Landing Page Personalization at Expedia Group

Pavlos Mitsoulis-Ntompos, Dionysios Varelas, Travis Brady, J. Eric Landry, Robert F. Dickerson, Timothy Renner, Chris Harris, Shuqin Ye, Abbas Amirabadi, Lisa Jones, Javier Luis Cardo
{pntompos,dvarelas,tbrady,johlandry,rdickerson,trenner,chrharris,sye,aamirabadi,ljones1,jlcardo}@expediagroup.com
Expedia Group

ABSTRACT

Recommender systems have become critical tools for e-commerce businesses in recent years and online travel platforms such as Expedia Group have made heavy use of them in production. Contemporary travel platforms benefit greatly from the use of recommender systems as very often the space of products (trips) is quite large and shopping cycles often stretch into the weeks.

Expedia Group has successfully trained and deployed multiple types of recommender systems in order to help our travelers find the perfect destination and property for them. In recommender systems literature, much attention is paid to the mathematical aspects of the field but here we focus on best practices in applying recommender systems in large-scale e-commerce for the improvement of browsing and shopping experiences.

In this paper, we describe how we personalize the user experience on a number of our core pages by exploiting existing internal recommender systems and relevant recommender system literature. Additionally we note several critical lessons learned about the importance of a robust machine learning platform, the need to apply engineering best practices and how best to integrate and test recommender systems in production.

KEYWORDS

Recommender Systems, Machine Learning, Travel, E-Commerce, Machine Learning Platform, Product Development

1 INTRODUCTION

Typically users of our sites (e.g. expedia.com, hotels.com, vrbo.com) begin their trip planning process on one of two pages: the home page or a "destination landing page" (DLP), a page created for a specific destination such as Paris, New York or Tokyo. Both home pages and destination landing pages contain separate sets of recommendations of destinations and properties in the form of horizontal-scrolling carousels. Not all travelers need both sets of recommendations since many already know their destination and are then determining which property to book. Yet others have some ideas about the type of trip they want to take (e.g. beaches in Texas), but are flexible as to the specific destination.

Many travelers will make multiple visits to our site before making a booking. In fact, repeat travelers make up a significant fraction of traffic to both landing and home pages. We can use recent signals such as properties viewed and destinations searched to reduce the shopping time for repeat travelers by recommending properties and destinations similar to those they have already searched for[1]. But for travelers identified as new to the site, we often have no historical data for them and therefore cannot use these strategies.

[1]Long term signals are much less useful, as they are often for a different trip.

For example:

- **Vrbo Home Page**: Travelers can land on this page by entering vrbo.com directly on their browser. Figure 1 shows an example of Vrbo's home page.
- **Hotels.com Home Page**: Travelers can land on this page by entering hotels.com directly on their browser.
- **Vrbo Landing Pages**: Landing pages refer to destination landing pages. Travelers usually land on destination landing pages via searching on search engines.
- **Hotels.com Landing Pages**: Similar to Vrbo, landing pages refer to destination landing pages. Figure 2 depicts a landing page for San Diego. Travelers usually land on destination landing pages via searching on search engines. In this example, a traveler probably searched for "san diego vacation homes".

Prior to the introduction of ML-driven personalization techniques our landing and home pages did not recognise or change the experience for travelers based on what we already knew about them. Returning travelers would often have to go through the long process of restarting their search, find properties they liked, and look for comparisons and alternatives themselves.

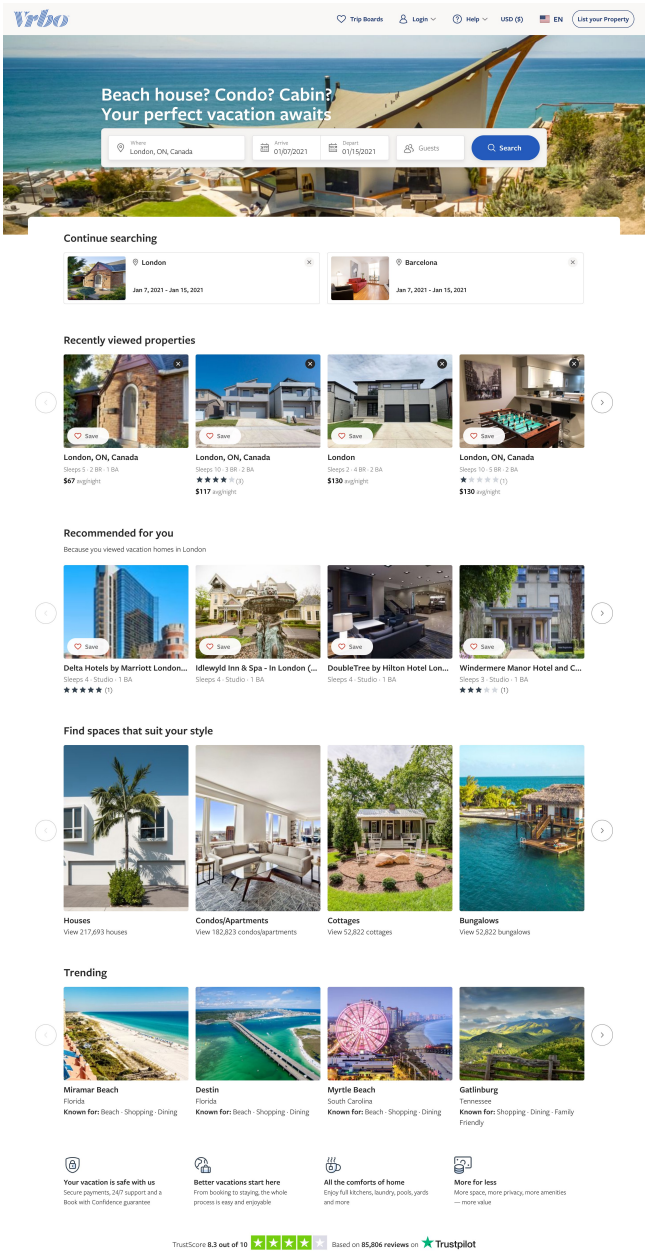
Our motivation for this work was to make the booking experience effortless and to enable travelers to make progress based on their previous visits. We hypothesize that by making contextually relevant destination and property recommendations to travelers based upon their recent browsing behavior we can substantially increase the likelihood of matching them with their ideal trip. We provide evidence to support this hypothesis by exploiting our internal Machine Learning (ML) platform and two systems already deployed within the organization for other internal use cases: a Property Recommender System (PRS) and a Destination Recommender System (DRS).

The proposed solution must address the following challenges:

(1) Software engineers from landing pages and home page team should easily integrate the two recommender systems.
(2) The recommender systems should provide predictions in a synchronous and an asynchronous fashion.
(3) The recommender systems should be able to leverage very recent history for each traveler.

1.1 Use Cases

Figure 3 shows a high-level abstraction of Traveler's Shopping Map. The shopping journey of travelers is non-linear, and there are many different traveler journeys. Some land on our websites and belong to *Discovery phase*, then they move to *Shop phase* and, finally, *Booking phase*. However, there are many travelers who know already where and when to travel, and at which property they want to stay and they start in the *Shop phase*. Others already have a shortlist of



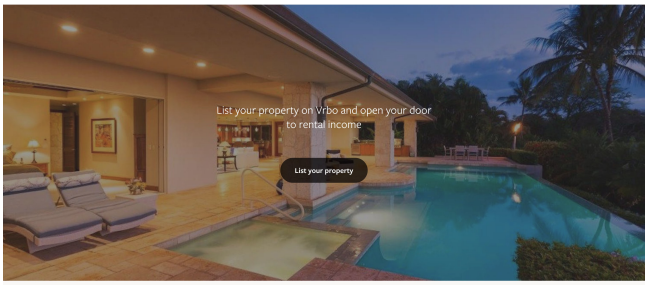


Figure 1: Vrbo Home Page

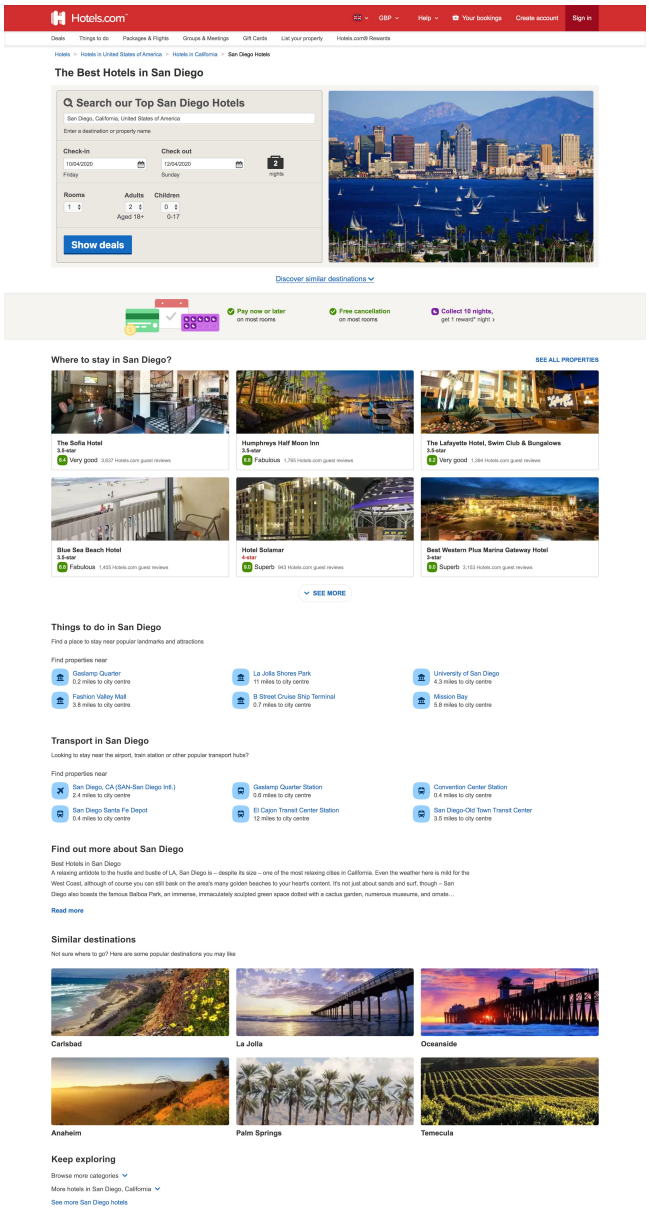


Figure 2: Hotels.com Landing Page

potential properties in mind, but haven't made a final selection and they begin in the *Booking phase*. The purpose of home and landing pages is to fulfill the needs of all travelers that are in Discovery and Shop phase. These are some examples of selection constraints for travelers in the Discovery and Shop phase:

(1) Traveler has not selected the date, destination or property (*Discovery*),
(2) Traveler knows the date and destination, but has not selected the property (*Discovery*),
(3) Traveler knows the destination, but has not selected the property and is flexible on the dates (*Discovery*),

(4) Traveler has selected the destination and date, and has a shortlist of properties (*Shop*),
(5) Traveler has selected the destination and property (or has a shortlist of either / both) and dates (*Shop*).

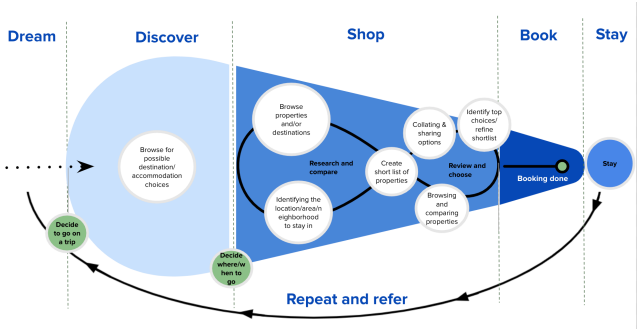


Figure 3: Traveler Shopping Map. In the Discovery phase, travelers are open to any of the following: dates, destination and/or properties. In the Shop phase, travelers are creating a shortlist of destinations and properties, and are conducting further research on them. The traveler books a specific property for specific dates in the Book phase, and physically stays at the booked property during the Stay phase.

1.1.1 Discovery. During the Discovery phase travelers can either be looking for a destination to travel to and/or a property to book within a destination they already decided to visit.

At Hotels.com and Vrbo we aim to facilitate this process by providing travelers destination and property recommendations in order to narrow down the plethora of available destinations and properties.

On the Hotels.com homepage, recommender systems are used on:

(1) The search box where travelers receive recommendations for nearby destinations based on their geographical location.
(2) The properties carousel where travelers are shown personalized recommendations based on their previous browsing sessions.
(3) The destination panel where travelers are shown personalized recommendations based on their previous browsing sessions.

Destination recommenders are used on destination landing pages where recommended destinations similar to the one users searched for are shown.

In the same manner, Vrbo's home page and landing pages show personalized property recommendations based on their previous browsing sessions.

1.1.2 Shop. In this phase, travelers have already shortlisted a number of properties and destinations. Many of them have even selected their destination, and they just need to make decision between a handful of properties.

In order to facilitate travelers in the *Shop* phase, we initially implemented and deployed a couple of heuristic algorithms. Many of our home and landing pages contain carousels showing recent searching and browsing behavior as well as separate recommendation carousels based on said recent behavior. We proved the hypothesis that recent activity is correlated with shortlisted properties and/or destinations using online experiments. Typically we choose the simplest possible solution to start as we believe that simple solutions can be very powerful in terms of providing better user experience and provide baselines for future ML based approaches.

2 IMPLEMENTING PERSONALIZATION FOR LANDING PAGES

To implement landing and home page personalization, we first tried a couple of heuristics. Applying simple methods allowed us to both establish an experimental baseline and address a number of technical issues before adding the complexity of machine learning. With a baseline established, the heuristic logic can then be replaced with machine learning models and tested for effectiveness and end-user impact. The aforementioned heuristic of most recent activity was employed for both *Discovery* and *Shop* phases.

After employing these heuristic approaches, we utilized recommender systems to further enhance personalization. These recommender systems are explained in the following subsections.

2.1 Property Recommender System

The goal of a Property Recommender System (PRS) is to identify the most similar properties to recent properties travelers have interacted with. Formally, let $P = \{p_1, ..., p_M\}$ denote the set of properties where M is the number of properties and $T = \{t_1, ..., t_N\}$ denote the set of travelers where N is the number of travelers. PRS can then be formulated as a problem of conditional probability distribution estimation

$$P\left(\text{traveler will click } p_k | \text{traveler already clicked } p_{i_1}, ..., p_{i_{m_i}}\right)$$

where the indices $1 \leq i_1,...,i_{m_i} \leq M$ represent an exhaustive list of indices for traveler t_i's previous clicks. So, m_i is the total number of clicks by t_i.

2.1.1 Model. We decided to model the aforementioned conditional probability distribution using Word2Vec and specifically the Continuous Bag-of-Words (CBoW) [10], [9] model. Similar approaches that used Word2Vec for item recommendations can be found in [5], [6], [11], [14]. Applied to property recommendations, CBoW predicts the target property p_k based on the properties p_{k-c} and p_{k+c} that appear around it in a given corpus (described in the following section), where c is the window size representing property context (Figure 4). The architecture and optimization targets are the same as in [9] but our use case differs from language modeling problems in the choice of corpus specifying the sequential data to which the model is applied.

2.1.2 Dataset and Application. For the experiments, anonymized clickstream data is collected for millions of travelers and properties. Specifically, for each traveler t_i we create training instances by collecting all the previously seen properties into sequences $p_{j_1}, ..., p_{j_{m_i}}$. From those training instances we slide a window in order to create training examples [10].

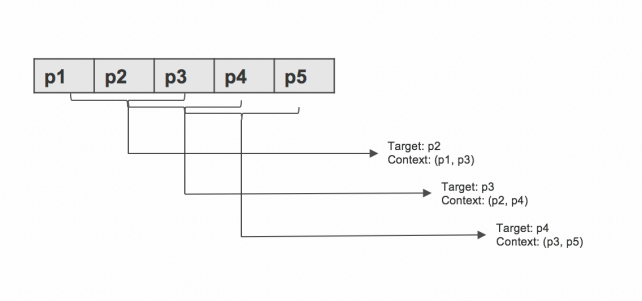


Figure 4: Training Dataset Batching

In our case, the order in which properties appear in the sets is not important as it is in many NLP tasks. Given that a traveler's choice to view a property is affected by the website's modules they interacted with during their shopping journey, viewing a property before another does not imply sequential preference. For this reason we wanted to ensure that our model is trained with as many possible permutations of the original sequences as possible. In offline experiments this resulted in a significant increase in normalized discounted cumulative gain (NDCG), precision, and recall.

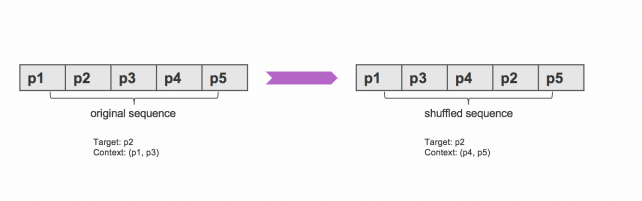


Figure 5: Training Dataset Shuffling. We shuffle the properties within the traveler sequence after each epoch. This significantly improved offline performance.

Finally, for each target property, 2048 properties are randomly sampled based on the probability [10]:

$$P(p_i) = \frac{f(p_i)^{3/4}}{\sum_{j=0}^{n} f(p_j)^{3/4}}$$

Where:

$f(p_i)$ denotes the frequency property i appears

2.1.3 Experiments and Results. For all experiments the optimization setup consists of:

(1) The window size was 3.
(2) Stochastic gradient descent was used for the minimization of the loss function.
(3) The batch size was 128.
(4) The learning rate was kept constant at 1.
(5) The model ran for 63 epochs.
(6) The embeddings weights were initialized using uniform distribution from -1 to 1.

After training, we extract the embedding layer from the resulting CBoW model. For each traveler, we look up the embeddings of each property they have visited and average them to create a "traveler profile". We then search the embedding space using cosine similarity and retrieve the properties closest to the traveler profile.

We compared these recommendations with the recommendations generated by a popularity-based recommender as a simple baseline. The popularity-based recommender calculates for each pair of properties their Jaccard similarity which for two properties p_i, p_j is defined as $JS(p_i, p_j) = \frac{f(p_i,p_j)}{f(p_i)+f(p_j)-f(p_i,p_j)}$ where f returns the number of times a property or two properties appear in all the traveler sequences. In order to generate personalized recommendations, we score each property by taking the average of its similarity score with all the properties a traveler has clicked on and return the top k properties based on that score. For example, if a traveler has clicked on properties p_1, p_2 then the score we assign to a property p_3 would be $\frac{1}{2}(JS(p_1, p_3) + JS(p_2, p_3))$

Table 1 compares the offline evaluation results of these two methods.

Table 1: Comparison between Model Settings

Algorithm	NDCG@20	Precision@20	Recall@20
Popularity based Baseline	0.16441	0.03766	0.23977
CBoW	**0.18412**	**0.03966**	**0.27548**

2.2 Destination Recommender System

Using similar data collection methods as in section 2.1, we leveraged session-based search data to build a co-search matrix, with the goal of recommending relevant destinations to travelers in the *Discovery* phase who have not yet chosen a destination.

On the homepage, the target is traveler's last destination in search history. We want to recommend similar locations with similar destination type (for example city, point of interest, or neighborhood). For example, if someone searches for Austin, TX, we would recommend destinations like San Antonio and Houston which are both cities in Texas similar to Austin.

On a destination landing page, the target is the destination that the traveler typed in the search engine. Since we know that the travelers coming from a search engine are for a particular destination, we recommend smaller locations that are within the target destination. For example, if someone searched for France, we would like to help the traveler know more about the place by recommending the most relevant cities/places in France such as Paris (a city) or Southern France (a region).

Powering both of the above use-cases, the destination recommender system (DRS) aims at identifying similar destinations to a set of destinations travelers have interacted with.

2.2.1 Model. Formally, we are trying to model the conditional probability distribution $P(d_j|d_i)$ where d_i is the destination that a traveler searched for and d_j is any destinations where $d_j \neq d_i$. The model we use on the homepage, destination landing page and Hotels.com search box is a matrix decomposition in which we decompose a very sparse preference matrix. The specifics of the preference matrix vary according to the application. On the home page

the preference matrix is traveler-destination, corresponding commonly to user-item matrices in wider literature [8], [15]. On destination pages the preference matrix is destination-destination, corresponding to item-item matrices. For the search box, the preference matrix is expressed between latitude/longitude coordinates of travelers and destinations.

The model decomposes the sparse preference matrix into two matrices of traveler and destination factors. These factors are dense latent space representations of each traveler and destination, denoted $x_u \in \mathbb{R}^f$ and $y_i \in \mathbb{R}^f$, respectively. To use these factors for recommendations, we simply take the dot product of the traveler and destination vectors to obtain a score $s = x_{u=k} \cdot y_{i=l}$, for traveler k and destination l, and rank the destinations by that score. This yields the most relevant destinations for the traveler.

For the destination-destination preference matrix, the process is identical except that the preference signal is expressed between destinations (described in the next section), and the latent traveler factor is now an "origin" destination vector. For the latitude/longitude preference matrix, the preference signal is expressed between the traveler's location and the destination's location, and the latent traveler factor is now a "geographic origin" vector.

Irrespective of how it is populated, the size and sparseness of the preference matrix presents a significant computational challenge. To solve the decomposition we use Alternating Least Squares (ALS) [12], which is a highly scalable, efficient, and well understood method for solving this type of problem.

2.2.2 Dataset and Application. On the homepage, we use recent destinations that travelers have searched for in order to build the traveler-destination preference matrix. Each destination searched by a traveler receives a positive preference. Factorizing this matrix into traveler and destination features as described above we generate personalized recommendations for each traveler that has searched for at least one destination in the last year.

On destination landing pages, we know the original intent destination of travelers. We leverage session-based search data in order to build a destination-destination preference matrix. Two destinations searched by one traveler in a single session receive a positive preference. Using the destination-destination preference matrix decomposition we provide recommendations for similar destinations.

On the Hotels.com search box, travelers are shown destination recommendations based on their geolocation. The preference matrix for this application is built almost identically to the traveler-destination matrix, except that the traveler rows are now replaced with traveler location rows. This enables recommendations to be provided for a traveler who has no search history.

2.2.3 Setup. The deployed ALS model on Homepage has a number of hyperparameters tuned via cross-validation:

(1) Number of latent factors, f, often on the order of tens
(2) Number of iterations represents the number of training loops performed, often tens will suffice
(3) Lambda, λ is problem dependent but always <1

3 RESULTS

Our top priority is to improve the experience of our users, and the machine learning models are only useful if they improve the traveler experience. Sophisticated models are not valuable for their own sake, they must provide significant value as measured by an online controlled experiment. In fact, it was shown in [2] that improvements in offline metrics do not necessarily drive an improved user experience for a whole host of reasons. At Vrbo and Hotels.com we have observed the same. We measure the impact of each new algorithm or system using our internal A/B test platform. For each model discussed in Section 2, the improvements on one of our core metrics are presented in Table 2. The definition of the core metric and the actual percentages are considered sensitive and cannot be revealed. The recommended destinations model at Hotels.com's home page had the lower improvement and is considered the benchmark. The Table 2 shows the relative improvements against this benchmark.

Table 2: Online Experiment Results

Brand	Model	Page	Improvement
Hotels.com	"Similar Properties"	home page	3.25
Hotels.com	"Recommended destinations"	home page	1.0
Hotels.com	"Similar destinations"	landing page	1.975
Hotels.com	"Search box recommendation"	home page	2
Vrbo	"Recently viewed properties"	home page	4.25
Vrbo	"Recent activity"	landing page	2.25
Vrbo	"Recommended for you"	home page	5
Vrbo	"Based on your recent searches"	landing page	2.425

There is a significant difference in orders of magnitude in results between Hotels.com and Vrbo. At Hotels.com, typical trip planning is relatively short in duration and with many travelers booking last minute trips. Vrbo on the other hand is a vacation rental focused site, where travelers are more likely to have long shopping cycles and may search for a wider variety of properties and destinations before making a final selection. This gives the recommendation systems much more exposure and therefore higher potential impact.

4 CHALLENGES AND SOLUTIONS

The models themselves, though important, are a small part of the overall system that provides and surfaces the recommendations to travelers. The complete system consists of multiple backend services and data stores as well as a front-end client. Additionally components for experimentation (the aforementioned A/B testing platform), instrumentation and monitoring are also required to ensure consistent uptime and technical correctness. In this section we present our initial attempts at machine learning integrations and the lessons we learned from those. We then describe our internal machine learning platform and how it addresses many of the challenges we faced early on. We also describe the machine learning platform's impact on overall data science efficiency, company-wide.

4.1 History

Initially at both Vrbo and Hotels.com, machine learning models were directly integrated with the product's codebase. At the time there were few machine learning models in production and therefore specialized infrastructure wasn't economical. Absent machine learning specific infrastructure, embedding the model code directly in the backend software stack is simpler.

There are a number of drawbacks to this approach that compound complexity as model updates become more frequent, however. Firstly, any model retraining or update would require redeploying the entire stack. For home and landing pages, this is not trivial. Those services are critical and must have near-perfect uptime, or users will be unable to search for properties and therefore unable to book. This slows down the speed of iteration and experimentation, which is critical for machine learning success.

Machine learning models are very different from traditional backend systems. The code for training ML models is usually written in Python, Scala or R, while most of the backend systems at Hotels.com and Vrbo are built in Java. This necessitates a model "handoff" from one runtime to the next, and makes apples-to-apples testing extremely difficult, particularly if there's data processing code required. Compute and memory requirements are also significantly higher than traditional backend systems. This means that adding machine learning directly into a backend system will likely change the requirements for the whole system, even if it's only needed for the ML model. Monitoring machine learning systems is more challenging as well. All the usual monitoring for uptime and latency apply, but it's also important to monitor predictions (as noted in [2]) as a means of detecting bias, train-test skew, and model drift. Finally, the skill sets between data scientists and backend developers are also very different. Managing software in production requires a completely different skill set than building a powerful model. With the model integrated directly into the backend, the data scientists need to know a lot about that system, and the backend developers must understand the computational and memory requirements of the model.

All of these differences slow down iteration and experimentation. Redeploying the entire backend stack for a critical system is slow and risky, even with appropriate application infrastructure. Translating the model from one runtime to another takes time, and testing is very tedious. The single most important thing a data scientist can do with a model to increase it's economic impact is to test it online as much as possible, yet at the start of these efforts that was extremely difficult. This necessitated an investment on the part of Vrbo and Hotels.com to develop specialized infrastructure specifically for deploying and monitoring ML models.

4.2 Machine Learning Infrastructure

Most companies with significant machine learning requirements and products have a platform to accelerate ML development and deployment. Some concrete examples include Tensorflow Extended (TFX) [1] from Google, Bighead from Airbnb [2], Michaelangelo from Uber [3], and FBLearner [4] from Facebook. We also developed a machine learning platform that serves models in real-time, streaming and batch contexts. We decided to favor a microservice approach since the systems ML is integrated into are mostly Java or Scala based. The microservice approach also solves the scaling / system requirements mismatch, as the ML model can be scaled independently from the integrating system. For the purposes of recommendation systems there are three relevant patterns to discuss: real-time synchronous, real-time precomputed (offline), and real-time precomputed (online). These are diagrammed in Figure 6. Here real-time refers to how the inferences are accessed, meaning large numbers of small one-at-a-time requests as opposed to bulk requests of the entire dataset (via SQL or Apache Spark [5], for example). The platform was built in stages, usually driven by data scientist / software engineering integration requirements, rather than dictated top-down all at once. The result is a series of loosely coupled components that were developed based on real-world use cases.

4.2.1 Real-time Synchronous Inference (RTSI). This is a straightforward request / response pattern. The caller sends the payload containing the model features, then the service invokes the model in-memory and performs the inference, sending the result back to the caller. This pattern is very simple and follows a RESTful way of calling the prediction logic of models. Deployment is entirely self-service, and requires only a small amount of code on the data scientist's part for instrumentation and packaging. It can support any Python, Java or Scala machine learning framework.

The obvious disadvantage to this pattern is clear when the model requires a significant amount of computational power, or when the required latency is low. Since the model is being invoked per request, latency can be quite high depending on the model.

4.2.2 Real-time Precomputed (Offline) (RTPOf). If the inference scores can be precomputed in an offline environment, they can be loaded into a key-value store as long as there's a suitable key. Examples of common keys at Vrbo and Hotels.com are property ID, traveler ID and destination ID. Then callers would only need to know the key to retrieve the scores. This transforms what was a complicated inference per-prediction into a lookup, reducing the latency by at least an order of magnitude.

This approach has drawbacks too. If the key space has a very high cardinality then precomputation and storage become impractical thus necessitating a move to the aforementioned synchronous inference. Another drawback is the frequency of updates to the key-value store. In many cases a simple daily or hourly update will suffice, however some use cases require the precomputed values to be updated continuously as new data arrives in the system. Updating traveler "profiles" based on their searches as they search on the site is an example of such a use case.

4.2.3 Real-time Precomputed (Online) - (RTPOn). By combining the real-time synchronous services with the key-value store, we are able to listen to a stream of values (via Apache Kafka[6]) and

[2] https://databricks.com/session/bighead-airbnbs-end-to-end-machine-learning-platform

[3] https://eng.uber.com/michelangelo/

[4] https://engineering.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/

[5] https://spark.apache.org/

[6] https://kafka.apache.org/

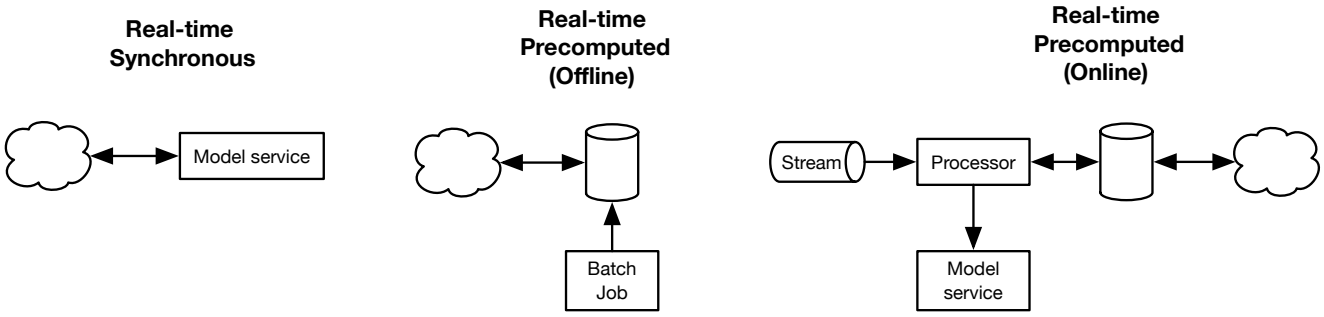


Figure 6: (a) Real-time synchronous, (b) real-time precomputed (offline), (c) real-time precomputed (online)

perform the predictions *as the inputs arrive*, rather than when the predictions are needed. The assumption of course is that the predictions won't be needed immediately, but some time after the features become available on the stream.

When the model input (including key for lookup) arrives on the stream, a stream processor service issues a synchronous request to the model service to retrieve the score. Then the stream processor inserts the score into the key-value store, where going forward it will be available at low latency without having to recompute the predictions. This results in continual updates to the key-value store at high frequency. Should the number of input events become large for a short period of time, it is possible to scale the model service to handle the traffic, or if a delay is acceptable, wait until the volume of stream events returns to normal levels.

Table 3: Mapping of Models to Inference Type

Model - Use Case	Inference Type
Hotels.com "Similar Properties"	RTSI
Hotels.com "Recommended destinations"	RTPOf
Hotels.com "Similar destinations"	RTPOf
Hotels.com "Search box recommendation"	RTPOf
Vrbo "Recently viewed properties"	RTPOn
Vrbo "Recent activity"	RTPOn
Vrbo "Recommended for you"	RTSI
Vrbo "Based on your recent searches"	RTSI

4.3 Productivity

The combination of straightforward deployment, standardized infrastructure toolkits, flexible ML framework support, and separation from the integrating system has significantly improved the iteration speed (and therefore effectiveness) of data scientists at Vrbo and Hotels.com. Since the development and adoption of the ML platform, the number of ML models deployed has grown approximately linearly.

5 DISCUSSION AND FUTURE WORK

At Vrbo and Hotels.com, we are always looking for new and better ways to lead travelers to their dream vacation. Correspondingly, our methods for personalizing landing pages and the home page are always evolving. We present ongoing and future work in this section.

5.1 Content Augmented Recommendation System - CaRS

Building upon the work done on Property Embeddings (section 2.1) and inspired by [4] we started developing a new recommendations framework in order to incorporate additional meaningful signals such as price or destination attributes (historic, romantic, etc.).

The aim of CaRS is to provide a recommender system that can be used for both property and destination recommendations. We're developing CaRS in a way that allows us to generate both personalized (traveler to item) and non-personalized (item to item) recommendations while maintaining only one model.

For the rest of this subsection, an *item* will refer to both properties and destinations. We consider a destination "booked" if a traveler booked a property within the destination.

The model uses sequences of clicked items as context and booked items as targets from traveler sessions. Travelers are represented as an aggregation (e.g. average) of the features of the items they have clicked on. We train a neural network to generate a score which represents how likely travelers are to book an item given the previous items they have clicked on. The function learned by the neural network can be used both with multiple items as input so as to personalize results and with a single item if no personalization is desired.

We use a pairwise loss; for each positive example we draw negative examples and learn the weights such that a higher score is assigned to positive examples than the negative ones. Inspired by [3], [13] we used our pretrained embeddings in order to identify similar items that weren't viewed by the travelers. This allowed us to improve the performance of the model with negative sampling based on the already estimated conditional probabilities provided by the embeddings. By sampling negatives near the decision boundary we can improve ranking metrics like NDCG.

Combining content features with decision boundary negative sampling, we saw an improvement in offline metrics like NDCG and recall when we compared CaRS to CBoW.

5.2 Multi-armed Bandits

Inspired by a paper on using multi-armed bandits for real-time optimization of web page layouts [7], we have recently adopted methods to optimize traveler experience via Reinforcement Learning (RL). The "arms" in this multi-armed bandit are various layouts or other visual attributes of the page. Since we are looking to serve layouts that will more often lead travelers to find a property, our reward function is a simple binary reward offering positive value whenever the traveler clicks. As previously stated in Section 2 not all travelers have the same behavior and so we seek to learn how traveler context influences preferences. We make use of contextual bandits to attempt to automatically discover ideal layouts tailored to users all exhibiting some shared characteristics such as traveler segmentation, purchase history, or device type. The model we build for the click-through-rate then has a certain amount of personalization built in by using features which are not only dependent on the particular layout that was served but also on some context of the traveler.

RL is fundamentally different in several ways from more traditional ML approaches. With RL, there is a feedback loop in which the model provides the front-end application with a layout and then observations of what results were obtained from that layout must be fed back into the model for it to update its parameters. This is a stateful, real-time coupling to both the backend and front-end systems serving the layouts which grows in complexity as the scope of observations increases. Reinforcement learning presents many new challenges for machine learning infrastructure that was designed primarily with supervised learning use cases in mind. However, as before simplicity is the best approach. By starting with initial experiments focused on LEARNING we have shown the ability to identify the patterns likely to succeed at larger scale whether related to deployement, offline evaluation or system health monitoring.

6 CONCLUSION

In this paper, we introduced a way to implement personalization for returning travelers while they are in the process of trip planning. Our contribution here is two-fold: first we share details on how the system was initially created and then ultimately run. Next we note lessons learned on the ways in which ML platforms can be especially valuable for large e-commerce companies. We emphasize the importance of simple solutions and of democratizing ML model predictions to a large and varied organization. And finally we conclude with details on extensions to personalization via Reinforcement Learning. Here we have begun to explore the usage of Multi-Arm Bandits to determine the best possible layouts for travelers that are in *Discovery* or *Shop* phases as well as improve the current Property Recommender System (PRS) with content features.

ACKNOWLEDGMENTS

The authors would like to thank Fedor Alexandre Parfenov, Kyler Eastman, Vasileios Vryniotis, Seiyang Chua for their contribution to this paper.

REFERENCES

[1] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 1387–1395. https://doi.org/10.1145/3097983.3098021

[2] Lucas Bernardi, Themistoklis Mavridis, and Pablo Estevez. 2019. 150 Successful Machine Learning Models: 6 Lessons Learned at Booking.Com. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*. Association for Computing Machinery, New York, NY, USA, 1743–1751. https://doi.org/10.1145/3292500.3330744

[3] Long Chen, Fajie Yuan, Joemon M. Jose, and Weinan Zhang. 2018. Improving Negative Sampling for Word Representation Using Self-Embedded Features. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. Association for Computing Machinery, New York, NY, USA, 99–107. https://doi.org/10.1145/3159652.3159695

[4] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA.

[5] Kallirroi Dogani, Matteo Tomassetti, Saúl Vargas, Benjamin Paul Chamberlain, and Sofie De Cnudde. 2019. Learning Embeddings for Product Size Recommendations. In *Proceedings of the SIGIR 2019 Workshop on eCommerce, co-located with the 42st International ACM SIGIR Conference on Research and Development in Information Retrieval, eCom@SIGIR 2019, Paris, France, July 25, 2019 (CEUR Workshop Proceedings)*, Jon Degenhardt, Surya Kallumadi, Utkarsh Porwal, and Andrew Trotman (Eds.), Vol. 2410. CEUR-WS.org. http://ceur-ws.org/Vol-2410/paper13.pdf

[6] Mihajlo Grbovic and Haibin Cheng. 2018. Real-Time Personalization Using Embeddings for Search Ranking at Airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. Association for Computing Machinery, New York, NY, USA, 311–320. https://doi.org/10.1145/3219819.3219885

[7] Daniel N. Hill, Houssam Nassif, Yi Liu, Anand Iyer, and S.V.N. Vishwanathan. 2017. An Efficient Bandit Algorithm for Realtime Multivariate Optimization. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '17* (2017). https://doi.org/10.1145/3097983.3098184

[8] Y. Hu, Y. Koren, and C. Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *2008 Eighth IEEE International Conference on Data Mining*. 263–272. https://doi.org/10.1109/ICDM.2008.22

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:cs.CL/1301.3781

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. (2013), 3111–3119.

[11] Pavlos Mitsoulis-Ntompos, Meisam Hejazinia, Serena Zhang, and Travis Brady. 2019. A Simple Deep Personalized Recommendation System. In *Proceedings of the Workshop on Recommenders in Tourism co-located with the 13th ACM Conference on Recommender Systems (RecSys 2019), Copenhagen, Denmark, September 16-20, 2019. (CEUR Workshop Proceedings)*, Vol. 2435. CEUR-WS.org, 22–26. http://ceur-ws.org/Vol-2435/paper4.pdf

[12] Gábor Takács and Domonkos Tikk. 2012. Alternating least squares for personalized ranking. (09 2012). https://doi.org/10.1145/2365952.2365972

[13] Ugo Tanielian and Flavian Vasile. 2019. Relaxed Softmax for PU Learning. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys '19)*. Association for Computing Machinery, New York, NY, USA, 119–127. https://doi.org/10.1145/3298689.3347034

[14] Flavian Vasile, Elena Smirnova, and Alexis Conneau. 2016. Meta-Prod2Vec: Product Embeddings Using Side-Information for Recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 225–232. https://doi.org/10.1145/2959100.2959160

[15] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-Scale Parallel Collaborative Filtering for the Netflix Prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM '08)*. Springer-Verlag, Berlin, Heidelberg, 337–348. https://doi.org/10.1007/978-3-540-68880-8_32