Lecture 10: Model Bias, Data Imbalance, and Real-World Fraud Detection

CSCI E-103: Reproducible Data Science and Machine Learning

Harvard University

- **Course:** CSCI E-103: Reproducible Data Science
- **Week:** Lecture 10
- **Instructors:** Anindita Mahapatra & Eric Gieseke
- **Objective:** Understand model errors, handle imbalanced data, and learn from a real-world fraud detection system at scale

Contents

1 The Central Theme: Data Problems Lead to Model Problems

Lecture Overview

This lecture explores the critical relationship between data quality and model performance. Poor data doesn't just affect accuracy—it can create biased, unfair, and even harmful models.

Key Topics:

- Machine Learning Errors: Bias, Variance, and Irreducible Error
- The Bias-Variance Tradeoff
- Model Bias: Why models can be unfair
- Class Imbalance: When your data is 99% one thing
- SMOTE: Synthetic Minority Oversampling Technique
- AutoML: Blackbox vs. Glassbox approaches
- Data Classification for PII detection
- Real-World Case Study: Fraud Detection at 15ms latency

2 Machine Learning Errors: The Three Components

Every ML model makes errors. Understanding **why** models err is crucial to improving them.

The Total Prediction Error

Total Error = Bias^2 + Variance + Irreducible Error

2.1 1. Bias (Underfitting)

Definition: Bias

Bias is the error introduced by approximating a real-world problem (which may be extremely complicated) with a too-simple model.
Symptom: The model fails to capture the true relationship between features and target.
Result: Poor performance on **both** training AND test data.

Example: High Bias Model

Imagine you're trying to predict house prices, which depend on many factors in complex, non-linear ways.
If you use simple linear regression (a straight line), you might get:

- Training accuracy: 60%
- Test accuracy: 58%

Both are bad! The model is too simple to capture reality.
Analogy: Trying to describe a curvy mountain road as "mostly flat."

2.2 2. Variance (Overfitting)

Definition: Variance

Variance is the error introduced when a model is too sensitive to small fluctuations (noise) in the training data.
Symptom: The model memorizes the training data instead of learning general patterns.
Result: Excellent performance on training data, **terrible** performance on test data.

Example: High Variance Model

Using an extremely complex model (like a deep neural network with no regularization) on limited data:

- Training accuracy: 99.5%
- Test accuracy: 62%

The model "cheated" by memorizing answers instead of learning rules.
Analogy: A student who memorizes all practice exam answers but can't solve new problems.

2.3 3. Irreducible Error

Definition: Irreducible Error

Irreducible Error is the inherent noise in the data that cannot be reduced regardless of the model. It represents the randomness in real-world phenomena.

This is the floor—no model can be more accurate than the noise allows.

2.4 The Bias-Variance Tradeoff

Important: The Fundamental Tradeoff

Bias and variance are **inversely related**:

- **Simple model** (fewer parameters) → High Bias, Low Variance
- **Complex model** (more parameters) → Low Bias, High Variance

The goal of ML is to find the **"sweet spot"**—a model complex enough to capture the true patterns (low bias) but not so complex that it fits noise (low variance).

Table 1: *Bias vs. Variance Characteristics*

Characteristic	High Bias (Underfitting)	High Variance (Overfitting)
Model complexity	Too simple	Too complex
Training error	High	Very low
Test error	High	High (much higher than training)
Cause	Not enough learning	Too much learning (noise included)
Fix	More complex model, more features	Regularization, more data, simpler model

3 Model Bias: When AI Becomes Unfair

This is a different kind of "bias"—not underfitting, but **unfairness**. When models systematically disadvantage certain groups of people.

3.1 Why Does Model Bias Happen?

3.1.1 1. Human Cognitive Bias in Data

Models learn from data. If data reflects human prejudices, the model learns those prejudices.

Example: Facial Recognition Accuracy Disparity

A NIST study found that facial recognition systems from major companies (Microsoft, IBM, etc.) had dramatically different accuracy rates:

- **Light-skinned males:** 90%+ accuracy
- **Dark-skinned females:** 60-70% accuracy

Root Cause: The training datasets were **under-represented** in dark-skinned faces, especially females. The model simply didn't have enough examples to learn from.

Key Insight: This isn't an algorithm problem—it's a **data collection** problem. The humans who created the dataset inadvertently (or sometimes systematically) included fewer examples of certain groups.

Example: Microsoft's Tay Chatbot

In 2016, Microsoft released a chatbot named "Tay" that learned from Twitter conversations. Within 24 hours, malicious users taught it to say racist, sexist, and Holocaust-denying statements.

Lesson: Models are only as good as their training data. Garbage in, garbage out—but magnified.

3.1.2 2. Poor Quality Training Data

- **Low resolution:** Images too blurry to distinguish (like the dog vs. arctic fox example)
- **Mislabeled data:** Human labelers may have their own biases (e.g., associating "evil" with dark colors)
- **Incomplete coverage:** Data missing important edge cases or scenarios

3.2 How to Reduce Model Bias

Strategies for Reducing Bias

1. **Ensure Representative Data (Most Important!):**
   - Audit your training data for demographic representation
   - Actively collect more data from underrepresented groups
   - Consider the source of your data and its inherent biases
2. **Use Appropriate Models:**

- Don't use linear models for non-linear relationships
- Tree-based algorithms are often better at handling bias

3. **Apply Weighting or Penalized Models:**
   - Give more weight to underrepresented groups
   - Use class weights in your loss function
4. **Extensive Hyperparameter Tuning:**
   - Don't just use defaults—optimize for fairness metrics
5. **Ensemble Methods (for reducing variance):**
   - Combine weak and strong learners
   - Random forests, boosting, etc.

4 Class Imbalance: The 99-1 Problem

A special and extremely common case of data problems.

Definition: Class Imbalance

Class Imbalance occurs when one class (label) in your training data is much more frequent than another.

Examples:

- Fraud detection: 99.9% legitimate, 0.1% fraud
- Cancer diagnosis: 98% healthy, 2% cancer
- Anomaly detection in manufacturing
- Network intrusion detection

4.1 The Accuracy Trap

Most ML algorithms optimize for **accuracy**—the percentage of correct predictions.

Why Accuracy is Misleading

Consider a fraud detection dataset with 99.9% legitimate transactions.

A model that **always predicts "legitimate"** will have:

Accuracy = 99.9%

Sounds great, right? But this model catches **zero fraud**. It's completely useless for its intended purpose.

The lesson: **Accuracy is a terrible metric for imbalanced data.**

4.2 Better Metrics: Precision, Recall, and F1

Definition: Confusion Matrix Terminology

- **True Positive (TP):** Model predicted Fraud, actually was Fraud
- **True Negative (TN):** Model predicted Legitimate, actually was Legitimate
- **False Positive (FP):** Model predicted Fraud, but was actually Legitimate (Type I Error)
- **False Negative (FN):** Model predicted Legitimate, but was actually Fraud (Type II Error)

Example: Cancer Screening

In cancer diagnosis, **recall is critical**. Missing an actual cancer case (false negative) can be fatal.

A model with:

- Precision: 90% (some false alarms)
- Recall: 99% (catches almost all cancers)

is far better than:

- Precision: 99% (rarely wrong when it says cancer)
- Recall: 70% (misses 30% of cancers!)

Table 2: *Key Metrics for Imbalanced Data*

Metric	Formula	When to Prioritize
Precision	$\frac{TP}{TP+FP}$	"Of things I flagged as fraud, how many were actually fraud?" Prioritize when **false positives are costly** (blocking legitimate customers)
Recall	$\frac{TP}{TP+FN}$	"Of all actual frauds, how many did I catch?" Prioritize when **false negatives are costly** (missing actual fraud/cancer)
F1 Score	$2 \times \frac{Precision \times Recall}{Precision + Recall}$	Harmonic mean of precision and recall. Use when **both matter**—the primary metric for imbalanced data

4.3 Resampling Techniques

Table 3: *Resampling Strategies*

Technique	How It Works	Pros/Cons
Undersampling	Remove samples from the majority class until balanced	Simple, but **loses valuable data**
Oversampling	Duplicate samples from the minority class	Simple, but can cause **overfitting** (memorizing duplicates)
SMOTE	Generate **synthetic** minority samples	Best of both worlds—creates new data points

4.4 SMOTE: Synthetic Minority Oversampling Technique

How SMOTE Works

SMOTE doesn't just copy existing minority samples—it **creates new synthetic ones**.

Algorithm:

1. Pick a minority class sample x_i
2. Find its k nearest neighbors (also minority class)
3. Randomly select one neighbor x_j
4. Create a new synthetic point **along the line** between x_i and x_j:

$$x_{new} = x_i + \lambda \cdot (x_j - x_i), \quad \text{where } \lambda \in [0, 1]$$

This effectively "fills in" the feature space around minority samples, giving the model a richer understanding of the minority class.

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# CRITICAL: Split BEFORE applying SMOTE!
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Apply SMOTE only to training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Now train your model
model.fit(X_train_resampled, y_train_resampled)

# Evaluate on original (non-resampled) test data
model.score(X_test, y_test)
```

Listing 1: Using SMOTE in Python

Critical SMOTE Warning

NEVER apply SMOTE before train/test split!

If you SMOTE the entire dataset first, then split, synthetic data may leak between train and test sets, causing artificially inflated test performance (data leakage).

Correct order:

1. Split data into train/test
2. Apply SMOTE to training set only
3. Evaluate on original (un-SMOTEd) test set

5 AutoML: Blackbox vs. Glassbox

Definition: AutoML

AutoML (Automated Machine Learning) automates the process of:

- Feature engineering
- Model selection
- Hyperparameter tuning
- Model evaluation

You provide data, it provides a trained model.

5.1 Blackbox AutoML

- **How it works:** Upload data, click "train," get a model
- **Examples:** DataRobot, some commercial tools
- **Problem:** You can't see how the model was built. When things go wrong (and they will), you can't diagnose or fix them. For enterprise use cases requiring auditability, this is a dealbreaker.

5.2 Glassbox AutoML (Databricks Approach)

- **How it works:** Same automation, but every step is exposed as a **notebook**
- **What you get:**
  1. **Data Exploration Notebook:** Automatic EDA with profiling
  2. **Best Model Notebook:** Full source code for the winning model
  3. **MLflow Integration:** All experiments tracked
  4. **SHAP Explanations:** Feature importance built in
- **Advantage:** "Citizen data scientists" can start quickly, but experts can take over and customize

Table 4: *Models Available in Databricks AutoML*

Classification	Regression	Forecasting
Decision Trees	Decision Trees	Prophet
Random Forests	Random Forests	Auto-ARIMA
Logistic Regression	Linear Regression	DeepAR
XGBoost	XGBoost	
LightGBM	LightGBM	

Example: AutoML Demo Walkthrough

Scenario: Customer churn prediction

Steps:

1. Select dataset (churn table from Unity Catalog)
2. Choose target column ("Churn")

3. Set evaluation metric (F1 Score—because churn is imbalanced!)
4. Set timeout (15 minutes)
5. Click "Start AutoML"

Results:

- AutoML runs Decision Trees, Random Forest, XGBoost, LightGBM in parallel
- **Best model:** LightGBM (based on F1 score)
- Click "View notebook for best model" to see full source code
- All experiments logged to MLflow
- SHAP analysis shows which features matter most

6 Data Classification: Protecting Sensitive Information

Models can inadvertently leak sensitive information (PII). If your training data contains social security numbers and someone asks the model about them...

Definition: PII - Personally Identifiable Information

Information that can identify an individual:

- Name, phone number, email address
- Social Security Number, driver's license
- IP address, location data
- Bank account numbers

PII must be detected and removed before model training.

6.1 Automated Data Classification in Databricks

Unity Catalog can automatically scan tables and detect sensitive columns:

1. Enable "Data Classification" at the catalog level
2. System scans all tables (takes about 15 minutes)
3. Results show which columns contain: Name, Phone, Email, SSN, IP Address, etc.
4. Tags are automatically applied for governance

This prevents accidentally feeding PII into models, which could create both legal liability and privacy leaks.

7 Real-World Case Study: Fraud Detection at Scale

Eric Gieseke shares a real production fraud detection system he built. This is where all the theory meets hard engineering constraints.

7.1 Business Requirements

Table 5: *Fraud Detection System Requirements*

Requirement	Target
Accuracy	High confidence—catch real fraud, don't block legitimate transactions
Latency	**15 milliseconds** (ms) response time
Throughput	50,000+ transactions per second (TPS)
Maintainability	Easy to add/modify features and rules

Example: Why 15ms is Incredibly Hard

For context:
- A typical hard disk seek time: 10-15ms
- A network round-trip: 1-100ms depending on distance
- Human blink: 100-400ms

The entire fraud decision—load customer history, compute features, run ML model, apply rules, return decision—must happen in the time it takes a hard disk to **find** data, let alone read it.

7.2 Features for Fraud Detection

The data science team developed hundreds of features:

Table 6: *Example Fraud Detection Features*

Feature Type	Examples
Transaction-based	Distance from customer's home address
	Difference from average transaction amount
	Number of transactions today
	Time since last transaction
Dimension-based	Merchant's average transaction amount
	Customer's transaction frequency
	Terminal's fraud history
	Geographic region risk score

7.3 Architecture: Lambda Architecture

To meet both 15ms real-time AND large-scale batch requirements, they used **Lambda Architecture**.

Lambda Architecture Overview

Lambda Architecture splits data processing into three layers:

1. Batch Layer (Slow but Complete):

- Stores all historical data (immutable)
- Runs nightly/hourly batch jobs (Hadoop/Spark)
- Computes features on entire dataset (e.g., "customer's average transaction over 1 year")
- Results stored in Feature Store

2. Speed Layer (Fast but Incremental):

- Processes real-time events as they arrive
- Uses Complex Event Processing (CEP)
- Computes real-time features (e.g., "transactions in last 10 minutes")

3. Serving Layer (Query Layer):

- Combines batch and speed results
- Serves queries with low latency
- Cassandra used for this layer

7.4 Key Innovation 1: Metadata-Driven Code Generation

- **Problem:** Hundreds of features need code for both batch (SQL) AND real-time (CEP language). Writing and maintaining both is error-prone.
- **Solution:** Define features as **metadata**, not code.

```
Feature: customer_avg_transaction_30d
Type: Average
Field: transaction_amount
Window: 30 days
Dimension: customer_id
```

- **Code Generator** reads metadata and automatically produces:
  - SQL for batch processing (Spark)
  - EPL for real-time processing (CEP)
- **Benefit:** Add a new feature by editing metadata, not writing two sets of code

7.5 Key Innovation 2: Circular Buffer (Ring Buffer)

Example: The Problem

To compute "customer's average transaction amount over last 7 days," you'd normally:

1. Query database for all transactions in last 7 days
2. Sum them up
3. Divide by count

For millions of customers, this is **impossible in 15ms**.

Circular Buffer Solution

Instead of storing individual transactions, store **aggregates per time bucket**.
Structure (7-day buffer):

Sun	Mon	Tue	Wed	Thu	Fri	Sat
(count, sum)	(count, sum)	(count, sum)	(count, sum)	(count, sum)	(count, sum)	(count, sum)

Example:

```
Sun: (3, $150)   Mon: (5, $280)   Tue: (2, $90)   ...
```

To compute 7-day average:

$$\text{Average} = \frac{\sum \text{sums}}{\sum \text{counts}} = \frac{150 + 280 + 90 + \ldots}{3 + 5 + 2 + \ldots}$$

Memory footprint: Just 14 numbers per customer per feature (2 numbers $\times$ 7 days)
When day changes: Overwrite oldest bucket with zeros, start fresh
This was novel enough that the company **patented** it.

7.6 Key Innovation 3: Cassandra as Feature Store

- **Why Cassandra?**
  - Write speed: Extremely fast (writes are "fire and forget")
  - **Read speed: 2.3ms achieved**—critical for 15ms budget
  - Scalable to petabytes
  - Supports wide rows (billions of columns per row)
- **Data Model:**
  - Only 2 tables: **Events** (fact table) and **Dimensions**
  - New features = new columns (schema-less flexibility)
  - Sparse data handled efficiently

7.7 System Flow Summary

1. **Fraud Analyst** defines new feature in Metadata Service
2. **Batch Processing** (nightly) computes historical feature values → stores in Cassandra
3. **Code Generator** creates real-time CEP code
4. **Customer** swipes card at merchant
5. **Payment Service** calls Fraud Detection Service
6. **Fraud Detection** (within 15ms):
   (a) Retrieves pre-computed features from Cassandra (2.3ms)
   (b) Computes real-time features using Circular Buffers (in-memory)
   (c) Runs ML model + rules

   (d) Returns "Approve" or "Decline"

7. **Customer** completes purchase (or doesn't)

7.8 Disaster Recovery

For mission-critical payment systems:

- Two data centers (US and Europe)
- If US fails, traffic automatically routes to Europe
- Slightly higher latency acceptable vs. complete outage
- Data replication between centers

8 Summary: One-Page Quick Reference

ML Errors: Bias vs. Variance

- **Bias (Underfitting):** Model too simple. Both train and test error high.
- **Variance (Overfitting):** Model too complex. Train error low, test error high.
- **Tradeoff:** Simple $\leftrightarrow$ Complex. Find the sweet spot.

Model Bias (Fairness)

- **Cause 1:** Human bias in data (under-representation of groups)
- **Cause 2:** Poor quality labels, mislabeled data
- **Fix:** Ensure representative, diverse training data (most important!)

Class Imbalance (99-1 Problem)

- **Problem:** Accuracy is meaningless (99% by always predicting majority)
- **Metrics:** Use **F1 Score** (harmonic mean of Precision and Recall)
- **Fix: SMOTE** (create synthetic minority samples)
- **Warning:** Apply SMOTE to training set **only**, after train/test split!

AutoML: Blackbox vs. Glassbox

- **Blackbox:** Magic model, no visibility (enterprise unfriendly)
- **Glassbox:** Full notebooks, MLflow tracking, SHAP explanations
- **Use case:** Quick baseline, data validation, citizen data scientists

Fraud Detection at 15ms

- **Architecture:** Lambda (Batch + Speed + Serving layers)
- **Key 1: Circular Buffer** - Store (count, sum) per time bucket, not individual records
- **Key 2: Cassandra** - 2.3ms reads for feature retrieval
- **Key 3: Code Generation** - Define features as metadata, auto-generate SQL/CEP
- **Lesson:** The difference between 15ms and 100ms is the difference between possible and impossible