twiml

AI Enterprise Workflow Study Group

Course 6, Week 1

5/16/2020

Agenda

- Check in
- Discussion
- Next steps

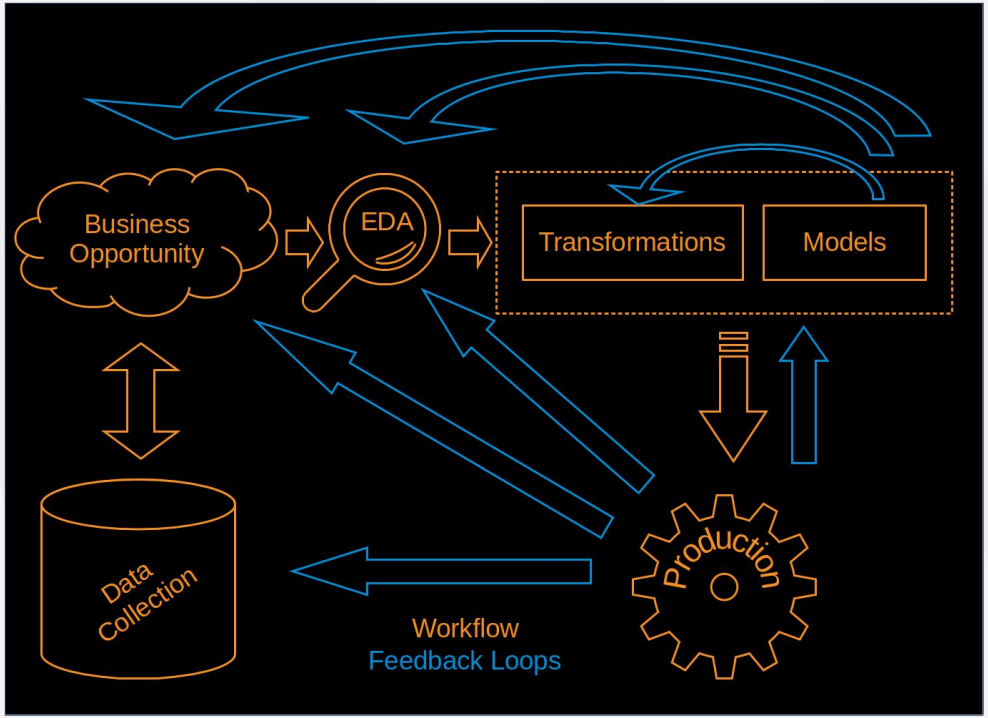
Course & Study Group Schedule

AI Enterprise Workflow Study Group		
Session	**Topic**	**Date**
Overview Webinar	Webinar with instructor, Ray Lopez	15-Feb
Course 1 Week 1	Course intro	22-Feb
Course 1 Week 2	Data ingestion, cleaning, parsing, assembly	29-Feb
Course 2 Week 1	Exploratory data analysis & visualization	7-Mar
Course 2 Week 2	Estimation and NHT	14-Mar
Course 3 Week 1	Data transformation and feature engineering	21-Mar
Course 3 Week 2	Pattern recognition and data mining best practices	28-Mar
Course 4 Week 1	Model evaluation and performance metrics	18-Apr
Course 4 Week 2	Building machine learning and deep learning models	25-Apr
Course 5 Week 1	Deploying models	2-May
Course 5 Week 2	Deploying models using Spark	9-May
Course 6 Week 1	Feedback loops and monitoring	16-May
Course 6 Week 2	Hands on with OpenScale and Kubernetes	23-May
Course 6 Week 3	Captsone project week 1	30-May
Course 6 Week 4	Captsone project week 2	6-Jun

Course 6 Week 1 learning objectives

- Discuss the different feedback loops in AI workflow
- Discuss the use of unit testing in the context of model production
- Describe processes for monitoring model performance in production
- Describe general principles for understanding models in business contexts

Feedback loops



Feedback loops

production –> business opportunity	Explore if a model has less of an impact on the business than originally anticipated this is often the first feedback loop that you will visit. The place where you discuss the opportunity cost of continued workflow iteration.
production –> data collection	Common feedback loop especially when using deep-learning models. Can be costly if labeling required.
production –> EDA	Once a model has been in production for some time, should investigate the relationship between model performance and the business metric.
production –> model selection and development	If a model performs poorly in production, e.g. due to latency issues or because there is an over-fitting issue, or starts out performing well but you see performance drift, it is reasonable to return to try a different model.

Unit testing for ML models

- Small/isolated units of code
- Boolean output
- Test coverage: The amount of source code that is actually tested when compared to the total amount of testable code is known as test coverage.
- Trade-off: unit tests put a stake in the ground; at odds with experimentation
- Not sure I agree with this (Test-time vs run-time errors):
    - “Unit tests also helps ensure that when software fails, it fails gracefully. This means it stops execution without causing additional errors and takes any steps, such as closing open connections or saving data to a file that may be necessary for recovery. This is an important aspect of software design that can save significant amounts of time when debugging a problematic query.”

Unit testing in Python

Unit testing libraries

- pytest
- nose
- unittest

Code coverage

- coverage

Related concepts

TDD

- Write test, run test to see failure, fix test, run again to see pass.

CI/CD

- Continuous Integration is the practice of merging all developers' changes frequently (usually daily) into a single central source often called the trunk.
- Continuous Delivery refers to the iteration on software in short cycles using a straightforward and repeatable deployment process.

Continuous Delivery = manual deployments vs. Continuous Deployment = automated

Minimal logging for ML models

- **runtime** - The total amount of time required to process the request. This is a factor that directly affects the end user's experience and should be monitored.
- **timestamp** - Timestamps are needed to evaluate how well the system handles load and concurrency. Additionally, timestamps are useful when connecting predictions to labels that are acquired afterwards. Finally, they are needed for the investigation of events that might affect the relationship between the performance and business metrics.
- **prediction** - The prediction is, of course, the primary output of a prediction model. It is necessary to track the prediction for comparison to feedback to determine the quality of the predictions. Generally, predictions are returned as a list to accommodate multi-label classification.
- **input_data_summary** - Summarizing information about the input data itself. For the predict endpoint this is the shape of the input feature matrix, but for the training endpoint the features and targets should be summarized.
- **model_version_number** - The model version number is used to better understand the influence of model improvements (or bugs) on performance.

Optional logging features

- **request_unique_id** - Each request that has been made should correspond to an entry in the log file. It is possible that a request corresponds to more than one entry in the log file for example if more than one model is called. This is also known as correlation_id.
- **data** - Saving the input features that were provided at the time of a predict request makes it much easier to debug broken requests. Saving the features and target at the time of training makes it easier to debug broken model training.
- **request_type** - Relevant attributes about the request (e.g. webapp request, browser request)
- **probability** - Probability associated with a prediction (if applicable)

Implementing logging

Ensure that your data are collected at the most granular level possible. This means each data point should represent one user making one action or one event.

Implementing logging

- Python logging module
- Elasticsearch
- Apache Commons Logging

Performance Drift

Types of performance drift

- **concept drift.** statistical distribution of a target variable changes over time.
- **sampling bias changes.** you built your model on a non-representative subset of the population than what is seen in reality.
- **selection bias changes.** the subset seen by your model shifts over time.
- **software changes.** software, dependency, framework changes.
- **data changes.** semantics/distribution of feature changes.

Ways to detect concept drift

- Outlier detection (Outliers are defined from the training data)
- Novelty detection (Assumes no outliers in training data)

Ways to detect feature distribution shift:

- Kullback–Leibler divergence
- Wasserstein_metric

Can be useful to use PCA to reduce dimensionality before detecting outliers or calculating distance/distribution shift.

Securing machine learning models

Adversarial threats

- poisoning attacks - injecting malicious data into the training set
- evasion attacks - exploit the blind spots of machine learning models at test time

Defending machine learning models involves certifying and verifying model robustness and model hardening with approaches such as:

- pre-processing inputs
- augmenting training data with adversarial examples
- leveraging runtime detection methods to flag any inputs that might have been modified by an adversary.

Resource:

- NIST: A Taxonomy and Terminology of Adversarial Machine Learning

Demo

- IBM: Adversarial Robustness Toolkit (ART)