Underfitting and Overfitting

Using MoDeVa to Analyze the Bias-Variance Tradeoff

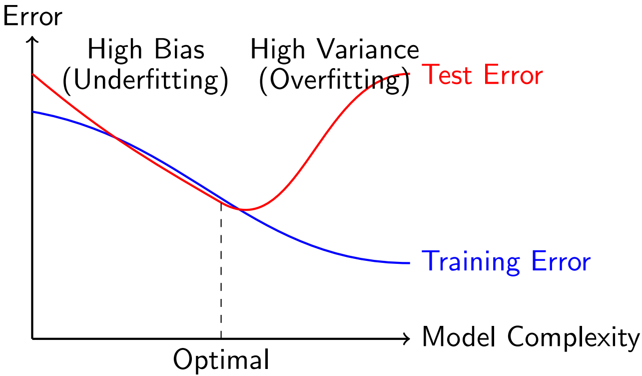
July 5, 2025

Outline

The Bias-Variance Tradeoff

Concept

The bias-variance tradeoff explains the relationship between a model's ability to fit training data and its generalization to unseen data.



Goal: Finding the sweet spot that minimizes both bias and variance to create a model that generalizes well to unseen data.

Empirical Risk Decomposition

The expected prediction error (empirical risk) for a model $\hat{f}$ at point x can be decomposed as:

$$\text{Err}(x) = \underbrace{(f(x) - E[\hat{f}(x)])^2}_{\text{Bias}^2} + \underbrace{E[(\hat{f}(x) - E[\hat{f}(x)])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible Error}}$$

- **Bias**: Systematic error from incorrect assumptions about the data (underfitting)
- **Variance**: Error from sensitivity to small fluctuations in training data (overfitting)
- **Irreducible Error**: Noise inherent in the problem (cannot be eliminated)

Key Insight

As model complexity increases, bias decreases but variance increases

Training vs. Test Error

Training Error

- Underestimates true error due to fitting noise
- Captures part of bias term
- Does **not** capture variance
- Decreases with model complexity

Test Error

- Includes both bias and variance
- Better estimate of true error
- Independent of training process
- U-shaped curve with model complexity

Generalization Gap

The difference between test and training error:

$$\text{Gap} = \text{Error}_{\text{test}} - \text{Error}_{\text{train}}$$

Overfitting Characterization

Underfitting (High Bias)

- Both training and testing errors are high
- Gap between them is small or negligible
- Model is too simple to capture patterns
- Predictions are consistently off in the same direction

Overfitting (High Variance)

- Training error is low
- Testing error is significantly higher
- Large gap between train and test errors
- Model captures noise rather than signal
- Predictions vary widely with small data changes

Key Insight

The generalization gap directly measures the degree of overfitting

Practical Applications of Gap Analysis

Model Selection

- Choose models minimizing gap while maintaining acceptable training error
- Use gap trends to guide complexity decisions
- Compare architectures based on gap stability

Training Process

- Monitor gap for early stopping
- Adjust regularization based on gap
- Balance model capacity against gap size
- Guide hyperparameter tuning

Performance Evaluation

- Compare models using both errors and gaps
- Consider gap stability across runs
- Account for dataset size effects
- Evaluate robustness via gap consistency

Local Generalization Gap

Concept

Instead of one global gap, we can analyze gaps in specific regions of the feature space to identify localized underfitting or overfitting.

Local generalization gap for region R in feature space:

$$\text{Gap}(R) = \frac{1}{|R_{\text{test}}|} \sum_{(x,y)\in R_{\text{test}}} L(y, \hat{f}(x)) - \frac{1}{|R_{\text{train}}|} \sum_{(x,y)\in R_{\text{train}}} L(y, \hat{f}(x))$$

where:

- R is a region in feature space
- $|R_{\text{train}}|$ is number of training points in R
- $|R_{\text{test}}|$ is number of test points in R
- $L(y, \hat{f}(x))$ is the loss function

Methods for Gap Slicing

1. Univariate Partitioning

- Analyze one feature at a time, partition feature into quantiles or bins
- Compute gap for each bin, identify features with high gap regions for each feature j:

$$R_j^k = \{x | q_k \leq x_j < q_{k+1}\}$$

2. Multivariate Region Detection

- Analyze feature interactions, create bins across multiple features
- Identify complex regions with high gaps for features i, j:

$$R_{i,j}^{k,l} = \{x | q_k \leq x_i < q_{k+1} \wedge q_l \leq x_j < q_{l+1}\}$$

Problematic Regions

Flag regions where: $\text{Gap}(R) > \mu_{\text{gap}} + \beta \cdot \sigma_{\text{gap}}$

Slicing Overfitting in MoDeVa

```
# Create a testsuite that bundles dataset and model
from modeva import TestSuite
ts = TestSuite(ds, model_lgbm) # store bundle of dataset and
    model

# overfit (gap) slicing for feature "season"
results = ts.diagnose_slicing_overfit(
    train_dataset="train",
    test_dataset="test",
    features="season",
    method = "quantile",  # binning method
    metric="MAE",         # error metric
    threshold=0.0065)     # gap threshold
results.table
# To visualize the results
results.plot()

```

This generates a table and visualization showing generalization gaps across different season values

Analyzing Multiple Features

```python
# Slicing for a Set of Features with automated binning
results = ts.diagnose_slicing_overfit(
    train_dataset="train",
    test_dataset="test",
    features=(("hr", ), ("workingday",), ("atemp", )),
    method="auto-xgb1",  # automated binning using XGBoost
    metric="MAE",
    threshold=0.0065)
results.table
# To visualize a single feature
results.plot(name="atemp")

```

This analyzes gaps across multiple features independently

Auto-XGB1 Binning

Uses depth-1 XGBoost tree to find optimal split points based on target relationship

Feature Interaction Analysis

```python
# 2-Feature Interaction Slicing: Two dimensional slicing
results = ts.diagnose_slicing_overfit(
    train_dataset="train",
    test_dataset="test",
    features=("hr", "atemp"),
    method="auto-xgb1",
    metric="MAE",
    threshold=0.0065)
results.table

```

This examines how combinations of feature values affect the generalization gap

Benefits:

- Identifies complex interactions causing overfitting
- Reveals feature combinations where model struggles
- Provides detailed insights beyond single-feature analysis
- Guides more targeted remediation strategies

Analyzing High-Gap Regions

```python
# Retrieving samples below threshold value
from modeva.testsuite.utils.slicing_utils import get_data_info
data_info = get_data_info(res_value=results.value)["hr"]
data_info

# Comparing distribution difference between high-gap and low-gap regions
data_results = ds.data_drift_test(
    **data_info,
    distance_metric="PSI",
    psi_method="uniform",
    psi_bins=10)
data_results.plot("summary")

```

This compares data distributions between regions with high and low generalization gaps

Model Comparison

```python
# Compare overfitting patterns between models
tsc = TestSuite(ds, models=[model_lgbm, model_xgb])
results = tsc.compare_slicing_overfit(
    train_dataset="train",
    test_dataset="test",
    features="hr",
    method="quantile",
    bins=10,
    metric="MAE")

```

This compares generalization gaps across different models for the same feature regions

What to look for:

- Which model shows smaller gaps overall?
- Do models have complementary strengths across regions?
- Are there regions where all models struggle (potential data issues)?
- Which model is more consistent across different regions?

Characterizing Weak Regions

1. **Data Sparsity**
   - Are high-gap regions sparsely represented in training data?
   - Insufficient samples lead to poor generalization
   - Analyze data density in problematic regions
   - Consider targeted data collection

2. **Complexity Measure**
   - Is the model too complex for available data?
   - Check model capacity vs. data volume
   - Analyze feature interactions and importance
   - Consider simpler models for sparse regions

3. **Uncertainty Assessment**
   - Do high-gap regions have inherently high noise?
   - Evaluate prediction variance in problematic areas
   - Check for conflicting examples in training data
   - Consider probabilistic approaches for uncertain regions

Connecting Overfitting and Robustness

Theoretical Connection

Overfitting and lack of robustness are related through the lens of local Lipschitz continuity

Manifestations:

- **Decision Boundary Complexity**
  - Overfit models create complex, wiggly decision boundaries
  - More sensitive to small perturbations
  - Higher local Lipschitz constants (larger changes for small input differences)
- **Feature Sensitivity**
  - Overfit models rely heavily on noise in features
  - Small input changes produce large output changes
  - Less stable predictions in deployment
- **Neighborhood Consistency**
  - Robust models give similar predictions for similar inputs
  - Overfit models show erratic local behavior
  - Affects adversarial robustness

Data-Centric Solutions

1. **Targeted Data Collection**
   - Focus on high-gap regions R
   - Use active learning to select informative samples
   - Apply stratified sampling based on gap size
   - Engage domain experts for difficult cases

2. **Data Cleaning**
   - Remove noisy samples in high-gap regions
   - Validate labels in problematic areas
   - Handle outliers affecting local gaps
   - Address class imbalance issues

Key Principle

Rather than just collecting more data, focus on improving data quality and representation in high-gap regions

Feature Engineering Solutions

1. Interaction Features

- Create new features for high-gap regions
- Combine features involved in problematic interactions
- Engineer domain-specific indicators
- Create polynomial or nonlinear transformations

2. Domain-Specific Transformations

- Apply log transforms for skewed features
- Use binning for nonlinear relationships
- Create feature combinations based on domain knowledge
- Apply constraints such as monotonicity

3. Feature Selection

- Weight features by gap reduction potential
- Remove noisy features contributing to overfitting
- Use L1 regularization to promote sparsity
- Consider different feature sets for different regions

Model-Centric Approaches

1. Alternative Modeling Frameworks

- Try different algorithms for high-gap regions
- Use simpler models where appropriate

2. Local Model Enhancement

- Tune hyperparameters for specific regions
- Apply different preprocessing per region
- Use locally weighted training
- Implement region-specific validation

3. Ensemble Strategies

- Implement Mixture of Experts (MoE)
- Train specialized models for difficult regions
- Weight models by local performance

4. Loss Function Adjustments

- Apply L1/L2 regularization
- Use gap-weighted loss functions
- Implement region-specific penalties
- Adjust sample weights based on gaps

Implementation Framework

Prioritization

- Rank regions by gap size
- Assess feasibility of solutions
- Consider implementation cost
- Focus on high business impact regions

Validation

- Monitor gap reduction
- Check for negative side effects
- Validate on holdout set
- Test impact on overall performance

Summary: Bias-Variance Tradeoff

- **Understanding Gaps**: The generalization gap (test error - train error) directly measures overfitting
- **Localized Analysis**: Slicing reveals regions in feature space with high generalization gaps
- **Targeted Remediation**: Address specific problematic regions rather than applying global solutions
- **Multiple Strategies**: Combine data-centric, feature engineering, and model-centric approaches
- **Systematic Implementation**: Prioritize, validate, and iterate to efficiently improve model generalization

Key Takeaway

Finding the optimal balance between bias and variance requires both global and local analysis of model performance to ensure consistent generalization across the entire feature space.