Quick Access: Building a Smart Experience for Google Drive

Sandeep Tata, Alexandrin Popescul,
Marc Najork, Mike Colagrosso
Google USA
1600 Ampthitheater Parkway
Mountain View, CA, USA
tata,apopescul,najork,mcolagrosso@google.com

Julian Gibbons, Alan Green, Alexandre Mah,
Michael Smith, Divanshu Garg,
Cayden Meyer, Reuben Kan
Google Australia
48 Pirrama Road
Pyrmont, NSW, Australia
juliangibbons,avg,alexmah,smithmi,divanshu,cayden,
reuben@google.com

ABSTRACT

Google Drive is a cloud storage and collaboration service used by hundreds of millions of users around the world. *Quick Access* is a new feature in Google Drive that surfaces the most relevant documents when a user visits the home screen. Our metrics show that users locate their documents in half the time with this feature compared to previous approaches. The development of Quick Access illustrates many general challenges and constraints associated with practical machine learning such as protecting user privacy, working with data services that are not designed with machine learning in mind, and evolving product definitions. We believe that the lessons learned from this experience will be useful to practitioners tackling a wide range of applied machine learning problems.

ACM Reference format:
Sandeep Tata, Alexandrin Popescul, Marc Najork, Mike Colagrosso, Julian Gibbons, Alan Green, Alexandre Mah, Michael Smith, Divanshu Garg, Cayden Meyer, and Reuben Kan. 2017. Quick Access: Building a Smart Experience for Google Drive. In *Proceedings of KDD'17, August 13–17, 2017, Halifax, NS, Canada.*, , 9 pages.
DOI: http://dx.doi.org/10.1145/3097983.3098048

1 INTRODUCTION

Searching for and gathering information is a common task that continues to consume a large portion of our daily life. This is particularly true when it comes to finding relevant information at work. Recent research [12] shows that time spent in finding the right documents and information takes up 19% of a typical work week and is second only to the time spent managing email (28%).

Hundreds of millions of people use Google Drive to manage their work-related and personal documents. Users either navigate through subdirectories or use the search bar to locate documents. In addition to the typical file-system metaphor of folders (directories), Drive provides views like "Recent", "Starred", and "Shared with me" for convenient access to files. However, the user still needs to navigate to a document through many clicks. To alleviate this problem, we built "Quick Access" – a smart feature in Google Drive that makes the most relevant documents available on its home screen. Figure 1 shows the home screen of the Google Drive app for Android, with the top portion devoted to a list of documents deemed most likely to be relevant. We use a machine learning (ML) model trained over past activity of Drive users to determine which documents to display in a given context. Our metrics show that Quick Access gets users to their documents nearly 50% faster than existing approaches (via navigation or search).

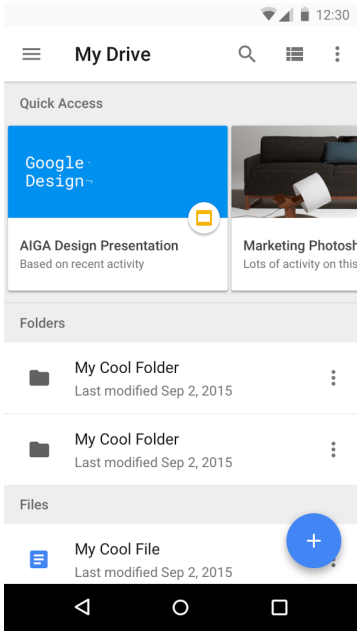


Figure 1: Quick Access screenshot

In this paper, we describe the development of this feature as an applied ML project focusing on aspects that are typically not discussed in the literature:

(1) Documents selected from private corpora.
(2) Existing data sources not set up for ML applications.
(3) Evolving product definition and user interface (UI).

KDD'17, August 13–17, 2017, Halifax, NS, Canada.
 978-1-4503-4887-4/17/08.
DOI: http://dx.doi.org/10.1145/3097983.3098048

The document corpora in Drive are private to their owners and vary in size from a few documents to tens of thousands. Further, users may choose to share a document either with specific users or with groups (e.g. an entire company). This distinguishes the setting from typical recommendation problems such as Netflix [7] and YouTube [13, 14], which use a shared public corpus. As a result, we do not use collaborative filtering and instead draw on other techniques to compute a ranked list of documents.

Private corpora also introduce additional challenges to developing ML models while maintaining a commitment to protect user privacy. As part of this commitment, visual inspection of raw data is not permitted, making initial data exploration, a common first step in applied ML, substantially more difficult. We used k-anonymity based aggregation approaches (see Section 4.1) and argue that further research in techniques that allow data cleaning and feature engineering on data without inspection will be valuable to building ML models on private datasets.

Setting up an ML pipeline on a data source that had never previously been used for ML applications required overcoming several non-trivial challenges. Data quality problems including incomplete logging, inconsistent definitions for certain event types, and multiple sources of training-serving skew (see Section 4.2.1) were significant obstacles to launching the product. These are not frequently discussed in the literature [23] and may be of much value to practitioners interested in setting up applied ML pipelines. We advocate a *dark-launch and iterate* technique to guard against such problems.

Finally, in order to maximize the usefulness of our predictions, we needed to figure out a UI that would be visible and helpful when predictions were good, but not distract or obstruct the user when they were not. We considered three UIs which emphasized completely different metrics. In this paper, we share the reasoning which led us to choose one over the other.

In addition to the above aspects, we also describe the use of deep learning [21] to model and solve this problem. We discuss how feature engineering tasks usually employed for more traditional algorithms may still be necessary when using deep learning in this setting – some classes of tasks (like modeling interactions) become redundant, while other, new types emerge that are particular to deep networks. We hope that some of the lessons we share here will be valuable to practitioners aiming to apply machine learning techniques on private corpora either when using more traditional tools or when using deep networks.

2 SYSTEM OVERVIEW

In this section, we provide a high-level overview of the system used to build Quick Access, shown diagrammatically in Figure 2. We describe the data sources used, the training and evaluation pipelines, and the services that apply the model in response to user requests.

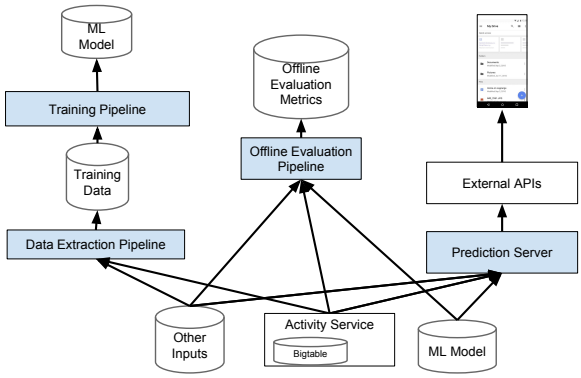


Figure 2: High-level system overview

2.1 Data Sources

The *Activity Service* provides access to all recent activity in Drive. It gathers all requests made to the Drive backend by any of the clients used to access Drive – including the web, desktop, and mobile clients, and the many third-party apps that work through the Drive API. The Activity Service logs high-level user actions on documents such as *create*, *open*, *edit*, *delete*, *rename*, *comment*, and *upload* events.

This data is made available for batch processing by placing it in Bigtable [10]. The Activity Service also maintains a Remote Procedure Call (RPC) endpoint for real-time access to the data for user-facing features.

Additional sources of input for document selection models include user context such as recent or upcoming meetings on the calendar, their Drive settings (e.g. personal vs. business account) and long-term usage statistics (e.g. total number of collaborators, total files created since joining). Figure 2 shows this as *Other Inputs*.

2.2 Training and Evaluation

The *data extraction pipeline* is responsible for scanning the Activity Service Bigtable and generating a *scenario* corresponding to each open event. A scenario represents a user visiting the Drive home screen, and is an implicit request for document suggestions. The pipeline is responsible for discarding data from users that should be excluded either for quality reasons (e.g. robots running crawl jobs and batch updates) or policy reasons (e.g. regulations governing enterprise users). Raw training data is stored in protocol buffers for each scenario along with the recent activity from the Activity Service. The data is split into train, test, and validation sets by sampling at the user level and the scenarios are converted into a set of positive and negative examples with suitable features. This is discussed in Section 3 in greater detail.

The training data generation pipeline is implemented as a Flume [9] job and runs on Google's shared infrastructure over $O(2000)$ cores for a few hours. An illustrative run over a sample extracted from 14 days of activity data considered $O(30\text{M})$ scenarios and produced $O(900\text{M})$ examples.

We used a distributed neural network implementation [15] to train over these examples, typically using $O(20)$ workers and $O(5)$ parameter servers. Once a model has been trained, the *evaluation pipeline* is used to compute various custom metrics like top-k accuracy for various k.

The *Prediction Service* responds to user requests by fetching the latest data from the Activity Service, computing the current set of features, and applying the model to score candidate documents. It returns an ordered list of documents which are displayed by the client. The service is integrated with an experiment framework that allows us to test multiple models on live traffic and compare their performance on key metrics.

3 MODELING

The document selection is modeled as a binary classification problem. For any given scenario, we generated one positive example with score 1 (corresponding to the document that was opened), and a sample from the $n-1$ negative examples with score 0, where n was the total number of candidate documents from the user's corpus. Note that this approach excludes from consideration documents a user has never visited, regardless of whether they have access to them (such as public documents). At serving time, scores in the interval $[0, 1]$ are assigned to candidate documents and used to rank them. Next, we discuss the candidate selection for training and inference.

3.1 Candidate Selection

Preliminary analysis showed that while users may have tens of thousands of documents (especially photos and videos for consumers who connect their phones to Google Drive storage), most of the *open* events are focused on a smaller working set of documents. We limited the candidates to documents with activity within the last 60 days. This choice was a consequence of several factors such as how long users have allowed Google to retain data, the cost of storing and retrieving longer user histories, and the diminishing returns from longer histories. For example, there was a strong recency bias in the *open* events, with more than 75% being for documents on which there was some activity within the last 60 days. This reduction in scope had the benefit of limiting the number of documents to be scored and sorted when serving predictions (thus also reducing latency for the user).

We used the same criterion to limit the pool of negative example candidates for the training set. Additionally, we selected from the candidate pool the k most recently viewed documents (for some $k \leq n$) to limit the disproportionate contribution of examples to the training data from highly active users.

3.2 Using Deep Networks

Having formulated a pointwise binary classification problem and a model to score each document, we considered several algorithms. Although scalable implementations of linear and logistic regression have previously been described in the literature [3, 22] and are commonly used in industry, we chose instead to use deep neural networks for three reasons:

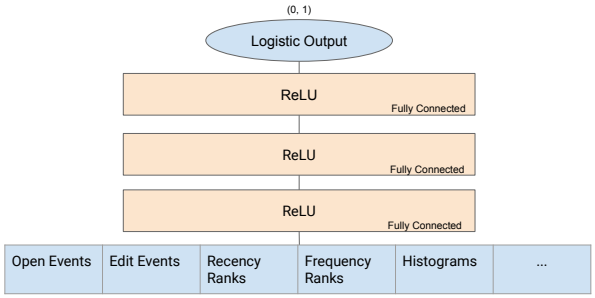


Figure 3: Network architecture used for training

(1) There has been a significant investment in tooling and scalability for neural networks at Google [1, 15].
(2) The promise of reduced effort for feature engineering given the well-known successes for neural networks in domains like speech, images, video, and even content recommendation [13].
(3) Informal comparisons showing that deep neural networks can learn a competitive model and be used for real-time inference with a resource footprint comparable to the alternatives.

The choice of deep neural networks for this problem was not automatic since the input features are heterogeneous compared to typical applications like images and speech.

3.3 Network Architecture

For convenience, we represented all the data associated with a given example using a vector of approximately 38K floats, many of which were zero. We employed a simple architecture with a stack of fully connected layers of Rectified Linear Units (ReLU) feeding into a logistic layer, which fed a linear combination of the activations from the last hidden layer into a sigmoid unit, to produce an output score in the interval $[0, 1]$.

Details of the network architecture such as the number of layers, the width of each layer, the activation function, the update algorithm, the initialization of the weights and biases in the layers were all treated as hyperparameters and were chosen using a grid search. We experimented with different activation functions including sigmoid, tanh, and ReLU. Unsurprisingly, we found ReLUs to work very well for our setting. Both sigmoid and tanh functions were either marginally worse or neutral when compared with the ReLU layers. We experimented with varying the widths of the hidden layers and the number of hidden layers. We tried several different learning algorithms including simple stochastic gradient descent, AdaGrad [16], and some momentum-based approaches. We found AdaGrad to be robust and in our setting, it did no worse than the momentum-based approaches.

The graph in Figure 4 shows the holdout AUCLoss (defined as $1-$ROC AUC, the area under the receiver operating

characteristic curve) for a set of hyperparameters after several hours of training. We observed improvements from adding more hidden layers up to four or five layers. Adding a sixth layer did not help in any of our metrics. While not shown here, using zero hidden layers produced a substantially worse AUCLoss, suggesting that logistic regression was not a competitive option without additional engineering of feature interactions. The right hyperparameters typically made a difference of about 2% to 4% in AUCLoss. However, improvements from feature engineering we did yielded substantially larger benefits, which we discuss next.

3.4 Feature Engineering

Feature engineering is an umbrella term that includes efforts like making additional input data sources available to the ML model, experimenting with different representations of the underlying data, and encoding derived features in addition to the base features. An interesting aspect of the data in this project was the heterogeneous nature of the inputs – these included categorical features with small cardinality (e.g. event type), large cardinality (e.g. mime type), and continuous features across different orders of magnitude (e.g. time of day, time since last open, total bytes stored in the service).

Data from the Activity Service consisted of several series of events, such as the history of *open*, *edit*, *comment*, etc. events for each document. We represented this data as a sparse, fixed-width vector by considering time intervals of a fixed-width (say, 30 minutes) over the period for which we were requesting events (say, 60 days) and encoding the number of events that fell into each bucket. Using 30-minute buckets over 60 days, each event type corresponded to a vector of length $60 \times 24 \times 2 = 2880$. In addition to this, we also considered finer-grained buckets (1-minute) for shorter periods of time. As one would expect, most of the entries in this vector were zero, since user activity on a given document does not typically occur in every 30-minute bucket every day.

In a further refinement, we distinguished between events that originated from various clients – native applications on Windows, Mac, Android, and iOS and web. This lead to a sparse vector for each combination of event type and client platform. We also produced similar vectors for events from *other* users on shared documents.

In addition, we computed histograms of the user's usage along several dimensions, such as the client type, event type, and document mime type. We encoded the time-of-day and day-of-week, to capture periodic behavior by users.

We used one-hot encoding for categorical and ordinal features, i.e. encoding with a vector where exactly one of k values was set to 1, with the rest at 0. These features were of fairly small cardinality, so using the one-hot encoding directly seemed a reasonable alternative to learning their embeddings. This included features like mime type, recency rank, frequency rank etc.

Table 1 shows a sketch of the relative improvements in the AUCLoss over the previous models as we added more features. Unsurprisingly, the effort had large benefits early, with diminishing returns later.

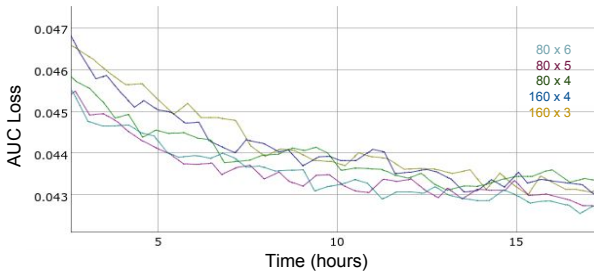


Figure 4: AUCLoss vs. time for networks with different architectures

3.4.1 Performance Optimization. In addition to improving accuracy, we found that some additional feature engineering over the base features significantly improved learning speed. For example, taking a logarithm of features with very large values improved convergence speed by nearly an order of magnitude. We also considered a dense encoding of time-series data where we encode the logarithm of the time elapsed since the event for the last k events of a given type. With this representation, we can represent up to k events using exactly k floating-point values instead of the much sparser representation that uses 2880 values as described above. We observed that this representation produced smaller training data sets that converged faster to approximately the same holdout loss as the sparser representation.

3.4.2 Impact of Deep Neural Nets. An exciting aspect of the recent breakthroughs in deep learning is the dramatic reduction in the need for feature engineering over existing data sources [18, 20]. For instance, in computer vision, hidden layers in the network tend to learn higher-level features like boundaries and outlines automatically without the need for these features to be explicitly constructed by experts in the domain [20]. Historically, several advances in computer vision and image recognition have relied on creative feature engineering that transformed the raw pixels into higher-level concepts. Deep learning models for image recognition [19] exchange the complexity of feature engineering for a more complicated network architecture.

Part of the allure of deep learning in other domains is that the intuition-intensive manual task of feature engineering can be exchanged for architecture engineering where the deep network is constructed using suitable building blocks that

Features added	Num. of features	AUCLoss improv.
Baseline, using open events	3K	N/A
Separate time series by event types	8K	30.1%
Separate by client; adjust granularity	24K	8.2%
Histograms of user events	26K	7.0%
Longer history	35K	5.1%
Document recency and frequency ranks	38K	3.8%

Table 1: AUCLoss improvements over the previous model from feature engineering.

are known be be useful for specific kinds of data. We found that in a task with heterogeneous input data such as ours, spending energy on feature engineering was still necessary, and provided substantial benefits as is evident from Table 1. However, the kinds of feature engineering required differed from what would be relevant when using more conventional modeling tools like sparse logistic regression. For instance, we did *not* need to explicitly materialize interactions between base features. Transforms such as normalization, denser representations, and discretization influenced both the model quality and the training performance.

4 CHALLENGES

The challenges in developing Quick Access arose from three areas: privacy protection requirements, incomplete or ambiguous data, and an evolving product definition. We discuss each of these in turn.

4.1 Privacy

Google's privacy policy requires that engineers do not inspect raw user data. Instead, policy only allows them to work with anonymized and aggregated summary statistics. This was enforced using techniques that only allowed binaries built with peer-reviewed code to have access to user data. Further, all accesses to user data was logged with the source code for the binary and the parameters with which it was invoked. The details of the infrastructure used to implement such policies are beyond the scope of this paper. While these safeguards slowed down the pace of development, it ensured data protection with a high degree of confidence.

As a consequence of the above policy, exploratory analysis, a common first step for any ML pipeline on a new data source, became extremely difficult. This phase is particularly important for identifying features to extract from the tens of thousands of fields that may be stored in the raw log data. Getting a sense for the data, and understanding how noisy certain fields were, and if some fields were missing or recorded in ways that differed from the documentation thus became more complicated.

In addition to these difficulties with initial exploration, debugging specific scenarios and predictions was made very challenging without access to the underlying data. Engineers were restricted to testing hypotheses on aggregate data for unexpected results when these happened to arise, rather than drawing inspiration from their own observations on the data. For instance, early in the development phase, we noticed that for a large chunk of users, the mean time between consecutive *open* events was under two seconds. We were not sure if this was an incorrect logging of timestamps, a double-logging problem for some clients that was bringing down the average, or a true reflection of the underlying data. To solve such problems, and enable some exploratory data analysis, we built additional infrastructure that reported aggregates while respecting k-anonymity [25]. For debugging, one of our most useful approaches was to build tools that allowed team members and colleagues to donate their own data for inspection. Such donated data was only visible to the engineers on the team, and only for a limited duration when we were developing the model. We computed the inter-open time for different mime types and were able to determine that the inter-open time was low when people swiped through consecutive images, leading to the client app pre-loading the next few images, which were getting logged as *open* events. This debugging task would have been much quicker if we were able to inspect the logs for a user who exhibited this behavior. Frustratingly, none of the engineers on the team had much image data in their work accounts, so this pattern did not appear in any of the donated datasets either. We believe that additional research on enabling engineers to build models on structured log data without directly inspecting the data would be valuable in increasing the adoption of ML techniques in a wide range of products.

4.2 Data Quality and Semantics

One of the biggest challenges we faced was the fact that our data sources were not "ML-ready". We were, for example, the first machine learning consumers of the Activity Service described in Figure 2. Since this service was owned by a different team, who in turn gathered data from logs generated by components owned by other teams, no data cleaning had been performed on this data for an ML project before ours.

As discussed in Section 3 the precise meaning of various fields in the logs was critical to determining if the event ought to result in a training example as well as the construction of relevant features. In the early phase, we discovered several documented fields that were not actually present in the logs, because existing users of the Activity Service did not need them. For instance, fields which denoted the mime type of a document for certain classes of events were missing since they were only used to display titles and links in the "Activity Stream" in Drive. Timestamp fields were also a source of concern since some were generated by the client (e.g. the Drive app on Android devices) while other timestamps were generated and logged on the server (e.g. time of request). Finally, since the various fields in the logs came from different systems owned by different teams, occasionally major changes to the values being logged could invalidate the features. For instance, the fields used to determine the type of client on which an event happened may be updated with major changes to Drive APIs invalidating models trained on older data.

In order to deal with data quality problems, we developed a tool that tracked the mean and variance for each of the feature values we gathered in our training data set and monitored these statistics. If we noticed that some of these statistics were off, we were able to tell that the fields underlying the corresponding features were not being correctly populated. This crude and simple check helped us gain confidence in the data sources even when the organization structure made it difficult to control the quality for an ML application.

4.2.1 Training-Serving Skew. In order to test the model in a production setting, we ran an internal *dark experiment* where the Drive client apps fetched our service predictions

for internal Google corpora, but instead of displaying, simply logged them. The app also logged the document that the user eventually opened. We measured the online performance of the model and to our surprise we found that the actual metrics were substantially worse than those that our offline analysis predicted.

Careful analysis comparing prediction scores for candidates in the offline evaluation pipeline and candidates being scored online showed that the distributions were very different. This led us to believe that either the training inputs were different from the inputs being provided to the model in production, or the real-life distribution of the requests was not reflected in the data we acquired.

After further analysis we discovered that the Activity Service produced different results when the data was requested using an RPC versus scanning the Bigtable (our source of training data). For instance, *comment* and *comment-reply* events were treated differently – in the Bigtable, they were treated as different event types, but when calling the RPC endpoint they were both resolved to *comment*. While this sort of discrepancy is pernicious for an ML application, it was a desirable feature for existing applications (e.g. the Activity Stream, which shows both as *comment* events). We discovered several other such transformations yielding different data depending on the access path.

Having discovered this major source of *training-serving skew*, we considered two approaches to tackle it. The first was to locate the list of transformations at the Activity Service and apply the same set of transformations in the data extraction pipeline. However, we quickly realized that this might impose a significant maintenance burden as the Activity Service modified this set of transformations. We instead devised a new pipeline to generate the training data by issuing RPCs to the Activity Service for each user, and replacing the data from the offline dataset with the results from the RPC. While this approach solved the problem of *skew* introduced by the transformations, historical values of features (e.g. total files created by the user) cannot be obtained for scenarios that occurred in the past. To prevent skew in such features, we advocate for gathering all training data at the prediction service (see Section 4.2.2 below).

Once we retrained our models on the newly generated training sets, we observed a marked improvement in the dark experiment. We did continue to observe some skews, largely because there was latency in propagating very recent (within the last few minutes) activity from the logs to the Activity Service. This meant that while the training data contained all events up to the instant of the scenario, at the time of applying the model, the last few minutes of activity were not yet available. We are in the process of re-architecting our data acquisition pipelines to log the data from the Activity Service (and other sources) at the time we serve predictions to be completely free of skew.

4.2.2 Dark-Launch and Iterate. For speculative projects where the data services feeding into the prediction are not owned by the same team building ML features, we advocate the approach of starting a dark experiment early with a basic set of features to gather clean training data and metrics. Even when offline dumps are available, if the data are complex and noisy, and owned by a different team with different goals and features to support, we believe that this approach serves to mitigate risk. Beyond initial feasibility analysis, isolating dependence on offline data dumps can result in higher quality training data, and captures effects such as data staleness, ensuring that the training and inference distributions presented to the model are as similar as possible.

4.3 Evolving Product Definition

As Quick Access evolved, we experimented with several different UIs. These iterations were motivated by both UI concerns and observed predictor performance. The original conception of Quick Access was as a zero-state search, populating the query box with predicted document suggestions. Because any user could conceivably see such predictions, our main goal was to improve accuracy@k (the fraction of scenarios correctly predicted in the top k documents, k being the maximum number of suggestions shown).

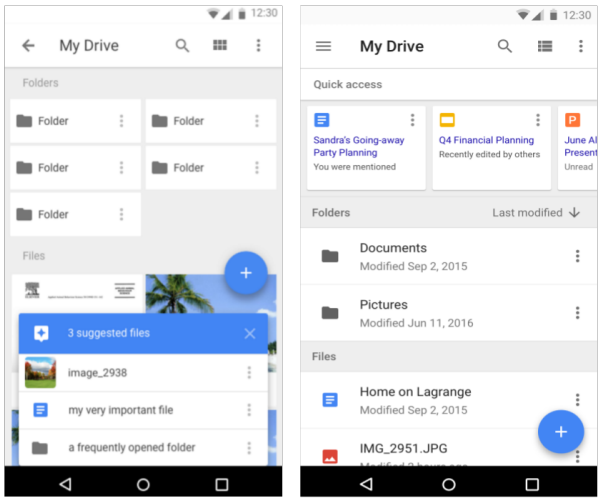
After assembling this UI and testing internally, we discovered that this was a poor user experience (UX). Zero-state search did not serve the bulk of users who are accustomed to navigating through their directory trees to locate files. Users who never used search would not benefit from this approach. We abandoned this UI since it would only help a small fraction of our users.

A second UI was designed to suggest documents proactively on the home screen, in the form of a pop-up list containing at most three documents, as shown in Figure 5 (A). Because suggestions were overlayed, however, we felt that it was critical predictions only appear when they were likely to be correct, minimizing the likelihood of distracting the user with poor suggestions. Consequently, our main metric shifted from accuracy@k to coverage@accuracy = 0.8, defined as follows:

(1) Letting $\mathcal{S}$ be a set of scenarios, define a *confidence function* $f : \mathcal{S} \longrightarrow [0, 1]$, which associates a single confidence score to each scenario. In practice, this was done using an auxiliary function $\mathbb{R}^k \longrightarrow [0, 1]$ computed from the top k prediction scores as inputs.
(2) Find the lowest value $t \in [0, 1]$ such that the accuracy over the scenario set $\mathcal{S}_t := \{s \in \mathcal{S} | f(s) \geq t\}$ was at least 0.8.
(3) Measure the *coverage*, defined as $\frac{|\mathcal{S}_t|}{|\mathcal{S}|}$.

By optimizing this metric, we would therefore be optimizing the number of scenarios receiving high-quality predictions (likely to be correct 80% of the time). The threshold t would be used at inference time to decide whether or not to show predictions.

Despite great progress optimizing this metric subsequent UX studies raised a significant concern. The complexity of the trigger made it difficult to explain to users when predictions would appear. We were worried about an inconsistent UX, both for individual users (suggestions appearing at some times but not others), and across users (people wondering if



(A) (B)

Figure 5: (A) Pop-up style UX considered, and (B) UX at release.

the feature was working if they did not receive suggestions while others were). We eventually settled on the current document list displayed on top of the screen, marking a return to the accuracy@k metric. Each such change required changes to the offline evaluation pipeline, and the ability to adapt to changing optimization objectives.

5 ONLINE METRICS

In this section, we discuss key metrics used to track the impact of Quick Access and how some of the expectations we held before launch did not bear out afterwards.

Since initial analysis of access patterns showed that users often revisit documents that they accessed in the recent past (to continue reading a document, collaborating on a spreadsheet, etc.) we chose the Most-Recently-Used (MRU) heuristic as a baseline. The choice of MRU was natural as it is a commonly used heuristic in many web products (including the "Recents" tab in Drive itself). We also considered a more complex baseline that incorporated document access frequency in addition to recency. Building such baseline would have involved designing a heuristic for optimally combining recency and frequency signals and determining the length of historic time interval for frequency calculation. Instead, we focused our effort on ML feature engineering which included frequency-based features.

Table 2 shows the approximate increase in accuracy@3 numbers the ML model achieved relative to the MRU baseline for users with various activity levels (classified by the logarithm of the number of documents accessed in the recent past).

Tracking performance online, we were later able to confirm online that our ML models indeed outperform the MRU heuristic for all classes of users. Furthermore, the relative improvement over the MRU heuristic was larger for more active users. We believe these users have more complex

User Class	Relative Improvement (ML over MRU)
Low Activity	3.8%
Medium Activity	13.3%
High Activity	18.5%

Table 2: Accuracy@3 for ML model vs. MRU for users with different activity levels

behavior, and are therefore harder to predict. As a result, the ML model out-performs the MRU heuristic by a larger margin for the power users.

We define a few terms common to the discussions below. A *scenario*, as mentioned earlier, denotes the user arriving at the Drive home screen and eventually opening a document. A *hit* is a scenario where the user clicked on a suggestion from Quick Access. A *miss* is a scenario where the user opened a document through other means (for instance, either by navigating, or through search). We use the term *hit rate* to refer to the ratio of the hits to the total scenarios.

5.1 Time-Saved Metric

A key objective for Quick Access is to present the documents a user is likely to need next as soon as they arrive, thereby saving the time it takes for our users to locate the content they are looking for. We measured the mean time elapsed from arriving at the Drive home screen to opening a document for hit scenarios as compared to miss scenarios. These numbers are presented in Table 3 for a 7-day period in 2017, over a representative fraction of the traffic to Drive.

As the data shows, people take half as long to navigate to their documents when using Quick Access when compared to alternatives. We also measured the time elapsed for an open before enabling the UI and observed that the average time taken was not statistically different from the 18.4 seconds reported above. As a result, we believe that faster opens can be attributed to the UI treatment in Quick Access.

5.2 Hit Rate and Accuracy

Pre-launch, our offline analysis indicated that we were likely to produce accurate predictions for a large class of users with non-trivial usage patterns. In order to build more confidence in our metrics, and to provide additional justification to product leadership, we performed a dark launch and tracked accuracy@3. We chose 3 because we expected this number of documents to be visible without swiping on most devices.

We hoped that this number would be close to the engagement rate we could expect once we enabled the UI. After launching, we measured the engagement using the hit rate.

Measure	Value
Average time elapsed for a hit	9.3 sec
Average time elapsed for a miss	18.4 sec
Percentage of time saved in a hit	49.5%

Table 3: Time-saved by Quick Access

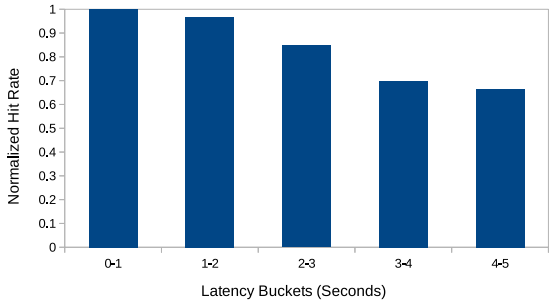


Figure 6: Hit rate for different latency buckets.

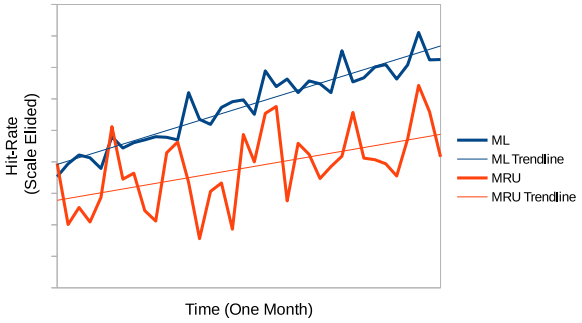


Figure 7: Improvement in hit rate over time

Surprisingly, we found that the actual hit rate was substantially lower than accuracy. Table 4 illustrates the approximate hit rate and accuracy we observed for the ML model as well as MRU the first few days after launching. The metrics are normalized to the MRU accuracy to elide proprietary statistics.

Instead of the hit rate equalling accuracy, it was substantially lower, at approximately half the accuracy. In other words, even when the right document was displayed in the Quick Access UI, users clicked on it only about half the time. Upon further analysis, we came up with two hypotheses to understand and address this anomaly: latency and the absence of thumbnails.

5.2.1 Latency. If the Quick Access predictions took too long to render, users may simply scroll down to locate their document instead of waiting. We tracked the hit rate separately for different latency buckets as shown in Figure 6.

Unsurprisingly, we found that as the latency increased, the hit rate decreased (of couse, accuracy remained unchanged). While the actual task of computing the predictions took very little time, some of the network calls had long tail latencies, especially on slow cellular networks. Based on this, we put in effort to improve the round-trip latency through optimization in the server-side and client-side code. These gradual improvements in latency, coupled with increasing familiarity by users with the feature, resulted in a steady increase in the post-launch hit rate over time even without improvements in model accuracy as shown in Figure 7. As a sanity check, we verified that this was true both for the production model as well as the MRU baseline.

5.2.2 Thumbnails. A second hypothesis to explain the gap between the hit rate and accuracy was the absence of thumbnails. Users used to seeing thumbnails for the rest of their documents might engage less with the new visual elements which displayed the title of the document instead of a thumbnail. In a fractional experiment, we were able to quickly verify that the new UX indeed produced a substantially higher hit rate. As the numbers in Table 5 show, for a fixed accuracy level, the hit rate relative to the accuracy increased from 0.533 to 0.624, an increase of 17%. The UI mock without thumbnails is shown in Figure 5 (B) in contrast to the current UI (Figure 1).

	Normalized Hit Rate	Normalized Accuracy
MRU	0.488	1.0
Model	0.539	1.10

Table 4: Initial Hit-Rate@3 and Accuracy@3

6 RELATED WORK

Quick Access draws on several areas of academic work as well as applied machine learning. Techniques for search ranking over private corpora using ML [6] are closely related. Efforts in products like Google Inbox and Facebook Messenger that try to provide a smart ordering of your private contacts when you try to compose a new message are similar in spirit. However, there are no published studies describing the techniques used and comparing them to heuristics.

Several recent papers attempt to model repeat consumption behavior in various domains such as check-ins at the same business or repeated watches of the same video. This body of work identified several important aspects that are predictive of repeat consumption such as recency, item popularity [5], inter-consumption frequency [8] and user reconsumption ratio [11]. We drew on these analyses to design several of the features that we feed into the neural network.

There is a rich body of work on the interaction of privacy with ML and data mining. Ideas such as k-anonymity [25] and differential privacy [17] have been studied in various data mining [4] and ML settings including for deep learning [2, 24]. In this paper, the problems we faced in building this system were centered around making sure that useful ML models could be developed without visual inspection for data cleaning or feature identification. In contrast to the questions considered in the literature, only the data that is already visible to a given user is used to compute suggestions.

	Normalized Hit Rate	Normalized Accuracy
Old UX	0.533	1.0
New UX	0.624	1.0

Table 5: Impact of UX change on Hit-Rate@3 and Accuracy@3

Consequently, this paper does not focus on issues around preventing private information from being inadvertently leaked through ML driven suggestions.

7 CONCLUSIONS

In building Quick Access, we set out to improve a critical user journey in Google Drive: getting the user to the file they were looking for faster. People get to their documents 50% faster with Quick Access than previously. With ML models leveraging deep neural networks, we improved accuracy over heuristics by as much as 18.5% for the most active users.

In grappling with the challenges this project presented, we learned several lessons we believe will be helpful to the applied ML community. When dealing with data sources that were not initially designed with ML products in mind, we advocate dark launching early to gather high-quality training data. Even when offline data dumps are available, this serves as a strategy to mitigate risks of training-serving skew. We also advocate the construction of tools to monitor aggregate statistics over the features to be alert to changes in the data source.

We believe more academic attention to the problem of building ML pipelines without being able to visually inspect the underlying data for data cleaning and feature identification could help practitioners make faster progress on deploying ML on private corpora.

Finally, having experimented extensively with heterogeneous inputs in deep neural networks, an area which has received relatively little theoretical coverage compared to homogeneous data (e.g. images and video), we feel that the area is ripe for exploration.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 265–283.

[2] Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 308–318.

[3] Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. 2014. A Reliable Effective Terascale Linear Learning System. *Journal of Machine Learning Research* 15, 1 (2014), 1111–1133.

[4] Rakesh Agrawal and Ramakrishnan Srikant. 2000. Privacy-preserving Data Mining. In *2000 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 439–450.

[5] Ashton Anderson, Ravi Kumar, Andrew Tomkins, and Sergei Vassilvitskii. 2014. The Dynamics of Repeat Consumption. In *23rd International World Wide Web Conference (WWW)*. 419–430.

[6] Michael Bendersky, Xuanhui Wang, Donald Metzler, and Marc Najork. 2017. Learning from User Interactions in Personal Search via Attribute Parameterization. In *10th ACM International Conference on Web Search and Data Mining (WSDM)*. 791–799.

[7] James Bennett, Charles Elkan, Bing Liu, Padhraic Smyth, and Domonkos Tikk. 2007. KDD Cup and Workshop 2007. *SIGKDD Explor. Newsl.* 9, 2 (2007), 51–52.

[8] Austin R. Benson, Ravi Kumar, and Andrew Tomkins. 2016. Modeling User Consumption Sequences. In *25th International Conference on World Wide Web (WWW)*. 519–529.

[9] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R. Henry, Robert Bradshaw, and Nathan Weizenbaum. 2010. FlumeJava: Easy, Efficient Data-parallel Pipelines. In *31st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 363–375.

[10] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. 2006. Bigtable: A distributed storage system for structured data. (2006), 205–2018.

[11] Jun Chen, Chaokun Wang, and Jianmin Wang. 2015. Will You "Reconsume" the Near Past? Fast Prediction on Short-term Reconsumption Behaviors. In *29th AAAI Conference on Artificial Intelligence (AAAI)*. 23–29.

[12] Michael Chui, James Manyika, Jacques Bughin, Richard Dobbs, Charles Roxburgh, Hugo Sarrazin, Geoffrey Sands, and Magdalena Westergren. 2012. *The social economy: Unlocking value and productivity through social technologies*. McKinsey Global Institute.

[13] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *10th ACM Conference on Recommender Systems (RecSys)*. 191–198.

[14] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. 2010. The YouTube Video Recommendation System. In *4th ACM Conference on Recommender Systems (RecSys)*. 293–296.

[15] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems 26 (NIPS)*. 1223–1231.

[16] John C. Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12 (2011), 2121–2159.

[17] Cynthia Dwork. 2006. Differential Privacy. In *33rd International Conference on Automata, Languages and Programming - Volume Part II (ICALP)*. 1–12.

[18] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdelrahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, and Brian Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 26 (NIPS)*. 1097–1105.

[20] Quoc V Le. 2013. Building high-level features using large scale unsupervised learning. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 8595–8598.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521 (2015), 436âĂŞ–444.

[22] Steffen Rendle, Dennis Fetterly, Eugene J. Shekita, and Bor-Yiing Su. 2016. Robust Large-Scale Machine Learning in the Cloud. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1125–1134.

[23] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems 29 (NIPS)*. 2503–2511.

[24] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-Preserving Deep Learning. In *22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 1310–1321.

[25] Latanya Sweeney. 2002. K-anonymity: A Model for Protecting Privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 10, 5 (2002), 557–570.