Big Data in Finance

Week 1: Foundations of Machine Learning

Dr Daniele Bianchi

Spring 2026

Today's Agenda

Introduction: What Are We Trying to Do?

Problem Setup

The Fundamental Tradeoff: Bias vs. Variance

Model Validation: Estimating Test Error

Common Pitfalls in Financial Machine Learning

Python for Machine Learning

A Bird's Eye View on Empirical Applications

Summary and Next Steps

Introduction: What Are We Trying to Do?

The Core Question of This Module

Can we make **better use** of data to improve **predictions** about financial outcomes?

Examples we'll tackle in this course:

- Which stocks will outperform next month and why?
- Will this borrower default on their loan and why?

The challenge: Financial markets are noisy, correlations and volatility change over time, and many claimed patterns do not hold up out-of-sample.

A Concrete Example: Predicting Stock Returns

Setup: You're a portfolio manager trying to pick winning stocks

Data-Rich Environment

- Company financials: earnings, book value, debt levels
- Market data: past returns, trading volume, volatility
- Valuation ratios: price-to-earnings, price-to-book
- Momentum: how has the stock performed recently?

What Do You Want to Predict?

- Next month's stock return (or just: will it beat the market?)

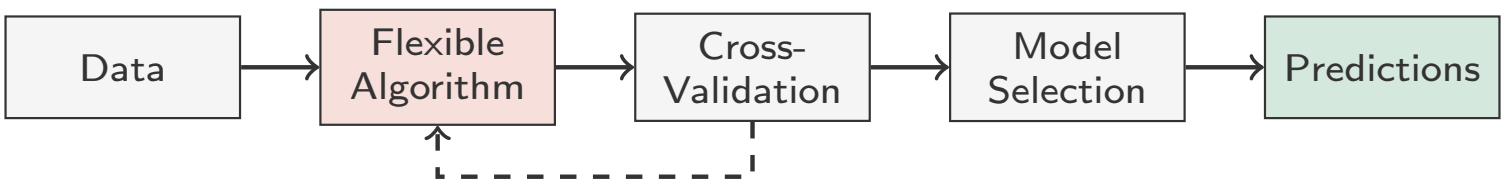
The Machine Learning (ML) Approach:

1. Collect historical data on stock characteristics and returns
2. Learn the relationship between characteristics and subsequent returns
3. Apply this relationship to make predictions as new data on characteristics arrive.

Learning from Data $\neq$ Fitting the Data

Traditional Econometrics

Machine Learning



	Econometrics	**Machine Learning**
Primary goal	Understand *why*	Predict *what*
Model structure	Often theory-driven	Mostly data-driven
Variables	Few, interpretable	Many, algorithm selects
Focus	Statistical significance	Out-of-sample accuracy

Punchline: ML for prediction, econometrics for causal understanding. Good practitioners combine both as much as possible.

Problem Setup

Notation: Keeping It Simple

Let's establish clear notation we will use throughout:

- x = **features** (the information we use to predict)
  - For a single observation: $x = (x_1, x_2, \ldots, x_p)$
  - Example: size, value ratio, momentum, profitability
- y = **target** (what we're trying to predict)
  - Example: next month's return
- n = number of observations (e.g., stock-months)
- p = number of features

Our data structure: We observe n examples:

$$(x^{(1)}, y^{(1)}),\ (x^{(2)}, y^{(2)}),\ \ldots,\ (x^{(n)}, y^{(n)})$$

Each $x^{(i)} = (x_1^{(i)}, \ldots, x_p^{(i)})$ is a vector of p features; each $y^{(i)}$ is the outcome.

The Goal: Learn a Prediction Rule

We want to find a function f that predicts Y from X:

$$\text{Predicted } Y = \widehat{f}(X)$$

N.B., $\widehat{f}$ signifies *"an estimate of the unknown f"*.

Concretely:

$$f(\text{size, value, momentum, ...}) = \text{predicted return}$$

What kind of function?

- **Linear:** $f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots$ (Lecture 2, 5)
- **Decision trees:** If size $>$ threshold, then... (Lecture 3, 6)
- **Neural networks:** Complex nonlinear combinations (Lecture 9, if time permits)

The challenge: Learn an f from historical data such that it works well on new, *unseen* data.

Why Prediction Is Hard: The Noise Problem

Reality: Even with perfect information, we cannot predict y exactly.

We can think of the data as generated by:

$$y = \underbrace{f(x)}_{\text{predictable part}} + \underbrace{\epsilon}_{\text{unpredictable noise}}$$

What is ϵ? Everything affecting y that is NOT in our features x:

- Information we don't have access to (e.g., private order flow)
- Events that occur *after* we make our prediction
- Measurement error in reported financials
- Pure randomness in how markets aggregate information

Key insight:

- $f(x)$ is what we can hope to learn (the "signal")
- ϵ is irreducible given our information set
- Better features can shrink ϵ, but never eliminate it

How Good Is Our Prediction? Measuring Error

Question: What do we mean by "as accurately as possible"?

Prediction error for a single observation:

$$\text{Error} = y - \widehat{f}(x) = \text{Actual} - \text{Predicted}$$

Problem: Errors can be positive or negative, so they cancel out

Solution: Square the errors (penalizes big mistakes more)

$$\text{Squared Error} = (y - \widehat{f}(x))^2$$

Average over all observations:

$$\text{Mean Squared Error (MSE)} = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \widehat{f}(x^{(i)}))^2$$

Lower MSE = More accurate predictions

The Learning Process in ML

Step 1: Choose a family of possible prediction functions

- Linear models, trees, neural networks, etc.

Step 2: Find the $\widehat{f}$ that minimizes prediction error on training data

$$\widehat{f} = \arg\min_{f} \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - f(x^{(i)}) \right)^2$$

Step 3: Evaluate on new data (we'll discuss this more soon)

Important!

The hat notation $\widehat{f}$ means "our estimate of f" — it's what we learn from data. The true f is unknown!

The Fundamental Tradeoff: Bias vs. Variance

A Critical Question

If we minimize error on our training data,
will predictions be good on new, **unseen** data?

Spoiler: Not necessarily!

Two distinct concepts:

- **Training error:** How well does $\widehat{f}$ fit the data we used to build it?
- **Test error:** How well does $\widehat{f}$ predict on new data?

Training error can be made *arbitrarily small* by using a sufficiently complex model. But this doesn't mean test error will be small!

What Makes Test Error Large?

Two sources of prediction error on new data:

1. **Bias** (Underfitting)
   - Model too simple
   - Misses real patterns
   - Training error $\approx$ Test error
   - Both higher than necessary

Example: Predicting returns using only firm size, when momentum and value also matter

2. **Variance** (Overfitting)
   - Model too complex
   - Fits noise, not just signal
   - Training error $<<$ Test error
   - Gap signals overfitting

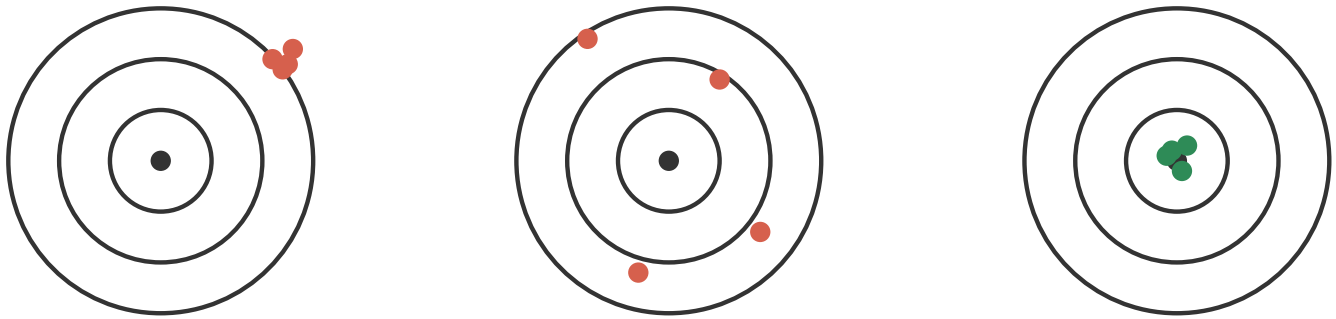
Example: Using 200 characteristics to predict returns with only 500 observations

The Tradeoff

Reducing bias (more complex model) typically increases variance, and vice versa.

Visualizing Bias and Variance

Thought experiment: What if we could repeat our analysis with different training samples?



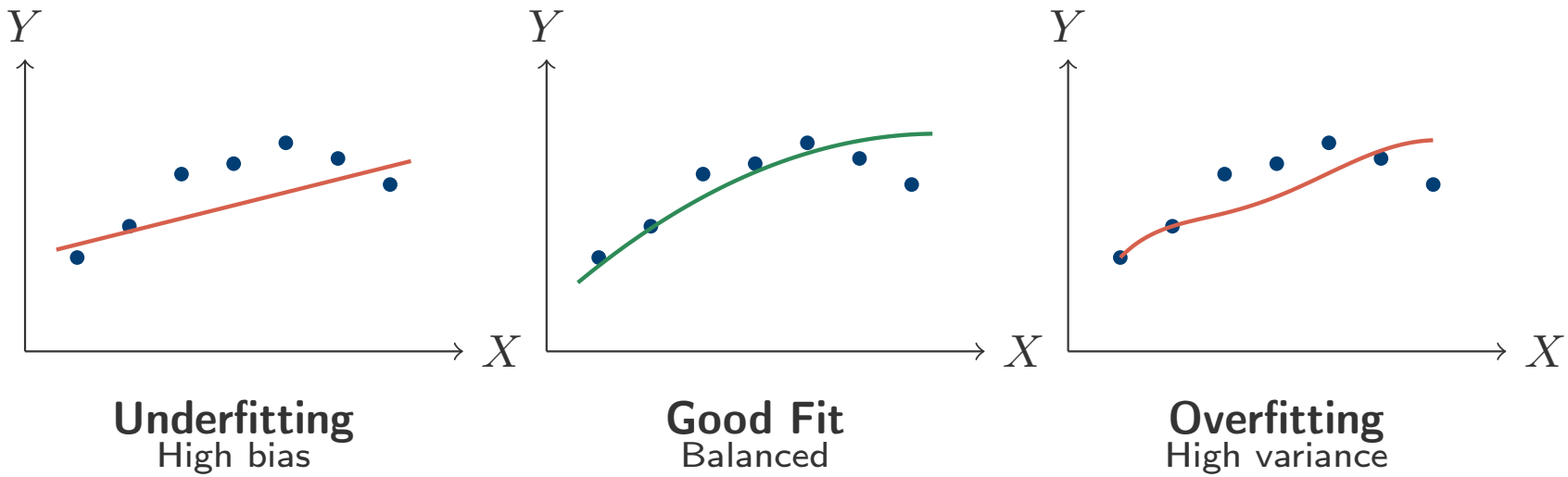
High Bias
(Consistently wrong)

High Variance
(Unstable predictions)

Low Both
(The goal)

- Bullseye = true relationship $f(X)$ we are trying to learn
- Each dart = prediction from model trained on a different sample
- **Bias:** How far is the *average* prediction from the truth?
- **Variance:** How much do predictions *scatter* across samples?

Visual Example: Fitting a Curve



- **Left:** Straight line misses the curvature in the data
- **Middle:** Smooth curve captures the overall pattern
- **Right:** Wiggly curve passes through every point — but will fail on new data

The Bias-Variance Decomposition

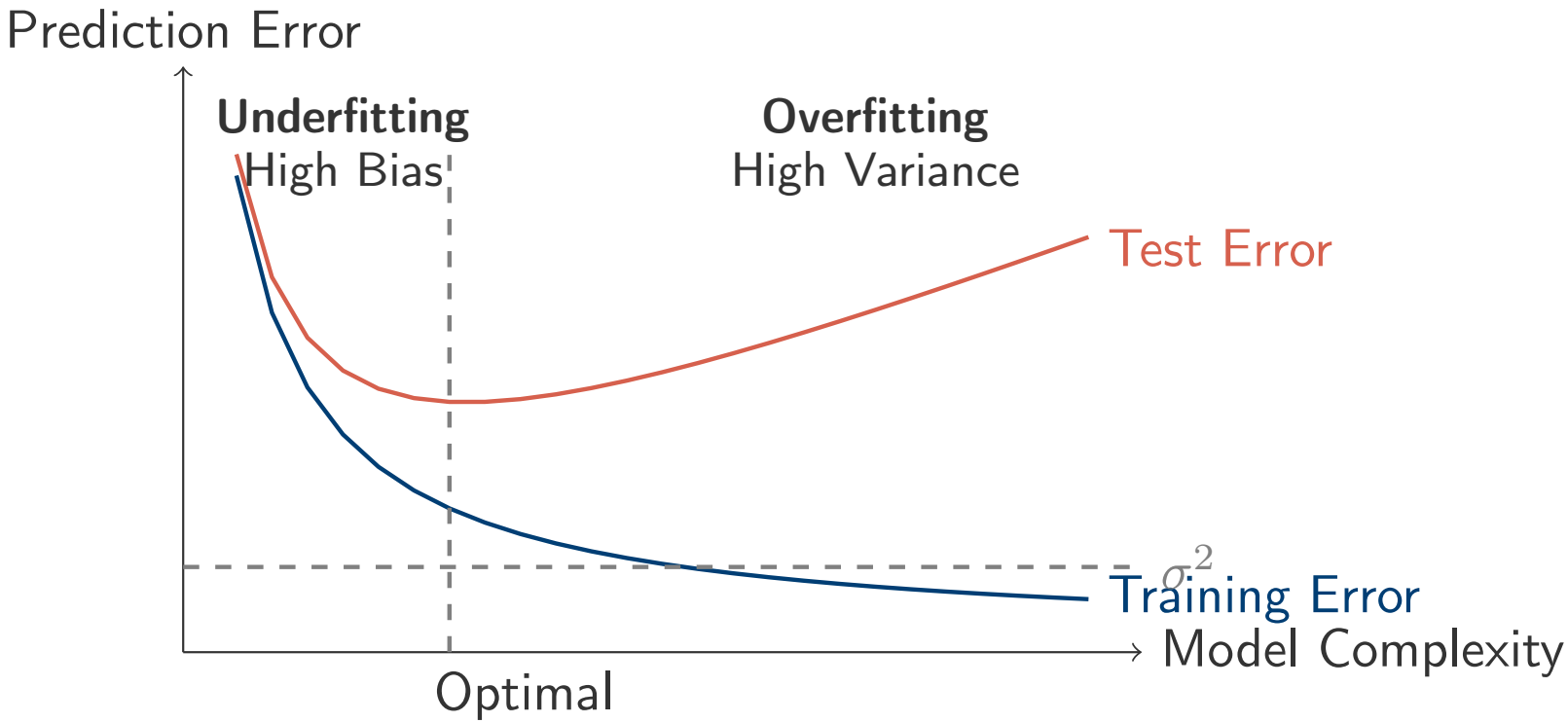
Mathematical fact: The expected test error decomposes into three parts

$$\underbrace{\mathbb{E}\left[(Y - \widehat{f}(X))^2\right]}_{\text{What we care about}} = \underbrace{\text{Bias}^2}_{\text{How Wrong is } \widehat{f}} + \underbrace{\text{Variance}}_{\text{How Unstable is } \widehat{f}} + \underbrace{\sigma^2}_{\text{Irreducible noise}}$$

- σ^2 is the inherent unpredictability — even the best model can't eliminate it
- We control Bias and Variance through model choice
- **Minimizing test error requires balancing Bias and Variance**

	Simple Model	**Complex Model**
Bias	High	Low
Variance	Low	High

Complexity vs. Error: The Classic Picture



Key observation:

- Training error always decreases with complexity
- Test error has a U-shape — there's a sweet spot
- **We must estimate test error to choose optimal complexity**

Why Overfitting Is Deadly in Finance

Finance is especially prone to overfitting because:

- Signal-to-noise ratio is very low (markets are near-efficient)
- Many candidate predictors (300+ characteristics documented)
- Limited independent observations (monthly data = 12 per year)
- Predictive relationships may change over time (regime shifts)

The evidence (Harvey, Liu & Zhu, 2016):

- Surveyed 300+ factors published in top finance journals
- Most factors that "worked" in-sample do not replicate
- Key culprit: researchers test many strategies, publish only winners

Mantra

If it seems too good to be true, it is probably overfitting

Model Validation: Estimating Test Error

The Validation Problem

We need to estimate test error, but we only have training data!

Naive approach: Use training error

- Problem: Training error is too optimistic (see previous discussion)
- Complex models can achieve zero training error while being useless

Solution: Hold out some data for validation

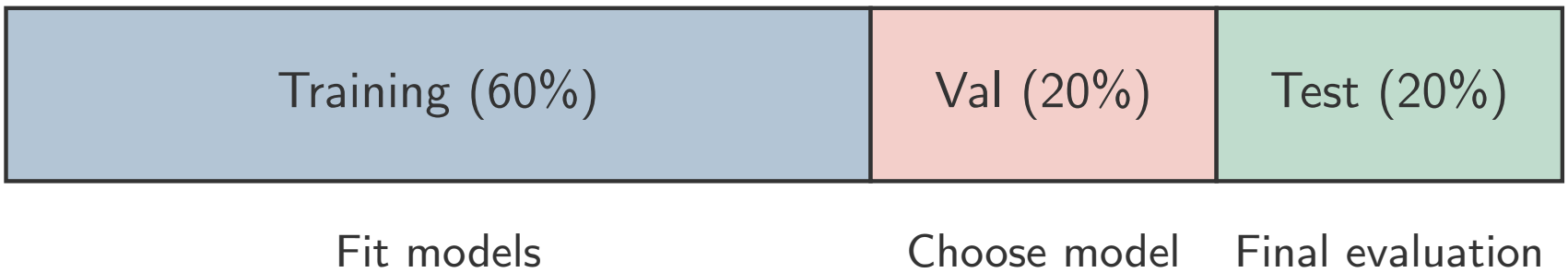
- Don't use it for training
- Use it to estimate how well we would do on new data

General Principle

The data you use to evaluate a model must be different from the data you used to build it

Train-Validation-Test Split

Standard approach: Divide your data into three parts



1. **Training set:** Fit different models
2. **Validation set:** Compare models, choose the best one
3. **Test set:** Evaluate final model (touch only once!)

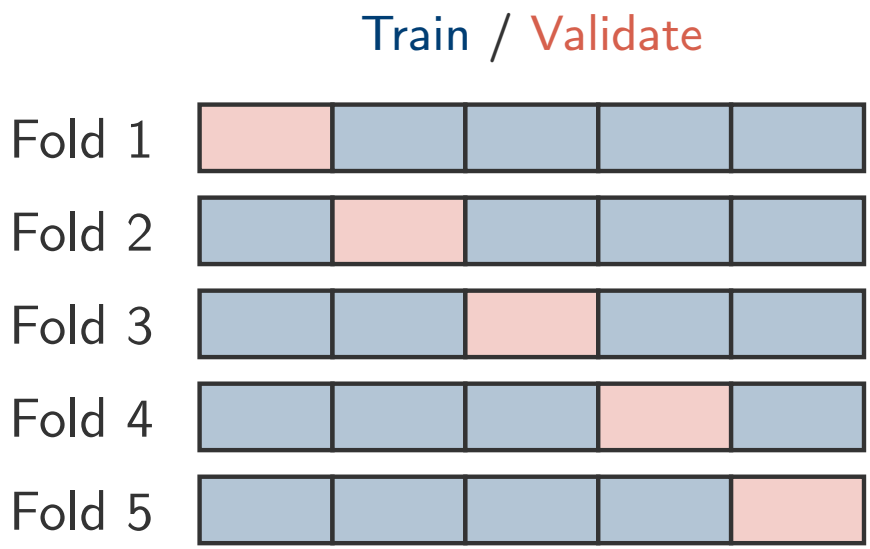
Why three sets?

- Validation set gets "used up" as we try different models
- Test set gives honest final assessment

Cross-Validation: Using Data More Efficiently

Problem: Holding out 40% of data reduces training sample

Solution: **K-Fold Cross-Validation**



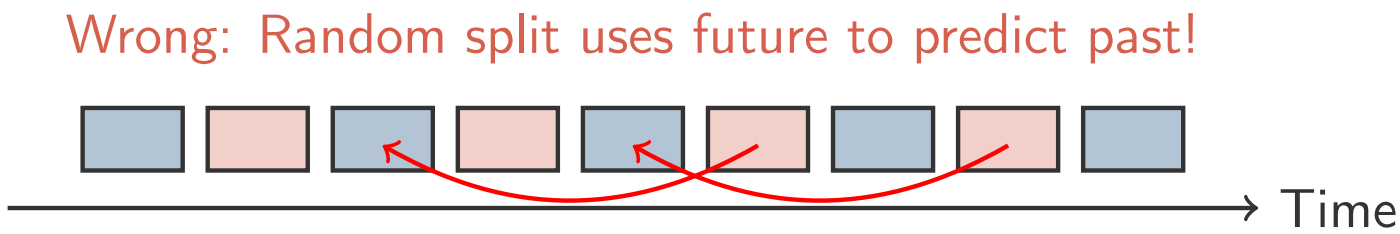
Procedure:

1. Split data into K equal parts ("folds")
2. For each fold: train on other K-1 folds, validate on this fold
3. Average the K validation errors

Common choice: K = 5 or K = 10

Problem: Random CV Doesn't Work for Time Series!

Financial data is ordered in time. Random splits cause problems:

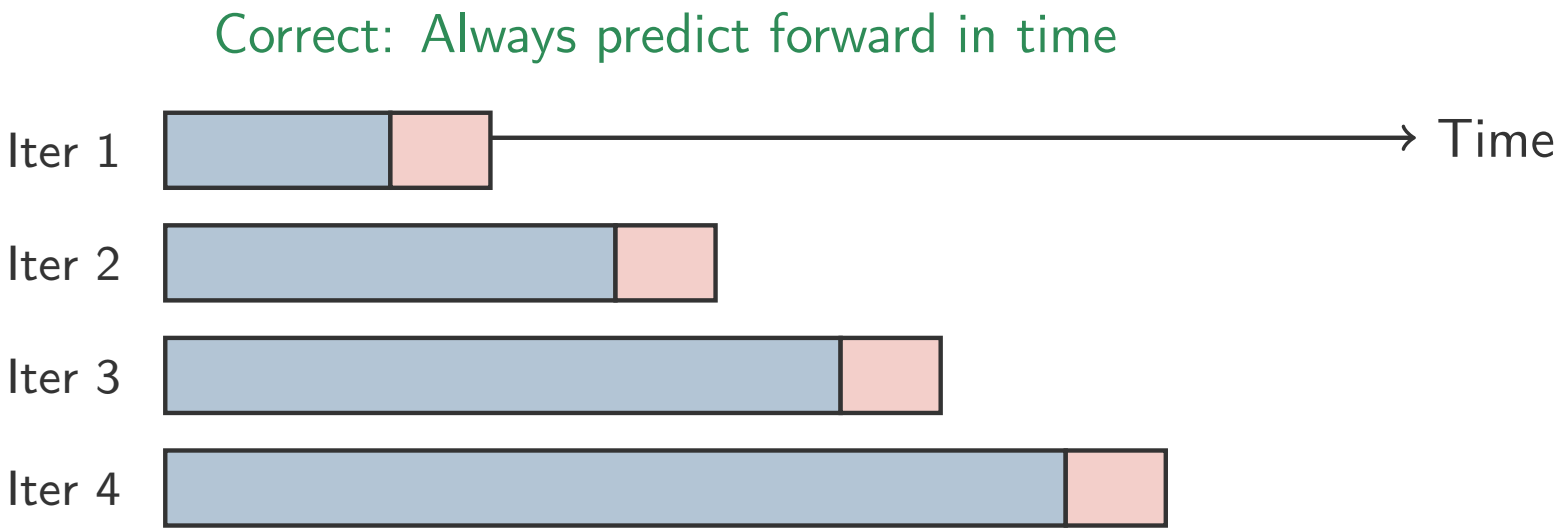


This creates look-ahead bias:

- Training on 2020 data to "predict" 2018 returns
- Model learns from information that would not exist at prediction time
- Overly optimistic performance estimates

Time-Series Cross-Validation

Solution: Only use past data to predict future data (Walk-Forward Validation)



Two variants:

- **Expanding window:** Use all past data (shown above)
- **Rolling window:** Fixed-size training window (discards old data)

The Golden Rule of Out-of-Sample Testing

The Rule

Once you look at test set results, the test set is contaminated.
You get **one shot** at honest out-of-sample evaluation.

Why?

- If you adjust model selection/estimation based on test results, you are implicitly fitting to the test set
- This is exactly the overfitting problem we are trying to avoid
- Performance will be overestimated in a realistic setting

Best practice:

1. Do all model selection on training + validation data
2. Lock in your final model choice
3. Only then evaluate on test set
4. Report the test result, **even if disappointing**

Common Pitfalls in Financial Machine Learning

Why Financial ML Is Especially Hard

Recall from previous slides: Overfitting is deadly in finance.

Three additional pitfalls specific to financial applications:

1. **Look-ahead bias** — Using information you would not have at prediction time
2. **Survivorship bias** — Only analyzing firms/funds that survived
3. **Data snooping** — Testing many strategies, reporting only winners

Each of these makes backtested performance look better than reality.

Key Insight

A strategy can fail for reasons beyond overfitting — even a properly validated model can *"be garbage in, garbage out."*

Look-Ahead Bias

Definition: Using information that would not have been available at the time the prediction was made.

Common sources:

- **Timing of data releases:** Using Q4 earnings announced in February to "predict" January returns
- **Restated financials:** Using corrected accounting data, not original reports
- **Point-in-time vs. revised data:** Economic indicators (e.g., GDP, Inflation, Unemployment) are often revised months later
- **Index membership:** Using current S&P 500 constituents for a 1990 backtest

Example:

- You build a model using book value reported in Compustat
- But Compustat shows the *final* value, not what was known at the first-time release
- Your model "knows" things investors could not have known

Look-Ahead Bias: How to Avoid It

Best practices:

1. **Use point-in-time databases** when available
   - CRSP, Compustat with proper date alignment
   - IBES for analyst forecasts with announcement dates
2. **Apply realistic lags**
   - Quarterly data: available ≈45 days after quarter end
   - Annual data: available ≈90 days after fiscal year end
3. **Be explicit about timing**
   - Document when each feature was knowable/observable
   - Build this into your data pipeline
4. **Use walk-forward validation** (as discussed in previous slides)

Rule of Thumb

When in doubt, add one (or more) lag(s). Being conservative hurts less than look-ahead bias.

Survivorship Bias

Definition: Analyzing only entities that "survived" to the present, ignoring those that failed, delisted, or disappeared.

Why it matters:

- Failed companies often had distinct characteristics before failure
- Excluding them biases your sample toward "winners"
- Relationships estimated on survivors may not hold in the full population, which also includes the "losers"

Classic examples:

- **Mutual funds:** Databases often drop funds that close; remaining funds look better on average
- **Hedge funds:** Voluntary reporting + fund closures = severe upward bias
- **Stocks:** Using only current exchange members misses delistings

Survivorship Bias: How to Avoid It

Best practices:

1. **Use survivorship-bias-free databases**
   - CRSP includes delisted stocks with delisting returns
   - Check if your data source retains "dead" entities
2. **Include delisting returns**
   - When a stock delists, there is often a final return (often negative)
   - Ignoring this overstates strategy performance
3. **Be skeptical of "cleaned" datasets**
   - Sometimes cleaning removes exactly the observations you need
   - Understand what was removed and why
4. **For credit risk:** Ensure you include loans/mortgages that defaulted, not just those that were fully or partly repaid

Data Snooping and Multiple Testing

Definition: Testing many hypotheses/strategies on the same data and selectively reporting the winners.

The problem:

- With 100 random strategies, $\sim$5 will appear significant at the 5% level by chance
- If you only report the winners, they look real
- This happens implicitly through researcher degrees of freedom

Forms of data snooping:

- Trying many feature combinations, keeping what works
- Testing multiple model specifications
- Optimizing hyperparameters on the same data used for evaluation
- "Peeking" at test data to guide model development

Harvey, Liu & Zhu (2016): Many published factors are likely false discoveries due to collective data snooping across the literature.

Data Snooping: How to Mitigate It

No perfect solution, but several defenses:

1. **Adjust for multiple testing**
   - Bonferroni correction: divide significance level by number of tests
   - More sophisticated: Benjamini-Hochberg for false discovery rate
   - **Chordia, Goyal, and Saretto (2020):** you need to adjust for portfolio return correlations.

2. **Pre-registration**
   - Commit to methodology before seeing results
   - Common in medicine, emerging in finance

3. **True out-of-sample testing**
   - Hold out data you genuinely never look at
   - Or use new data that arrives after model development

4. **Economic plausibility**
   - Does the pattern make sense? Is there a "story" to be told?
   - Pure statistical patterns without economic rationale are suspect

Summary: A Checklist for Honest Backtests

Before trusting any ML strategy in finance, verify:

✓ **No look-ahead bias**
- All features use only information available at prediction time

✓ **No survivorship bias**
- Sample includes failed/delisted entities

✓ **Proper validation**
- Time-series CV, not random splits
- Clean separation of train/validation/test

✓ **Multiple testing adjustment**
- Account for all strategies tried, not just the winner

✓ **Economic sensibility**
- Results have plausible interpretation

If a study does not address these, treat results with skepticism.

Python for Machine Learning

Why Python?

Python has become the dominant language for ML in finance:

- Rich ecosystem of ML libraries
- Strong data manipulation tools
- Good integration with databases and APIs
- Large community and documentation
- Used by major quant firms and fintech companies

Alternatives:

- **R:** Strong for statistics, less so for production deployment
- **Julia:** Fast, but smaller ecosystem
- **MATLAB:** Common in academia, expensive, less ML support

This course: All implementation in Python using industry-standard libraries

Core Libraries We Will Use

Library	Purpose	Key Use
`numpy`	Numerical computing	Arrays, linear algebra
`pandas`	Data manipulation	DataFrames, time series
`scikit-learn`	Machine learning	Models, pipelines, CV
`matplotlib`	Visualization	Basic plots
`seaborn`	Visualization	Statistical graphics
`xgboost`	Gradient boosting	Tree ensembles
`lightgbm`	Gradient boosting	Fast tree ensembles
`shap`	Interpretability	Model explanations

Focus: We use `scikit-learn` as our primary ML library.

- Consistent API across all models
- Built-in cross-validation and pipelines
- Good documentation and examples

The Scikit-Learn API Pattern

All scikit-learn models follow the same pattern:

```
from sklearn.linear_model import Ridge

# 1. Create model with hyperparameters
model = Ridge(alpha=1.0)

# 2. Fit to training data
model.fit(X_train, y_train)

# 3. Make predictions
predictions = model.predict(X_test)

# 4. Evaluate
score = model.score(X_test, y_test)
```

This works for any model: `Lasso`, `RandomForestRegressor`, etc.

Once you learn this pattern, switching between models is trivial.

Pipelines: Combining Steps

Real ML workflows have multiple steps: (1) Handle missing values and outliers, (2) Scale features and pre-processing, and (3) Fit model

Pipelines bundle these together:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler()),
    ('model', Ridge(alpha=1.0))
])

pipeline.fit(X_train, y_train)
predictions = pipeline.predict(X_test)
```

Benefits: Prevents data leakage, easy to save/deploy, cleaner code

Time-Series CV in Scikit-Learn

Scikit-learn provides `TimeSeriesSplit`:

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import cross_val_score

# Create time-series splitter
tscv = TimeSeriesSplit(n_splits=5)

# Cross-validate respecting time order
scores = cross_val_score(model, X, y, cv=tscv)

print(f"Mean CV Score: {scores.mean():.4f}")
```

Implementing the walk-forward validation from previous slides becomes trivial in Python.

Note: Data must be sorted by time before using `TimeSeriesSplit`!

A Bird's Eye View on Empirical Applications

Two Running Applications

Throughout this course, we apply ML methods to two problems:

1. **Market Timing**
   - Forecast the aggregate stock market return over time
   - A *time-series* problem: when is the market expected to do well or poorly?
   - Use macroeconomic and valuation predictors to time the market
   - Lectures 2, 3, 4, 7, 9

2. **Credit Risk**
   - Predict loan/mortgage defaults
   - Classification problem
   - Handle class imbalance
   - Build credit scorecards and implement credit assessment and monitoring tools
   - Lectures 5, 6, 8

Why these two?

- Cover regression and classification
- Represent major industry applications
- Rich datasets available

Application 1: Market Timing

The setup:

- Target: next month's aggregate market excess return, $r_{m,t+1}$
- Predictors: macroeconomic and valuation variables known at time t
- A *time-series* forecasting problem

Typical predictors (15–20 from the literature):

- **Valuation ratios:** dividend-price, earnings-price, book-to-market
- **Interest rates:** T-bill rate, term spread, default spread
- **Other:** stock variance, inflation, net equity expansion

Key questions we will address:

- Can we beat the historical mean out-of-sample?
- Is there genuine economic value in timing the market?
- Do ML methods help, or do they just overfit?

Application 1: From Forecasts to Allocations

A return forecast is only useful if it improves decisions.

Typical workflow:

1. Forecast next month's market return $\widehat{r}_{m,t+1}$
2. Translate the forecast into an equity allocation (e.g., shift between stocks and bonds)
3. Rebalance as new information arrives
4. Measure performance against a buy-and-hold benchmark

Performance metrics:

- Out-of-sample R^2 vs. the historical mean (often tiny — even $\sim$0.5% per month is economically meaningful)
- Sharpe ratio of the timing strategy
- Certainty-equivalent / utility gains for a mean-variance investor
- Turnover and transaction costs

Spoiler: Generating genuine out-of-sample economic value (net of transaction costs) is more difficult than you would think.

Application 2: Credit Risk

The setup:

- Data: LendingClub peer-to-peer loans
- Features: Borrower characteristics, loan terms
- Target: Default (yes/no) — a *classification* problem

Types of features:

- **Borrower:** Income, employment length, home ownership
- **Credit history:** FICO score, delinquencies, credit utilization
- **Loan:** Amount, interest rate, purpose, term

Key challenges:

- **Class imbalance:** Defaults are rare ($\sim$10–20%)
- **Probability calibration:** Need accurate default probabilities
- **Interpretability:** Regulators require explanations
- **Fairness:** Cannot discriminate on protected characteristics

Application 2: From Predictions to Decisions

Credit predictions feed into business decisions:

Use cases:

1. **Approve/reject:** Set threshold on predicted default probability
2. **Pricing:** Charge higher interest to riskier borrowers
3. **Portfolio management:** Allocate capital across risk buckets
4. **Regulatory capital:** Calculate required reserves

Performance metrics:

- AUC-ROC (ranking quality)
- Precision/Recall at decision threshold
- Expected loss vs. actual loss
- Calibration: Do 10% predicted defaults actually default 10%?

Reference: Khandani, Kim & Lo (2010) show ML methods significantly outperform traditional credit scorecards.

Summary and Next Steps

Key Takeaways from Lecture 1

1. ML focuses on prediction; we learn $\widehat{f}$ from data
2. Bias-variance tradeoff governs model selection
3. Time-series CV is essential for financial applications
4. Beware of look-ahead bias, survivorship bias, and data snooping
5. Python + scikit-learn provide a consistent ML workflow
6. We will apply methods to market timing and credit risk

The Bottom Line

ML offers powerful tools, but finance demands extra caution. A disciplined and rigorous approach to validation is essential.

Readings and Next Steps

Readings for Today's Lecture:

- James et al. (2023), *An Introduction to Statistical Learning*, Chapters 2 and 5
- Harvey, Liu & Zhu (2016), "...and the Cross-Section of Expected Returns," *Review of Financial Studies*
- Gu, Kelly & Xiu (2020), "Empirical Asset Pricing via Machine Learning," *Review of Financial Studies* (Introduction and Section 2)
- Chordia, Goyal & Saretto (2020), "Anomalies and False Rejections," *Review of Financial Studies*

Week 2 topics:

- OLS limitations in high dimensions
- Ridge regression (L2 penalty)
- Lasso (L1 penalty, sparsity)
- Elastic Net (combining L1 and L2)
- Application: Market Timing and Factor Investing

Questions?