FMv2++

Goals

1. Recommend resources to users based on user-history and resource-properties
2. Quality must be robust to low data-volume for recommendable candidates, including:
    a. Customers with strict business rules (i.e. "only recommend things published in the past 3 days")
    b. Customers with sparse viewership
3. Quality must be robust to integration issues
    a. Issues around identity should not impact the recommendations of correctly-identified users
4. Quality should be consistent across users of varying history lengths
    a. Recommendation quality for *unidentified* users should be as good or better than the popular baseline
    b. Recommendation quality should increase with a user's history-length (assuming the user has been properly identified)

Challenges + Assumptions

Sparse Viewership

The most common approach to the recommendations problem is to build separate models for each recommendable entity, and score them independently. While this works reasonably well for vast datasets, it breaks down for sites with low-to-moderate traffic (i.e. all of our customers).

- Requires N parameters for every recommendable item, resulting in O(n^2) parameters
- Shared features across items are learned independently
- Scores coming from each model are at best difficult to compare, and at worst completely unrelated

To avoid this, we've opted to use Factorization Machines to produce our recommendations. In an FM, every feature has a bias term (equivalent to a simple logistic-regression parameter), and an interaction term. The score for a given (user, item) pair is the sum of bias-scalars and interaction-scalars for each *pair* of non-zero features.

As such, we train a *single* model, where each candidate includes one or more interaction features, allowing us to pose our problem as "score = f(user, item)", rather than "score = f(user | item)". Because the interaction vectors are shared across all training samples, the overall model complexity reduces to O(n*k) (where the hyper-parameter k represents the embedding rank).

Implicit Feedback

Like the models that came before it, FMv2 is trained to predict "viewed" events. This decision is mostly around convenience - we get ample "viewed" events from all of our customers and they are simple to interpret.

The implicit assumption here is that people want to see more of what they look at, and less of what they don't. Of course, this is not always the case - people may view content they don't enjoy, only to leave shortly afterwards. Similarly, if *not* viewing an item implies disinterest, then how can we make recommendations at all?

In reality, relevance is not a 1/0 target - rather than saying an item is or isn't a good recommendation, we use BPR Loss to train our models. Instead of learning whether a recommendation is good or bad, FMv2 attempts to learn which of 2 candidates is *better*. This gives the model room to learn how to sort recommendations by relevance. By learning which of two items is "better", we never have to explicitly penalize our model for trying to recommend something a user has not yet viewed. Similarly, there is room for the model to learn that some viewed items better reflect a user's tastes than others.

When compared to standard negative-sampling techniques using sigmoid loss, we found that using BPR loss greatly improves both absolute and top-k accuracy.

Cold-Start

This is a special-case of the sparse-viewership problem where the sparsity of views is tightly correlated with the recency of recommendable items.

Because recent items have lower view-counts than older articles, models trained to recommend specific articles will learn that these are not worth recommending. For customers with strict recency-rules, the bias towards older content decreases the relevance of many recommendations *and* s ubverts the intent of said recency restrictions.

Possible solutions include:

- Increase regularization coefficients for bias terms
    - Pros - easy to implement
    - Cons - extremely sensitive to hyper-parameters; requires regular per-customer tuning; interaction vectors trained on very little data

- Re-weight training samples to increase "importance" of new content
  - Pros - many resources regarding best practices (e.g. bootstrap resampling)
  - Cons - hardest to implement; interaction vectors overfit to the very few training samples we see
- **Recommend content features *instead of* producing embeddings for each item** ⭐
  - Pros - interaction vectors for content-features trained on the entire dataset; no per-item bias terms; allows us to easily add new features
  - Cons - longer training time; more preprocessing needed; requires thoughtful feature-engineering

Mangled Identity Resolution

We have found that our identity-resolution logic frequently associates multiple "humans" with the same identity, such that individual BSINs can have thousands of viewed events on a given day. FMv2 addresses these issues by using a number of heuristics:

1. Remove BSINs having:
   a. large numbers of views/day (addresses mangled identity, bot traffic)
   b. large numbers of sessions/day (addresses mangled identity)
   c. events coming from multiple devices over the course of a single session (addresses mangled identity)
   d. extremely long session lengths (addresses bot traffic)
2. Weight BSINs equally during training
   a. Users with large numbers of views have the same "importance" as those without (reduces the impact of mangled BSINs not detected during preprocessing)

Popular Baseline

Popularity works really, really well for a lot of customers - especially those with strict business rules. As such, it's an incredibly valuable feature to include in our models.

Because popularity is a function of time, including it in FMv2 impacted the model's implementation in the following ways:

1. Heterogeneous candidate sampling
   a. The common practice for most candidate-sampling strategies is to sample a single set of negative candidates to be contrasted with an entire batch of positive candidates. Because popularity values need to be looked up per (recommendation-time, candidate), this turned out to be more complicated than it was worth.
   b. Sampling sets of negative candidates independently for each sample allows us to sample negatives according to their popularity at recommendation-time (very similar to how candidates are produced as inference time)
2. Learned discretization of continuous features
   a. The importance of popularity as a feature is extremely non-linear. What is considered 'popular' vs 'maybe-popular' vs 'not-popular' is heavily based on overall traffic-patterns for each customer. Rather than choosing these cutoffs on a per-customer basis, FMv2 learns these boundaries by passing "log(popularity_counts)" through a 1-by-K neural layer with sigmoid optimization (where K is a hyper-parameter governing the number of popularity "buckets").
3. Recency features:
   a. The same bucketization technique used for popularity was also used for item recency (i.e. "sample.rec_time - candidate.pubDate")
   b. Including recency enables models to learn where lower-popularity might be a symptom of the cold-start problem

Future Projects

Feedback

As mentioned in Implicit Feedback, a "viewed" event does not necessarily imply that a user *enjoyed* the content they saw, or that the content will encourage them to continue browsing that site. To bring our models to the next level, we need more concrete metrics to optimize against.

Both "liked" and "shared" are very obvious signs that a person wants to see more similar content. While it is safe to assume that predicting likes is better than predicting views, "liked" events occur rarely and only for a small set of sites.

One option is to find a site which sends us "liked" events, and work with them to send us rich metadata in "viewed" events (i.e. time-on-page, scroll-activity). By finding features which correlate strongly with likes, we may be able to enrich our "viewed" events with "p(like|user,item)". Better yet, we could eventually extend these insights to sites that don't offer "liked" events at all.

Once we can both "p(like)" and "p(view)", introducing serendipity to our recommendations becomes a much more tractable problem - we can recommend that items that people are least likely to view on their own, but most likely to enjoy. How this would be implemented is an area for future research.

Neural Networks

One of the limitations of Factorization Machines is that they do not easily allow for "composite" features (user A viewed item X with title Y at time Z). While research into tensor-analogues of factorization machines has been done, the implementations are far more complex and require much more careful tuning.

Given the sequential nature of our data, I feel that an RNN would be a far better way of processing events. In my opinion, sequence and timing are the two most important features that we do not yet make use of.

In terms of sequence: a person who views "pants pants pants pants shirt shirt shirt belt" has probably found the pants+shirt they want, and are instead looking for a belt or other accessory. On the other hand, a person who views "pants shirt belt shirt pants shirt shirt" could be looking for any of the above. An LSTM-based network would be a convenient way to explicitly learn this.

Timing is a similar issue: a person who views "playstation playstation playstation nintendo" is probably *mostly* interested in playstation... However, a person who views "playstation playstation playstation (no activity for a month) nintendo" may have just bought a switch!

Though the examples above are a bit contrived, they highlight the expressive power of RNNs that other classes of models lack. By treating data as a sequence of structured features, each event we see can be accompanied by its own context, which differs from the context of other events. We may find that users who look at older articles are less picky about recency; or that the types of items a person looks at are different on their phone, laptop, or desktop; we might even find that the items a user views in the morning are better indicators of what they'll view later than the items they view at night. Long story short - we have many interesting features to consider that our current models are incapable of leveraging.