Feature Engineering Part 3: Comprehensive Review

DSC 208R - Data Management for Analytics

Algorithm Selection / Architecture Selection

Algorithm Selection in Classical ML

- In traditional (non-Deep Learning) Machine Learning, the user typically selects models/algorithms *ab initio* (from the beginning).
- **Best Practice**: Start by training simple models (e.g., Logistic Regression) as baselines. Then, gradually try more complex models (e.g., XGBoost) if needed.
- **Ensembles**: A common technique is to build diverse models and aggregate their predictions to improve overall performance and robustness.

Architecture Selection in Deep Learning

- This is more critical in Deep Learning compared to classical ML.
- The neural network architecture serves as the "inductive bias" (in classical ML terms) and directly controls both feature learning and the bias-variance trade-off.
- **Transfer Learning**: For some applications, off-the-shelf pre-trained Deep Learning models (e.g., from HuggingFace Models) can be used, with fine-tuning for specific tasks. This is known as transfer learning.
- For other applications, the effort saved in hand-crafted feature engineering is replaced by the complexity of neural architecture engineering.

Automated Model Selection / AutoML

Automated Machine Learning (AutoML) aims to automate parts of the ML pipeline.

- **Scope**: Hyper-parameter tuning and most of feature engineering are largely automated in practice today. Neural architecture search is also a growing area of AutoML.
- **Components**: AutoML platforms typically offer capabilities for automatic algorithm selection, feature engineering, and hyper-parameter tuning.
- **Scalability**: These platforms are designed for scalable model building.
- **Examples of AutoML Platforms**: H2O, Azure Machine Learning, Amazon SageMaker, DataRobot.
- **Specialized Tools**:
  - *Decision Tree-oriented*: XGBoost, LightGBM (dmlc, Microsoft).
  - *Deep Learning-oriented*: TensorFlow, PyTorch.
  - *General ML (in-memory)*: scikit-learn, R.
  - *Disk-based/Distributed*: MADlib (on RDBMS), Spark MLlib, Dask.

Scalable ML Inference

Once an ML model is trained, it needs to make predictions on new, often large, datasets.

The Prediction Function

A trained ML model is essentially a prediction function: $f : \mathcal{D}_X \to \mathcal{D}_Y$, mapping input data ($X$) to output predictions (Y).

Scaling Inference

- **Assumption 1**: An individual input example fits entirely in DRAM (main memory).
- **Assumption 2**: The prediction function f (i.e., the model parameters) fits entirely in DRAM.
- **Trivial Access Pattern (if both assumptions hold)**: If both assumptions are true, scaling inference is relatively simple: perform a single file scan of the data, apply the per-tuple function f, and write the output. This can be easily parallelized, for example, using a Map-only function in a MapReduce-like framework.
- **Complex Access Pattern (if assumptions fail)**: If either assumption fails (e.g., very large examples or very large models), the access pattern becomes more complex. This typically involves breaking down the function f into smaller internal computations and staging data access for partial results.

The Lifecycle of ML-based Analytics

ML-based analytics is an iterative process, involving several key stages.

1. **Data Acquisition**: Collecting raw data from various sources.
2. **Data Preparation**: Cleaning, transforming, and pre-processing the data to make it suitable for analysis. This includes data integration, missing value imputation, and feature engineering.
3. **Model Training and Evaluation**: Building and training ML models, and rigorously evaluating their performance using appropriate metrics. This stage often involves iterations of algorithm selection, architecture selection (for DL), and hyper-parameter tuning.
4. **Serving (Deployment)**: Deploying the trained model into a production environment so it can make real-time or batch predictions.
5. **Monitoring**: Continuously monitoring the deployed model's performance, data drift, and concept drift to ensure it remains effective over time. This may trigger retraining or model updates.

This lifecycle emphasizes that ML is not a one-shot process but a continuous cycle of data handling, model building, deployment, and maintenance.