Chapter 10. MLOps in Practice: Marketing Recommendation Engines

Makoto Miyazaki

Recommendation engines have become very popular in the last 20 years, from the first Amazon book recommendations to today's generalized use in digital shops, advertisements, and music and video streaming. We have all become accustomed to them. However, throughout the years, the underlying technologies behind these recommendation engines have evolved.

This chapter covers a use case that illustrates the adaption of and need for MLOps strategies given the particularities of a fast-paced and rapidly changing machine learning model life cycle.

The Rise of Recommendation Engines

Historically, marketing recommendations were human-built. Based on qualitative and quantitative marketing studies, marketing moguls would set up rules that statically defined the impression (in the sense of advertising views) sent to a customer with given characteristics. This technique gave rise to the marketing data mining urban legend that a grocery chain discovered that men who bought diapers on Thursdays and Saturdays were more likely to buy beer as well and hence placing the two next to each other will increase beer sales.

Overall, recommendation engines created manually presented numerous bottlenecks that resulted in a significant amount of wasted money: it was hard to build rules based on many different customer features because the rule creation process was manual, it was hard to set up experiments to test many different kinds of impressions, and it was hard to update the rules when the behavior of the customers changed.

The Role of Machine Learning

The rise of ML has brought a new paradigm to recommendation engines, allowing for the elimination of rules based on human insight. A new class of algorithm called *collaborative filtering* dominates the field. This algorithm is able to analyze customer and purchase data with millions of customers and tens of thousands of products to perform recommendations without any prior marketing knowledge. By identifying efficiently what customers that look like the current customer bought, marketers can rely on automatic strategies that outperform traditional ones both in cost and efficiency.

Because the process of building strategies is automatic, it is possible to update them regularly and to compare them using A/B testing or shadow scoring schemes (including the way to choose the impression among all possibilities). Note that these algorithms may be combined with more classical business rules for various reasons—e.g., avoiding the filtering bubble, not selling a product in a given geographical area, or preventing the use of a specific association that is statistically meaningful but unethical to use (like proposing alcohol to a recovering alcoholic), to name a few.

Push or Pull?

When implementing a recommendation engine, it is important to keep in mind that its structure will depend on whether the recommendations are pushed or pulled. Push channels are the easiest to handle; for example, they can consist of sending emails or making outbound calls.

The recommendation engine can be run on a regular basis in batch mode (typically between once a day and once a month), and it is easy to split the customer dataset into several parts to perform analysis within a sound experimental design. The regularity of the process allows for regular review and optimization of the strategy.

Pull channels are often more effective because they provide information to customers when they need it—for example, when doing an online search or when calling a customer service line. Specific information from the session can be used (e.g., what the user has searched for) to precisely tailor the recommendation. Music streaming platforms, for instance, provide pull-channel recommendations for playlists.

Recommendations can be prebaked, as illustrated in the in-depth example in this chapter, or made in real time. In the latter case, a special architecture has to be set up to compute recommendations on the fly.

Comparing strategies in a pull context is more challenging. First, the customers who call in on a given channel are likely to differ from the average customer. In simple cases, it is possible to randomly choose the strategy to use for each recommendation, but it also happens that the strategy needs to be used consistently over a given period for a given customer. This usually results in an unbalanced proportion of recommendations made with each strategy, which makes the statistical treatment to decide which one is the best more complex. However, once a good framework is set, this allows a very quick improvement cycle, as many strategies can be tested in real time.

Data Preparation

The customer data that is usually accessible to a recommendation engine is composed of the following:

- Structural information about the customer (e.g., age, gender, location)
- Information about historical activities (e.g., past views, purchases, searches)
- Current context (e.g., current search, viewed product)

Whatever the technique used, all customer information has to be aggregated into a vector (a list of fixed size) of characteristics. For example, from the historical activities, the following characteristics could be extracted:

- Amount of purchases during the last week or the last month
- Number of views during past periods

- Increase in spending or in views during the last month
- Previously seen impressions and customer's response

In addition to customer data, the recommendation context can also be taken into account. For example, days to summer for seasonal products like above-ground swimming pools or days to monthly pay day, as some customers delay purchases for cash flow reasons.

Once the customer and context data is formatted, it is important to define the set of possible recommendations, and there are many choices to make. The same product may be presented with different offers, which may be communicated in different ways.

It is of the utmost importance not to forget the "do not recommend anything" option. Indeed, most of us have the personal experience that not all recommendations have a positive impact. Sometimes not showing anything might be better than the alternatives. It's also important to consider that some customers may not be entitled to see certain recommendations, for example depending on their geographical origin.

Design and Manage Experiments

To leverage the continuous improvement potential of recommendation engines, it is necessary to experiment with different strategies within a sound framework. When designing a prediction model for a recommendation engine, the data scientist might well focus on a simple strategy, such as predicting the probability that a given customer clicks on a given recommendation.

This may seem a reasonable compromise compared to the more precise approach of trying to gather information about whether the customer purchased the product and whether to attribute this purchase to a given recommendation. However, this is not adequate from a business perspective, as phenomena like cannibalization may occur (i.e., by showing a low-margin product to a customer, one might prevent them from buying a high-margin product). As a result, even if the predictions were good and resulted in increased sales volume, the overall revenue might be reduced.

On the other hand, bluntly promoting the organization’s interest and not the customer’s could also have detrimental long-term consequences. The overarching KPI that is used to assess if a given strategy yields better results should be carefully chosen, together with the time period over which it is evaluated. Choosing the revenue over a two-week period after the recommendation as the main KPI is common practice.

To be as close as possible to an experimental setting, also called A/B testing, the control group and the experimental groups have to be carefully chosen. Ideally, the groups are defined before the start of the experiment by randomly splitting the customer base. If possible, customers should not have been involved in another experimentation recently so that their historical data is not polluted by its impact. However, this may not be possible in a pull setting in which many new customers are coming in. In this case, the assignment could be decided on the fly. The size of the groups as well as the duration of the experiments depend on the expected magnitude of the KPI difference between the two groups: the larger the expected effect, the smaller the group size and the shorter the duration.

The experimentation could also be done in two steps: in the first one, two groups of equal but limited size could be selected. If the experimentation is positive, the deployment could start with 10% on the new strategy, a proportion that can be raised progressively if results are in line with expectations.

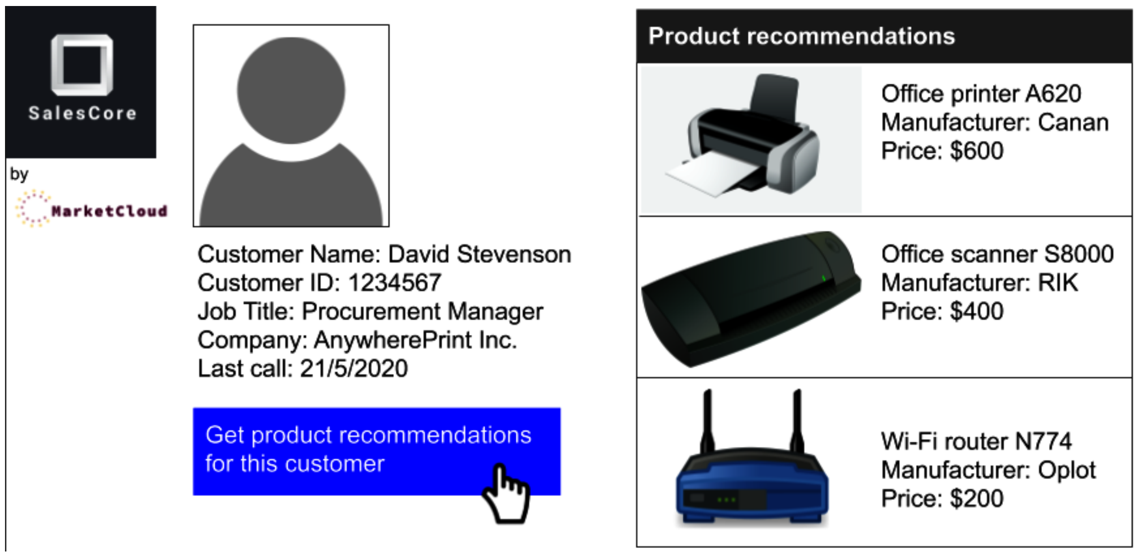
Model Training and Deployment

To better illustrate the MLOps process for this type of use case, the following sections focus on the specific example of a hypothetical company deploying an automated pipeline to train and deploy recommendation engines. The company is a global software company (let’s call it MarketCloud) headquartered in Silicon Valley.

One of MarketCloud’s products is a software-as-a-service (SaaS) platform called SalesCore. SalesCore is a B2B product that allows its users (businesses) to drive sales to customers in a simple manner by keeping track of deals, clearing tedious administration tasks off their desks, and creating customized product offers for each customer (see Figure 10-1).

From time to time, MarketCloud's clients use the cloud-based SalesCore while on a call with their customers, adjusting their sales strategy by looking at past interactions with the customers as well as at the product offers and discounts suggested by SalesCore.

MarketCloud is a mid-sized company with an annual revenue of around $200 million and a few thousand employees. From salespeople at a brewing company to those in a telecommunication entity, MarketCloud's clients represent a range of businesses.



Customer Name: David Stevenson
Customer ID: 1234567
Job Title: Procurement Manager
Company: AnywherePrint Inc.
Last call: 21/5/2020

Office printer A620
Manufacturer: Canan
Price: $600

Office scanner S8000
Manufacturer: RIK
Price: $400

Wi-Fi router N774
Manufacturer: Oplot
Price: $200

Figure 10-1. Mock-up of the SalesCore platform, the basis of the theoretical company on which this section's example is based

MarketCloud would like to automatically display product suggestions on SalesCore to the salespeople trying to sell products to the customers. Suggestions would be made based on customers' information and their past interaction records with the salesperson; suggestions would therefore be customized for each customer. In other words, SalesCore is based on a recommendation engine used in a pull (inbound calls) or push (outbound calls) context. Salespeople would be able to incorporate in their sales strategy the suggested products while on a call with their customers.

To implement this idea, MarketCloud needs to build a recommendation engine and embed it into SalesCore's platform, which, from a model training and deployment standpoint, presents several challenges. We'll present these challenges in this section, and in the next section we'll show MLOps strategies that allow the company to handle each of them.

Scalability and Customizability

MarketCloud's business model (selling software for client companies to help them sell their own products) presents an interesting situation. Each client company has its own dataset, mainly about its products and customers, and it doesn't wish to share the data with other companies.

If MarketCloud has around four thousand clients using SalesCore, that means instead of having a universal recommender system for all the clients, it would need to create four thousand different systems. MarketCloud needs to come up with a way to build four thousand recommendation systems as efficiently as possible since there is no way it can handcraft that many systems one by one.

Monitoring and Retraining Strategy

Each of the four thousand recommendation engines would be trained on the customer data of the corresponding client. Therefore, each of them would be a different model, yielding a different performance result and making it nearly impossible for the company to manually keep an eye on all four thousand. For example, the recommendation engine for client A in the beverage industry might consistently give good product suggestions, while the engine for client B in the telecommunication sector might seldom provide good suggestions. MarketCloud needed to come up with a way to automate the monitoring and the subsequent model retraining strategy in case the performance degraded.

Real-Time Scoring

In many situations, MarketCloud's clients use SalesCore when they are talking to their customers on the phone. The sales negotiation evolves every single minute during the call, and the salesperson needs to adjust the strategy during the interaction with the customer, so the recommendation engine has to be responsive to real-time requests.

For example, imagine you as a salesperson are on a call with your customer to sell telecommunication devices. The customer tells you what his office looks like, the existing infrastructure at the office such as an optic fiber, the type of WiFi network, and so forth. Upon entering this information in SalesCore, you want the platform to give you a suggestion for the products that your customer could feasibly purchase. This response from the platform needs to be in real time, not 10 minutes later, after the call, or on the following day.

Ability to Turn Recommendations On and Off

Responsible AI principles acknowledge that retaining human involvement is important. This can be done through a human-in-command design,[1] by which it should be possible *not* to use the AI. In addition, adoption is likely to be low if users cannot override AI recommendations. Some clients value using their own intuition about which products to recommend to their customers. For this reason, MarketCloud wants to give its clients full control to turn the recommendation engine on and off so that the clients can use the recommendations when they want.

Pipeline Structure and Deployment Strategy

To efficiently build four thousand recommendation engines, MarketCloud decided to make one data pipeline as a prototype and duplicate it four thousand times. This prototype pipeline consists of the necessary data preprocessing steps and a single recommendation engine, built on an example dataset. The algorithms used in the recommendation engines will be the same across all four thousand pipelines, but they will be trained with the specific data associated with each client (see Figure 10-2).

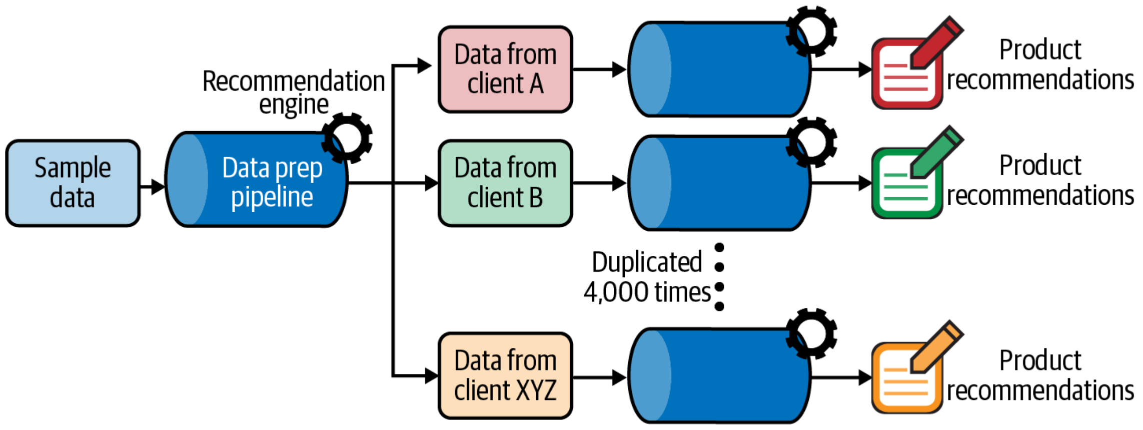


Figure 10-2. Image of data pipeline structure for MarketCloud's recommendation engine project

In this way, MarketCloud can efficiently launch four thousand recommendation systems. The users will still retain some room for customization, because the engine is trained with their own data, and each algorithm will work with different parameters—i.e., it's adopted to the customer and product information of each client.

What makes it possible to scale up a single pipeline to four thousand pipelines is the universal schema of the dataset. If a dataset from client A has 100 columns whereas client B has 50, or if the column "number of purchased items" from client A is an integer whereas the same column from client B is a string, they would need to go through different preprocessing pipelines.

Although each client has different customer and product data, at the point that this data is registered on SalesCore, it acquires the same number of columns and the same data types for each column. This makes things easier, as MarketCloud simply needs to copy a single pipeline four thousand times.

Each recommendation system embedded in the four thousand pipelines will have different API endpoints. On the surface, it looks like when a user clicks the "show product recommendations" button, SalesCore displays a list of suggested products. But in the background, what is happening is that by clicking the button, the user is hitting the specific API endpoint associated with the ranked product lists for the specific customer.

Monitoring and Feedback

Maintaining four thousand recommendation systems is not an easy task, and while there have already been many MLOps considerations until this point, this is maybe the most complex part. Each system's performance needs to be monitored and updated as needed. To implement this monitoring strategy at a large scale, MarketCloud can automate the scenario for retraining and updating the models.

Retraining Models

Clients obtain new customers, some of the customers churn, every once in a while new products are added to or dropped from their catalogs; the bottom line is that customer and product data are constantly changing, and recommendation systems have to reflect the latest data. It's the only way they can maintain the quality of the recommendation, and, more importantly, avoid a situation such as recommending a WiFi router that is outdated and no longer supported.

To reflect the latest data, the team could program a scenario to automatically update the database with the newest customer and product data, retraining the model with the latest datasets every day at midnight. This automation scenario could then be implemented in all four thousand data pipelines.

The retraining frequency can differ depending on the use case. Thanks to the high degree of automation, retraining every night in this case is possible. In other contexts, retraining could be triggered by various signals (e.g., signification volume of new information or drift in customer behavior, be it aperiodic or seasonal).

In addition, the delay between the recommendation and the point in time at which its effect is evaluated has to be taken into account. If the impact is only known with a delay of several months, it is unlikely that retraining every day is adequate. Indeed, if the behavior changes so fast that retraining it every day is needed, it is likely that the model is completely outdated when it is used to make recommendations several months after the most recent ones in the training data.

Updating Models

Updating models is also one of the key features of automation strategies at scale. In this case, for each of the four thousand pipelines, retrained models must be compared to the existing models. Their performances can be compared using metrics such as RMSE (root-mean-square error), and only when the performance of the retrained model beats the prior one does the retrained model get deployed to SalesCore.

Runs Overnight, Sleeps During Daytime

Although the model is retrained every day, users do not interact directly with the model. Using the updated model, the platform actually finishes calculating the ranked list of products for all the customers during the night. On the following day, when a user hits the "show product recommendations" button, the platform simply looks at the customer ID and returns the ranked list of products for the specific customer.

To the user, it looks as if the recommendation engine is running in real time. In reality, however, everything is already prepared overnight, and the engine is sleeping during daytime. This makes it possible to get the recommendation instantly without any downtime.

Option to Manually Control Models

Although the monitoring, retraining, and updating of the models is fully automated, MarketCloud still leaves room for its clients to turn the models on and off. More precisely, MarketCloud allows the users to choose from three options to interact with the models:

- Turn on to get the recommendation based on the most updated dataset
- Freeze to stop retraining with the new data, but keep using the same model
- Turn off to completely stop using the recommendation functionality of SalesCore

Machine learning algorithms attempt to convert practical knowledge into meaningful algorithms to automate processing tasks. However, it is still good practice to leave room for users to rely on their domain knowledge, as they are presumed to be far more capable of identifying, articulating, and demonstrating day-to-day process problems in business.

The second option is important because it allows users to stay in the current quality of the recommendation without having the recommendation engines updated with the newer data. Whether the current model is replaced with a retrained one depends on the mathematical evaluation based on metrics such as the RMSE. However, if users feel that the product recommendations on SalesCore are already working well for pushing sales, they have the choice not to risk changing the recommendation quality.

Option to Automatically Control Models

For those that don't want to manually handle the models, the platform could also propose A/B testing so that the impact of new versions is tested before fully switching to them. Multi-armed bandit algorithms (an algorithm that allows for maximization of the revenue of a user facing multiple slot machines, each with a different probability to win and a different proportion of the money given back on average) are used for this purpose.

Let's assume that several model versions are available. The goal is to use the most efficient one, but to do that, the algorithm obviously has to first learn which is the most efficient. Therefore, it balances these two objectives: sometimes, it tries algorithms that may not be the most efficient to learn if they are efficient (exploration), and sometimes it uses the version that is likely to be the most efficient to maximize the revenue (exploitation). In addition, it forgets past information, as the algorithm knows the most efficient today may not be the most efficient tomorrow.

The most advanced option consists in training different models for different KPIs (click, buy, expected revenue, etc.). A method inspired from ensemble models would then allow for the solving of conflicts between models.

Monitoring Performance

When a salesperson suggests a customer buy the products recommended by SalesCore, the interaction of the customer with the recommended products as well as whether the customer bought them or not is recorded. This record can then be used to keep track of the performance of the recommender system, overwriting the customer and product dataset with this record to feed the most updated information to the model when it is retrained.

Thanks to this ground truth recording process, dashboards showing model performance can be presented to the user, including performance comparison from A/B testing. Because the ground truth is obtained quickly, data drift monitoring is secondary. A version of the model is trained every night, but, thanks to the freeze mechanism, the user can choose the active version based on the quantitative information. It is customary to keep the human in the loop on these high-impact decisions where the performance metrics have a hard time capturing the full context around the decision.

In the case of A/B testing, it is important that only one experiment be done at a time on a group of customers; the impact of combined strategies cannot be simply added. With such considerations in mind, it is possible to build a sound baseline to perform a counterfactual analysis and derive the increased revenue and/or the decreased churn linked to a new strategy.

Apart from this, MarketCloud can also monitor the algorithm performance at a macro level, by checking how many clients froze or turned off the recommender systems. If many clients turned off the recommender systems, that’s a strong indicator that they are not satisfied with the recommendation quality.

Closing Thoughts

This use case is peculiar in the sense that MarketCloud built a sales platform that many other companies use to sell products, where the ownership of the data belongs to each company, and the data cannot be shared across companies. This brings a challenging situation where MarketCloud must create different recommender systems for each of the users instead of pooling all the data to create a universal recommendation engine.

MarketCloud can overcome this obstacle by creating a single pipeline into which data from many different companies can be fed. By having the data go through an automated recommendation engine training scenario, MarketCloud created many recommendation engines trained on different datasets. Good MLOps processes are what allow the company to do this at scale.

It's worth noting that though this use case is fictionalized, it is based on reality. The real-life team that tackled a similar project took around three months to finish. The team used a data science and machine learning platform to orchestrate the duplication of a single pipeline to four thousand copies and to automate the processes to feed corresponding datasets to each pipeline and train the models. Of necessity, they accepted trade-offs between the recommendation quality and scalability to efficiently launch the product. If the team had carefully crafted a custom recommendation engine for each of the four thousand pipelines by, for example, choosing the best algorithm for each client, the recommendation engines would have been of a higher quality, but they would have never been able to complete the project with such a small team in such a short period of time.

[1] For an explanation of human-in-command design, see Karen Yeung, "Responsibility and AI" (Council of Europe study, DGI(2019)05), 64, footnote 229.