Overview of the Different Approaches to Putting Machine Learning Models in Production

September 13, 2019 by Julien Kervizic

There are different approaches to putting models into production with benefits that can vary dependent on the specific use case. Take, for example, the use case of churn prediction. It is beneficial to have a static value that can be easily looked up when someone calls customer service, but there is some extra value that could be gained if, for specific events, the model could be re-run with the newly acquired information.

There are generally different ways to both training and serving models into production:

- **Train**: one-off, batch, and real-time/online training
- **Serve:** Batch, Realtime (Database Trigger, Pub/Sub, web-service, inApp)

Each approach has its own set of benefits and tradeoffs that need to be considered.

Here are the topics that are covered in this article:

Do you like this in-depth educational content on applied machine learning? Subscribe to our Enterprise AI mailing list **to be alerted when we release new material.**

One-off Training

Data scientists prototyping and doing machine learning tend to operate in their environment of choice Jupyter Notebooks, which is essentially an advanced GUI on a repl, that allows you to save both code and command outputs.

With this approach, it is more than feasible to push an ad-hoc trained model from some piece of code in Jupyter to production. Different types of libraries and other notebook providers help further tie the link between the data scientist's workbench and production.

Model Format

Pickle converts a Python object to a bitstream and allows it to be stored to disk and reloaded at a later time. It provides a good format to store machine learning models provided that their intended applications are also built in Python.

ONNX, the Open Neural Network Exchange format, is an open format that supports the storing and porting of predictive models across libraries and languages. Most deep learning libraries support it, and sklearn also has a library extension to convert their models to ONNX's format.

PMML, or Predictive model markup language, is another interchange format for predictive models. Like with ONNX, sklearn also has another library extension for converting the models to PMML format. However, it has the drawback of only supporting certain types of prediction models. PMML has been around since 1997, and so it has a large footprint of applications leveraging the format. For instance, applications such as SAP can leverage certain versions of the PMML standard (e.g., CRM applications such as PEGA).

POJO and MOJO are H2O.ai export formats that intended to offer an easily embeddable model into Java application. They are, however, very specific for using with the H2O platform.

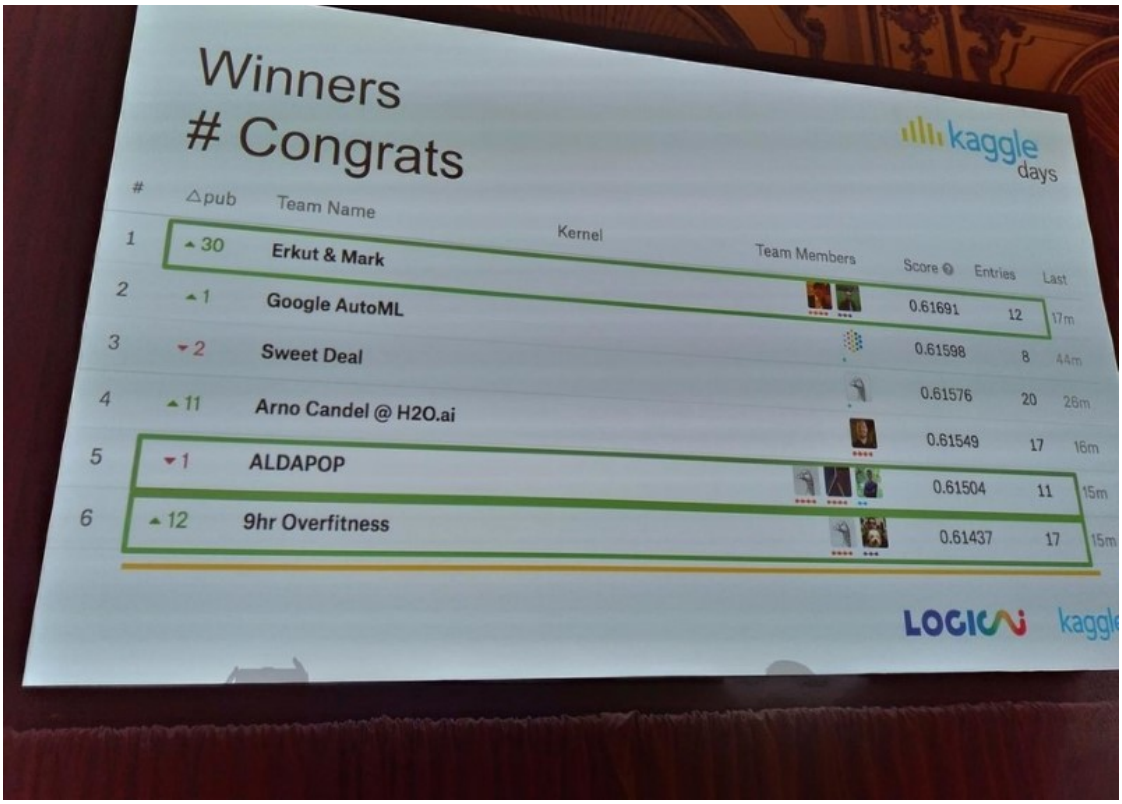
Training

For one-off training of models, the model can either be trained and fine-tuned ad hoc by data scientists or trained through AutoML libraries. Having an easily reproducible setup helps to push into the next stage of model deployment, i.e., batch training.

Batch Training

While not fully necessary to implement a model in production, batch training allows having a constantly refreshed version of your model based on the latest train.

Batch training can benefit a lot from AutoML type of frameworks as AutoML enables you to perform/automate activities such as feature processing, feature selection, model selection, and parameter optimization. The recent performance of AutoML systems has been on par or better compared to the most diligent models built by data scientists.



LinkedIn post

Using the AutoML solutions allows for a more comprehensive model training than what was typically done before their ascent: simply retraining the model weights.

Different technologies exist for supporting this continuous batch training. These could be, for instance, set up through a mix of airflow to manage the different workflows and AutoML libraries such as tpot. Different cloud providers offer their solutions for AutoML that can be put in a data workflow. Azure, for instance, integrates machine learning prediction and model training with their data factory offering.

Real-time Training

Real-time training is possible with 'Online Machine Learning' models. Algorithms supporting this method of training include K-means (through mini-batch), Linear and Logistic Regression (through Stochastic Gradient Descent), and Naive Bayes classifier.

Spark has StreamingLinearAlgorithm / StreamingLinearRegressionWithSGD to perform these operations. Sklearn has SGDRegressor and SGDClassifier that can be incrementally trained. In sklearn, the incremental training is done through the partial_fit method, as shown below:

```
import pandas as pd
from sklearn import linear_model

X_0 = pd.DataFrame([[0,0], [1,0]] )
y_0 = pd.DataFrame([[0], [0]])

X_1 = pd.DataFrame([[0,1], [1,1], [1,1]])
y_1 = pd.DataFrame([[1], [1], [1]])

clf = linear_model.SGDClassifier()

clf.partial_fit(X_0, y_0, classes=[0,1])
print(clf.predict([[0,0]])) # -&amp;amp;amp;amp;gt; 0
print(clf.predict([[0,1]])) # -&amp;amp;amp;amp;gt; 0

clf.partial_fit(X_1, y_1, classes=[0,1])
print(clf.predict([[0,0]])) # -&amp;amp;amp;amp;gt; 0
print(clf.predict([[0,1]])) # -&amp;amp;amp;amp;gt; 1
```

to new data and noise, and model performance needs to be monitored on the fly. In offline training, you can filter points of high leverage and correct for this type of incoming data. That is much harder to do when you are constantly updating your model training based on a stream of new data points.

Another challenge that occurs with training models online is that they don't decay historical information. This means that in case there are structural changes in your datasets, the model will need to be anyway re-trained and that there will be a big onus in model lifecycle management.

Batch vs. Real-time Prediction

When looking at whether to set up a batch or real-time prediction, it is important to understand why making real-time prediction can be essential. It can potentially update scores when a significant event happens – for instance, the system can update the churn score of a customer after they call a contact center. These benefits need to be weighted against the complexity and cost implications that arise from making real-time predictions.

Load implications

Choosing a real-time prediction approach requires a way to handle peak load. Depending on the approach taken and how the prediction ends up being used, choosing a real-time approach might also require to have a machine with the extra computing power available for providing a prediction within a certain SLA. This contrasts with a batch approach where the predictions computing can be spread out throughout the day based on available capacity.

Infrastructure Implications

Selecting a real-time approach puts a much higher operational responsibility. People need to be able to monitor how the system is working, be alerted when there is an issue as well as take some consideration concerning failover responsibility. For batch prediction, the operational obligation is much lower. Some monitoring is definitely needed, and altering is desired, but the need to monitor arising issues is much lower.

Cost Implications

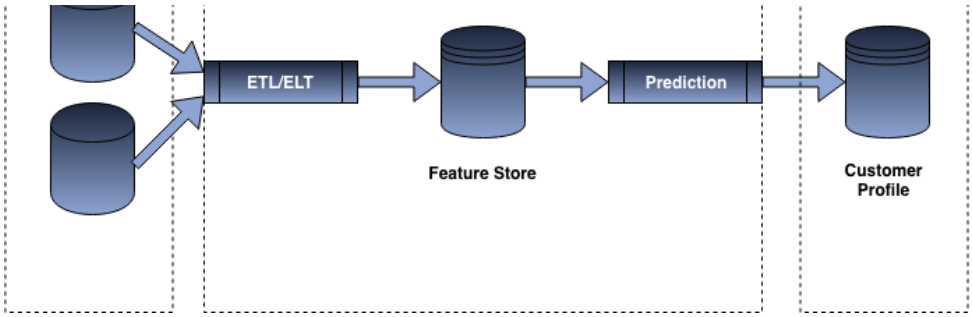
Going for real-time predictions also has costs implications. The need for more computing power without the ability to spread the load throughout the day can force into purchasing more computing capacity than you would need or to pay for a spot price increase. Depending on the approach and requirements taken, there might also be extra cost because of the need to have more powerful compute capacity for meeting SLAs. Furthermore, there will tend to be a higher infrastructure footprint when choosing real-time predictions. One potential caveat there is when it was chosen to rely on app prediction – for that specific scenario, the cost might actually end up being cheaper than going for a batch approach.

Evaluation Implications

Evaluating the prediction performance in a real-time manner can be more challenging than for batch predictions. For instance, how would you evaluate performance when you are faced with a succession of actions in a short burst producing multiple predictions for a given customer? Evaluating and debugging real-time prediction models is significantly more complex to manage. It requires a log collection mechanism that will allow collecting the different predictions and features that yielded the score for further evaluation.

Batch Prediction Integration

Batch predictions rely on two different sets of information, one is the predictive model, and the other one includes the features that we feed into the model. In most types of batch prediction architecture, ETL is performed to either fetch pre-calculated features from a specific datastore (feature-store) or perform some type of transformation across multiple datasets to provide the input to the prediction model. The prediction model then iterates over all the rows in the datasets providing the different score.

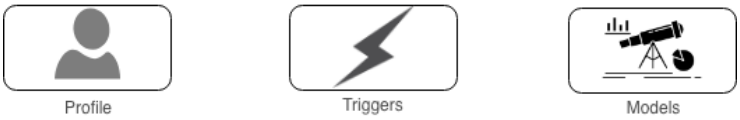


Example flow to model serving for batch prediction

Once all the predictions have been computed, we can then "serve" the score to different systems wanting to consume the information. This can be done differently depending on the use case for which we want to consume the score. For instance, if we want to consume the score on a front-end application, we will most likely push the data to a "cache" or NoSQL database such as Redis so that we can offer milliseconds responses, while for certain use cases such as the creation of an email journey, we might just be relying on a CSV SFTP export or a data load to a more traditional RDBMS.

Real-Time Prediction Integration

Being able to push the model into production for real-time applications requires three base components: a customer/user profile, a set of triggers, and predictive models.



Profile: The customer profile contains all the related customer attributes as well as some other attributes (e.g., counters) necessary for making a given prediction. That is required for customer level prediction to reduce the latency of pulling the information from multiple places as well as to simplify the integration of machine learning models in production. In most cases, a similar type of data store would be needed to fetch the data needed to power the prediction model effectively.

Triggers: Triggers are events causing the initiation of the process. For churn, these could include, for instance, calling to a customer service center, checking information within your order history, etc.

Models: models need to be pre-trained and typically exported to one of the three formats previously mentioned (pickle, ONNX or PMML) to be something that we could easily port to production.

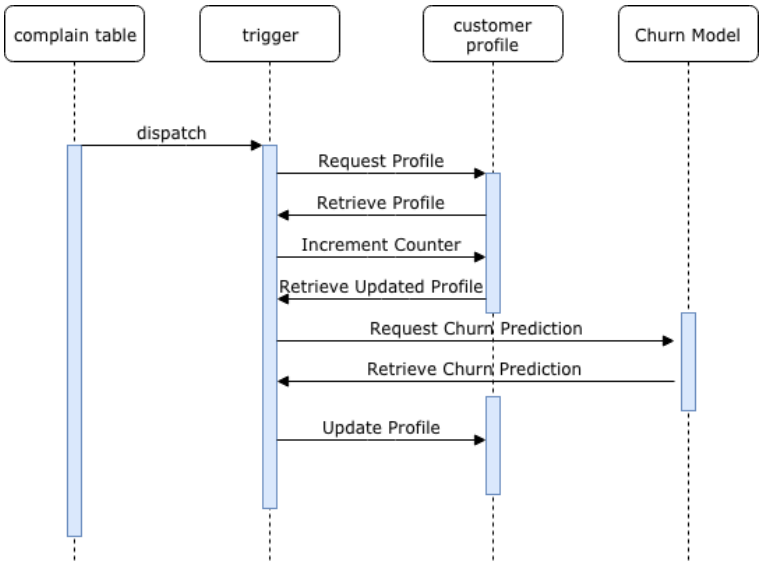
There are quite a few different approaches to putting models for scoring purpose in production:

- *Relying on Database integration:* many database vendors have made a significant effort to tie up advanced analytics use cases within the database, either by direct integration of Python or R code or by importing the PMML model.
- *Exploiting a Pub/Sub model*: The prediction model is essentially an application feeding with a data stream and performing certain operations, such as pulling customer profile information.
- *Webservice:* Setting up an API wrapper around the model prediction and deploying it as a web-service. Depending on the way the web-service is set up, it might or might not do the pull or data needed to power the model.
- *inApp:* it is also possible to deploy the model directly into a native or web application and have the model run on local or external data sources.

Database integrations

If the overall size of your database is fairly small (< 1M user profile) and the update frequency is occasional, it can make sense to integrate some of the real-time update processes directly within the database.

This can be coupled with Postgres' Triggers Mechanism to perform a run of the database and update the churn score. For instance, if a new entry is made to a complaint table, it would be valuable to have the model be re-run in real-time.



Sequence flow

The flow could be set up in the following way:

New Event: When a new row is inserted in the complaint table, an event trigger is generated.

Trigger: The trigger function would update the number of complaints made by this customer in the customer profile table and fetch the updated record for the customer.

Prediction Request: Based on that it would re-run the churn model through PL/Python and retrieve the prediction.

Customer Profile Update: It can then re-update the customer profile with the updated prediction. Downstream flows can then happen upon checking if the customer profile has been updated with new churn prediction value.

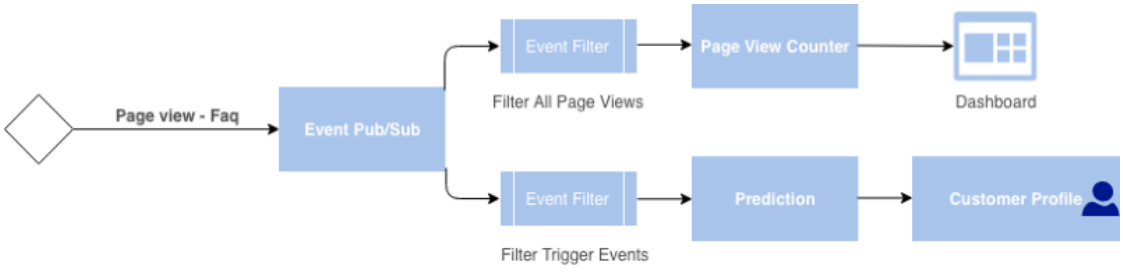
Technologies

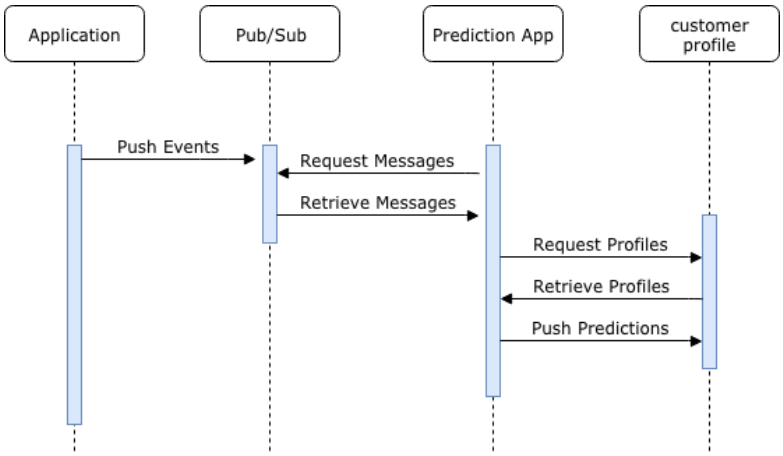
Different databases are able to support the running of Python script. That is the case of PostGres which has a native Python integration as previously mentioned, but also of MS SQL Server through its Machine Learning Service (in Database). Other databases, such as Teradata, can run R/Python script through an external script command, while Oracle supports PMML model through its data mining extension.

Pub/Sub

Implementing real-time prediction through a pub/sub model allows handling the load through throttling properly. For engineers, it also means that they can just feed the event data through a single "logging" feed, to which different applications can subscribe.

An example, of how this could be set up is shown below:





Sequence flow

Event messages are pushed to the pub/sub topic as they occur. The prediction app polls the topic for new messages. When the prediction app retrieves a new message, it requests and retrieves the customer profile and use the message and the profile information to make a prediction, which it ultimately pushes back to the customer profile for further use.

A slightly different flow can be set up, where the data is first consumed by an "enrichment app" that adds the profile information to the message and then pushes it back to a new topic to finally be consumed by the prediction app and pushed onto the customer profile.

Technologies

The typical open-source combination that you will find supporting this kind of use cases in the data ecosystem is a combination of Kafka and Spark streaming, but a different setup is possible on the cloud. On Google, a pub-sub/dataflow (Beam) provides a good alternative to that combination. On Azure, a combination of Azure-Service Bus or Eventhub and Azure Functions can serve as a good way to consume the messages and generate these predictions.

Web Service

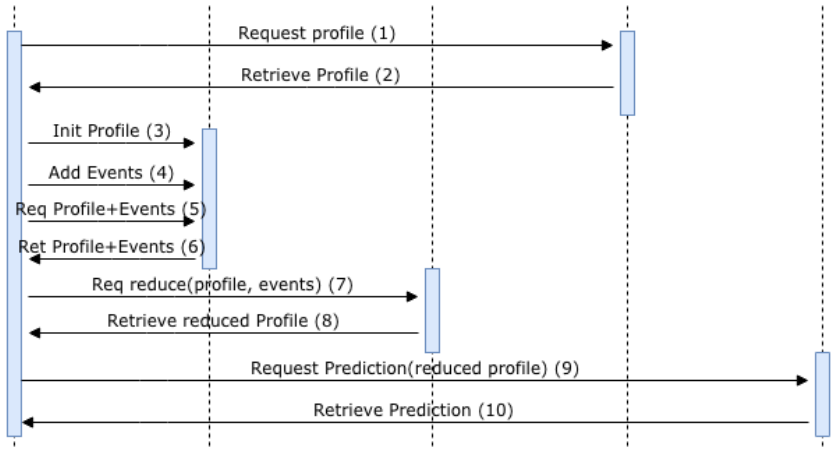
We can implement models in production as web-services. Implementing prediction models as web-services are particularly useful in engineering teams that are fragmented and that need to handle multiple different interfaces such as web, desktop, and mobile.

Interfacing with the web-service could be set up in different ways:

- By providing an identifier and having the web-service pull the required information, compute the prediction and return its' value;
- By accepting a payload, converting it to a data frame, making the prediction and returning its value.

The second approach is usually recommended in cases when there is a lot of interaction happening, and a local cache is used to buffer the synchronization with the backend systems, or when needing to predict a different grain than a customer id, for instance when making session-based predictions.

The systems making use of local storage, tend to have a reducer function, which role is to calculate what would be the customer profile, should the event in local storage be integrated back. As such, it provides an approximation of the customer profile based on local data.



Sequence Flow

The flow for handling the prediction using a mobile app, with local storage can be described in 4 phases:

Application Initialization (1 to 3): The application initializes, and makes a request to the customer profile, and retrieves its initial value back, and initializes the profile in local storage.

Applications (4): The application stores the different events happening with the application into an array in local storage.

Prediction Preparation (5 to 8): The application wants to retrieve a new churn prediction, and therefore needs to prepare the information it requires to provide to the Churn Web-service. For that, it makes an initial request to local storage to retrieve the values of the profile and the array of events it has stored. Once they are retrieved, it requests a reducer function providing these values as arguments. The reducer function outputs an updated* profile with the local events incorporated back into this profile.

Web-service Prediction (9 to 10): The application makes a request to the churn prediction web-service, providing the difference of the updated*/reduced customer profile from step 8 as part of the payload. The web-service can then use the information provided by the payload to generate the prediction and output its value back to the application.

Technologies

There are quite a few technologies that can be used to power a prediction web-service:

Functions

The functionality of AWS Lambda, Google Cloud, and Microsoft Azure (although Python support is currently in Beta) offers an easy-to-set-up interface for deploying scalable web-services.

For instance, on Azure, a prediction web-service could be implemented through a function that looks roughly like this:

```
import logging

import azure.functions as func
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.externals import joblib

def main(req: func.HttpRequest) -&amp;amp;gt; func.HttpResponse:
    logging.info('Python HTTP trigger function processed a request.')
    if req.body:
        try:
            logging.info("Converting Request to DataFrame")
            req_body = req.get_json()
            df_body  = pd.DataFrame([req_body])

            logging.info("Loadding the Prediction Model")
            filename = "model.pckl"
            loaded_model = joblib.load(filename)
```

```
            logging.info("Subselecting the dataframe")
            df_subselect = df_body[feature_names]

            logging.info("Predicting the Probability")
            result = loaded_model.predict_proba(df_subselect)
            # We are looking at the probba prediction for class 1
            prediction = result[0][1]

            return func.HttpResponse("{prediction}".format(prediction=prediction), status_code=200)

        except ValueError:
            pass
    else:
        return func.HttpResponse(
             "Please pass a name on the query string or in the request body",
             status_code=400
        )
```

azure_function_prediction_function.py hosted with ♥ by GitHub

Container

An alternative to functions is to deploy a Flask or Django application through a docker container (Amazon ECS, Azure Container Instance, or Google Kubernetes Engine). Azure, for instance, provides an easy way to set up prediction containers through its Azure Machine Learning service.

Notebooks

Different notebooks providers such as Databricks and Dataiku have notably worked on simplifying the model deployment from their environments. These have the feature of setting up a web service to a local environment or deploying to external systems such as Azure ML Service, Kubernetes engine, etc.

in App

In certain situations, when there are legal or privacy requirements that do not allow for data to be stored outside of an application, or there exist constraints such as having to upload a large number of files, leveraging a model within the application tend to be the right approach.

Android-ML Kit or the likes of Caffe2 allow to leverage models within native applications, while Tensorflow.js and ONNXJS allow for running models directly in the browser or apps leveraging javascript.

Considerations

Beside the method of model deployment, there are quite a few important considerations to keep in mind when putting a model to production.

Model Complexity

The complexity of the model itself is the first consideration to have. Models such as linear regression or logistic regression are relatively easy to apply and do not usually take much space to store. Using more complex models, such as neural networks or complex ensemble decision trees, ends up taking more time to compute, more time to load into memory on cold start and proves more expensive to run

Data Sources

It is important to consider the difference that could occur between the data source in production and the one used for training. While the training data need to be in sync with the context it would be used for in production, it is often impractical to recalculate every value to become perfectly in-sync.

Experimentation framework

Wrapping Up

Choosing how to deploy predictive models into production is quite a complicated issue with different ways to handle the lifecycle management of the predictive models, different formats to store them, multiple ways to deploy them, and very vast technical landscape to pick from.

Understanding specific use cases, the team's technical and analytics maturity, the overall organization structure and its interactions, help come to the right approach for deploying predictive models to production.

This article was originally published on Medium and re-published to TOPBOTS with permission from the author.

Enjoy this article? Sign up for more updates on applied ML.

We'll let you know when we release more technical education.

Email Address *

Name *

First

Last

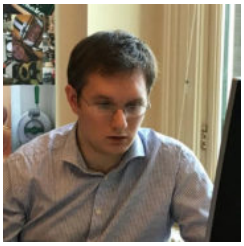
Company *

What business use cases are you applying AI to? *

- ☐ Administration
- ☐ Analytics
- ☐ Customer Support
- ☐ Finance
- ☐ HR
- ☐ Legal
- ☐ Marketing
- ☐ Operations
- ☐ Product
- ☐ Research
- ☐ Sales
- ☐ Other (Please Describe Below)

What is your biggest challenge with applied AI? *

SUBMIT



About Julien Kervizic

Julien is living at the interstice of business, data, and technology. Currently, he holds a position of Senior Enterprise Data Architect at GrandVision NV. His broad experience in analytics comes from working in such top tech companies as Amazon, Facebook, and Veon.

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name

Email

Website

POST COMMENT

Search the site ...

Learn Applied AI

We create and source the best content about applied artificial intelligence for business. Be the FIRST to understand and apply technical breakthroughs to your enterprise.

First

Last

Company

Email

SUBMIT

FOLLOW US

POPULAR ARTICLES

10 Leading Language Models For NLP In 2022

NeurIPS 2021 – 10 Papers You Shouldn't Miss

A Guide To Knowledge Graphs

Why Graph Theory Is Cooler Than You Thought

BERT Inner Workings

Pretrain Transformers Models in PyTorch Using Hugging Face Transformers

More Articles

Brands

Business

China

Commerce

Computer Vision

Conversational AI

Customer Service

Cybersecurity

Data Science & Engineering

Design

Education

Ethics & Safety

Finance

Gaming

Healthcare

HR & Recruiting

Infrastructure

Leadership & Management

Manufacturing

Marketing

Natural Language Processing

Reinforcement Learning

Research

Retail & CPG

Society

Technical Guide

Technology