An Open-Source Recipe for Real-Time, Adaptive Classifiers in a Planetary-Scale Network

Chris Kappler
Facebook, Inc.
Traffic Infrastructure / CDN / Boston
ckappler@fb.com

Dejan Curcic
Facebook, Inc.
Traffic Infrastructure / CDN / Menlo Park
curcic@fb.com

Abstract

This paper demonstrates how to use open-source software to deploy a set of adaptive classifiers that satisfy the unique demands of Facebook's Traffic Infrastructure. We combine techniques from sub-disciplines of Machine Learning (ML), Service-Oriented Architecture (SOA) and control systems to embed an ML microservice tier that supports multiple simultaneous classifier models, tolerates version skew in clients and servers, and adapts classifier precision and recall in real time to match operational conditions such as network congestion and available cache space.

1 Embedding Machine Learning in the Network

The load balancers, caches, and optical networks that make up Facebook's Traffic infrastructure form a unique set of challenges for deploying ML classifiers. The hardware and software run under very high load, serving hundreds of billions of latency-sensitive requests for the people using our services each day. This infrastructure is shared over multiple products including Oculus, WhatsApp, Instagram, and the web and mobile versions of Facebook. These services are always-on and are maintained through a process of *continuously deployed* binaries and configurations. Finally, the traffic infrastructure is distributed in thousands of cities around the world, with unique physical and technical constraints in each location.

The contributions of this paper can therefore be summarized as a low-cost approach to satisfy the requirements of this environment, including:

- Support for continuous deployment and A/B testing via robust version-skew handling
- Extrinsic adaptation of the classification threshold based on operational conditions
- A multi-tenant classifier microservice with per-product features and classifiers
- Emphasis on low-latency classification (e.g. 1% to a networking Round-Trip-Time)
- Emphasis on low-cost classification (e.g. less than 5% load to existing compute clusters)
- Operational safe-guards including an easy-to-find "Off Switch"

2 Adapting ML Classifiers as an SOA Microservice

A service refers to a client/server application running on one or more machines and conforming to a service description. This paper demonstrates how to bring the robustness, maintainability, and deployability that are expected of production services to systems using open-source ML libraries, whose typical use cases are biased towards offline analyses in the data warehouse.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

```
// Thrift Struct
struct Work {
  1: i32 num1 = 0,
  2: i32 num2,
  3: Operation op,
  4: optional string comment,
}
```

```
service Calculator extends
      shared.SharedService {
  i32 calculate(
    1:i32 logid,
    2:Work w,
  ) throws (1:InvalidOperation ouch),
}
```

Figure 1: A Thrift Structure and Service Declaration from the Thrift Tutorial [17]

We start by adopting an Interface Description Languages (IDL) like Apache Thrift [18], Google's Protocol Buffers [16], or Microsoft Bond [10]. Each of these libraries benefits from years of experience with service-oriented architectures deployed in their respective companies.

One key feature of these IDLs is their support for *version skew* for continuous deployment environments where clients and servers must continue to operate correctly in spite of multiple versions of each being active at any time.

This IDL syntax prefixes the member variable of structures and method signatures with an integer identifier, in addition to its name and type, so that clients and servers can easily detect which version of a thrift interface is being used by the other.

2.1 Allowing Version Skew in a Classifier Service

Machine Learning and Data Mining libraries like WEKA do not typically support concepts like version skew. In order to deploy a robust classifier microservice, these libraries needed to be augmented with this capability.

WEKA libraries can import training and test data in CSV format or using its `.arff` file format. CSV files do not contain any kind of implicit schema to describe the types of the different columns of data. WEKA's `.arff` file format has an in-file header to achieve this, but with multiple files from multiple sources, the in-file header is repeated in each file and creates incompatibilities more often than not.

The ideal is to have a shared schema, rather than one copy of the schema in each file. Our ML service framework creates a WEKA `.arff` containing only a single Instance row in addition to the schema header. Subsequently, for each training generation of the model, this `schema.arff` file is revised in a reverse-compatible way by extending the lists of attributes and symbolic values for each nominal attribute.

2.2 Robust Processing of Instance Attributes

In addition to sharing a data-type description over multiple files and over multiple training cycles, a classification service can use its schema to protect a model from client requests that are incompatible with that schema. This can occur cases where client includes an attribute that is not yet part of the trained model, where the client does not include an attribute that the model is expecting, or where the client uses an unknown value for a nominal attribute.

Some classifiers allow *empty* or *missing* attributes in both training and in classification. Naive Bayes and BayesNet in particular offer a graceful handling of missing values, both in training and in classification, by operating on only the sub-set of attributes that are available and ignoring the missing ones.

The general recommendation for missing values is based on the *significance* of the value's absence. If the absence of a value has significant predictive value, then missing values should be marked as a placeholder value such as -1 for numerics or "missing-value" for nominals.

Since, in all of the missing-value cases above, there is no predictive significance to version skew, then the best classification results will be obtained by using one of the classifiers that can operate on a sub-set of attributes.

```
Instance inst = (Instance)schema.instance(0).copy();
for (int ai = 0; ai < schema.numAttributes(); ai++) {
    Attribute a = schema.attribute(ai);
    Object val = jsonReqestInstance.opt(a.name());
    if (Objects.isNull(val)) {
        inst.setMissing(ai);
    } else {
        boolean success = convertTypeAndUpdateInst(inst, a, ai, val);
        if (!success) {
          inst.setMissing(ai);
        }
    }
}
```

Figure 2: Example WEKA Code for Normalizing an Input Instance to a Schema

2.3 Hosting Multiple Models and Versions in a Multi-tenant Classification Service

Since Facebook's traffic infrastructure is shared over multiple product families that respond to different sets of HTTP requests, each product family can potentially run in its own server pools. For example, Facebook www backend runs in PHP while Instagram's backend is implemented in Python/Django. This means that different product families will have access to different feature attributes for classifying their content.

The product family has a significant impact on the costs and rewards of a classification decision. For example, instant messaging is more sensitive to the latencies, but there is also a lower risk of the receiver ignoring the selected content. This yields higher rewards and lower expected costs for a speculative caching decision.

Finally, there is the practical issue of separation of concern and per-product isolation. By letting each team develop their own predictive models and roll them out independently of each-other, a large engineering organization can achieve a higher feature velocity and each team can operate with focused models of their own system's needs.

To handle this, the classifier service obtains a list of pre-trained WEKA classifier models that it can use to respond to client Thrift requests. By importing immutable, thread-safe models in the java code, the classification service can support the multiple high-performance execution threads assumed by the auto-generated Thrift server code. Each classification request specifies a name of the trained model that is meant to handle it, so at any time, there may be tens of models and hundreds of requests running in a single server.

3 Characterizing a Classifier Model for Operational Thresholds

ML classifiers are used in systems that have some tolerance of false positives and false negatives. However, that tolerance can change based on operational conditions. For this reason, we require an ML classifier service that can deliver tunable degrees of certainty at different times of day or when operating under specific conditions.

To achieve a tunable classifier service, we utilize WEKA classifiers that are capable of outputting probabilities, such as BayesNet. Given these probabilities, we characterize a trained model by the precision and recall that it can deliver on the test set for a every classification threshold.

The chart in Figure 3 shows the cumulative percentage of true positives and true negatives that could be detected for each classification threshold between 0 and 1. This chart is obtained by sorting each instance in the test set by the probability value returned by applying the trained binary classifier to it. This chart extends the concepts of an ROC curve [6, p.186] to show three dimensions rather than two.

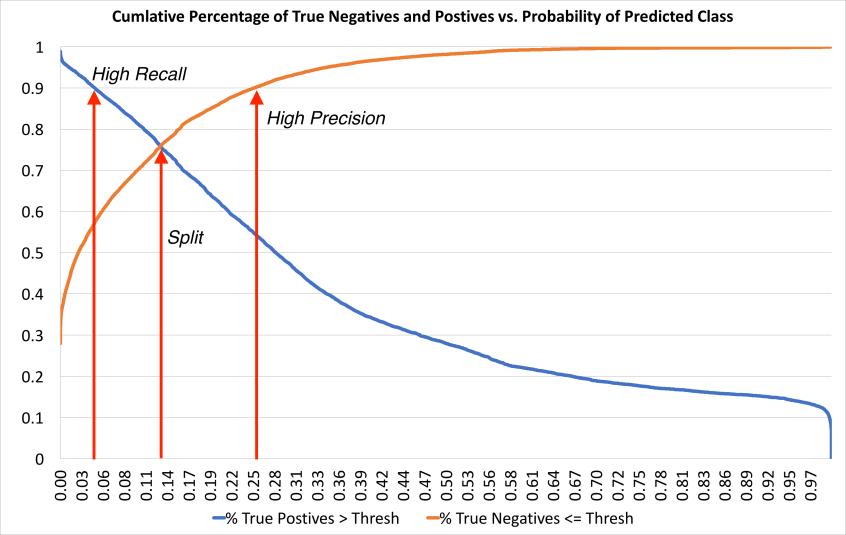

Figure 3: Counting True Positives and True Negatives in the Test Set

3.1 Tunable Classification Threshold vs. Cost-Sensitive Learning

Since our goal is to deploy lightweight ML classifiers into high-load, high-value, operational systems, such as Facebook's Traffic infra, certain design trade-offs become biased towards operational transparency.

For example, we could have used a single classification threshold and trained the model to avoid wasting resources during congestion. This would be done by adding endpoint congestion levels into the feature attributes and providing heavy weights to any negative instances that wasted bandwidth during high-load periods. Instead, we chose to use operational constraints, like congestion to select a tunable classification threshold.

There are a few benefits to this in the production deployment. First, machine learning offers statistical, not absolute guarantees. Though the system might be trained to back off and protect the network, we don't have any way to guarantee that the training data will always be completely sufficient to insure this. Second, since all models need to respect operational parameters, keeping them extrinsic, rather than learned, simplifies multiple models. Finally, the explicit control serves as an easy-to-find "Off Switch" for the speculative behavior in the system. This allows operational teams to troubleshoot systems and eliminate potential sources of uncertainty when handing a critical issue.

3.2 Selecting Threshold Values

From the chart we can detect the obvious intersection of true positives with true negatives at a classification threshold of 0.13. Moving to the left one could select a hypothetical high-recall classification threshold at 0.04 that would accurately label 90% of true positives. Likewise, moving to the right, one could select a high-precision classification threshold of 0.25 that would accurately label 90% of the true negatives.

The exact precision and recall would depend on how many positive vs. negative instances exist in the test set, and the cost of operating at any one of these thresholds would be a function of the cost of a false positive. Though our system can tolerate high false positive rates under some circumstances, not every system will chose its thresholds around the intersection point as we did.

3.3 Running a Real-Time Control System to Drive Per-Tier Precision/Recall Selection

Once we have multiple classification thresholds to chose from for a trained model, a second microservice can be created which monitors key parts of a planetary-scale networking infrastructure

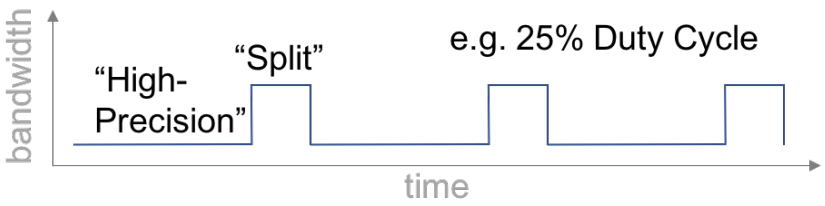

Figure 4: Oscillating Between Two Thresholds

and adapts the classification tier to current operational conditions. At Facebook, one of the principal tools for monitoring live systems is called ODS [14][20].

Given such monitoring tools, the services and tiers that will be most effected by the output of the classifier can be tracked in real time. For each monitored tier or cluster, the tuning microservice can maintain a target classification mode at one of 5 levels:

- Open mode defaulting to true classification and no false negatives
- High-Recall mode biased towards false positives
- Split mode erroring equally towards false positives and false negatives
- High-Precision mode biased toward false negatives
- Closed mode defaulting to false classification and no false positives

The specific values of the thresholds are not crucial since a control system such as a simple PID controller [1] will often oscillate between two thresholds yielding a hybrid threshold. In the example in Figure 4, a system that spends one-quarter of its time at the "Split" threshold filtering 76% of false negatives and three-quarters of its time at the "High-Precision" threshold filtering 90% of false negatives will yield a combined filtering rate of 86.5% of false negatives. This duty-cycle calculation can be done for any time-averaged walk of threshold states.

4 Case Study: Creating a Proactive CDN by "Priming" Certain Content

The Facebook Content Distribution Network (CDN) delivers photos and videos to people who use Facebook. The software stack for the CDN includes several layers of caches and multiple backend storage systems [23] [13] [15] for *static content*.

Caches reduce load and latency in the network by storing popular content close to the people who want to see it. Not all requests can be satisfied by caches. *Dynamic requests* obtain the ever-changing content on the site like posts from friends. That dynamic content comes in the form of HTML or JSON and refers to static resources using photo and video URLs. Web browsers and mobile devices then make static requests to *GET* those photo and video resources included in user posts.

4.1 ML Filtering to Select High-Value Content for Proactive Caching

Read-through caches are good for serving popular content, but not all content is popular. In fact most content is not shared with more than a handful of people. For these cases, we need a way to try to move that content to where it can be used just before it is needed. This is called cache priming and tries to identify when and were the system would perform better by pushing content into caches rather than waiting for it to be pulled by users.

Facebook recently crossed over a big milestone, welcoming 2 billion active users each month [11]. To celebrate we released a personalized goodwill video for each person [12] that showed some images of friends. This is an example of content that is both latency sensitive and "unpopular" by definition, since each person had his or her own personalized video that they would probably not watch more than once. Priming these videos to the edge is an easy decision. By priming the caches, people could start viewing the video sooner, with lower latency, given that the probability that a user will watch

at least the beginning of the video is high. This improved cache hit rate by 20% for these goodwill videos in Thailand.

More nuanced decisions need to be made for photos and on-demand videos that are posted by our friends. There is a cost to priming content that hasn't been requested by users. It takes up network bandwidth and space in the caching devices. This is where the core demand for machine learning and data mining came into this project.

4.2 Adding Tracing and Tagging to obtain Training Data

In order to detect high-value vs. low-value content for priming, offnet and edge caches need to participate by logging when URLs are followed by users in the form of an HTTP GET. Because a single edge cache might process over 1 billion GET events per hour, logging all GETs is infeasible. Instead, a small percentage of photo and video URLs are designated for multi-hop tracing.

Each product platform logs data for these same URLs during the dynamic request. These logs contain attributes that can be gleaned from the context of the dynamic request. At the end of each day, the dynamic attributes logs and the CDN logs are joined in Hive [8] to determine which attributes correspond to URLs that users follow with HTTP GETs by clicking on the photo or video.

4.3 Related Work

This work was done as part of a Proactive CDN [19] at Facebook. Proactive CDNs are discussed in [5], though the paper does not propose specific techniques such as machine learning for creating its "demand profiles". A survey of recent work applying Machine Learning for Networking (MLN) is available in [21], including techniques for resource management. CDNs are not directly cited in this work nor is Layer-7 Networking, such as the interplay of dynamic and static content. Using network control systems as a means for selecting a classification threshold is not discussed.

Selection of a classification threshold is common in data-mining literature for Cost-Sensitive classification, as is sorting the instances of a test set by their predicted probability of being in a certain "class" [6, p.184]. Automatically employing a tunable classification threshold is discussed in [4] for gene sequence analysis. Other than the tunable threshold, the selection of high-value content based on a contextual features and digital cost function is similar to that in Resourceful Contextual Bandits [3].

Measurement of the impact of photo and video caching decisions is described in [2]. These same techniques provide training data for our classifiers with only minor modifications. Though not part of the current paper, ongoing work in on this system seeks to leverage popularity prediction like that proposed in [7].

4.4 Initial Results

We use Apache Thrift [18] and WEKA [22] ML libraries, leveraging important characteristics of each to compute 10^5 classifications per second on a single server and 10^7 classifications per second on a service tier with latencies under $500\mu s$.

The initial classifier service with a trained BayesNet classifier was able to perform at very high request rates with acceptable precision and recall for the proactive CDN project. The classification service was deployed, co-resident on servers that carry out cache priming. Though Facebook has other at-scale ML services [9], the WEKA-based classifier service is small enough to allow it to be embedded with other service tiers with acceptable classification latency and memory consumption.

As outlined above, multiple target levels are used for precision and recall based on a per-destination control system that monitors available bandwidth and cache space. The open and high-recall modes are used when bandwidth and cache space are abundant. The high-precision mode is used when these resources are available but scarce. The closed mode was also employed when a given edge cache had more than 70% of its target network bandwidth consumed by other traffic, or while troubleshooting.

Excessive priming would ultimately hurt cache-hit ratio by causing thrashing of the cache storage. Priming experiment yielded a steady 2% - 3% cache hit-rate increase in the edge pops where they were deployed. This indicates that the filtering of false positives was sufficient, even during aggressive priming phases.

Acknowledgments

Multiple Facebook engineers contributed to this work including: Huapeng Zhou, Federico Larumbe, Fangfei Zhou, Alexander Kramarov, Doug Porter, Tim Wu, David Kim, Srikanth Sastry, and others.

References

[1] Kiam Heong Ang, Gregory Chong, and Yun Li. "PID control system analysis, design, and technology". In: *IEEE transactions on control systems technology* 13.4 (2005), pp. 559–576.

[2] Qi Huang et al. "An Analysis of Facebook Photo Caching". In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP '13. Farminton, Pennsylvania: ACM, 2013, pp. 167–181. ISBN: 978-1-4503-2388-8. DOI: 10.1145/2517349.2522722. URL: http://doi.acm.org/10.1145/2517349.2522722.

[3] Ashwinkumar Badanidiyuru, John Langford, and Aleksandrs Slivkins. "Resourceful contextual bandits". In: *Conference on Learning Theory*. 2014, pp. 1109–1134.

[4] Douglas B Rusch et al. "Bioinformatics for Genomes and Metagenomes in Ecology Studies". In: *Infectious Microecology*. Springer, 2014, pp. 203–226.

[5] John Tadrous, Atilla Eryilmaz, and Hesham El Gamal. "Proactive Content Download and User Demand Shaping for Data Networks". In: *IEEE/ACM Trans. Netw.* 23.6 (Dec. 2015), pp. 1917–1930. ISSN: 1063-6692. DOI: 10.1109/TNET.2014.2346694. URL: http://dx.doi.org/10.1109/TNET.2014.2346694.

[6] Ian H. Witten et al. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2016. ISBN: 978-0128042915.

[7] Linpeng Tang et al. "Popularity Prediction of Facebook Videos for Higher Quality Streaming". In: *Proceedings of the 2017 USENIX Annual Technical Conference. USENIX*. 2017.

[8] *APACHE HIVE ™*. URL: https://hive.apache.org/.

[9] *Applied Machine Learning*. URL: https://research.fb.com/category/applied-machine-learning/.

[10] *Bond-over-gRPC*. URL: https://microsoft.github.io/bond/manual/bond_over_grpc.html.

[11] Josh Constine. *Facebook now has 2 billion monthly users... and responsibility*. URL: https://techcrunch.com/2017/06/27/facebook-2-billion-users/.

[12] *Facebook crosses 2 billion monthly users and it has a video for you*. URL: http://indianexpress.com/article/technology/social/facebook-crosses-2-billion-monthly-users-and-it-has-a-video-for-you-4726324/.

[13] Federico Larumbe and Abhishek Mathur. *Under the hood: Broadcasting live video to millions*. URL: https://code.facebook.com/posts/1653074404941839/under-the-hood-broadcasting-live-video-to-millions/.

[14] Ran Leibman. *Monitoring at Facebook*. URL: https://www.slideshare.net/DevopsCon/monitoring-at-facebook-ran-leibman-facebook-devopsdays-tel-aviv-2015.

[15] *Networking & Traffic*. URL: https://code.facebook.com/networking-traffic/.

[16] *Protocol Buffers*. URL: https://developers.google.com/protocol-buffers/.

[17] Mark Slee. *Thrift Tutorial*. URL: https://git-wip-us.apache.org/repos/asf?p=thrift.git;a=blob_plain;f=tutorial/tutorial.thrift.

[18] Mark Slee, Aditya Agarwal, and Marc Kwiatkowski. *Thrift: Scalable Cross-Language Services Implementation*. URL: https://thrift.apache.org/static/files/thrift-20070401.pdf.

[19] Sumanth Sukumar. *Facebook CDN: Minimizing latency and jitter for content delivery*. URL: https://atscaleconference.com/videos/facebook-cdn-minimizing-latency-and-jitter-for-content-delivery%E2%80%A8/.

[20] Liyin Tang, Vinod Venkataraman, and Charles Thayer. *Facebook's Large Scale Monitoring System Built on HBase*. URL: https://conferences.oreilly.com/strata/stratany2012/public/schedule/detail/25540.

[21] Mowei Wang et al. *Machine Learning for Networking: Workflow, Advances and Opportunities*. URL: https://arxiv.org/abs/1709.08339.

[22] *Weka 3: Data Mining Software in Java*. URL: https://www.cs.waikato.ac.nz/ml/weka/.

[23] Huapeng Zhou et al. *The Evolution of Advanced Caching in the Facebook CDN*. URL: https://research.fb.com/the-evolution-of-advanced-caching-in-the-facebook-cdn/.