AI in a Web-based Survey Instrument: A Low Latency, Real-time Prediction Serving Service

**Keywords:** Model Deployment, Machine Learning, Continuous Integration, Serving Systems

# Introduction

## Machine learning (ML) is being deployed in a growing number of applications which demand text categorization at the time of data collection. In this paper, we introduce a low-latency prediction serving system which uses an application programming interface (API), in conjunction with a web-based survey instrument. We discuss applying a ML model to the Annual Wholesale Trade Survey (AWTS) to code open-ended remarks. The AWTS is a U.S. Census Bureau survey of companies that have significant activity in wholesale trade. Merchant wholesale companies provide estimates on sales, e-commerce, end-of-year inventories, inventories held outside the United States, purchases, and total operating expenses. AWTS includes open-ended questions that require respondents to formulate a textual response. The remarks text field offers space to explain any significant year-to-year changes, to clarify responses, or indicate where data were estimated.

Referral coding categorizes AWTS open-ended responses into groups that can then be used in analysis. The coding process is susceptible to the judgment and interpretation of the coder, it is something that must be done diligently and with standardized procedures. We consider Natural Language Processing (NLP) techniques to help code open-ended responses, which currently are reviewed by human analysts twice. First, during data collection at the National Processing Center (NPC) and finally during post-collection data review at Census Headquarters (HQ) in Suitland, Maryland.

Continuous integration (CI) is a software engineering practice that helps a team manage the development life cycle. An agile development process is used to build, test, integrate, and deploy a ML application with CI. Little work has been done around CI of ML models [1]. No study, to our best knowledge, has applied these efforts for the production of official statistics for National Statistics Offices (NSOs). The question we address in this paper is: *Given unstructured survey remarks text, can we accurately predict referral codes in real-time under heavy query load?*

* We present a novel system architecture for low-latency and real-time inference at scale for NSOs. We demonstrate the advantage of using NLP techniques and a CI pipeline for automating survey coding. This system is currently being served in a U.S. Census Bureau web-based survey instrument.

# Methods

## ML Model

Generalized linear model (GLM) is a widely used class of models for statistical inference and response prediction problems. For example, in order to recommend targeted content to a user or optimize for revenue, many web advertising companies use logistic regression (LR) models to predict the probability of the user’s clicking on an item (e.g., ad, news article, job) [2]. We also deploy LR for categorizing or coding text responses in our survey instrument. Using the popular scikit-learn Python library, we train a prediction model using the labelled AWTS remarks text, integrate that model into our web application, which is then deployed to a production server environment.

## System Framework

We are calling the system that sits in between the text categorization application and ML model, a prediction serving system [3]. Our objective was to construct an end-to-end system which would make predictive model development, deployment, and retraining seamless, reducing the time from research to production for integrating predictive insights into survey collection. To achieve these goals, we built an Application Programming Interface (API) web service wrapping a serialized ML model. Web APIs have made it easy for cross-language applications to work well. If a front-end developer needs to use a ML Model to create an ML powered web application, they would just need to get the URL Endpoint from where the API is being served [4]. Trained models can be stored by serializing them — or how it is often called in Python, by ‘pickling’ them. Pickling a model means converting it into a binary file that can be stored, copied, moved, transferred, and eventually loaded in order to retrieve the original model.

ML model profiling is required to estimate the number of CPUs and the amount of memory required to serve predictions at a specific throughput. After profiling, we determined the largest consumer of CPU time was the deserialization of the model file(s) to python objects. We found API Response latency could be minimized by keeping the model in computer memory and utilizing a job queue in our application design. This process begins with an API customer sending a GET request using Hypertext Transfer Protocol (HTTP) protocol. The Apache web server software processes customer HTTP requestsand delivers HTTP responses back to customers*.* A PHP API server drops a job into an inbox when a web request is made, and then immediately and repeatedly begins checking the outbox for a job. This job contains the AWTS remarks text data in a JSON file. We also start our Python ML script up at the beginning of the day, it is running continuously to keep our deserialized model file in RAM. Figure 1 shows our python start function that keeps



def start(name, predict):

#get the name of this api

server\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

sasha\_dir = os.path.dirname(server\_dir)

inbox = sasha\_dir + "/jobs/" + name + "/inbox/"

outbox = sasha\_dir + "/jobs/" + name + "/outbox/"

while True:

found\_job = look\_for\_jobs(predict, inbox, outbox)

if not found\_job:

sleep(0.1) #Sleep for 100 milliseconds

**Figure 1. The ML python script looks in a folder we called inbox, and it looks for jobs. Before dropping into a while True loop, we have already deserialized the ML model file and have it in RAM. The prediction task begins when our start function finds a job in the inbox and ends with the classified text being dropped into the outbox as a JSON file.**

the deserialized model file in memory. The PHP API server is listening to the outbox, as soon as that job appears in the outbox, it is finally returned to the customer. In summary, customers make a simple HTTP request, this invokes our deserialized model and we return back the prediction response. Our prediction service is able to process 40 requests per second with a round-trip time below 0.3 seconds.

## Decoupling the Serving System

A primary goal of our serving system is to decouple applications from models and allow them to evolve independently from each other, while still providing the full functionality needed to serve applications and to deploy models. From the perspective of the front-end developer building the application, and the DevOps team, they should be able to focus on building reliable low latency applications. That means this system needs to provide stable, performant APIs that can meet service level agreements. The system should also provide the right knobs, to scale to different types of workloads and application demands.

From the perspective of the data scientist, they should be able to focus on making accurate predictions. General-purpose systems allow us to support lots of models and lots of frameworks simultaneously [3]. We did not want to tie the hands of data scientists; we want them to leverage the full rich ecosystem of tools that are being developed at an astonishing pace. This simplifies the model deployment process for data scientists and lets them be as oblivious as possible to system performance and workload demands. Figure 2 shows this high-level design.

|  |
| --- |
| Online  Offline |

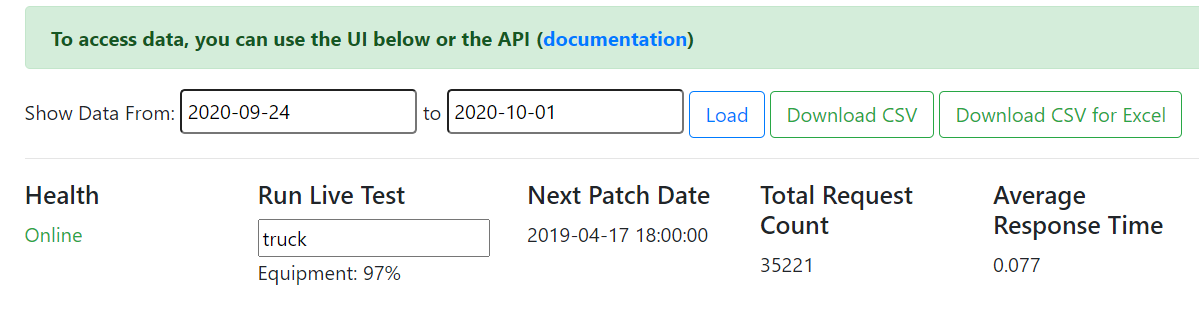
**Figure 2. Decoupled serving system to train our ML models, integrate them with a web application, and deploy into production**

## Deployment Pipeline

Development and Operations (DevOps) practices increase an organization’s ability to deliver applications and services at high velocity. This speed enables organizations to better serve their customers. There is also a constant need to monitor model drift and retrain models as more data is made available. A versioning system allows for continuous iteration on models without breaking or disrupting applications that are using older versions. Our system requires a two-step DevOps process: (1) ML developers commit code to a Git-versioned repository, (2) then a Jenkins Continuous Integration (CI) process builds, tests, and validates the most recent master branch. If everything meets deployment criteria, a Continuous Delivery (CD) pipeline releases the latest valid version of the model to customers. As a result, data scientists can focus on refining models, release engineers can build and govern the software delivery process, and end users can receive the most functional code available, all within a system that can be reviewed and rolled back at any time [2].

# Results

We have validated our solution and operationalized it for the Annual Capital Expenditures Survey (ACES). After two years in production, results show the effectiveness of our proposed system design and deployment approach for classifying ACES open-ended text responses. Once deployed, our web application reporting tool (Figure 3) allows survey analysts to select a date in the survey life cycle, and the tool will output the classification results, and the probability associated with each prediction. Analysts can also use this UI to review real-time survey model responses. In Figure 3, typing the text ‘truck’ into the ‘Live Test’ interface, gives you the result ‘Equipment’ with a confidence of 97%. The text categorization done at the time of data collection has significantly reduced the workload of staff, by 60% to 80% for manual review of written responses.

**Figure 3. The API reporting tool helps staff test the ML model’s results**

ACES API

# Conclusions

Machine Learning is becoming the primary mechanism by which information is extracted from big data, and a primary pillar that Artificial Intelligence is built upon. Accordingly, more and more teams are using ML/AI for a range of applications of official statistics. To advance these innovations, we have developed a CI system for integrating ML into production. The system provides a workflow that will shorten operationalizing ML time when there are multiple models developed and deployed on it, by reusing the environment toolsets, technologies, and engineering pipelines of an already deployed pre-trained model. The system draws on reusable code, rather than reinventing the development pieces. We are refining this system and applying this model to classify AWTS remarks text. Our hope is that this approach will significantly reduce the manual review of open-ended questionnaire responses to the AWTS and accelerate automation processes within NSOs.

# References.

[1] C. Renggli, B. Karlas, and B. Ding, Continuous Integration of Machine Learning Models with ease.ml/ci: Towards a Rigorous Yet Practical Treatment, SysML Conference (2019), 1–19.

[2] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang, Laser: A Scalable response prediction platform for online advertising, WSDM Conference (2014), 173–182.

[3] C. Sun, N. Azari, and C. Turakhia, Gallery: A Machine Learning Model Management System at Uber, EDBT Conference (2020), 474–485.

[4] A. Arora, A. Nethi, and P. Kharat, ISTHMUS: Secure, Scalable, Real-time and Robust Machine Learning Platform for Healthcare, arXiv preprint arXiv:1909.13343 (2019), 1–11.