Continuous Data Pipeline for Benchmarking & Data Set Augmentation

How to use open data to continuously benchmark and improve your production models

Ivan Aguilar - 03.30.23

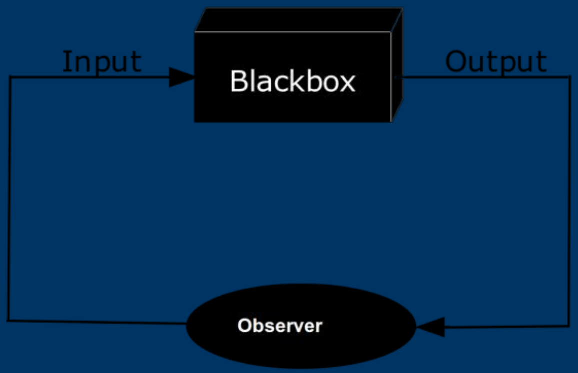
teleskope

About me

Data Scientist at Teleskope.ai
Formerly MLE at Forge.ai
Baseball Fan
Hobby Potter

ivan@teleskope.ai

The Problem



How do we make sure our model is doing what we think it should be doing?

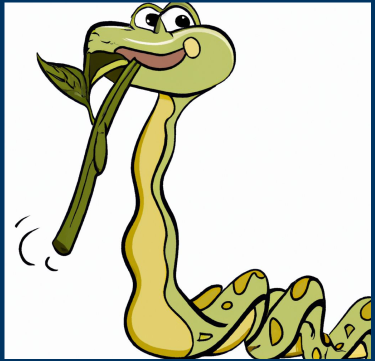
Why is this a problem?

- Confidential Customer Data
- Data Drift
  - Training data can drift away from production environment data
- Blind spots in training
- Set it and forget it
- Shifting Priorities
  - Model maintenance might become a lower priority

Usual Approaches

- Quality Training
  - High quality data and the right architecture for your approach is essential
- Spot Check Results
  - At arbitrary times check arbitrary amounts of data to get an idea of our model performance
- User Feedback
  - Once a model is live getting validation from users is essential
- Hope for the Best

Our Approach



How do we operationalize our quality control loop and stay up to date with our performance?

Open Source Data API's

- Find a list of APIs
  - Curated lists on GitHub
  - Sites you frequently use already
- Pick APIs that have data resembling your production data
  - In this example we look at conversation data
- Determine copyright requirements and terms you need to follow
  - Be a good internet user and make sure to not abuse these services

Celery

How to manage your tasks

- Task queue
  - Good for distributing work across servers/nodes
  - Create cron jobs
  - Keep workload balanced
  - Very quick set up and support for many queue systems
  - Can help you persist results to a database

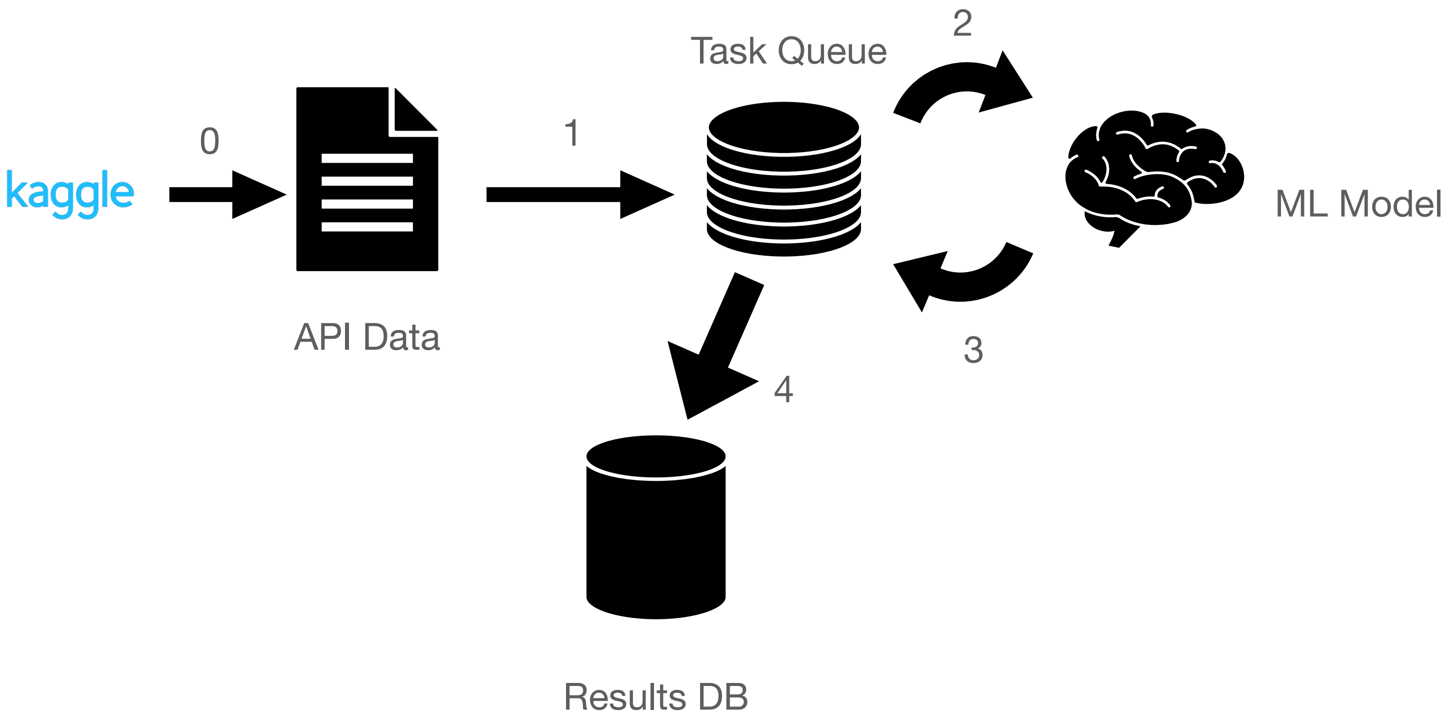
Integrations

Connect to external APIs + internal model

- Fetching Data
  - Use API to gather data and store it as tasks in the queue
  - Control volume of data and frequency of calls within the task queue
- Running Classifications
  - Pull tasks off of the queue and generate classifications
  - Persist the results in a database for the annotation and review tasks

Task Overview

Gathering Data and Generating Classifications

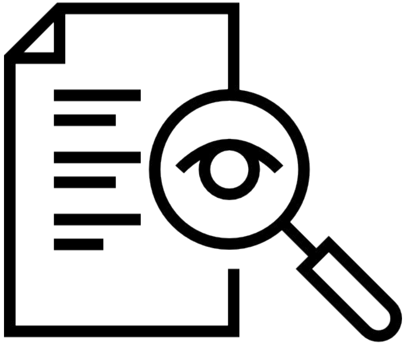


Now What?

Once we have all of this data from these APIs and our results, what can we use this for?

Using Results

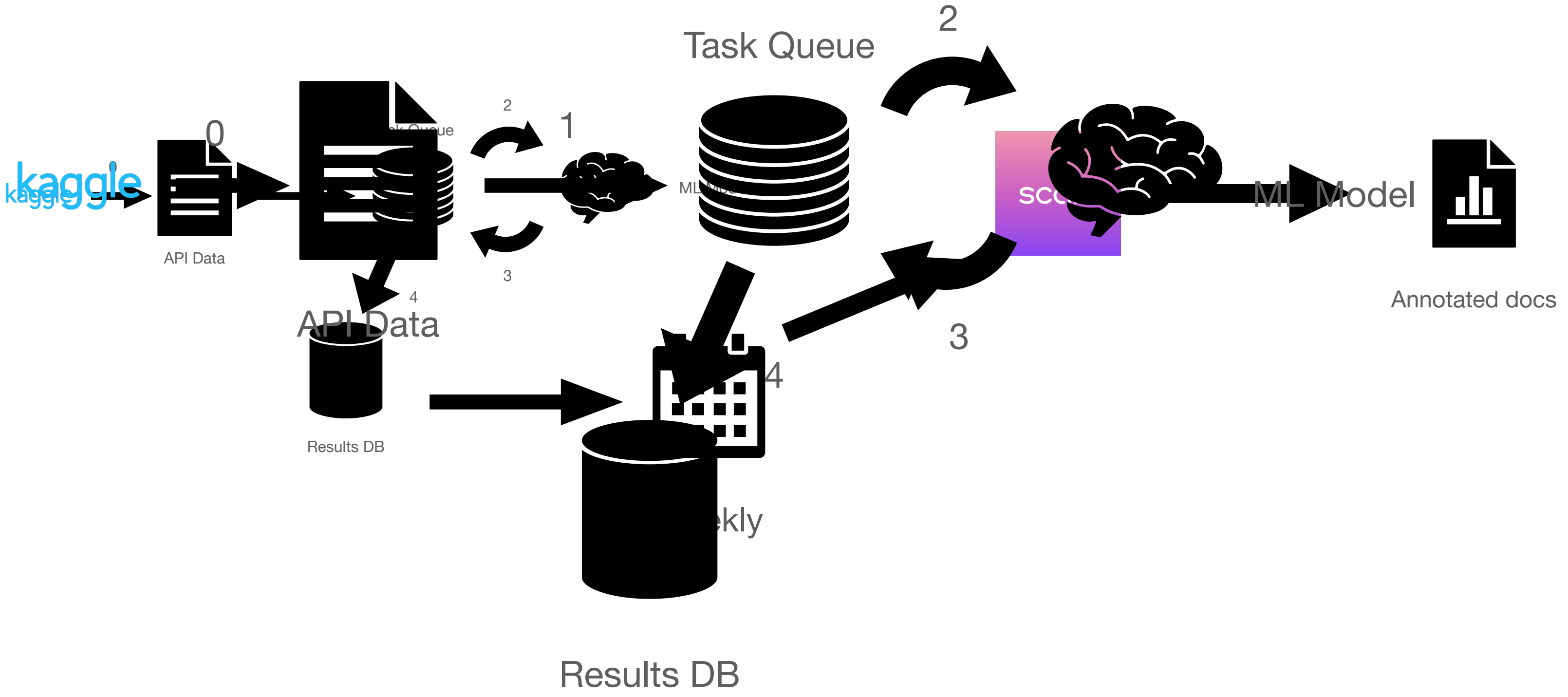
Scale.AI / doccano



- Scale AI
  - Pay as you go annotation shop with API
  - Quick annotation of your data with customizable review and annotator count
- doccano
  - Open source annotation tool with API for internal annotation
  - Bare bones app that lets you annotate your own documents

Annotations Overview

Gathering Annotations



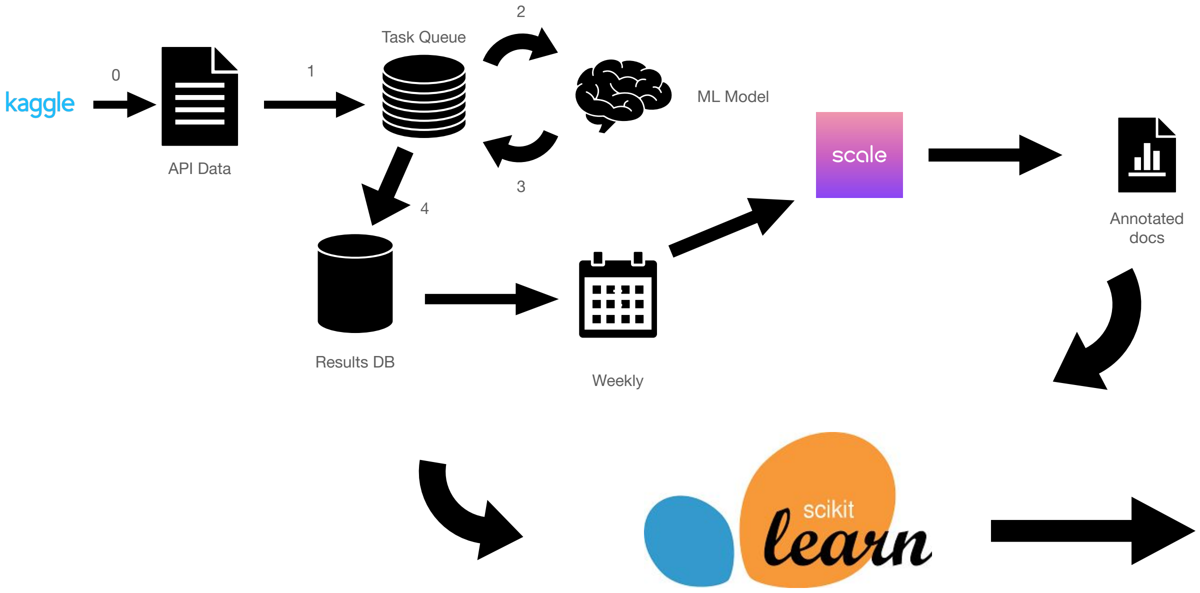
Using Results

Scale.AI / doccano

- With the annotations have task compute performance metrics
    - Accuracy
    - Element Type Precision/Recall Splits
    - Comparison of previous models
- Additional training data for future model improvements
    - Essential for keeping track of where model needs to improve and what kind of data is needed to improve coverage

Annotations Overview

Gathering Annotations




```
Performance by label (#match, #model, #ref) (precision, recall, F1):
    B-NP: (12000, 12358, 12407) (0.9710, 0.9672, 0.9691)
    B-PP: (4707, 4872, 4805) (0.9661, 0.9796, 0.9728)
    I-NP: (13984, 14484, 14359) (0.9655, 0.9739, 0.9697)
    B-VP: (4466, 4662, 4653) (0.9580, 0.9598, 0.9589)
    I-VP: (2549, 2698, 2643) (0.9448, 0.9644, 0.9545)
    B-SBAR: (448, 498, 534) (0.8996, 0.8390, 0.8682)
    O: (5939, 6113, 6174) (0.9715, 0.9619, 0.9667)
    B-ADJP: (322, 403, 438) (0.7990, 0.7352, 0.7658)
    B-ADVP: (711, 835, 866) (0.8515, 0.8210, 0.8360)
    I-ADVP: (54, 82, 89) (0.6585, 0.6067, 0.6316)
    I-ADJP: (110, 137, 167) (0.8029, 0.6587, 0.7237)
    I-SBAR: (2, 15, 4) (0.1333, 0.5000, 0.2105)
    I-PP: (34, 42, 48) (0.8095, 0.7083, 0.7556)
    B-PRT: (80, 102, 106) (0.7843, 0.7547, 0.7692)
    B-LST: (0, 0, 4) (0.0000, 0.0000, 0.0000)
    B-INTJ: (1, 1, 2) (1.0000, 0.5000, 0.6667)
    I-INTJ: (0, 0, 0) (******, ******, ******)
    B-CONJP: (5, 7, 9) (0.7143, 0.5556, 0.6250)
    I-CONJP: (10, 12, 13) (0.8333, 0.7692, 0.8000)
    I-PRT: (0, 0, 0) (******, ******, ******)
    B-UCP: (0, 0, 0) (******, ******, ******)
    I-UCP: (0, 0, 0) (******, ******, ******)
Macro-average precision, recall, F1: (0.639239, 0.602512, 0.611086)
Item accuracy: 45422 / 47321 (0.9599)
Instance accuracy: 1176 / 2011 (0.5848)
Elapsed time: 0.940000 [sec] (2140.4 [instance/sec])
```

Using Metrics And Reviewed Data

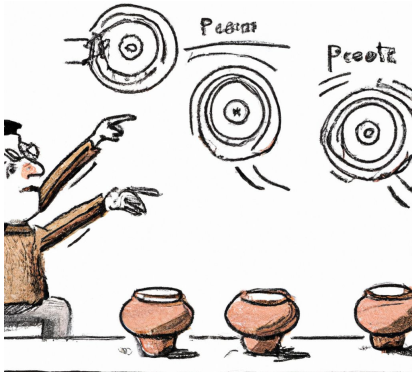
- Dashboard to have regularly updated metrics on performance
- Using the findings to target additional training data sources
- Targeted improvements
- Additional data to train with from your review set

Final Thoughts

- Usual Approaches are still important
  - Customer feedback essential for fine tuning down the line
  - Initial training set and annotation guide are critical
- Visibility is Key
  - Testing your model with continuous data provides insight which might be otherwise hidden
  - Detecting changes in the performance as leading indicator of maintenance needs
- Initial Success at Teleskope
  - Reducing FP and improving precision/recall for traditionally noisy elements

Thanks!

Teleskope team and/or **open source contributors** and/or **audience!**



Some Links

- https://github.com/public-apis/public-apis
- https://github.com/Kaggle/kaggle-api
- https://github.com/scaleapi/scaleapi-python-client
- https://github.com/doccano/doccano-client
- https://docs.celeryq.dev/en/stable/getting-started/introduction.html