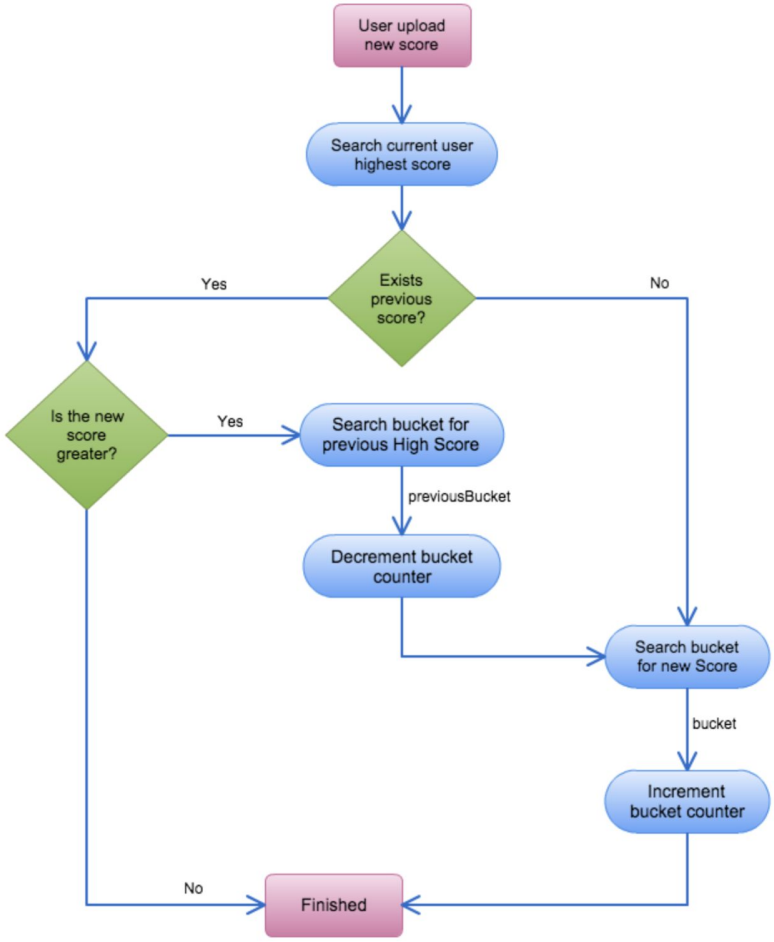
Design a ranking system for online game

Game is counted by arounds. After each round each player will get a score. Each user can add other users as friends, and the number of friends is arbitrary. After each round, two tables pop up: 1) the user and/or his/her friends top 10 according to score ranking; 2) user’s rank among all the players (several million), top 10 scores, last 10 scores.
Algorithm part:
If total point is given (or largest possible score), say 10,000 pts, maintain a length of 10001 BIT (binary index tree), if a user gets 90,000 pts, use BIT to count how many is higher than 90,000 pts. For top 10 and bottom 10, use two heaps to maintain. Consistency is more important, i.e., when a new score is obtained and tries to update the heap, others cannot change it. All the scores will be sent in a queue waiting to update the heaps and BIT. For friends’ rank, pull the score for all the friends and then sort.
Or we can use red-black tree (self-balancing binary search tree, search, insert, delete are all O(logN)) and bucket sort.
Bucket sort (http://arturogutierrez.github.io/creating-an-efficient-leaderboard). Obtaining the rank of a user in all users (millions) is doing a query of all Scores better than a given score and count them.

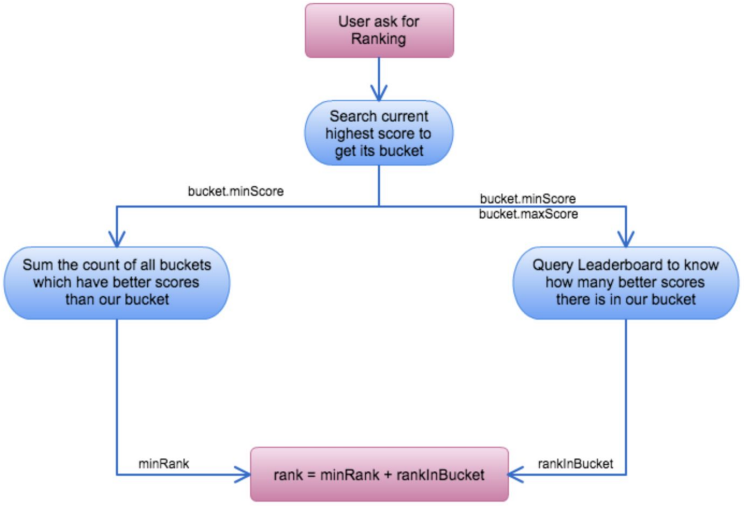
So, how can we do it?. I had the idea of using counters per score ranges, I called each counter as `Bucket`. The idea is counting how many scores there is in a range of scores on `Leaderboard` and save the count in the bucket.



How to distribute the ranges of the scores? Or the range of each bucket? If the scores are distributed linearly, it is easy. According to long tail theorem, 80% is not in top 20%, most are low scores, so low scores should have finer granularity. Also, potentially we can dynamically adding or deleting buckets depending on the number of users increasing or decreasing. And most users are not active, so majorities of the buckets are not changing. So it is a read heavy system.



ranking for a user:



Final rank = bucketRank + rankInBucket
bucketRank is the weighted sum of bucket whose lower range is greater than the user's score.
System part:
How to scale up with distributed system. Bucket sort is naturally suitable for distributed system.

Design a system of detecting fake news

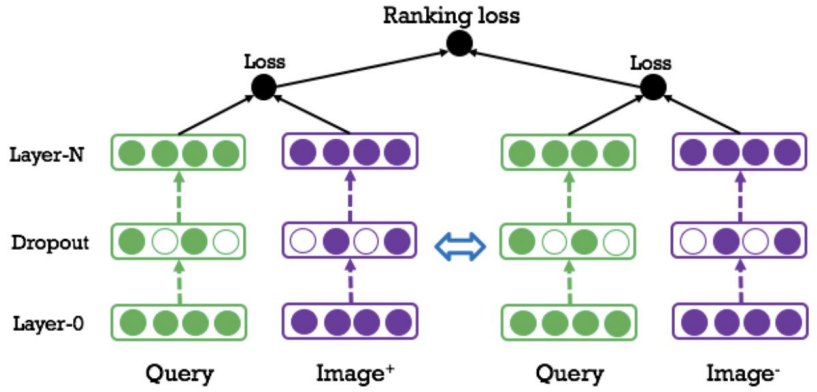
From NLP perspective: certain words may be used in fake news more often than in real news. Eg., 'establishment'. Some words may be used in real news more often, eg., 'quote', 'said'...
Can also detect sentiment…
Article on steps for detecting fake news.
Dataset: Kaggle published a dataset with 13k fake news.

Photo Search in FB

Photo Search was built with Unicorn, an in-memory and flash storage indexing system designed to search trillions of edges between tens of billions of users and entities
Graph search: built to retrieve objects from the social graph based on the relationships between them ==> presents engineering challenges when constraining the query to a relevant subset, sorting and scoring the results for relevancy, and then delivering the most relevant results. To augment this approach, the Photo Search team applied deep neural networks to improve the accuracy of image searches based on visual content in the photo and searchable text

Each of the public photos uploaded to Facebook is processed by a distributed real-time system called the image understanding engine. The engine is built on top of ResNet, we can train models and store useful information ahead of time, which enables low-latency responses to user queries. It produces high-dimensional float vectors of semantic features, too computationally intensive for indexing and searching. ***ITQ and locality-sensitive hashing tech,*** binary representation as compact embedding. The object tags and semantic embeddings populate Unicorn with an index for search queries.

It is impossible to apply complex ranking model to the entire photo store at FB scale. A relevance model applied to the tags and embeddings estimates relevance and produces low-latency query results. Relevancy is assessed with rich query and photo concept signals by comparing the concept sets with a similarity function. The relevance model exploits multimodal learning to learn a joint embedding between a query and image. The inputs to the model are the embedding vectors of the query and the photo result. The objective of training is to minimize classification loss.



adding ranking loss

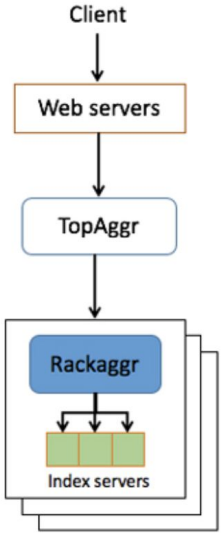
Query understanding:
Query intents (which type of scenes should be retrieved); syntactic analysis (sentence's grammatical constituents); entity linking: (often represented by a page, help identify photos about specific concepts); rewriting query knowledge (extract concepts, provide semantic interpretation, not only extending query meaning, but also bridge the gap between different vocabularies used by query and result); query embedding (transfer learning on top of word2vec representation, which maps similar queries to nearby points)

Verticals and query rewriting

When someone types a query and hits search, a request is generated and sent to our servers. The request first goes to the web tier, which collects various contextual information about the query. The query and associated context get sent to a top aggregator tier that rewrites the query into an s-expression, which then describes how to retrieve a set of documents from the index server.

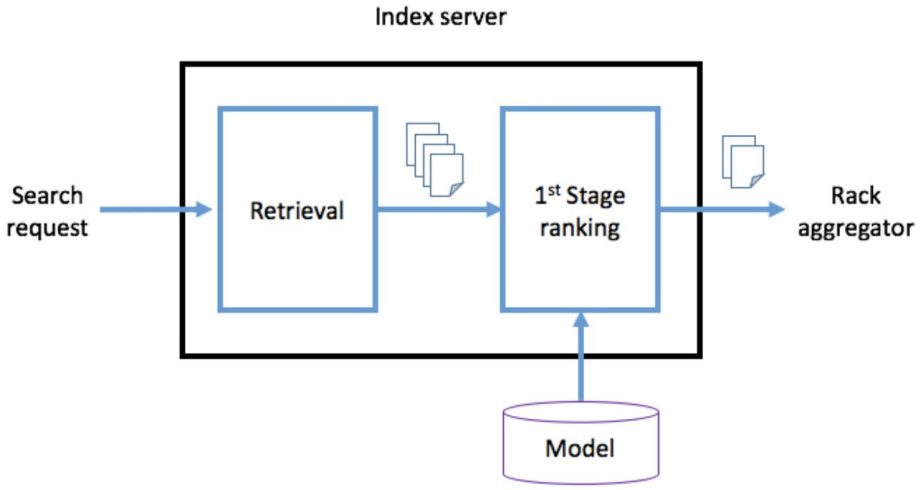
Based on the query intent, a triggering mechanism is employed using a neural network model to decide which verticals — for example, news, photos, or videos — are relevant to avoid unnecessary requests processed on less relevant verticals. For example, if a person queries the term "funny cats," the intent would search and return more results from the photos vertical and skip querying results from the news vertical.

If a query about Halloween triggers both the intent for public photos and photos of friends in Halloween costumes, both the public and social photo verticals will be searched. Photos shared among the searcher's friends and public photos ranked as relevant will be returned. Two independent requests are made because social photos are highly personalized and require their own specialized retrieval and scoring. Photo privacy is protected by applying Facebook's systemwide privacy controls to the results. The diagram below depicts a module where the top section is social and the bottom is public.



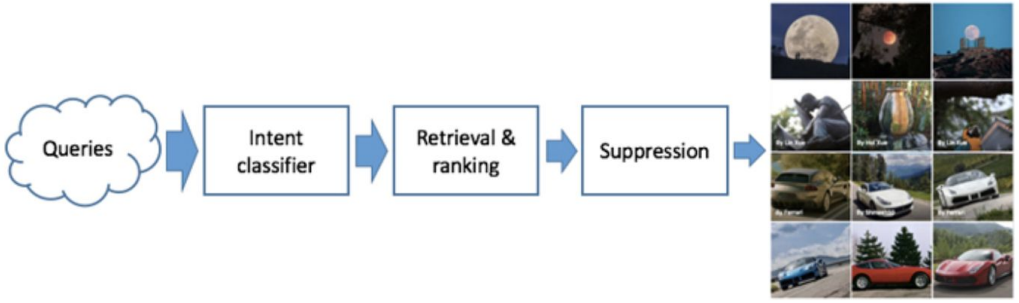
First-stage ranking

After the index servers retrieve documents according to the s-expression, the machine-learned first-stage ranker is applied to those documents. The top M documents with the highest scores are sent back to the rack aggregator tier, which performs the merge sort of all documents it receives and then returns the top N results to the top aggregator tier. The main goal of the first-stage ranking is to make sure that the documents returned to the rack aggregator preserve relevance to the query. For example, for the query “dog,” the photos with dogs should be ranked higher than those without dogs. The latency from the complexity of the retrieval and ranking stage is balanced to serve relevant photos on the order of milliseconds.



Second-stage re-ranking

After the ranked documents are returned to the top aggregator, they go through another round of signals calculation, deduplication, and ranking. The signals describing the distribution of the entire result are calculated, detecting outlying results. Next, the documents are deduplicated of visually similar results using image fingerprints. A deep neural network then scores and ranks the final order of the photo results. The collection of ranked photos, referred to as a module, is then passed to the results page UI.



Fine-tuning relevance ranking for Photo Search

The assessment of a query's relevance to a photo and vice versa is a core problem of Photo Search that extends beyond the scope of text-based query rewriting and matching. It requires a comprehensive understanding of the query, author, post text, and visual content of the photo result. Advanced relevance models incorporating state-of-the-art ranking, natural language processing, and computer vision techniques were developed to fine-tune the relevance of those results, giving us a novel image taxonomy system capable of delivering fast, relevant results at scale.

ranking loss:
* from fast triplet hashing paper:

hinge ranking loss:

$$\mathcal{L}(\boldsymbol{z}_i, \boldsymbol{z}_j, \boldsymbol{z}_k) = \max(0,\, q/2 - (d_H(\boldsymbol{z}_i, \boldsymbol{z}_j) - d_H(\boldsymbol{z}_i, \boldsymbol{z}_k)).$$

where zi and zj are similar pair, and zi and zk are dissimilar pair, q is the code length. Minimizing the above loss function indicates that it encourages that similar pair distance ideally should be q/2 smaller than dissimilar pair.

CVPR'17 (Discretely Coding Semantic Rank Orders for Supervised Image Hashing)

$$\begin{aligned}\Lambda(\mathbf{x}_i, \widehat{\mathbf{x}}_j) &= \sum_{l=1}^{k} I(||\mathbf{b}_i - \widehat{\mathbf{b}}_j||_{\mathrm{H}} \geq ||\mathbf{b}_i - \widehat{\mathbf{b}}_l||_{\mathrm{H}}) \\ &= \sum_{l=1}^{k} I(\mathbf{b}_i^{\top}(\widehat{\mathbf{b}}_l - \widehat{\mathbf{b}}_j) \geq 0) \\ &\approx \sum_{l=1}^{k} g(\mathbf{b}_i^{\top}(\widehat{\mathbf{b}}_l - \widehat{\mathbf{b}}_j)),\end{aligned}$$

g() is the sigmoid function to relax the indicator function I(), it also considers the word embedding for the labels.

(normalized) Discounted cumulative gain

Discounted Cumulative Gain [edit]

The premise of DCG is that highly relevant documents appearing lower in a search result list should be penalized as the graded relevance value is reduced logarithmically proportional to the position of the result. The discounted CG accumulated at a particular rank position p is defined as:[2]

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)}$$

Previously there has not been any theoretically sound justification for using a logarithmic reduction factor[3] other than the fact that it produces a smooth reduction. But Wang et al. (2013)[4] give theoretical guarantee for using the logarithmic reduction factor in NDCG. The authors show that for every pair of substantially different ranking functions, the NDCG can decide which one is better in a consistent manner.

An alternative formulation of DCG[5] places stronger emphasis on retrieving relevant documents:

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

The latter formula is commonly used in industry including major web search companies[2] and data science competition platform such as Kaggle.[6]

These two formulations of DCG are the same when the relevance values of documents are binary;[3]:320 $rel_i \in \{0, 1\}$.

From the CVPR'15 paper directly. Using NDCG as the metric. Z is the ideal DCG (IDCG), which has the highest score.

by hash functions, such as the Normalized Discounted Cumulative Gain (NDCG) score [10], which is a popular measure in the information retrieval community and defined as:

$$NDCG@p = \frac{1}{Z}\sum_{i=1}^{p}\frac{2^{r_i}-1}{\log(1+i)}, \qquad (2)$$

where p is the truncated position in a ranking list, Z is a normalization constant to ensure that the NDCG score for the correct ranking is one, and r_i is the similarity level of the i-th database point in the ranking list. Directly optimizing

Also mean average precision (mAP)

Precision [edit]

Main article: Precision and recall

Precision is the fraction of the documents retrieved that are relevant to the user's information need.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Recall [edit]

Main article: Precision and recall

Recall is the fraction of the documents that are relevant to the query that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

F1 score:

$$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{(\text{precision} + \text{recall})}$$

Mean average precision [edit]

Mean average precision for a set of queries is the mean of the average precision scores for each query.

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q} \mathrm{AveP(q)}}{Q}$$

where Q is the number of queries.

Average precision [edit]

Precision and recall are single-value metrics based on the whole list of documents returned by the system. For systems that return a ranked sequence of documents, it is desirable to also consider the order in which the returned documents are presented. By computing a precision and recall at every position in the ranked sequence of documents, one can plot a precision-recall curve, plotting precision $p(r)$ as a function of recall r. Average precision computes the average value of $p(r)$ over the interval from $r = 0$ to $r = 1$:[9]

$$\mathrm{AveP} = \int_0^1 p(r)dr$$

That is the area under the precision-recall curve. This integral is in practice replaced with a finite sum over every position in the ranked sequence of documents:

$$\mathrm{AveP} = \sum_{k=1}^{n} P(k)\Delta r(k)$$

where k is the rank in the sequence of retrieved documents, n is the number of retrieved documents, $P(k)$ is the precision at cut-off k in the list, and $\Delta r(k)$ is the change in recall from items $k-1$ to k.[9]

This finite sum is equivalent to:

$$\mathrm{AveP} = \frac{\sum_{k=1}^{n} (P(k) \times \mathrm{rel}(k))}{\text{number of relevant documents}}$$

Social recommendation in FB

Similar to simply posting and commenting, no extra mechanism to make a recommendation. Understand the text and extract the most likely places. Also ensure the feed story remains consistent as the story dynamically updates.
The challenge of recommendation extraction is to determine – without additional input from the commenter – that the recommended place is xxx. Using conversational understanding (CU) to identify relevant post and find the corresponding text in the comments that contains the recommendation.

CU feed the initial query from posts to local places search engine (FBNY), which merges user-specific info from commenter's social graph (check-in history, post location, etc.) with places graph to produce the most likely place matches. Resolve a place name specific to the area, insert an attachment to the comment.

A single recommendation post can appear in multiple surfaces (e.g., News Feed and Timeline). Within our current architecture, each surface has a separate model for that single post, and simple story updates (e.g., Likes and comment counts) are propagated across all views. Recommendations posts, however, introduce an additional level of complexity that needed to be managed programmatically.
Update UI and data model that backs the UI
Fetch little data
GraphQL only good at updating scalar object, need to update more complex objects.

!!! To maintain the consistency of the complex dynamic story updates that the new type of post requires, we manually managed the caching of recommendations in both Android and iOS versions of the app, separately, across each view. While we realized that it was redundant to have code on the client that performs essentially the same model manipulations that occurred on the server, we believed that making sure things look consistent across multiple locations and reducing the amount of data we consume was a worthwhile tradeoff that enabled us to release Recommendations as quickly as possible.

Recommendation vs check-ins. Check-ins are based on coordinates (latitude and longitude to the center of the desired city). Recommendation is uniformly distributed over the entire city: tweaking distance to allow for equal weighting throughout the city.

Do not want to surface poor matches. Implement a suppression layer in backend based on the query string, the distance of the place from the requestor, and relative score to other places

Building scalable systems to understand content
FBLearner Flow: general purpose ML experiment platform. Lumos: built on top of FBLearner Flow, for image and video understanding. Active language for accessibility: automatic alt text (AAT) for photos, for visually impaired people. Human annotations are involved.

Faiss: A library for efficient similarity search
- Quick search for multimedia documents that are similar to each other

- Fastest k-selection alg
- k-nearest-neighbor graph constructed on 1 billion high-dimensional vectors

Building Express Backbone: Facebook's new long-haul network
Before, data centers were connected by a single wide-area (WAN) backbone network known as the classic backbone (CBB), carrying both user (egress) traffic and internal server-to-server traffic.
Machine-to-machine traffic increases and may interfere with and impact the regular user traffic, affecting reliability goals.
How to solve? Split the cross-data center vs Internet-facing traffic into different networks, Express Backbone (EBB)

You are given a large set of transcripts, use them find the best student

1. **Problem formulation**
* can this problem be formulated as classification? Regression? (unsupervised) clustering? (if have both labeled and unlabeled data) semi-supervised?
Given the transcript, the information is only about students, the course they took and corresponding grades. Students can have different backgrounds (different majors). Not all students took all the or the same classes, some took more, some took fewer; This could be considered as a ranking problem. And the best student is top-1 student. (there might be more than one possible solutions)

2. **Data and features**
* what kind of data is provided? In what form? What useful features do you think you can extract? How would you extract the features?
since we only have grades of classes, averaging them and simply ranking them probably would be fair due to the reasons above. **Using graph**!
Express student/transcript data as a directed graph. Each vertex is a student. If A and B took the same class, and A achieved higher grade, add an edge from B to A (B, A), implying "B endorses A's academic performance". The graph can be weighted as well. The weight can be the score ratio, the bigger the ratio, the stronger the endorsement. Can also consider the average grade for that class, the lower the stronger the endorsement. The size of the class can also be considered.

3. **Model**
* what models do you think are fit to solve the problem? Why? Compare several candidates. What are the input and output of the model?
With graph as the input of the model, we can use PageRank, to find the student with highest rank, just like ranking webpages. (PageRank Wikipedia)

In the general case, the PageRank value for any page **u** can be expressed as:

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)},$$

i.e. the PageRank value for a page **u** is dependent on the PageRank values for each page **v** contained in the set $\mathbf{B_u}$ (the set containing all pages linking to page **u**), divided by the number $L(v)$ of links from page **v**.

if considering damping factor d (an imaginary surfer who is randomly clicking on links will eventually stop clicking. The probability, at any step, that the person will continue is a damping factor d),

So, the equation is as follows:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

where $p_1, p_2, \ldots, p_N$ are the pages under consideration, $M(p_i)$ is the set of pages that link to p_i, $L(p_j)$ is the number of outbound links on page p_j, and N is the total number of pages.

The PageRank values are the entries of the dominant right eigenvector of the modified adjacency matrix. This makes PageRank a particularly elegant metric: the eigenvector is

$$\mathbf{R} = \begin{bmatrix} PR(p_1) \\ PR(p_2) \\ \vdots \\ PR(p_N) \end{bmatrix}$$

where **R** is the solution of the equation

$$\mathbf{R} = \begin{bmatrix} (1-d)/N \\ (1-d)/N \\ \vdots \\ (1-d)/N \end{bmatrix} + d \begin{bmatrix} \ell(p_1,p_1) & \ell(p_1,p_2) & \cdots & \ell(p_1,p_N) \\ \ell(p_2,p_1) & \ddots & & \vdots \\ \vdots & & \ell(p_i,p_j) & \\ \ell(p_N,p_1) & \cdots & & \ell(p_N,p_N) \end{bmatrix} \mathbf{R}$$

where the adjacency function $\ell(p_i, p_j)$ is 0 if page p_j does not link to p_i, and normalized such that, for each j

$$\sum_{i=1}^{N} \ell(p_i, p_j) = 1,$$

i.e. the elements of each column sum up to 1, so the matrix is a stochastic matrix (for more details see the computation section below). Thus this is a variant of the eigenvector centrality measure used commonly in network analysis.

4. **Train and test**

* how do you train your model if necessary? what evaluation metrics do you think is meaningful to assess the performance of the model?

Accuracy/precision/recall/mAP/F1 score/ cross-validation

Manually run some test cases to show it makes sense. Time complexity is O(m+n), m is the number of nodes, n is the number of edges (?).

Can use hierarchy to treat local network as a big node (?) specifically, each major might be a big node.

Use Hadoop MapReduce

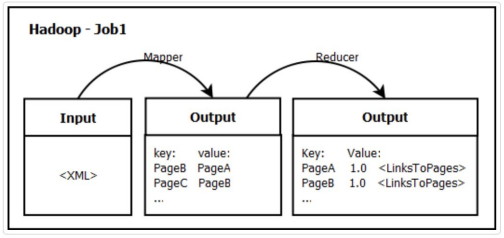
3 steps: parsing, calculating, ordering

Parse the big wiki xml into articles in Hadoop Job 1.

In the Hadoop mapping phase, get the article's name and its outgoing links.

In the Hadoop reduce phase, get for each wikipage the links to other pages.

Store the page, initial rank and outgoing links.



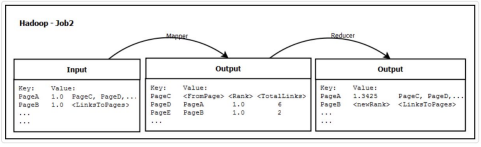
Hadoop Job 2 will calculate the new pageRank.

In the mapping phase, map each outgoing link to the page with its rank and total outgoing links.

In the reduce phase calculate the new page rank for the pages.

Store the page, new rank and outgoing links.

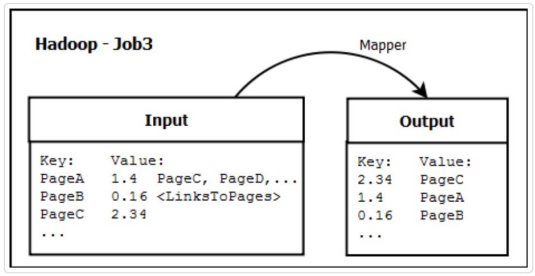
Repeat these steps for more accurate results.



Hadoop Job 3 will map the rank and page

Store the rank and page (ordered on rank)

See the top 10 pages!

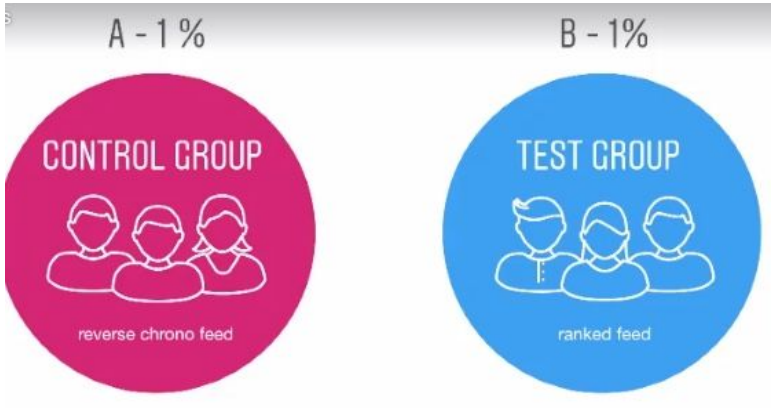


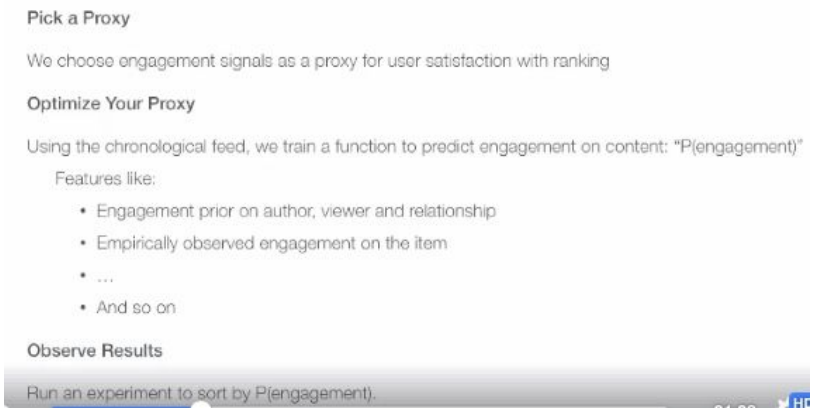
5. **Refinement and improvement**

* what are the potential problems with the selected model? How to improve/solve them? Overfit? Bias vs. variance. Learning to rank?

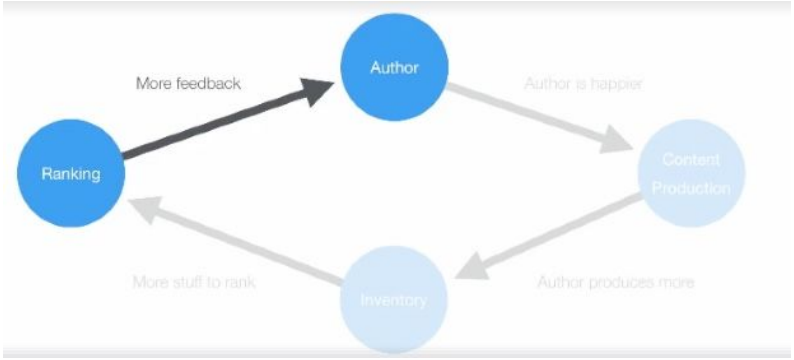
Instagram feed ranking

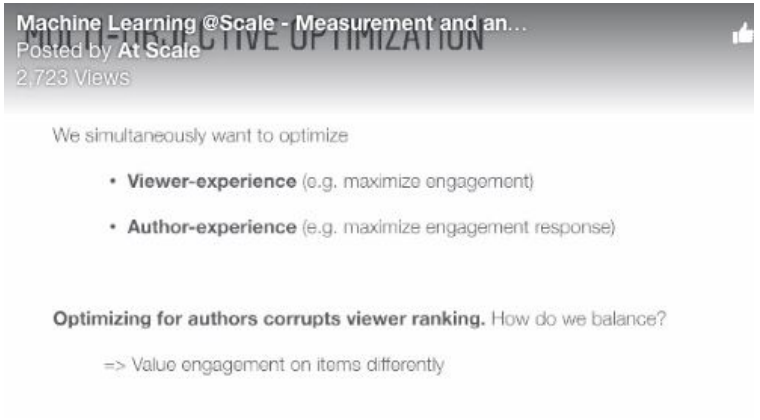
Chronological blocks (ie., 11am block, 1pm block…). Rank within the block: 'interesting' posts ranked higher.

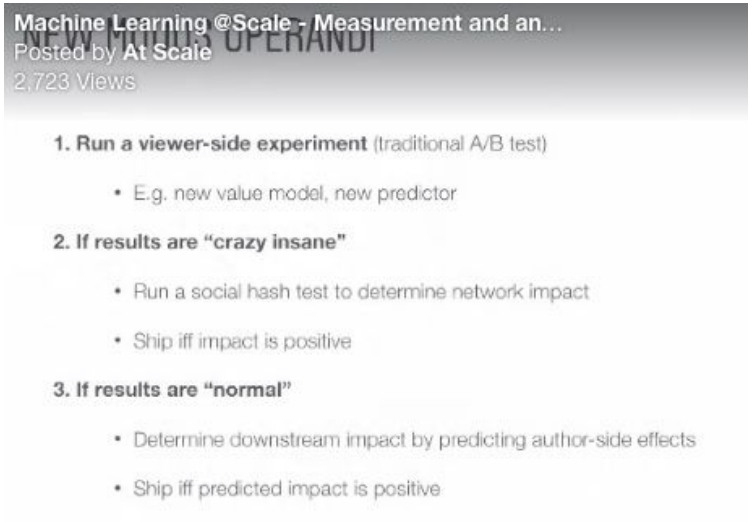


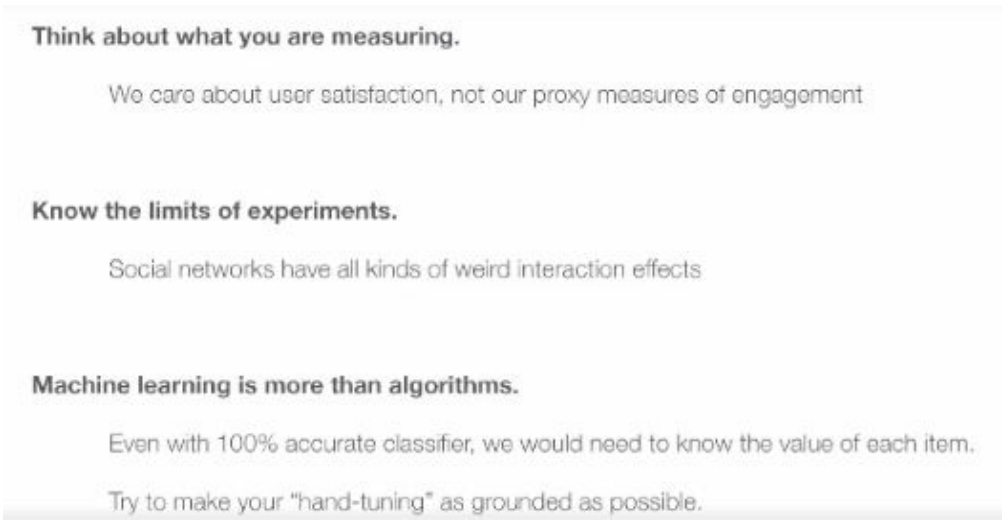


Ranking in social media creates an inconvenient loop:

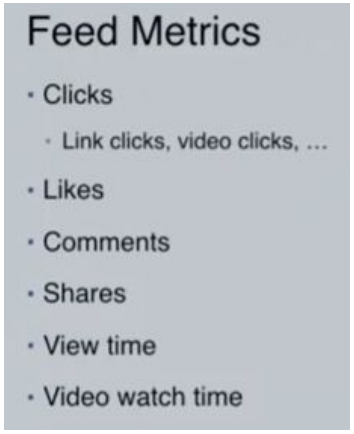


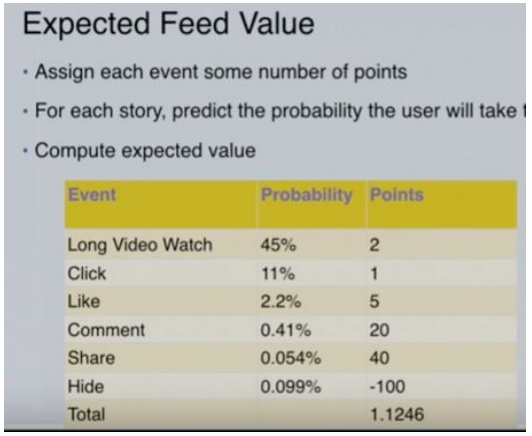


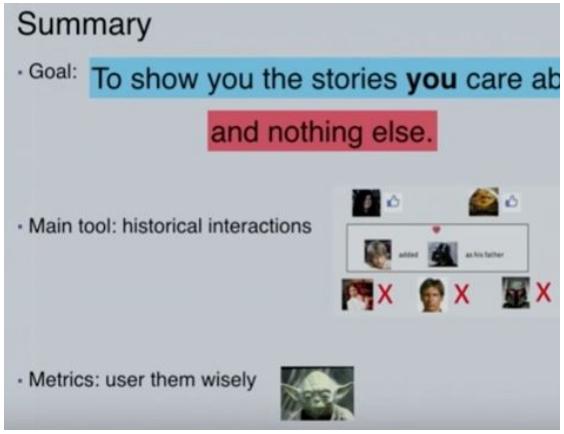




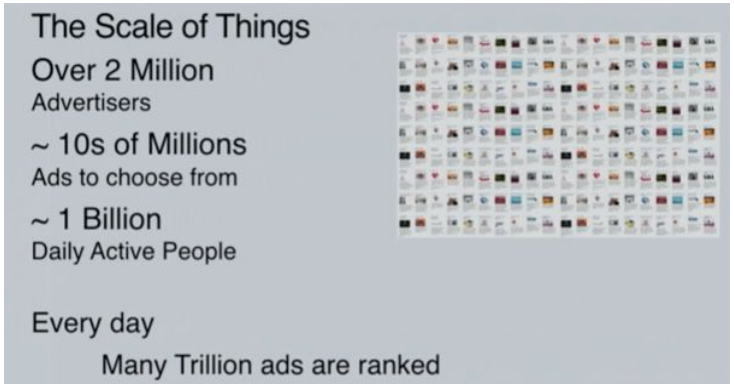
FB newsfeed ranking

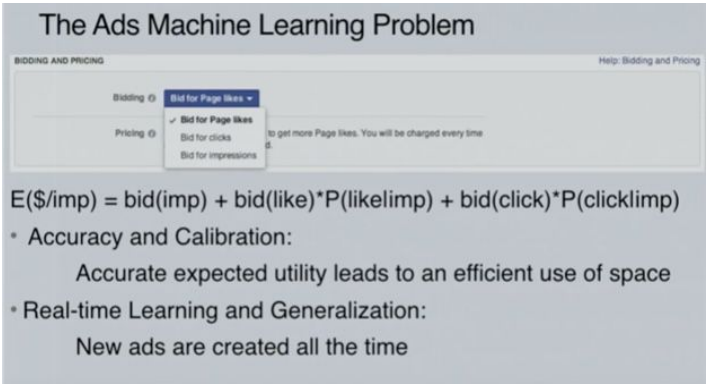




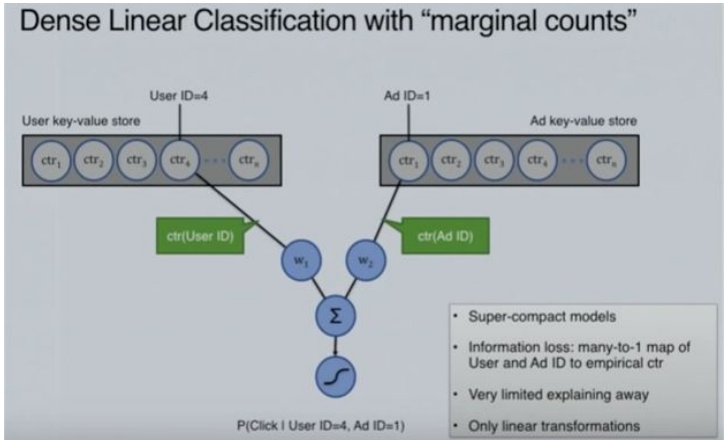


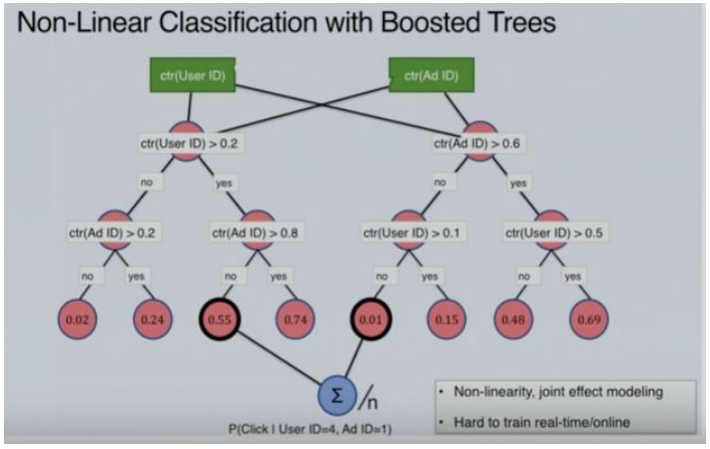
FB advertising



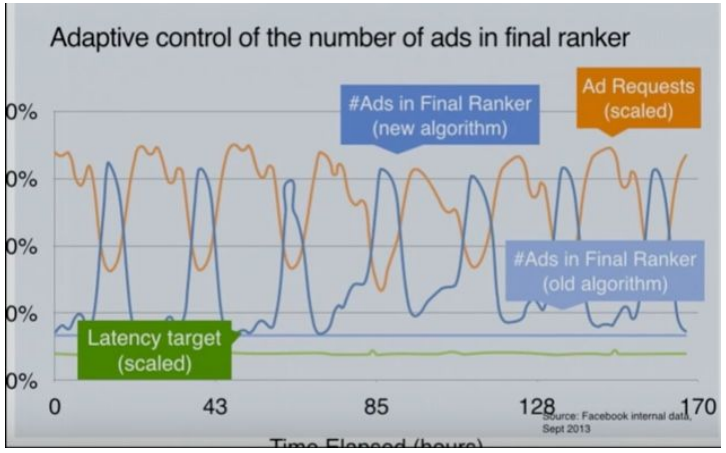


Must be fast. Starts with simple model: logistic regression, then tried boosted trees with logistic regression.
No theory background, but able to try different things quickly and find a good one.



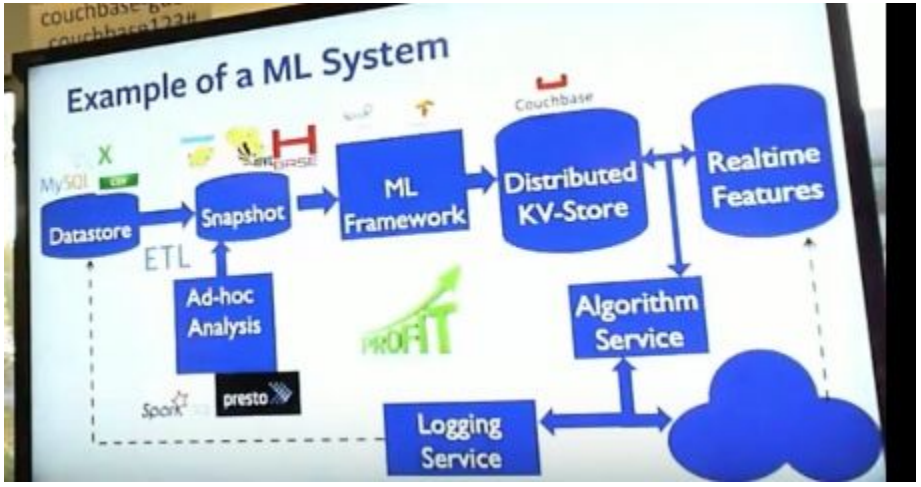


Efficient ranking: compute ranking at valley time (ie., rank ads when there are fewer ranking requests at night)



FB ML video [link]:

Data: more data the better. How to store/retrieve efficiently.
Algorithm: distributed. Eg., mapReduce. First have code run on one machine. Then use a model to distribute work using driver-worker.
Fault tolerant: ML model runs in iteration, what if at certain stage data is gone? Build model that is resilient to fault. Ie., Hadoop(HDFS). Spark: RDD(resilient distributed dataset).
Deployment: maintain, update. How to estimate the model performance in scale (ie., country test…)



Snapshot: daily snapshot(eg., happens at night). Used to evaluate deployed model.
Distributed KV-store: store trained models, retrieve them fast.
Realtime features: most features can be obtained from snapshot. However, certain cases require realtime features, such as ‘what is trending’. So at deployment, keep the realtime feature counter somewhere, and given a new query(task), combine trained model with realtime features and produce result.
Logging service: store results and send them back to datastore.

When to retrain model:
Ads: retrain frequently.
CTR: can be divided into different sub-tasks, thus could consist of different sub-modules, eg., user modeling. The sub-modules should be separated and retrain depending on need.

In practice: keep track of performance(historic models) before the change.
Lessons learned from practice:

-Keep track of interactions between sub-modules.
-Run A/B test; bucket test(past model vs updated model)
-Run A/A test! Ie., observe the performance change when the model is not changing!
-Ad-hoc analysis. Realtime feedback.(FB uses Spark and Pesto)
-Adding/validating features.
-Gap between online/offline metrics. Model is trained on sample data. Always a gap between offline/online data.

Missing data:
-If data size is large, can ignore.
-Make missing data a new categorical feature.
-If linear, take mean to fill in…