Large Scale Personalized Categorization of Financial Transactions

Christopher Lesner, Alexander Ran, Marko Rukonic, Wei Wang
Intuit Inc., Mountain View, CA

Abstract

A major part of financial accounting involves tracking and organizing business transactions over and over each month and hence automation of this task is of significant value to the users of accounting software. In this paper we present a large-scale recommendation system that successfully recommends company specific categories for several million small businesses in US, UK, Australia, Canada, India and France and handles billions of financial transactions each year. Our system uses machine learning to combine fragments of information from millions of users in a manner that allows us to accurately recommend user-specific Chart of Accounts categories. Accounts are handled even if named using abbreviations or in a foreign language. Transactions are handled even if a given user has never categorized a transaction like that before. The development of such a system and testing it at scale over billions of transactions is a first in the financial industry.

Introduction

A major part of financial accounting involves tracking and organizing of business transactions using a customizable filing system, which accountants call *chart of accounts* (CoA). Every business transaction must be filed into some suitable CoA account. This is a regular and tedious chore for millions of accounting software users. Assuming it takes a human about 3 seconds to pick the right CoA account for a typical financial transaction, last year the users of Intuit's accounting software would have spent well over a thousand man years on just this task if it were not automated by our systems.

Assigning correct categories to financial transactions is important because errors on this task can lead to incorrect financial statements, increased audit risk, tax and other regulatory penalties, misinformed financial decisions and displeased business owners / creditors / investors. For these reasons the reliable automation of this task is of significant economic value for everyone involved: business owners, their accountants, vendors of accounting software, etc.

In this paper we present a large-scale recommendation system that successfully recommends company specific categories for several million small businesses in US, UK, Australia, Canada, India and France and handles billions of financial transactions each year. Our system uses machine learning to combine fragments of information from millions of users in a manner that allows us to accurately recommend user-specific Chart of Accounts categories. Accounts are handled even if named using abbreviations or in a foreign language. Transactions are handled even if a given user has never categorized a transaction like that before. The development of such a system and testing it at scale over billions of transactions is a first in the financial industry.

The rest of this paper is organized as follows: First we classify the problem and identify a candidate solution from related prior research. Next, we discuss what was necessary to adopt this solution to our specific problem in order to build the working system that has been serving millions of Intuit QuickBooks customers for well over a year. We then share results of experiments and our experience with large scale model training and deployment. Here we emphasize the importance of different data representations: one suitable for model training and another for model deployment. Finally, we cover some practical aspects of building a large-scale production system such as dealing with firm real-time latency deadlines, and selecting and optimizing servers for model builds vs. model deployment. Last, we conclude with a discussion of user impact, benefits and what we learned.

Understanding the Problem

Financial accounts track how much money was transferred on a given date with a certain counterparty but unlike an invoice or a receipt, a transaction from a bank or credit card account generally does not have information about items purchased or the services involved.

Despite this, it is still often possible to tell what a financial account transaction is about just by the attributes that are available: (1) the financial institution that recorded the

Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

transaction, (2) the financial account description, (3) the date and time of the transaction, (4) the monetary amount and (5) the counterparty with whom the transaction took place. While transaction counterparty is the most important attribute, it can be challenging to extract from financial account transaction descriptions. This is because these descriptions are strings of unknown structure that may contain a mix of (1) counterparty name, (2) transaction location, (3) time/date, (4) payer identity, (5) transaction reference ID #s, (6) payment method and (7) other details. The structure and content of these descriptions may also depend on the specific route from the point of sale to the bank that issued the payment card.

A billion transactions with about ten million unique counterparties yield about four hundred million unique transaction descriptions and after normalization (case folding, digit folding, etc.) this reduces to about one hundred million unique descriptions. As a consequence the transactions of a single grocery merchant (at one location) appear in well over 300 different formats. Fig 1 shows a few examples.

SAFEWAY STORE 00000000

SAFEWAY STORE 00000000 SAN FRANCISCOCA

SAFEWAY STORE 00000000 SAN FRANCISCO CA

SAFEWAY STORE 00000000 SAN FRANC

SAFEWAY STORE 00000000 SAN FRANCISCO CA 00000 US

CHECK CRD PURCHASE 00/00 SAFEWAY STORE 00000000 SAN FRANCISCO CA 000000XXXXXX0000 000000000000000 ?MCC=0000 00

CHECK CRD PURCHASE 00/00 SAFEWAY STORE 00000000 SAN FRANCISCO CA 000000XXXXXXxxxxxxxxxxxx000 ?MCC=0000 000000000DA00

SAFEWAY STORE 00000000 SAN FRANCSAFEWAY STORE 00000000 SAN FRANCISCO00000000000 000 Oct 00 @ 0:00pm

SAFEWAY STORE 00000000 SAN FRANCISCO, CA 00.00 USD @ 0.000000

PURCHASE
SAFEWAYSTORE 00000000 SAN FRANCISCO CA

SAFEWAY STORE 00000000 - SAN FRANCISCO, CA Reference Number:0000000000000000000000 Merchant Name: SAFEWAY STORE 00000000 Merchant Information:SAN FRANCISCO CA Category: Retail/Department Stores

PURCHASE/ADVANCE (CHG-SAFEWAY STORE 00000000 SAN)To Principal: $00.00

SAFEWAY STORE 00000000POS PURCHASE MERCHANT PURCHASE TERMINAL 000000 SAFEWAY STORE 0000 0000 SAN FRANC CA 00-00-00 XXXXXXXXXXXX0000

Figure 1: Examples of transaction description variability.

The State of Practice

It is possible to think of categorization of financial transactions as a supervised classification problem. There are billions of transactions that have been categorized by small businesses in the past that can be used as labeled data and so a training and test set can be constructed for supervised learning.

Unfortunately, learning from 10^9 examples (categorized financial transactions) how to classify on the order of 10^8 unique items (counterparties identified from transaction descriptions) into 10^6 (distinct accounts) is not likely to result in a useful system simply because there is not nearly enough data.

One could instead construct company specific classifiers using as a labeled set the transactions that the business owner has categorized in the past. Unfortunately, that also does not change fundamentally the fact that there is not enough data to achieve good coverage for future transactions. Specifically, using historical data we know that on average about 50% of the transactions are with new counterparties. This means that even if future transactions with the same counterparty are always categorized correctly and new counterparty transactions are assigned to the most popular account, the overall accuracy of such a classifier would not exceed 60% in the common case.

Nevertheless company-specific classifiers are useful and that is the state of the practice today among other vendors of accounting software. While this simple approach solves the easy part of the problem it is possible to do much better by properly integrating the knowledge of how different businesses categorize transactions into their own personalized CoA.

Personalized Tag Recommendation

A well-studied problem that has a similar structure is known as personalized tag recommendation (Hu, Lim, and Jiang 2010), (Abel et al. 2011), (Steffen et al. 2009). A typical use case involves users tagging a collection of resources for social sharing (Jaeschke et al. 2007) such as for example when users collaborate to tag websites, news or research articles, photos, etc. (Sigurbjornsson and Van Zwol 2008).

Unlike recommender systems that rely on a single shared vocabulary as labels for resource categories, with personalized tag recommendation users can have their own, unique vocabulary of tags with new ones created when necessary.

Personalized tag recommenders answer the question: “Which personal tag of this user is most likely to apply to this resource given the set of tags that have been assigned to this resource by other users?” In financial account domain this corresponds to the question: “Which CoA account of this user is most likely to apply to this financial transaction given the set of accounts to which this transaction has been assigned by other users?”

A common approach to such questions is based on normalized tag co-occurrence frequencies in resource annotations using for example Jaccard index over tags when applied to the same resource.

For example, if a_i and a_j are two tags that have been applied to a number of different resources, their similarity

can be expressed as the ratio between intersection of the corresponding resource sets to their union:

$$J_{ij} = J\left(a_i, a_j\right) = \frac{\left|a_i \cap a_j\right|}{\left|a_j \cup a_j\right|} \quad (1)$$

where $|a_i|$ stands for the set of resources to which tag a_i has been applied. Therefore, the problem of CoA account recommendation can be thought of as a tensor:

$$\mathbf{U} \times \mathbf{I} \times \mathbf{T} \quad (2)$$

where **U** stands for the users, **I** for set of items and **T** for the set of tags. A popular solution uses tensor factorization, as described for example in (Steffen et al. 2009).

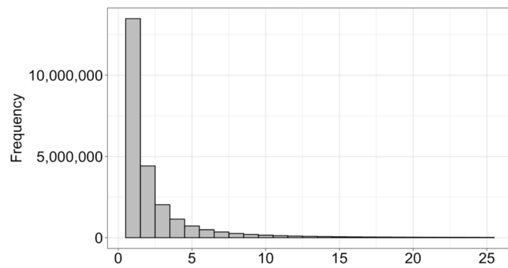

Figure 2: Counterparties per account

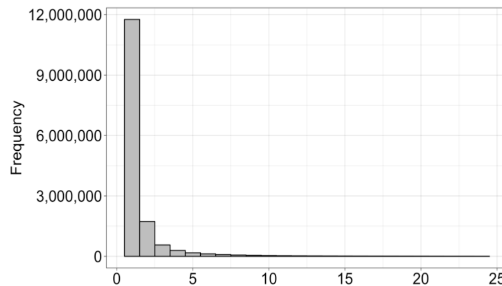

Figure 3: Accounts per counterparty.

Personalized CoA Recommendation

Personalized tag recommendation had to be adopted to the scale and structure of our problem domain:

1. **Domain Scale**: While in a typical social tagging system 10^6-10^8 of items (resources) are tagged by 10^5-10^6 users using 10^4-10^5 unique tags, personalized transaction categorization handles 10^7 counterparties (items) for 10^6 users with 10^8 accounts (tags). This 10^3-10^4 difference in scale requires careful consideration of representation to maintain feasibility of even the most effective of currently known approaches.
2. **Domain Structure**: An important observation to be made is that the number of distinct accounts exceeds the number of distinct counterparties by a factor of 10-10^2. At the same time the distribution of counterparties per account and distribution of accounts per counterparty have a definite asymmetry. See Fig.2 and Fig.3.

As a result of scale difference and asymmetry of distributions it is much more efficient to collect counterparty co-occurrence statistics over accounts than account co-occurrence statistics over counterparties.

Account Likelihood Ranking Model

Personalized transaction categorization assigns transaction t_i to account a_j according to maximum likelihood given company specific CoA accounts and the transactions that have been assigned to these CoA accounts so far.

For users who have already categorized transactions, we use their previous counterparty => CoA accounts assignment to guide future counterparty assignments – that is, if the exact counterparty had been categorized before by a given user, then their last used CoA account is our top recommendation. Otherwise we recommend the CoA account with the collection of counterparties having the highest co-occurrence with the transaction that is being classified.

More formally this can be described as follows. Let each counterparty be represented as an n-dimensional vector of its normalized co-occurrence with other counterparties:

$$\vec{T_i} = (J_{1i}, J_{2i}, \dots, J_{ii}, \dots, J_{ni}) \quad (3)$$

Where:

$$i = 1, 2, \dots, n$$

$$0 \leq J_{ij} \leq 1$$

Then the likelihood for a counterparty to be categorized into an account is given by:

$$P\left(\boldsymbol{a}_i \middle| \boldsymbol{t}_j\right) \propto \sum_{\boldsymbol{T}_k \in \boldsymbol{a}_i} \boldsymbol{J_{jk}} \quad (4)$$

The final prediction then amounts to selecting n accounts with the highest co-occurrence score.

We tested several different measures of counterparty co-occurrence and evaluated their performance with a validation dataset.

One measure for counterparty co-occurrence is Kulczynski similarity index:

$$J_{ij} = \frac{1}{2}\left(\frac{\left|t_i \cap t_j\right|}{|t_i|} + \frac{\left|t_i \cap t_j\right|}{\left|t_j\right|}\right) \quad (5)$$

Another is Jaccard index:

$$J_{ij} = \frac{|t_i \cap t_j|}{|t_i \cup t_j|} = \frac{|t_i \cap t_j|}{|t_i| + |t_j| - |t_i \cap t_j|} \quad (6)$$

where $|t_i|$ is the number of accounts that have transactions with counterparty t_i ; $|t_i \cap t_j|$ is the number of accounts that have transactions with both counterparty t_i and t_j ; $|t_i \cup t_j|$ is the number of accounts that have transactions with either counterparty t_i or t_j.

Jaccard index being not null invariant is affected strongly by asymmetry in frequencies of counterparties. That is, the Jaccard similarity of a common counterparty with an infrequent counterparty approaches 0 even though the two counterparties are very likely to co-occur in the same account. In other words, Jaccard index loses information from and for common counterparties.

The Kulczynski similarity index on the other hand is null-invariant and preserves information even in the case of asymmetric counterparty frequencies. In our experiments however, we found that using Jaccard index gave us a slightly better categorization accuracy. One explanation is that our strategy gives preference to accounts that already contain transactions with the given counterparty. If the user has already categorized a given counterparty before, that previous account assignment will be re-used as the prediction when this counterparty needs to be categorized again.

That is for each user k,

$$J_{ii} = \begin{cases} 1, & \text{if user } k \text{ categorized } t_i \text{ before} \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

This strategy likely applies to transactions with a common counterparty thus masking the lack of null-invariance of the Jaccard index.

Though the counterparty co-occurrence matrix $(\vec{T}_1, \vec{T}_2, \dots, \vec{T}_n)^T$ could be in the order of 10^7 x 10^7, it is very sparse and can be efficiently reduced to fewer than 10^9 non-zero elements, making real time account scoring relatively efficient.

In the real-time prediction, given a transaction t from user k, for every account a_i which had transaction history, we can calculate:

$$J_{a_i} = \sum_{m \neq t} \hat{J}_{tm} \quad (8)$$

where $\hat{J}_{tm}$ is the Jaccard index between counterparty t and m (m denotes all other counterparties that are coupled with counterparty t). And finally choose the account as:

$$a = \underset{a_i}{\operatorname{argmax}}(J_{a_i}) \quad (9)$$

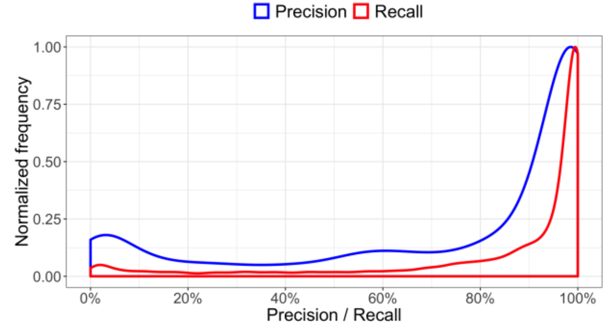

Figure 4: Distribution of precision and recall at individual account level for transaction categorization.

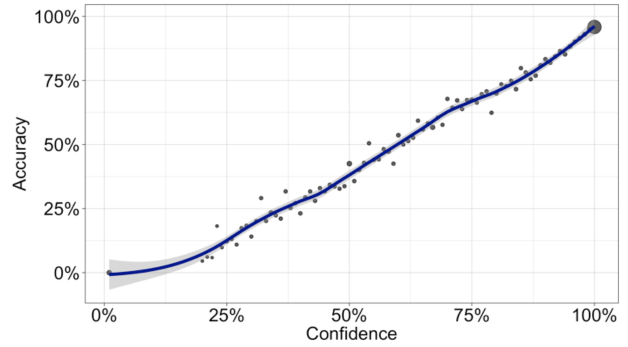

Figure 5: Categorization accuracy vs. overall model confidence. Larger dots represent more transactions at given confidence.

Results from Experiments

Experiments were carried out using a sample of 10^9 financial account transactions matched with their user assigned CoA accounts.

Datasets were selected by full calendar month:

1. **Training**: 12 months of transactions were used to generate the counterparty co-occurrence matrix
2. **Validation**: the month of transactions right after training was used to calibrate model confidence predictions.
3. **Test**: the month of transactions right after validation was used to measure model performance.

Three metrics were tracked:

1. **Accuracy**: automatically categorized transactions accepted by users without changes divided by the total number of transactions imported by the users from financial accounts.
2. **Precision**: number of transactions being categorized correctly divided by the total number of predicted transactions in a given user's account.
3. **Recall**: number of transactions being categorized correctly divided by the actual total number of transactions in a given user's account.

As shown in Fig. 4 we care about the distribution of precision and recall, rather than just precision / recall as a single number. Measured on the test set the model performs consistently with accuracy above 70% having both high precision and recall across users' CoA accounts.

Model Confidence Prediction

Communicating prediction confidence to our users through the UI allows them to build trust in the CoA account recommendations they review. The idea being that when users witness that high confidence predictions are almost never wrong, their review of these transactions can be quick and more of their time can be invested reviewing and correcting just those transactions with low confidence CoA recommendations.

Our model's confidence is predicted in two ways:

1. For **counterparties previously categorized**, we predict confidence using a linear function of the duration since the last time a transaction with the same counterparty was categorized by this same user. The intuition being that the more recently a given user categorized a given counterparty the more likely they are to do it the same way again. For our datasets a simple linear function fits our validation results reasonably well.
2. For **counterparties not previously categorized** model confidence is predicted using the ratio of the likelihood of most appropriate account vs. alternative accounts.

Fig. 5 shows that the confidence of our recommendations is highly correlated with the accuracy of recommendations measured from user feedback. As intended, better recommendations have higher predicted confidence and users can indeed trust them more.

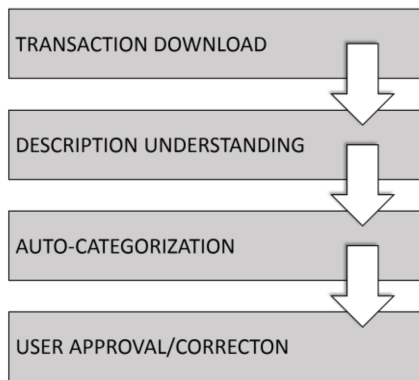

Figure 6: Stages of financial transaction processing.

How the Model is Used

QuickBooks offers users the ability to connect their financial accounts (banks / credit unions / investment / etc.) to download transactions. What happens next is illustrated in Fig 6. Upon download, each transaction undergoes analysis to understand what it represents (withdrawal/deposit, purchase/income, loan payment or disbursement, money transfer, fee, etc.) and who the transaction is with (who the counter-party is). Next our account likelihood ranking model is applied and transactions are tentatively filed (auto-categorized) with respect to each user's CoA filing system[1]. In the final step users get an opportunity to accept/correct how their transactions have been filed and their corrections are used to update the account likelihood ranking model next time it is rebuilt.

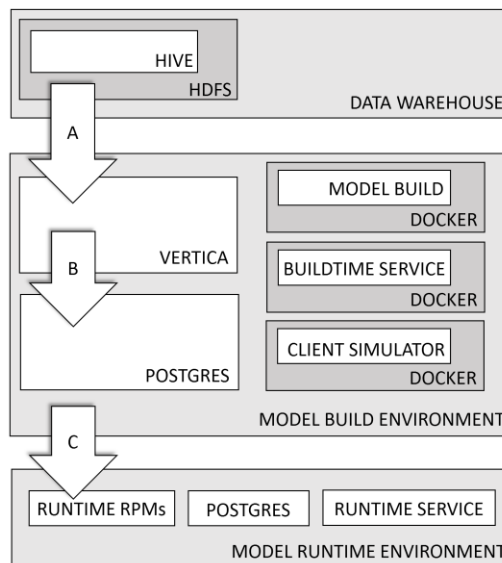

Figure 7: Model Build Environment.

How the Model is Built

To keep production models fresh (and EU GDPR compliant) we regularly rebuild them. This process has three main steps as shown in Fig 7.

Data Extraction

Model builds start with extraction of just the table columns that pertain to financial account transactions and CoA accounts. From a data warehouse these columns are transferred to Vertica (Fig 7 step A) where additional projec-

[1] Although accounting standards apply to CoAs, in practice what each business tracks for accounting varies. Many users have distinct accounts for their various business locations, rental properties, business vehicles, etc. Consider how different a flower shop is from a law office from a cement factory. CoAs also vary because what users are legally required to track for government reporting varies between jurisdictions (county / state / country – QuickBooks has users worldwide), and even in the same jurisdiction CoAs can be impacted by new laws. For example, when tax law changes what is and isn't deductible, impacted business may have accounts in their CoA that need to be split or new ones created.

tions are added so that our model build data access patterns are sequential (*Knowledge Representation* section explains why this is important).

Model Build

The model build (computing the counterparty co-occurrence sparse matrix – from here on also called the **coupling table**) is carried out in Vertica as controlled by a Python orchestration service. Once model tables are created they are transferred from Vertica to Postgres (Fig 7 step B) – in this step our knowledge representation is switched from column store to row store. (See *Knowledge Representation* section).

Model Acceptance Testing

After model data is in Postgres an instance of the build time model service is started, and a model service client simulator is launched for model acceptance testing – it replays a month of transactions. Model coverage and accuracy metrics are tracked and the model build is halted unless these metrics have acceptable values. On successful test completion the model is compressed into RPM package files for distribution (Fig 7 Step C). The final step is to install the RPMs on a node having hardware matching what is used in production and to again launch the client simulator to replay transaction history this time however for model latency acceptance testing. Model acceptance testing is split like this for two reasons:

1. Latency tests are not reliable unless they are performed using OS and hardware matching production runtime environment. (Further explained in *Firm Real-Rime Deadlines* section.)
2. Model coverage and accuracy tests do not need production hardware so these tests are launched right away. If there is a model accuracy or coverage drop (due to for example a change in some upstream system that we do not control) automated tests catch this early.

Firm Real-Time Deadlines

Some transactions involve merchants coupled to a small number of other merchants. These are quick to classify especially when the merchants involved are popular. Other transactions involve merchants weakly coupled to hundreds of merchants or to merchants which are relatively rare. Such transactions take longer to classify because each extra merchant requires a new b-tree index search and the more obscure the merchant, the lower down in the cache hierarchy the coupling table entries for that merchant are likely to be.

On busy production servers popular merchants are likely cached in RAM or even CPU, obscure merchants however may be in parts of the index not cached. For this reason some transactions can take 10^2 times longer to categorize than others.

Due to this variability models must be latency tuned to operate under firm real time deadlines. Deadlines are firm because failing to show users their transactions on time is far worse than if these transactions are missing account recommendations.

Model Latency Tuning

Latency tuning involves pruning those entries from the model tables which are least likely to influence recommendations. Values that are tiny for example are unlikely to make a difference.

With coupling tables smaller, fewer b-tree search steps are needed and a larger portion of the coupling table b-tree index can be cached so index searches are shorter and faster. Yet small coupling tables contain less information and as coupling table size is reduced model coverage and model accuracy both suffer.

During latency tuning we adjust this tradeoff between model latency (due to coupling table size) and model coverage/accuracy. Our goal in latency tuning is to make sure models rarely if ever exceed firm real-time latency deadlines. If a deadline is missed, account predictions are late they cannot be used; late predictions – even if correct – are always counted as being incorrect.

Tuning coupling table sizes for latency also requires that the tuning process sends transaction requests that are representative of what happens in production:

1. transaction counterparties must be as diverse
2. counterparty order should be representative

Latency tuning with just a few transaction counterparties is misleading because after the first hit the coupling table entries associated with these are now high up the cache hierarchy. A similar cache effect occurs even if you use all possible counterparties but fail to mix up their order. To avoid both of these problems, we tune models by using a sequence of requests that plays back actual production model usage from history.

Build vs. Runtime Servers

Our model is regularly refreshed to reflect changes in the real world and comply with regulations such as GDPR. To enable regular and timely model updates the build process has to be performance optimized as well. However, the characteristic patterns of data access during model training are quite different from the interactive context at runtime:

1. **Model build servers** are selected and optimized for sequential large IO throughput. These have RAID10 with small chunks and wide stripes (~12 HDDs work in parallel). Filesystems are created with large records and OS scheduler policies are set to favor

throughput over latency. Four+ CPU socket servers with NUMA work well.

2. **Model runtime servers** are selected and configured to maximize number of small IO operations per second (IOPS). RAM is maximized and SSDs are used for model data storage. The file system holding the model is created with small records, and OS scheduler policies are set to favor latency over throughput. We avoid NUMA due to RAM latency overheads it can impose.

Model runtime servers are dedicated for just one task, so no other process competes for IO or cache. Virtual memory / swap are either disabled or model process memory is locked to prevent being swapped out. This is all done so that once a classifier node is running model response latencies stay low and predictable.

Knowledge Representation

We represent knowledge differently when building models vs. when using them:

1. During model builds knowledge is represented inside a column store database (Vertica) using projections in a de-normalized format with the same data stored in various sort orders so access is sequential, cache friendly and takes advantage of efficient column-wise compression boosting effective IO throughput.
2. For model deployment knowledge is represented using tables in a row store database (Postgres). Here tables are stored clustered on their primary key and additional b-tree indexes are built such that the need to access data beyond what is indexed is rare (“index only scans”).

The reason for this difference is twofold:

1. During model builds the data access patterns are known in advance so in-memory and on-disk layouts of data can be optimized for cache hierarchy locality. However, when the model runs in production we do not know in advance which users, accounts and counterparties will be involved in any incoming request hence our knowledge representation must be optimized to answer any request quickly.
2. When the model runs in production there is a firm real time latency deadline -- requests must be handled in milliseconds because users are waiting; latency concerns dominate over throughput concerns. On the other hand, when a new model is being built users are not waiting so latency is not a concern and instead throughput concerns dominate because they drive model refresh cost.

Model Deployment

Our model operates as a web service API deployed using a cluster of identical classifier nodes all behind a load balancer as illustrated in Fig 8. Incoming requests first go to the load balancer which then forwards the request to an available classifier node. If the continuous load on the least busy classifier node is too high, additional classifier nodes are added. If the continuous load on the busiest classifier node falls, the oldest classifier node is removed from the load balancer pool and stopped. If a classifier node malfunctions (e.g. timeouts on requests) the load balancer automatically replaces it with a new node thus healing the service. This healing functionality is also used for zero downtime upgrades such as when fresh models are deployed – old classifier nodes are purposefully killed one at a time and the load balancer replaces them with upgraded versions.

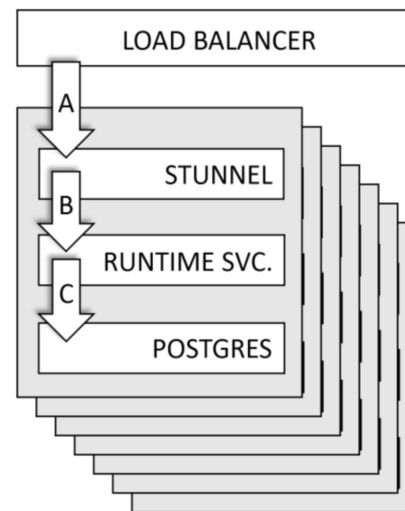

Figure 8: Model Runtime Environment.

We use a shared nothing architecture because it makes service deployment and scaling straightforward. For example, when the number of incoming requests doubles the number of running classifier instances is doubled. When the number of incoming requests drops in half the number of running classifier nodes is dropped in half. The ratio is not quite exact because classifier node startup takes several minutes so a number of classifiers nodes are always kept around to handle spikes in demand while new nodes are created.

User Impact and Benefits

The described ML-based categorization service was first deployed to production for English language QuickBooks (USA, Canada, UK: 2+ million users) in November 2016.

Non-English global regions (France, India, etc.) were added in August 2017. Compared to the legacy systems that it replaced, the new ML service classifies more transactions (even those that are hard to classify) and at the same time it does so at a higher accuracy.

Specifically, in direct A/B comparison tests the ML service results in 56% fewer uncategorized transactions and 28% fewer errors. This represents a remarkable reduction in the number of errors that must be corrected and in the amount of manual work millions of QuickBooks users have to do to file their financial transactions.

For a sense of scale, if – without automation – it takes 3 seconds to choose the right account for a financial transaction then last year the users of Intuit's accounting software would have spent well over 1,000 man years on this task.

Conclusions

We have presented a new method for automatic categorization of financial transactions for small business accounting.

We shared lessons learnt with respect to differences in data access patterns during model training and runtime production deployment. We explained how these differences can be effectively supported by adopting column-oriented data store for model training and row-oriented store for runtime deployment.

We also discussed the requirements related to real-time constraints on model runtime performance and suggested ways to satisfy such constraints.

Our system has been deployed at scale and it handles billions of financial transactions for millions of small businesses each year. Our solution combines fragments of information from millions of users in a manner that allows us to accurately recommend user-specific Chart of Accounts categories. Accounts are handled even if named using abbreviations or in a foreign language. Transactions are handled even if a given user has never categorized a transaction like that before. The development of such a system and testing it at scale over billions of transactions is a first in the financial industry.

Our work greatly benefited from research in machine learning for personalized tag recommendations and tensor factorization.

Our ability to scale the system relies heavily on training and deploying the model on top of mature database technology that also supports efficient regular model updates necessary to adapt to new merchants, new small businesses and constant change in business and accounting practices.

References

Abel, F.; Araujo, S.; Gao, Q.; and Houben, G. 2011. Analyzing Cross-System User Modeling on the Social Web. In *Web Engineering: 11th International Conference*, 28-43. Springer-Verlag, Berlin.

Jaeschke, L. B.; Marinho, A.; Hotho, L.; Schmidt-Thieme, L.; and Stumme, G. 2007. Tag recommendations in folksonomies. In *Pro-ceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 506-514. Lecture Notes in Computer Science, vol 4702. Springer, Berlin, Heidelberg.

Hu, M.; Lim, E.; and Jiang, J. 2010. A Probabilistic Approach to Personalized Tag Recommendation. In *2010 IEEE Second International Conference on Social Computing*, 33-40. IEEE Computer Society Washington, DC, USA.

Sigurbjornsson, B., and Van Zwol, R. 2008. Flickr tag recommendation based on collective knowledge. *In Proceedings of the 17th Inter-National Conference on World Wide Web*, 327-336. ACM New York, NY, USA.

Steffen, R.; Marinho, L.; Nanopoulos, A.; and Schimdt-Thieme, L. 2009. Learning optimal ranking with tensor factorization for tag recommendation. In *KDD '09: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 727-736. ACM New York, NY, USA.