INSAID

THE WONDERFUL WORLD OF RECOMMENDER SYSTEMS

I recently gave a talk about recommender systems at the (http://www.meetup.com/Data-Science-Sydney/) (the slides are available (http://yanirs.github.io/talks/the-wonderful-world-of-recommender-systems)). This post roughly follows the outline of the talk, expanding on some of the key points in non-slide form (i.e., complete sentences and paragraphs!). The first few sections give a broad overview of the field and the common recommendation paradigms, while the final part is dedicated to debunking five common myths about recommender systems.

Motivation: Why should we care about recommender systems?

The key reason why many people seem to care about recommender systems is *money*. For companies such as Amazon, Netflix, and Spotify, recommender systems drive significant engagement and revenue. But this is the more cynical view of things. The reason these companies (and others) see increased revenue is because they deliver actual *value* to their customers – recommender systems provide a scalable way of personalising content for users in scenarios with many items.

Another reason why data scientists specifically should care about recommender systems is that it is a true data science problem. That is, at least according to my favourite definition of data science as the intersection between software engineering, machine learning, and statistics. As we will see, building successful recommender systems requires all of these skills (and more).

Defining recommender systems

When trying to the define anything, a reasonable first step is to ask Wikipedia. Unfortunately, as of the day of this post's publication, Wikipedia defines recommender systems too narrowly, as "a subclass of information filtering system that seek to predict the 'rating' or 'preference' that a user would give to an item" (I should probably fix it, but this wrong definition helped my talk flow better – let me know if you fix it and I'll update this paragraph).

The problem with Wikipedia's definition is that there's so much more to recommender systems than rating prediction. First, *recommender* is a misnomer – calling it a discovery assistant is better, as the so-called recommendations are far from binding. Second, *system* means that elements like presentation are important, which is part of what makes recommendation such an interesting data science problem.

My definition is simply:

Recommender systems are systems that help users discover items they may like.

Recommendation paradigms

Depending on who you ask, there are between two and twenty different recommendation paradigms. The usual classification is by the type of data that is used to generate recommendations. The distinction between approaches is more academic than practical, as it is often a good idea to use hybrids/ensembles to address each method's limitations. Nonetheless, it is worthwhile discussing the different paradigms. The way I see it, if you ignore trivial approaches that often work surprisingly well (e.g., popular items, and "watch it again"), there are four main paradigms: collaborative filtering, content-based, social/demographic, and contextual recommendation.

Collaborative filtering is perhaps the most famous approach to recommendation, to the point that it is sometimes seen as synonymous with the field. The main idea is that you're given a matrix of preferences by users for items, and these are used to predict missing preferences and recommend items with high predictions. One of the key advantages of this approach is that there has been a huge amount of research into collaborative filtering, making it pretty well-understood, with existing libraries that make implementation fairly straightforward. Another important advantage is that collaborative filtering is independent of item properties. All you need to get started is user and item IDs, and some notion of preference by users for items (ratings, views, etc.).

The major limitation of collaborative filtering is its reliance on preferences. In a cold-start scenario, where there are no preferences at all, it can't generate any recommendations. However, cold starts can also occur when there are millions of available preferences, because pure collaborative recommendation doesn't work for items or users with no ratings, and often performs pretty poorly when there are only a few ratings. Further, the underlying collaborative model may yield disappointing results when the preference matrix is sparse. In fact, this has been my experience in nearly every situation where I deployed collaborative filtering. It always requires tweaking, and never simply works out of the box.

Content-based algorithms are given user preferences for items, and recommend similar items based on a domain-specific notion of item content. The main advantage of content-based recommendation over collaborative filtering is that it doesn't require as much user feedback to get going. Even one known user preference can yield many good recommendations (which can lead to the collection of preferences to enable collaborative recommendation). In many scenarios, content-based recommendation is the most natural approach. For example, when recommending news articles or blog posts, it's natural to compare the textual content of the items. This approach also extends naturally to cases where item metadata is available (e.g., movie stars, book authors, and music genres).

One problem with deploying content-based recommendations arises when item similarity is not so easily defined. However, even when it is natural to measure similarity, content-based recommendations may end up being too homogeneous to be useful. Such recommendations may also be too static over time, thereby failing to adjust to changes in individual user tastes and other shifts in the underlying data.

Social and demographic recommenders suggest items that are liked by friends, friends of friends, and demographically-similar people. Such recommenders don't need any preferences by the user to whom recommendations are made, making them very powerful. In my experience, even trivially-implemented approaches can be depressingly accurate. For example, just summing the number of Facebook likes by a person's close friends can often be enough to paint a pretty accurate picture of what that person likes.

Given this power of social and demographic recommenders, it isn't surprising that social networks don't easily give their data away. This means that for many practitioners, employing social/demographic recommendation algorithms is simply impossible. However, even when such data is available, it is not always easy to use without creeping users out. Further, privacy concerns need to be carefully addressed to ensure that users are comfortable with using the system.

Contextual recommendation algorithms recommend items that match the user's current context. This allows them to be more flexible and adaptive to current user needs than methods that ignore context (essentially giving the same weight to all of the user's history). Hence, contextual algorithms are more likely to elicit a response than approaches that are based only on historical data.

The key limitations of contextual recommenders are similar to those of social and demographic recommenders – contextual data may not always be available, and there's a risk of creeping out the user. For example, ad retargeting can be seen as a form of contextual recommendation that follows users around the web and across devices, without having the explicit consent of the users to being tracked in this manner.

Five common myths about recommender systems

There are some common myths and misconceptions surrounding recommender systems. I've picked five to address in this post. If you disagree, agree, or have more to add, I would love to hear from you either privately or in the comment section.

The accuracy myth Offline optimisation of an accuracy measure is sufficient for creating a successful recommender **Reality** Users don't really care about accuracy

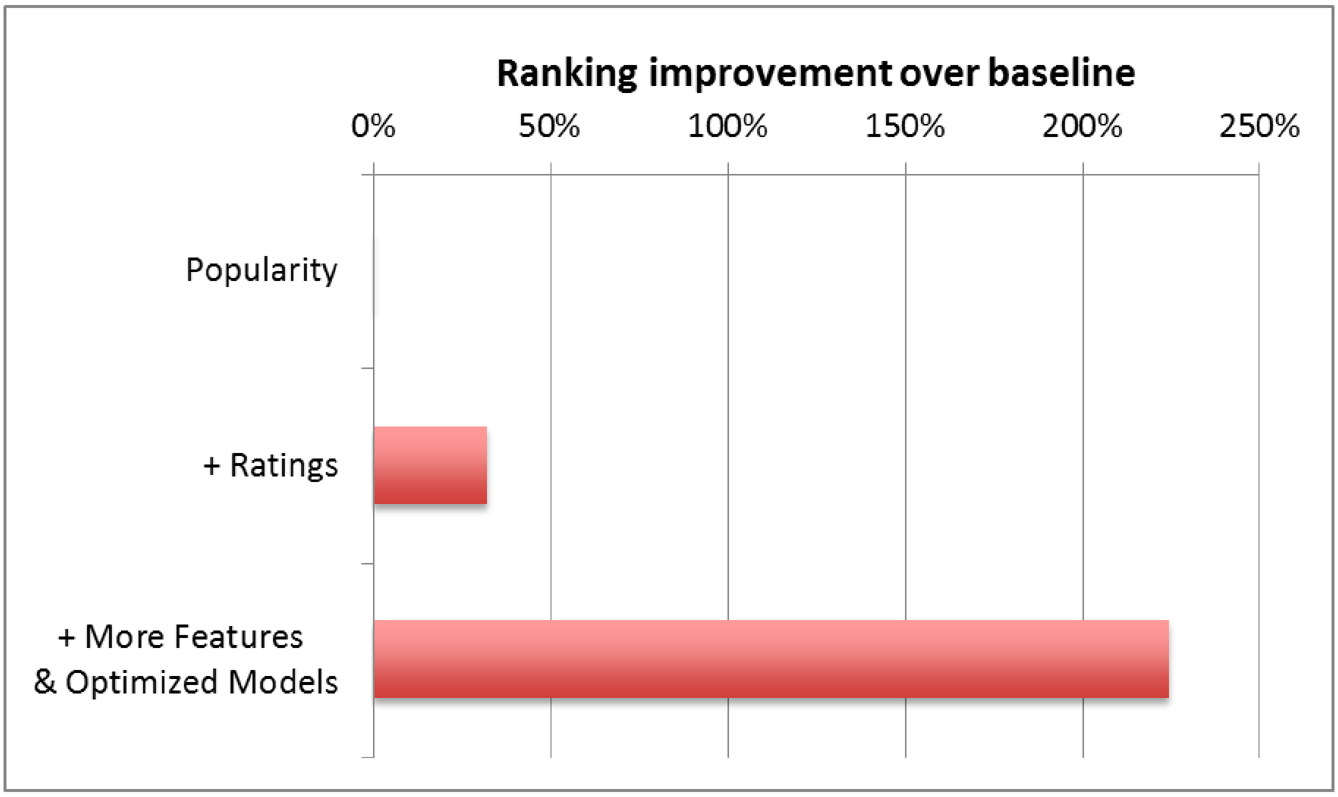
This is perhaps the most prevalent myth of all, as evidenced by Wikipedia's definition of recommender systems. It's somewhat surprising that it still persists, as it's been almost ten years since McNee et al.'s influential paper on the damage the focus on accuracy measures has done to the field.

It is therefore worth asking where this myth came from. My theory is that it is a feedback loop between academia and industry. In academia it is pretty easy to publish papers with infinitesimal improvements to arbitrary accuracy measures on offline datasets (I'm also guilty of doing just that), while it's relatively hard to run experiments on live systems. However, one of the moves that significantly increased focus on offline predictive accuracy came from industry, in the form of the $1M Netflix prize, where the goal was to improve the accuracy of Netflix's rating prediction algorithm by 10%.

Notably, most of the algorithms that came out of the three-year competition were never integrated into Netflix. As discussed on the Netflix blog:

> You might be wondering what happened with the final Grand Prize ensemble that won the $1M two years later... We evaluated some of the new methods offline but the additional accuracy gains that we measured did not seem to justify the engineering effort needed to bring them into a production environment.
>
> Our business objective is to maximize member satisfaction and month-to-month subscription retention... Now it is clear that the Netflix Prize objective, accurate prediction of a movie's rating, is just one of the many components of an effective recommendation system that optimizes our members' enjoyment.

The following chart says it all (taken from the second part of the blog post quoted above):



An important question that arises is: If users don't really care about predictive accuracy, what do they care about? The answer is that predictive accuracy has some importance (as evidenced by the above chart), but it is not the only thing. In my opinion, the key consideration is UI/UX. You can have the most accurate recommendations in the world, but no one would know about it (or care) if they are not served in a timely manner through a friendly interface.

Of course, even with a great user interface and accurate predictions, there are other issues that require attention when designing recommender systems. Examples include diversity (showing various types of items), serendipity/novelty (showing non-obvious recommendations that users don't already know about), and coverage (being able to generate recommendations for all users and items). Many other considerations are covered in an excellent survey by Guy Shani and Asela Gunawardana.

It's also worth noting that there is an inherent problem with common accuracy measures. Specifically, when using a measure like root mean square error, a rating prediction algorithm can be made to perform better by reducing errors on low ratings. This is rather pointless, because items with low ratings will not be shown to users in any case.

Finally, a key issue that arises with offline evaluation is that there are biases in offline datasets that do not necessarily carry over to online scenarios. For instance, in many cases there is an implicit assumption that data is missing at random, when it really isn't, e.g., the fact that users took the effort to watch and rate a movie already tells us a lot about a bias they have towards this movie (the team that won the Netflix prize used this bias to their advantage). Hiding this rating and trying to predict it is not the same as predicting a rating for a movie that is picked at random from the entire set of movies.

The black box myth You can build successful recommender systems without worrying about what's being recommended and how recommendations are being served **Reality** UI/UX is king, item type is critical

A good recommender *system* has to consider how users interact with the recommendations. For example, the number of displayed recommendations should inform the optimisation procedure (e.g., are you aiming for precision@1 or precision@10?). How these recommendations are laid out (e.g., horizontally/vertically) tends to influence user interaction. In addition, being able to explain the reasons for the recommendations can yield easy wins. Finally, in many cases there are constraints on the amount of time that can be spent generating recommendations.

In addition to UI/UX, the design of good recommender *systems* has to account for what's being recommended. For example, music tracks and short videos can be played many times, so it's probably a good idea to recommend items that the user has already seen. On the other hand, items like washing machines and cars don't get consumed as often. If a user has just bought a washing machine, they're unlikely to want another one anytime soon (but they may want a dryer or a clothes line).

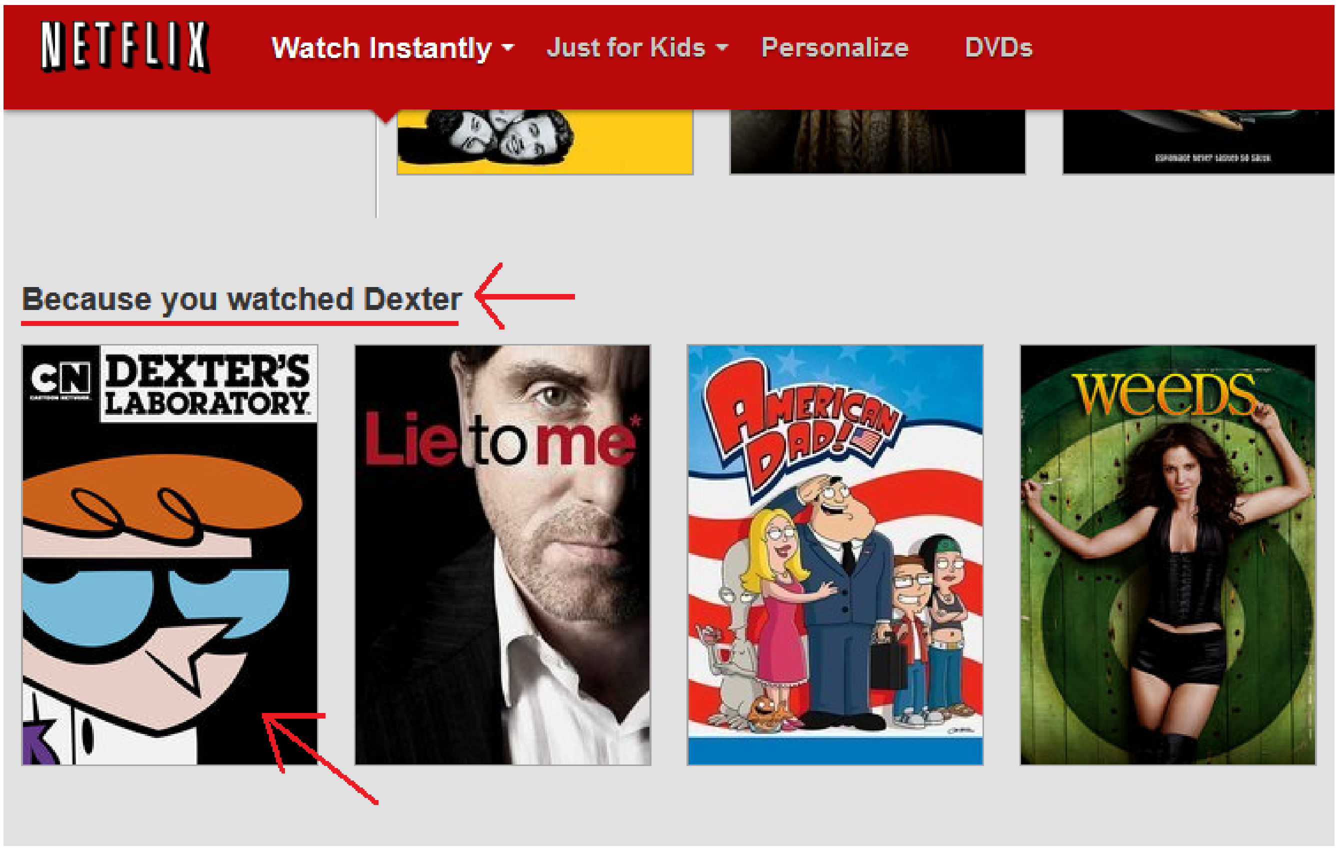
Hynt is a recommender-system-as-a-service for e-commerce whose development I led up until the middle of last year. The general idea is that merchants simply add a few lines of JavaScript to their shop pages and Hynt does the hard work of recommending relevant items from the store, while considering the user and page context. Going live with Hynt reaffirmed many well-known UI/UX lessons. Most notably:

- *Above the fold is better than below.* Engagement with Hynt widgets that were visible without scrolling was higher than those that were lower on the page.
- *More recommendations are better than a few.* Hynt widgets are responsive, adapting to the size of the container they're placed in. Engagement was more likely when more recommendations were displayed, because users were more likely to find something they liked without scrolling through the widget.
- *Fast is better than slow.* If recommendations load faster, more people see them, which increases engagement. In Hynt's case speed was especially important because the widgets load asynchronously after the host page finishes loading.

Another important UI/UX element is explanations. Displaying a plausible explanation next to a recommendation can do wonders, without making any changes to the underlying recommendation algorithms. The impact of explanations has been studied extensively by Nava Tintarev and Judith Masthoff. They have identified seven different aims of explanations, which are summarised in the following table (reproduced from their survey of explanations in recommender systems).

AIM	DEFINITION
Transparency	Explain how the system works
Scrutability	Allow users to tell the system it is wrong
Trust	Increase user confidence in the system
Effectiveness	Help users make good decisions
Persuasiveness	Convince users to try or buy
Efficiency	Help users make decisions faster
Satisfaction	Increase ease of usability or enjoyment

Explanations are ubiquitous in real-world recommender systems. For example, Amazon uses explanations like “frequently bought together”, and “customers who bought this item also bought”, while Netflix presents different lists of recommendations where each list is driven by a different reason. However, as the following Netflix example shows, it is worth making sure that the explanations you provide don’t make you look stupid.



The solved problem myth The space of recommender systems has been exhaustively explored **Reality** Development of new methods is often required

When I finished my PhD, about three years ago, I joined a small startup called Giveable as the first employee (essentially part of the founding team that was formed after Adam Neumann, the original founder, graduated from AngelCube and raised some seed funding). Giveable’s original product was a webapp where users could connect with their Facebook account and find gifts for their friends.

giveable

At the time, there wasn’t much published research on gift recommendation, and there was more or less nothing about the specific problem of recommending gifts for Facebook friends using liked pages. Here are some of the ways this problem differs from classic recommendation scenarios.

- *Need to consider giver and receiver.* Unlike traditional scenarios, the recommended items aren’t consumed by the user to whom they’re shown. In practice, this meant that we had to ensure the items are *giftable*, and take into account the relationship between the giver and the receiver. For example, the type of gift your mum may give you is different from gifts your partner may give you.
- *Likes are historical, sparse, and often nonsensical.* This is best illustrated by an example: What does liking a page such as Tony Abbott – Worst PM in Australian History tell us about gifts the user may like? Tony

Abbott is no longer prime minister (thankfully), so it's historical, and while this page is quite popular, there are many other pages out there that are difficult to interpret and are liked by only a handful of people (this video is a good summary of why Tony is disliked, for those who are unfamiliar with Australian politics).
- *Likes are not for recommended items.* As the above example shows, just because you like disliking Tony, it doesn't exactly lead to useful gifts. Even with things that are more related to interests, such as authors and bands, the liked pages aren't recommendable as gifts.
- *Likes are not always available offline.* This was an important engineering consideration: We didn't have much time to generate recommendations from the point where a new user gave us permission to view their likes and the likes of their friends. Ideally, recommendation generation would take less than a second from the time we got all the data from Facebook. This puts a strong constraint on the types of algorithms we could use.

The key to effectively addressing the Giveable recommendation problem was doing as much processing offline as possible. Specifically:

- Similar pages were inferred using Latent Dirichlet Allocation (which can be seen as a collaborative filtering technique). This made it possible to use information from pages that are not directly linked to giftable products, e.g., for the above Tony Abbott example, people who dislike him are likely to be left-leaning, which implies many other interests.
- Facebook pages were matched to giftable products with heuristics + Mechanical Turk + machine learning. This took a few iterations of what was essentially partly-manual semi-supervised learning, where we obtained high-confidence matches through heuristics and manual tagging, and then used this to train a classifier that was used to classify uncertain matches. The results of classification on a hold-out set were then verified through manual tagging of subsamples.
- We enriched the page and product data with structured information from the Freebase knowledge graph (which has since been deprecated). This allowed us to easily match giftable products to liked pages, e.g., books to authors.

The online part included taking a receiver's liked pages, inferring likes for similar pages, and matching all these pages to a ranked and diversified list of giftable product recommendations. These recommendations came with explanations, which were quite important in this case because the giver of a gift has to know why they're giving it.

The silver bullet myth Optimising a single measure or using a single algorithm is sufficient for generating a good recommendation list **Reality** Hybrids work best

Netflix provides another example for how focusing on a single algorithm or measure of success is far from sufficient. In a recent blog post, they talk about how they use multiple algorithms to optimise the order of different recommendation lists and each list's internal ranking, while considering device-specific UI constraints, relevance, engagement, diversity, business requirements, and more.

An example from my experience comes from Giveable (which ended up evolving into Hynt), where a single list was generated by mixing the outputs of the following recommendation approaches: contextual, direct likes, inferred likes, content-based, social, collaborative filtering of products, previously viewed items, and popular interests/products. The weight of each algorithm in the mix was static – it was either set manually or through A/B testing, and then left as a hardcoded constant.

This kind of static mix can get you very far, but there's a better way that I haven't gotten around to implementing before leaving to work on other things. This way is described in a series of posts on bandits for recommenders by Sergey Feldman of RichRelevance. The general idea is to train recommendation models offline using a small number of strategies/paradigms. Online, recommendations are served from strategies that maximise clickthrough and revenue, given a context of features that describe the user, merchant, and web page where the RichRelevance widget is embedded. Rather than setting static weights for the strategies, the bandit model continuously adjusts the weights, while balancing between exploring new strategy weights and exploiting strategies that have been known to work well in a specific context. This allows the overall recommendation engine to adjust to changes in reality and in the underlying data.

The omnipresence myth Every personalised system is a recommender system **Reality** This one is kinda true, but not necessarily useful…

The first conference I attended as a PhD student was the 18th International Conference on User Modeling, Adaptation and Personalization (UMAP), back in 2010. The field of recommender systems was getting increased attention, and Peter Brusilovsky, who has been working in the UMAP field for decades, argued that recommender systems are the new expert systems. This was partly because the hype was causing people to broaden the definition of the field to allow them to say that they're working on recommender systems.

I don't think it's incorrect that personalisation and recommender systems are different things. However, one problem that this may cause is making people think that common recommendation techniques would apply in scenarios where they're unlikely to work. For example, web search can be seen as a recommender system for pages that gives a high weight to the user's intent, as captured by the query. Hence, when personalising web search, it seems sensible to use collaborative filtering techniques. This was indeed my experience with the Yandex search personalisation competition: employing a matrix factorisation approach that was inspired by collaborative filtering turned out to be a waste of time compared to domain-specific methods.

In conclusion, recommenders are about as murky as data science. Just like data science, the boundaries of recommender systems are hard to define and they are sometimes over-hyped. This hype may lead to people investing in a recommender system they don't really need, just like the common issue of premature investment in data science. However, the hype is based on real value, which can definitely be delivered by recommender systems when they are used correctly.