Statistical Learning Theory

Foundations for Data Science Applications

Diogo Ribeiro
ESMAD – Escola Superior de Média Arte e Design
Lead Data Scientist, Mysense.ai

November 16, 2025

Outline

From Statistics to Machine Learning

The Evolution of Learning from Data:

- Classical Statistics (1900-1970):
  - Fixed parametric models
  - Hypothesis testing framework
  - Small sample theory
  - Focus on inference and explanation
- Machine Learning (1980-present):
  - Algorithmic approach
  - High-dimensional data
  - Prediction-focused
  - Computational methods

Modern Data Science

- Massive datasets
- Complex patterns
- Real-time decisions
- Business impact

The Challenge

How do we learn reliable patterns from finite data that generalize to unseen examples?

Real-World Learning Problems

Application	**Input (X)**	**Output (Y)**	**Goal**
Netflix Recommendations	User history, ratings	Movie preferences	Predict ratings
Medical Diagnosis	Symptoms, test results	Disease presence	Classification
Financial Trading	Market data, news	Price movements	Forecast returns
Fraud Detection	Transaction features	Fraud/legitimate	Binary classification
Drug Discovery	Molecular structure	Biological activity	Regression

Common Pattern

We observe training examples $(x_1, y_1), \ldots, (x_n, y_n)$ drawn from unknown distribution $P(X, Y)$ and want to predict Y for new X.

The Fundamental Learning Setup

Mathematical Framework:

$$\text{Input space:} \quad \mathcal{X} \subseteq \mathbb{R}^d \tag{1}$$

$$\text{Output space:} \quad \mathcal{Y} \tag{2}$$

$$\text{Hypothesis class:} \quad \mathcal{H} = \{h : \mathcal{X} \to \mathcal{Y}\} \tag{3}$$

$$\text{Loss function:} \quad \ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+ \tag{4}$$

Unknown joint distribution:
$(X, Y) \sim P(X, Y)$
Training data: $S = \{(x_i, y_i)\}_{i=1}^n$ i.i.d. from P

Risk Functions:

Population Risk

$$R(h) = \mathbb{E}_{(X,Y)\sim P}[\ell(h(X), Y)]$$

True generalization error

Empirical Risk

$$R_n(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(h(x_i), y_i)$$

Training error

The Goal

Find $h^* \in \operatorname{argmin}_{h\in\mathcal{H}} R(h)$ but we only observe $R_n(h)$

Risk Decomposition: Understanding Prediction Error

Fundamental Decomposition:
For any learning algorithm $\hat{h}$ trained on dataset S:

$$R(\hat{h}) = R(\hat{h}) - R(h^*_{\mathcal{H}}) + R(h^*_{\mathcal{H}}) - R^* + R^* \tag{5}$$

$$= \underbrace{R(\hat{h}) - R(h^*_{\mathcal{H}})}_{\text{Estimation Error}} + \underbrace{R(h^*_{\mathcal{H}}) - R^*}_{\text{Approximation Error}} + \underbrace{R^*}_{\text{Bayes Risk}} \tag{6}$$

where:

- $R^* = \inf_f R(f)$ is the Bayes risk (irreducible error)
- $h^*_{\mathcal{H}} = \text{argmin}_{h \in \mathcal{H}} R(h)$ is the best function in our class

Bayes Risk	Approximation Error	Estimation Error
Noise in data	**Model bias**	**Finite sample effects**
• Measurement error	• Limited hypothesis class	• Random sampling

Common Loss Functions

Regression Tasks:

Squared Loss: $\ell(y, \hat{y}) = (y - \hat{y})^2$ (7)

Absolute Loss: $\ell(y, \hat{y}) = |y - \hat{y}|$ (8)

Huber Loss: $\ell_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$ (9)

Properties:

- Squared: Differentiable, sensitive to outliers
- Absolute: Robust, non-differentiable at 0
- Huber: Best of both worlds

Classification Tasks:

0-1 Loss: $\ell(y, \hat{y}) = \mathbb{I}[y \neq \hat{y}]$ (10)

Hinge Loss: $\ell(y, \hat{y}) = \max(0, 1 - y\hat{y})$ (11)

Logistic Loss: $\ell(y, \hat{y}) = \log(1 + e^{-y\hat{y}})$ (12)

Cross-entropy: $\ell(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log$ (13)

Trade-offs:

- 0-1: What we care about, but non-convex
- Hinge: Convex surrogate, sparse solutions
- Logistic: Smooth, probabilistic interpretation

Bias-Variance Decomposition

Consider regression with squared loss. For a fixed point x, decompose the expected squared error:

$$\mathbb{E}[(\hat{f}(x) - y)^2] = \mathbb{E}[(\hat{f}(x) - f(x) + f(x) - y)^2] \tag{14}$$

$$= \mathbb{E}[(\hat{f}(x) - f(x))^2] + \mathbb{E}[(f(x) - y)^2] + 2\mathbb{E}[(\hat{f}(x) - f(x))(f(x) - y)] \tag{15}$$

$$= \mathbb{E}[(\hat{f}(x) - f(x))^2] + \sigma^2 + 0 \tag{16}$$

where $f(x) = \mathbb{E}[Y|X = x]$ and $\sigma^2 = \mathrm{Var}[Y|X = x]$.

Further decomposition:

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)] + \mathbb{E}[\hat{f}(x)] - f(x))^2] \tag{17}$$

$$= \mathrm{Var}[\hat{f}(x)] + (\mathbb{E}[\hat{f}(x)] - f(x))^2 \tag{18}$$

$$= \text{Variance} + \text{Bias}^2 \tag{19}$$

Final Decomposition

Understanding Bias and Variance

Definition (Bias)

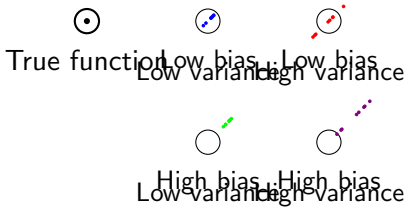
$$\text{Bias}[\hat{f}(x)] = \mathbb{E}[\hat{f}(x)] - f(x)$$

Systematic error - how far off is our method on average?

Definition (Variance)

$$\text{Variance}[\hat{f}(x)] = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]$$

Random error - how much does our method vary across datasets?



The Tradeoff

Complex models: Low bias, high variance
Simple models: High bias, low variance

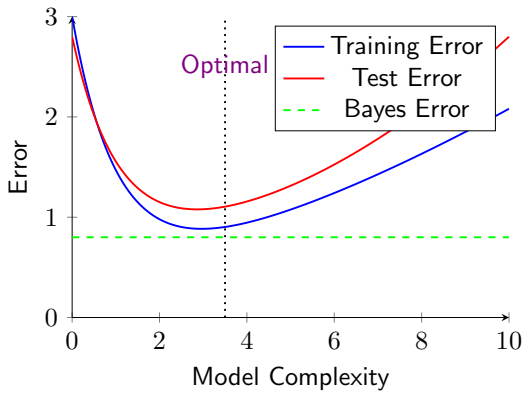
Interactive Demonstration: Polynomial Regression

Example: Fitting polynomials of different degrees to a noisy sine wave

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline

def true_function(x):
    return 1.5 * np.sin(2 * np.pi * x)

def generate_data(n_samples=50, noise_std=0.3):
    X = np.random.uniform(0, 1, n_samples)
    y = true_function(X) + np.random.normal(0, noise_std, n_samples)
    return X.reshape(-1, 1), y

# Generate multiple datasets for bias-variance analysis
n_datasets = 100
degrees = [1, 4, 15]
X_test = np.linspace(0, 1, 100).reshape(-1, 1)
y_test_true = true_function(X_test.ravel())

for degree in degrees:
    predictions = []
    for _ in range(n_datasets):
        X_train, y_train = generate_data()
        model = Pipeline([
            ('poly', PolynomialFeatures(degree)),
            ('linear', LinearRegression())
        ])
```

Model Complexity and the U-Shaped Curve

Training vs Test Error:



Key Observations:

- Underfitting region: Both training and test error are high
- Sweet spot: Test error is minimized
- Overfitting region: Gap between training and test error grows

Practical Implications

- Use validation to find optimal complexity
- Regularization helps control overfitting
- More data allows more complex models
- Early stopping prevents overfitting

Modern Puzzle

PAC Learning Theory

Definition (PAC Learnability)

A hypothesis class $\mathcal{H}$ is **PAC-learnable** if there exists an algorithm A and polynomial function $p(\cdot,\cdot,\cdot,\cdot)$ such that:
For any distribution D over $\mathcal{X}$, any target concept $c \in \mathcal{H}$, and any $\epsilon, \delta > 0$:
If $m \geq p(1/\epsilon, 1/\delta, \text{size}(c), \text{size}(\mathcal{X}))$, then

$$\mathbb{P}[\text{error}(A(S)) \leq \epsilon] \geq 1 - \delta$$

where S is a training set of size m drawn i.i.d. from D.

Interpretation:

- **Probably**: With high probability $(1 - \delta)$
- **Approximately**: Within ϵ
- **Correct**: Low generalization error

Sample Complexity

The function $m(\epsilon, \delta)$ tells us how many examples we need to learn with accuracy ϵ and

VC Dimension: Measuring Hypothesis Class Complexity

Definition (Shattering)

A set of points $\{x_1, \ldots, x_k\}$ is **shattered** by hypothesis class $\mathcal{H}$ if for every possible labeling $\{y_1, \ldots, y_k\} \in \{0, 1\}^k$, there exists $h \in \mathcal{H}$ such that $h(x_i) = y_i$ for all i.

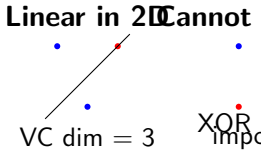
Definition (VC Dimension)

The **VC dimension** of $\mathcal{H}$ is the size of the largest set that can be shattered by $\mathcal{H}$.

$$\text{VC}(\mathcal{H}) = \max\{k : \exists \text{ set of size } k \text{ that can be shattered by } \mathcal{H}\}$$

Examples:

- **Linear classifiers in** $\mathbb{R}^d$: VC dim $= d + 1$
- **Decision trees of depth** h: VC dim $\approx 2^h$
- **Neural networks**: Complex, depends on architecture
- **Nearest neighbor**: Infinite VC dimension



Generalization Bounds

The power of VC theory: Provides distribution-free generalization guarantees.

Theorem (VC Generalization Bound)

Let $\mathcal{H}$ be a hypothesis class with VC dimension d. Then with probability at least $1-\delta$, for all $h \in \mathcal{H}$:

$$R(h) \leq R_n(h) + \sqrt{\frac{8d\log(2n/d) + 8\log(4/\delta)}{n}}$$

Sample Complexity: For (ϵ, δ)-PAC learning:

$$m = O\left(\frac{d + \log(1/\delta)}{\epsilon^2}\right)$$

Key insights:

- Linear dependence on VC dimension

For Finite Hypothesis Classes

If $|\mathcal{H}| = N < \infty$, then with probability $1-\delta$:

$$R(h) \leq R_n(h) + \sqrt{\frac{\log(N) + \log(1/\delta)}{2n}}$$

Sample complexity: $m = O\left(\frac{\log(N)+\log(1/\delta)}{\epsilon^2}\right)$

The Model Selection Challenge

Scenario: We have multiple hypothesis classes $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_K$ and need to choose the best one.

Why is this hard?

- Training error is optimistically biased
- More complex models always fit training data better
- Need unbiased estimate of generalization error
- Limited data for evaluation

Classical approach:

- Information criteria (AIC, BIC)
- Analytical penalties for complexity

The Holdout Method

1. Split data: Train (60%) / Validation (20%) / Test (20%)
2. Train each model on training set
3. Evaluate on validation set
4. Select model with best validation performance
5. Report final performance on test set

Problems with Holdout

- Wastes data (especially problematic for small datasets)

k-Fold Cross-Validation Algorithm:

Algorithm 1 k-Fold Cross-Validation

1: Split data into k roughly equal folds
2: **for** $i = 1$ to k **do**
3: Use fold i as validation set
4: Use remaining $k - 1$ folds as training set
5: Train model and compute validation error e_i
6: **end for**
7: Return $\text{CV}_k = \frac{1}{k} \sum_{i=1}^{k} e_i$

Common choices:

- $k = 5$ or $k = 10$ (good bias-variance tradeoff)
- $k = n$ (Leave-One-Out CV, LOOCV)

```python
from sklearn.model_selection import
    cross_val_score
from sklearn.linear_model import Ridge
import numpy as np

# Generate sample data
from sklearn.datasets import make_regression
X, y = make_regression(n_samples=100,
                       n_features=20,
                       noise=0.1,
                       random_state=42)

# Test different regularization strengths
alphas = np.logspace(-4, 2, 20)
cv_scores = []

for alpha in alphas:
    model = Ridge(alpha=alpha)
    scores = cross_val_score(
        model, X, y,
        cv=5,
        scoring='neg_mean_squared_error'
    )
    cv_scores.append(-scores.mean())

# Select best alpha
best_alpha = alphas[np.argmin(cv_scores)]
print(f"Best alpha: {best_alpha:.4f}")
```

Advanced Cross-Validation Techniques

Nested Cross-Validation:

- Outer loop: Model assessment
- Inner loop: Hyperparameter selection
- Provides unbiased estimate of generalization
- Essential for fair model comparison

Stratified Cross-Validation:

- Maintains class proportions in each fold
- Important for imbalanced datasets
- Reduces variance in estimates

Group Cross-Validation

When data has natural clusters (e.g., patients, companies):

- Ensure same group doesn't appear in train and validation
- Prevents data leakage
- More conservative but realistic estimates

Bootstrap Methods

Alternative to CV:

- Sample with replacement
- Out-of-bag samples for validation
- Good for small datasets

Case Study 1: Feature Selection for House Price Prediction

Problem: Predict house prices with 80+ features, many potentially irrelevant.

```python
import pandas as pd
from sklearn.model_selection import validation_curve
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston

# Load data (Boston housing as example)
X, y = load_boston(return_X_y=True)

# Create pipeline with feature selection
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('selector', SelectKBest(f_regression)),
    ('regressor', LinearRegression())
])

# Test different numbers of features
param_range = range(1, X.shape[1] + 1)
train_scores, val_scores = validation_curve(
    pipe, X, y,
    param_name='selector__k',
    param_range=param_range,
    cv=5,
    scoring='neg_mean_squared_error'
)
```

Case Study 2: Regularization in High-Dimensional Regression

Problem: Gene expression data with 5,000 features and 100 samples.

```python
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import validation_curve
import numpy as np

# Simulate high-dimensional data
np.random.seed(42)
n_samples, n_features = 100, 5000
X = np.random.randn(n_samples, n_features)
true_coef = np.zeros(n_features)
true_coef[:10] = np.random.randn(10)  # Only first 10 are relevant
y = X @ true_coef + 0.1 * np.random.randn(n_samples)

# Compare Ridge and Lasso
alphas = np.logspace(-3, 2, 20)

ridge_train, ridge_val = validation_curve(
    Ridge(), X, y, param_name='alpha', param_range=alphas,
    cv=5, scoring='neg_mean_squared_error'
)

lasso_train, lasso_val = validation_curve(
    Lasso(max_iter=2000), X, y, param_name='alpha', param_range=alphas,
    cv=5, scoring='neg_mean_squared_error'
)

print("Ridge vs Lasso in high-dimensional setting:")
print("- Ridge: Continuous shrinkage, keeps all features")
print("- Lasso: Sparse solutions, automatic feature selection")
```

Case Study 3: Model Selection in Practice

Problem: Credit scoring with multiple algorithm choices.

Model	CV Score	Std Error	Train Time	Interpretable?
Logistic Regression	0.845	0.012	0.1s	Yes
Random Forest	0.867	0.015	2.3s	Partial
Gradient Boosting	0.874	0.011	45s	Partial
Neural Network	0.871	0.018	15s	No
SVM (RBF)	0.863	0.014	8s	No

Statistical Considerations:

- Is difference between 0.874 and 0.871 significant?
- Paired t-test on CV folds
- Practical vs statistical significance
- Account for multiple comparisons

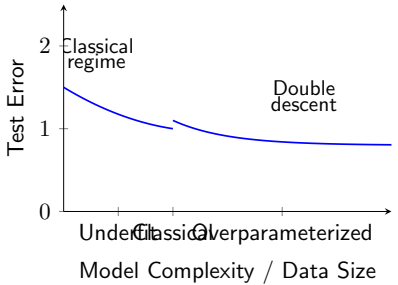
Business Considerations:

- Interpretability requirements (regulation)
- Prediction speed (real-time scoring)
- Training cost (model updates)
- Maintenance complexity

Beyond Classical Theory: Modern Challenges

The Double Descent Phenomenon:

- Classical theory: U-shaped test error curve
- Modern observation: Second descent in overparameterized regime
- Challenges bias-variance decomposition
- Common in deep learning



High-Dimensional Statistics:

- $p \gg n$ scenarios (genomics, finance)
- Curse of dimensionality
- Blessing of dimensionality (concentration)
- Sparsity assumptions crucial

Modern Theory Needs

- Implicit regularization in SGD
- Benign overfitting conditions
- Role of initialization and architecture
- Distribution-dependent bounds

Emerging Frontiers in Learning Theory

Causal Learning:

- Beyond correlation to causation
- Structural causal models
- Invariant risk minimization
- Domain adaptation and robustness

Meta-Learning:

- Learning to learn across tasks
- Few-shot learning
- Model-agnostic meta-learning (MAML)
- Bayesian optimization for hyperparameters

Continual Learning:

- Learning without forgetting

Federated Learning:

- Distributed learning with privacy
- Communication constraints
- Non-IID data distributions
- Differential privacy guarantees

Robust Learning:

- Adversarial examples
- Distribution shift
- Worst-case guarantees
- Certified defenses

Practical Implications

Modern ML applications require:

Key Takeaways

Fundamental Concepts:

- Risk decomposition: Understand sources of error
- Bias-variance tradeoff: Balance simplicity and complexity
- PAC learning: Formal guarantees for learnability
- VC dimension: Measure of hypothesis class complexity
- Cross-validation: Practical model selection tool

Practical Guidelines:

- Start with simple baselines

Modern Challenges:

- High-dimensional data requires new techniques
- Deep learning challenges classical theory
- Robustness and fairness are crucial
- Causality matters for decision-making

The Big Picture

Statistical learning theory provides:

- Principled foundation for ML
- Tools for understanding when/why methods work
- Guidance for method selection
- Framework for developing new algorithms

Next Steps in the Data Science Track

Immediate Next Topics:

1. **Feature Engineering & Selection**
   - Automated feature engineering
   - Dimensionality reduction
   - Feature importance methods
2. **Causal Inference**
   - From correlation to causation
   - Experimental design
   - Observational causal methods
3. **Model Interpretability**
   - SHAP, LIME, and friends
   - Global vs local explanations
   - Interpretable model classes

Hands-on Projects:

- Implement bias-variance decomposition from scratch
- Build cross-validation framework
- Apply theory to real dataset
- Compare multiple algorithms systematically

Further Reading:

- Hastie, Tibshirani, Friedman: *Elements of Statistical Learning*
- Shalev-Shwartz, Ben-David: *Understanding Machine Learning*
- Vapnik: *The Nature of Statistical Learning Theory*

Thank You

Questions & Discussion

Diogo Ribeiro
ESMAD – Escola Superior de Média Arte e Design
Lead Data Scientist, Mysense.ai

dfr@esmad.ipp.pt
https://orcid.org/0009-0001-2022-7072

Slides and code available at:
github.com/diogoribeiro7/academic-presentations

Next: Feature Engineering & Selection