**Cheat Sheet for SEAS 8500 – Fundamentals of AI Enabled Systems- Final – Spring 2024 – Michael Wacey**

Chapter 6 – Model Development and Offline Evaluation

Model Selection: Avoid the state of the art trap, Start with the simplest model, Avoid Human Biases in selecting models, Evaluate good performance now versus good performance later (A learning curve of a model is a plot of its performance—e.g., training loss, training accuracy, validation accuracy—against the number of training samples it uses), Evaluate tradeoffs (false positives and false negatives trade-off), Understand your model’s assumptions(Prediction Assumption, IID – Independent and Identically Distributed), Smoothness, Tractability, Boundaries, Conditional Independence, Boundaries, Conditional Independence, Normally Distributed.

Ensemble, Base Learners, Bagging (bootstrap aggregating) (create each learner with sampling with replacement), (Classification – vote, regression – average) (Improves unstable methods) (Random Forest), Boosting (improves weak learners) (Train serially on misclassified data) (GBM, XGBoost, LightGBM), and stacking (Meta Learner).

Experiment Tracking and Versioning: The process of tracking the progress and results of an experiment is called experiment tracking. The process of logging all the details of an experiment for the purpose of possibly recreating it later or comparing it with other experiments is called versioning. Things to track:

The loss curve corresponding to the train split and each of the eval splits. // The model performance metrics that you care about on all nontest splits, such as accuracy, F1, perplexity. // The log of corresponding sample, prediction, and ground truth label. This comes in handy for ad hoc analytics and sanity check. // The speed of your model, evaluated by the number of steps per second or, if your data is text, the number of tokens processed per second. // System performance metrics such as memory usage and CPU/GPU utilization. They’re important to identify bottlenecks and avoid wasting system resources. // The values over time of any parameter and hyperparameter whose changes can affect your model’s performance, such as the learning rate if you use a learning rate schedule; gradient norms (both globally and per layer), especially if you’re clipping your gradient norms; and weight norm, especially if you’re doing weight decay.

Debugging ML Models: They fail silently, difficult to confirm bug, and many components. Causes: Theoretical Constraints, Poor implementation of model, Poor choice of Hyperparameters, Data Problems, Poor choice of features.

Approach to debugging: Start small and add components, Overfit a single batch, Set a random seed.

Paralelism: Data, Model, Workload, Pipeline

AutoML: Soft AutoML – Hyperparameter tuning, Hard AutoML – Includes the model selection

Phases: Before, Simple, Optimize, Complex

Five Baselines: Random, Simple Heuristic, Zero Rule, Human, Existing Solutions

Evaluation Methods: Perturbation Test, Invariance Tests (output should not change unexpectedly), Directional Expectation Tests (Output should change as expected with input), Model Calibration (can use Platt Scaling from Scikit Learn to calibrate), Confidence Measurement (Only one on a single prediction), Slice Based Evaluation (aggregation can conceal and contradict actual situations.) (identifying slices: Heuristic-based, Error analysis, slice finder),

Model Calibration Example: We’ll walk through two examples to show why model calibration is important. First, consider the task of building a recommender system to recommend what movies users will likely watch next. Suppose user A watches romance movies 80% of the time and comedy 20% of the time. If your recommender system shows exactly the movies A will most likely watch, the recommendations will consist of only romance movies because A is much more likely to watch romance than any other type of movies. You might want a more calibrated system whose recommendations are representative of users’ actual watching habits. In this case, they should consist of 80% romance and 20% comedy.44

Chapter 7 – Model Deployment and Prediction Service

ML App Logic: Data Engineering, Feature Engineering, Model, Metrics

Myth 1: You Only Deploy One or Two ML Models at a Time, 2: If We Don’t Do Anything, Model Performance Remains the Same, 3: You Won’t Need to Update Your Models as Much, 4: Most ML Engineers Don’t Need to Worry About Scale

Model Compression: low-rank optimization, knowledge distillation, pruning, and quantization

Local optimization techniques: vectorization, Parallelization, loop tiling, operator fusion

Chapter 8 – Data Distribution Shifts and Monitoring

Software System Failures: Dependency Failure, Deployment failure, Hardware Failure, Downtime or crashing

ML Specific Failures: Production data differing from training data, Edge Cases, Degenerate Feedback Loops

Data Distribution Shifts

P(X, Y) = P(Y|X)P(X), P(X, Y) = P(X|Y)P(Y)

Covariate shift - When P(X) changes but P(Y|X) remains the same. This refers to the first decomposition of the joint distribution.

Label shift - When P(Y) changes but P(X|Y) remains the same. This refers to the second decomposition of the joint distribution.

Concept drift - When P(Y|X) changes but P(X) remains the same. This refers to the first decomposition of the joint distribution.21

Address data shifts – large data sets, adapt to target without labels, retrain

Stateful and stateless retraining

Chapter 9 – Continual Learning and Test in Production

Four stages of continual learning: 1: Manual, stateless retraining, 2: Automated retraining, 3: Automated, stateful training, 4: Continual learning

Test in production: Shadow Deployment, A|B Testing, canary release, Interleaving, Bandits, Contextual Bandits

Chapter 10 – Infrastructure and Tooling for MLOps

Layers – Storage and compute, Resource Management, ML Platform, Development Environment

Graphical user interface, text, chat or text message

Description automatically generated

Jupyter execution – Papermill, Commuter

Dockerfile 🡪 Docker Image 🡪 Docker Container

Cron, schedulers, orchestrators

Schedulers use a DAG

Airflow – monolithic, DAG is not parametized, DAG is static

Argo, Prefect improved on Airflow – Argo workflow in YAML, Argo only on Kubenetes

Kubeflow and Metaflow - One component of Kubeflow is Kubeflow Pipelines, which is built on top of Argo, and it’s meant to be used on top of K8s. Metaflow can be used with AWS Batch or K8s. Metaflow allows you to work seamlessly with both dev and prod environments from the same notebook/script. You can run experiments with small datasets on local machines, and when you’re ready to run with the large dataset on the cloud, simply add @batch decorator to execute it on AWS Batch.

Model Store - Model definition, Model parameters, Featurize and predict functions, Dependencies. Data, Model generation code.

* What frameworks it used
* How it was trained
* The details on how the train/valid/test splits were created
* The number of experiments run
* The range of hyperparameters considered
* The actual set of hyperparameters that final model used

Feature Store