Reconstructing and Querying ML Pipeline Intermediates

Sebastian Schelter (University of Amsterdam)
https://ssc.io
ML-Specific Data Debugging

- ML-specific data debugging methods identify subsets of the input data with poor accuracy, negative impact on fairness or label errors (e.g., SliceFinder, Gopher, Fairlearn, DataScope)

- Designed for a single static input dataset with attributes to slice the data, aligned with features and predictions in matrix form

- Difficult to apply to end-to-end ML pipelines, which do not expose/store required intermediate data

→ Data scientists have to manually construct an appropriate evaluation dataset for each pipeline and analysis method

Can we automatically apply such debugging methods to ML pipelines?

Chung: SliceFinder - Automated data slicing for model validation, ICDE’19.
Pradhan: Interpretable explanations for fairness debugging, SIGMOD’22.
Bird: Fairlearn - a toolkit for assessing and improving fairness in AI, MSR Tech Report
Karlaš: Data Debugging with shapley importance over end-to-end machine learning pipelines, arXiv
Automatically Constructing Evaluation Datasets

- Treat ML pipeline as dataflow computation turning multiple relational inputs into matrix outputs (features, labels, predictions)
- Compute record-level provenance during pipeline execution
- Store relational inputs, matrix outputs and provenance information in a DB, generate “evaluation” views based on provenance
- Materialise custom evaluation datasets for external debugging libraries based on these views (or query them directly)
- Prototypical implementation for pandas/ sklearn and pyspark pipelines, internally leverages DuckDB:

  https://github.com/amsterdata/freamon

```python
# Execute sklearn pipeline, capture intermediates and provenance
view_generator = from_sklearn_pipeline('classify-product-reviews.py')

# Materialize a view over the test labels and predictions, sliceable by two attributes from the test input
test_view = view_generator.materialize_test_view(
    sliceable_by=['category', 'rating'],
    with_features=False, with_y=True, with_y_pred=True)

# Compute fairness metrics from the view via the fairlearn library
fairness_metrics = fairlearn.metrics.MetricFrame(
    metrics=('recall': sklearn.metrics.recall_score),
    y_true=test_view.y, y_pred=test_view.y_pred,
    sensitive_features=(test_view.category, test_view.rating>3)
print(fairness_metrics.by_group)

# Compute Slicefinder statistics via an aggregation query
view_generator.execute_query(""
SELECT category, rating>3 AS toprated,
AVG(cross_entropy_loss(y, y_pred)) AS avg_loss,
VARIANCE(cross_entropy_loss(y, y_pred)) AS var_loss,
COUNT(*) as size
FROM virtual_test_view
GROUP BY GROUPING SETS ((category,rating>3),(rating>3),(category))")
```python
def healthcare_pipeline(split_year, min_age, counties):
    # Data loading
    patients = pd.read_csv("s3:///...")
    if len(counties) > 0:
        patients = patients[patients.age>min_age]
    # Integration and filtering
    treatments = pd.read_csv("s3:///...")
    histories = patients.join(treatments, on="patient_id")
    histories = histories[['age', 'weight', 'smoker', 'vegetarian', 'notes', 'had_complications']]  
    # Temporal train/test split
    train = histories[histories.date<split_year]
    test = histories[histories.date>=split_year]
    # Declaratively defined (nested) feature encoding pipeline
    pipeline = Pipeline([  
        ('features', ColumnTransformer([  
            (StandardScaler(), ['age', 'weight']),  
            (Pipeline([SimpleImputer(), OneHotEncoder()]), ['smoker', 'vegetarian']),  
            (HashingVectorizer(n_features=100), 'notes')]),  
        # ML model for learning
        ('learner', LogisticRegression())])
    # Train and evaluate model
    model = pipeline.fit(train, train.had_complications)
    return model, model.score(test, test.had_complications)
```

---

**Diagram:**
- **1. Pipeline representation as SPJ (Select-Project-Join) query**
- **2. Fine-grained provenance for reconstructing intermediate train and test data**
- **3. Data access by pipeline query**
- **4. View over intermediate data for model training and testing, in relational and corresponding matrix form**

**Equations:**
- \( R_{\text{train}} = \pi (\sigma (R_{\text{patients}} \times R_{\text{habits}} \times R_{\text{treatments}})) \)
- \( X_{\text{train}} \rightarrow y_{\text{train}} \)
- \( R_{\text{test}} = \pi (\sigma (R_{\text{patients}} \times R_{\text{habits}} \times R_{\text{treatments}})) \)
- \( X_{\text{test}} \rightarrow y_{\text{test}} \rightarrow y_{\text{pred}} \)

**Backup Slide**