دانشکار

{ یک یادگیری کارساز }

بوتکمپ

Artificial Intelligence

MLOps

Session 4

Session Overview

Data Management & Experiment Tracking

- **Recap:** Environment Management & Dependencies
- **Today's focus:** Systematic data and experiment management
- **Learning objectives:**
  - Understand data versioning challenges
  - Master experiment tracking fundamentals
  - Learn data lineage concepts
  - Apply tracking tools in practice

The Data Challenge

VisionaryAI's Growing Dataset Problem

Scenario: Mobile phone defect detection system

- **Key Questions:**
  - Which dataset version produced the best model?
  - How to track changes between versions?
  - Can you reproduce results with exact data?

Month 1

- 10,000 component images

Month 2

- +5,000 images (better lighting)

Month 3

- +8,000 images (new phone model)

Traditional vs ML Data Management

Why ML is Different

Traditional Software	Machine Learning
Version **code**	Version **code + data + models**
Small text files	Gigabytes/Terabytes
Stable structure	Evolving schemas
Code-driven changes	Data-driven changes

ML Data Challenges:

- Size and format diversity
- Complex relationships and dependencies
- Frequent changes and updates

Data Versioning Fundamentals

- **Definition:** Systematic tracking of dataset changes over time
- **Similar to Git:** but designed for large ML datasets
- **VisionaryAI NLP Example:**
  - v1.0: 50K support tickets (Jan-Mar)
  - v1.1: +20K tickets, found 5K duplicates
  - v1.2: Removed duplicates, added categories
  - v2.0: Added sentiment + priority labels

Data Versioning Strategies

Approaches to Track Data Changes

1. Snapshot Approach	2. Incremental Approach	3. Hash-based Versioning
• Store complete dataset versions • ✅ Simple to understand • ❌ High storage requirements	• Store only changes between versions • ✅ Storage efficient • ❌ More complex implementation	• Unique identifier based on content • Automatic version creation on changes

Data Versioning Metadata

Essential Information to Track

- **For Every Data Version:**
  - **Timestamp** of creation
  - 👤 **Author** who created version
  - **Description** of changes made
  - **Schema** information
  - 📊 **Quality metrics** (completeness, validity)

Example:

```
Version: v2.3
Created: 2024-01-15 14:30:00
Author: sarah.chen@visionaryai.com
Description: Added 1000 new defect examples for screen
scratches
Schema: Added 'scratch_severity' column (int, 1-5 scale)
Quality: 98.5% complete, 0.2% invalid labels
```

Data Lineage Introduction

Understanding Data Flow

- Definition: Complete trail of data through ML pipeline
- From raw sources to final models

Why it matters:
Debugging, compliance, reproducibility, impact analysis

VisionaryAI Computer Vision Pipeline:

```
Raw Images → Quality Filter → Augmentation → Feature Extraction → Training Data
    ↓              ↓                ↓                ↓                   ↓
metadata.json  filter_log    aug_config.yml   features.pkl    train_v2.3.h5
```

Types of Data Lineage

What to Track in Your Data Flow

1. Schema Lineage
   - Structure changes over time
   - Column additions, removals, type changes
   - Example: Helpdesk data: 5 → 15 columns over 6 months

2. Operational Lineage
   - Which processes created/modified data
   - Execution times, success/failure status
   - Resource usage and performance

3. Business Lineage
   - Business rules and logic applied
   - Regulatory requirements
   - Data ownership and access controls

Lineage Tracking Challenges

Common Complexity Scenarios

Complex Transformations

- Multiple processing steps
- Feature engineering pipelines
- External data integration

Multiple Data Sources

- User logs + Purchase history + Product catalog
- External demographic data
- Third-party APIs

Temporal Dependencies

- Time-series data requirements
- Historical data impact on current models

The Experiment Chaos Problem

Without Systematic Tracking

Scenario: VisionaryAI data scientist, 2 weeks of work:

- 5 different architectures
- 3 preprocessing approaches
- 4 learning rates
- 2 augmentation strategies
- = 120 different experiments!

Result: Scattered files, lost results, no comparisons

- `model_v3_final_ACTUALLY_FINAL.ipynb`
- `best_model.pkl`, `really_best_model.pkl`
- Screenshots and memory-based results

Problems Without Experiment Tracking

Common Issues:

- Lost results: "I got 95% accuracy but can't remember how“
- Unreproducible: Can't recreate successful conditions
- Duplicate work: Accidentally repeating experiments
- No baseline: Can't determine actual improvements
- Team conflicts: Multiple people, same problems

Impact:
Wasted time, missed opportunities, frustration

Systematic Experiment Management

The Solution Framework

Core Components:

1. Experiment Design

- Clear hypothesis
- Defined variables
- Success metrics
- Baseline comparison

2. Controlled Execution

- Consistent environment
- Automated logging
- Progress monitoring

3. Complete Documentation

- Parameters and metrics
- Artifacts and environment
- Notes and insights

Experiment Design Example

VisionaryAI NLP Ticket Classification

- **Hypothesis:** “Pre-trained embeddings improve classification accuracy”
- **Variables:**
  - Embedding type: Word2Vec, GloVe, BERT
  - Embedding dimensions: 100, 200, 300
- **Metrics:**
  - Accuracy, F1-score, Inference time
- **Baseline:**
  - TF-IDF + Logistic Regression (82% accuracy)

Experiment Organization Strategies

Keeping Experiments Manageable

1. Hierarchical Structure

```
VisionaryAI_Defect_Detection/
├── baseline_models/
├── cnn_experiments/
│   ├── resnet_variants/
│   ├── custom_architectures/
├── data_augmentation_tests/
└── ensemble_methods/
```

2. Tagging System

- `baseline`, `production-candidate`, `hyperparameter-tuning`

3. Naming Conventions

- `YYYY-MM-DD_project_approach_version`

The Three Pillars of Tracking

Parameters (Inputs)

- Model settings: architecture, hyperparameters
- Data settings: version, preprocessing, splits
- Environment: software versions, hardware, seeds

Metrics (Outputs)

- Performance: accuracy, loss, F1-score
- Operational: training time, memory usage
- Business: cost per prediction, revenue impact

Artifacts (Files)

- Models: trained files, architectures
- Evaluation: plots, confusion matrices
- Configuration: config files, notebooks

Parameter Tracking Example

VisionaryAI Recommendation System Parameters

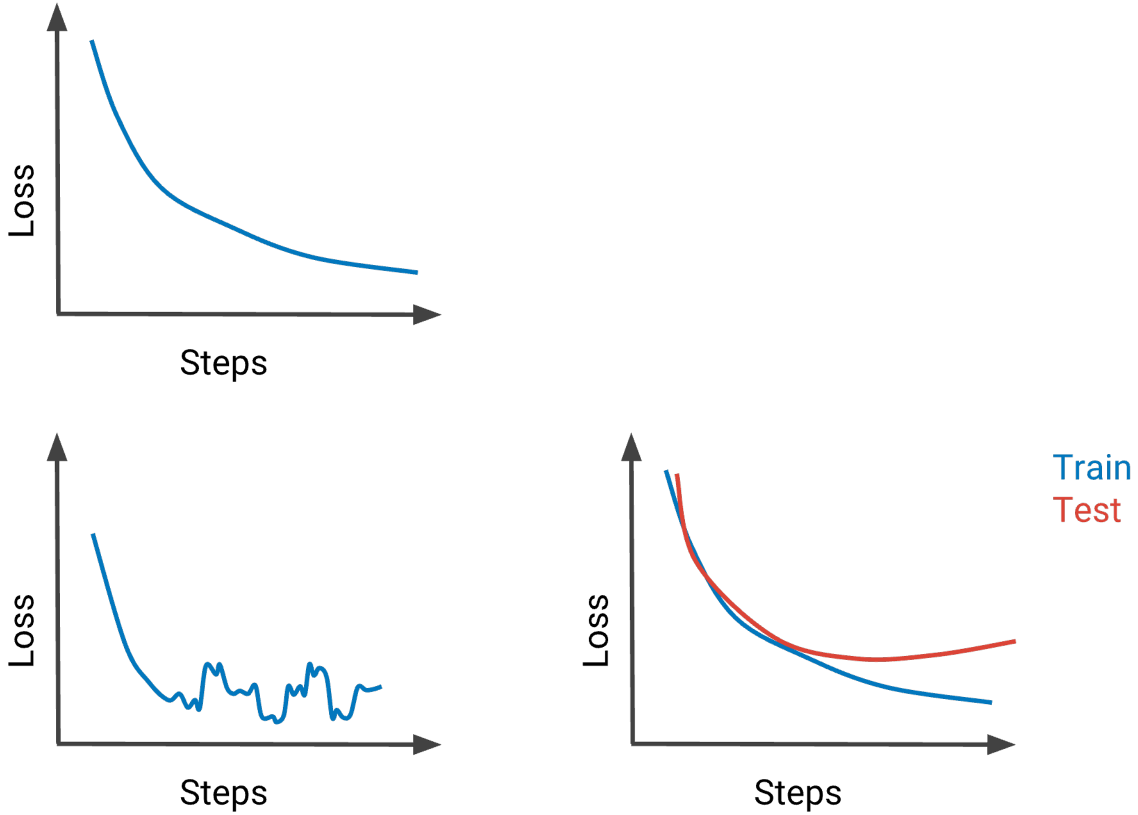
```
parameters = {
    'model_type': 'collaborative_filtering',
        'embedding_dim': 50,
        'regularization': 0.01,
        'learning_rate': 0.001,
        'batch_size': 256,
        'data_version': 'v2.1',
        'train_split': 0.8,
        'random_seed': 42
    }
```

Key principle:
Track everything that affects results

Metrics Tracking Over Time

Beyond Final Results

- **Track throughout training:**
    - Training loss curves
    - Validation performance trends
    - Learning rate schedule effects
    - Resource utilization patterns
- **Multiple metric types:**
    - Technical performance metrics
    - Operational efficiency metrics
    - Business impact metrics



Artifact Management Best Practices

Organizing Experiment Outputs

Artifact Types:

- Models: .pkl, .h5, .pth files
- Evaluation: confusion matrices, ROC curves
- Data: processed datasets, quality reports
- Configuration: config files, requirements.txt

Best Practices:

- Consistent naming conventions
- Version coupling with experiments
- Storage efficiency (compression)
- Access control for sensitive artifacts

Tools Overview

Solutions for Data & Experiment Management

- **Data Versioning:**
  - DVC (Data Version Control): Git-like for data
  - Pachyderm: Enterprise data lineage
  - Delta Lake: Data lakehouse with versioning
- **Experiment Tracking:**
  - MLflow: Open-source ML lifecycle management
  - Weights & Biases: Cloud-based experiment tracking
  - Neptune: Metadata store for ML experiments
  - TensorBoard: Visualization and tracking

Code Demo Preview

Practical Implementation

What we'll demonstrate:

1. Data Versioning Scenario

- Track dataset changes using DVC concepts
- Handle multiple data versions
- Reproduce experiments with specific data

2. MLflow Experiment Tracking

- Set up systematic experiment logging
- Track parameters, metrics, and artifacts
- Compare multiple experiment runs

3. Experiment Dashboard

- Analyze experiment results
- Identify best performing models
- Export results for team sharing

Key Takeaways

5 Critical Points:

Data versioning is critical
- Models depend on data quality and consistency

Experiment chaos is real
- Systematic tracking prevents wasted effort

Three pillars matter
- Always capture parameters, metrics, artifacts

Lineage enables debugging
- Understand data flow for troubleshooting

Tools automate tracking
- Leverage existing solutions (DVC, MLflow)

Impact:
More reproducible, efficient, and collaborative ML development

Next Session Preview

Session 5: Model Development & Testing

- Code organization for ML projects
- Configuration-driven development
- Testing strategies for ML code
- Model validation frameworks
- Bias detection and fairness

Connection:

- Proper data management enables better model development and testing

Questions & Discussion

Discussion Topics:

- Data versioning challenges in your projects?
- Experiment tracking pain points?
- Tool preferences and experiences?
- Team collaboration strategies?

Next Steps:

- Download code examples from demo
- Try MLflow with your own experiments
- Set up data versioning for current projects


THANK YOU
FOR YOUR ATTENTION!