Machine Learning Introduction

BSAD 8310: Business Forecasting — Lecture 7

Department of Economics

University of Nebraska at Omaha

Spring 2026

Why ML for Forecasting?

Classical methods are powerful — but structured. What happens when the structure breaks?

Six lectures have built a powerful toolkit. This lecture examines where its assumptions break down.

What we have (L01–L06):

- Benchmarks and regression (L01–L02)
- ETS for trend + seasonality (L03)
- ARIMA/SARIMA for autocorrelation (L04)
- VAR, ARIMAX, ECM for multivariate dynamics (L05)
- Rigorous walk-forward evaluation (L06)

Classical models assume: linearity, fixed lag structure, Gaussian errors, known seasonality. When these fail, **model misspecification** inflates forecast error beyond what better data could fix.

Socratic: which assumption fails first when forecasting social-media-driven demand spikes for a new product?

Gap	Classical limit	ML remedy
Non-linearity	Linear $f(x)$ assumed	Trees, nets
Many predictors	OLS collapses $(k \gg T)$	Regularization
Interactions	Hand-specified only	Learned
Non-Gaussian	Normality assumed	Distribution-free

(Hastie et al. 2009): the statistical learning perspective on non-linearity and interactions.

ML does **not** replace ARIMA. It extends the toolkit for problems where: (1) predictors number in the hundreds, (2) relationships are non-linear, or (3) structure is unknown *a priori*.

100,000 time series, 61 methods (Makridakis et al. 2020): nine of the top twelve methods were *hybrid* (classical + ML).

Method	Rank	sMAPE
Hybrid ES-RNN (ML + ETS)	1	11.374
Theta (classical)	2	11.551
FFORMA (ML ensemble)	3	11.720
Naive 2 (seasonal)	12	13.56
Auto ARIMA	14	13.58

Retail, electricity, and supply-chain forecasting now routinely use gradient-boosted trees and ensemble combinations of ARIMA + ML.

But: pure ML often *underperforms* naive on short series ($n < 100$). Context matters.

The Bias-Variance Tradeoff

Every modeling decision shifts error between bias and variance. This is the central tension in all of statistical learning.

Let $\hat{f}$ be an estimated model, f the true function, and x_0 a new observation.

$$\mathrm{MSE}(\hat{f}(x_0)) = \left(\mathrm{Bias}[\hat{f}(x_0)]\right)^2 + \mathrm{Var}[\hat{f}(x_0)] + \sigma^2$$

(Hastie et al. 2009)

Three distinct sources of error — each unpacked on the next slide.

Two of the three are **controllable** by the modeler. One is fixed by the data-generating process.

(Each term unpacked on the next slide.)

- **Bias^2** (underfitting): systematic error from wrong model class.
  Example: Bias$^2 \approx 0.42$. Reduce by increasing model complexity.
- **Variance** (overfitting): sensitivity to the particular training sample.
  Example: $\mathrm{Var} \approx 0.31$. Reduce by regularization or more data.
- σ^2 (irreducible): determined solely by the DGP; neither better models nor additional observations can eliminate it.
  Example: $\sigma^2 = 0.07$ — fixed.

Two levers you control:

Complexity $\uparrow$:
Bias$^2 \downarrow$, $\mathrm{Var} \uparrow$

Regularization $\uparrow$:
Bias$^2 \uparrow$, $\mathrm{Var} \downarrow$

σ^2 is fixed by the DGP.

Both extremes hurt.

AR(1) fit to a threshold non-linear series: the model systematically underestimates peaks and overestimates troughs every cycle.

Low variance: refit on any subsample and the errors look the same.

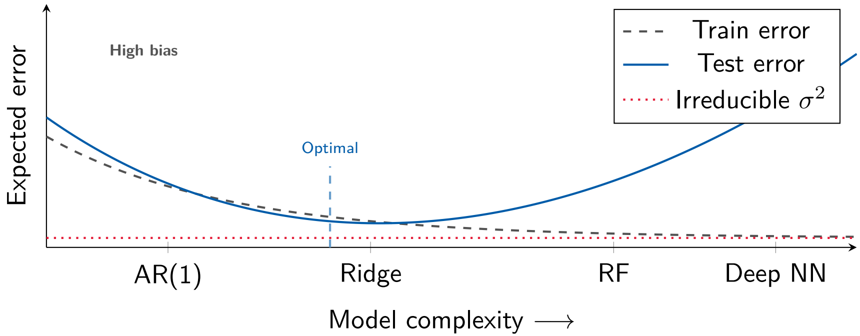
High variance — overfitting:

Degree-15 polynomial fits every wiggle in the training set. Train RMSE ≈ 0.02. Test RMSE $\approx 8\times$ train RMSE.

Small change in training data $\Rightarrow$ large change in $\hat{f}$.

Reducing bias *typically* increases variance. We control the tradeoff via **model complexity** and **regularization**.

Train error always falls with complexity. Test error has a U-shape.



Cross-validation finds the optimal complexity point without ever touching the test set.

Model	**Bias**	**Var.**	**Good when**
AR(1)	High	Low	Long, stable series
ARIMA (auto)	Medium	Low–Med	General TS
Ridge (many X)	Med	Low	Many predictors
OLS (unregul.)	Low	High	$n \gg p$ only
Deep LSTM	Low	High	Very long series

This directly motivates LASSO (L08): deliberately increasing bias via shrinkage to reduce variance on short, noisy series.

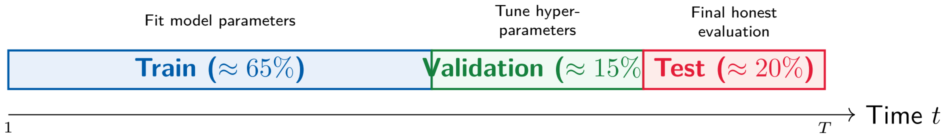
For short series ($n < 200$):

Bias is cheaper than variance.

Prefer parsimonious models. Add complexity only when cross-validation confirms out-of-sample improvement.

Train / Validation / Test Discipline

One split is not enough. Three sets, three roles — and the rules are strict.



Why three sets, not two?

If you tune on the test set — even once — it becomes a second validation set, and you no longer have an honest generalization estimate.

The test set is a **time capsule**. Open it exactly once, at the very end, to report the final number.

Leakage occurs when information from outside the training window contaminates the model, producing optimistic in-sample performance that does not generalize out-of-sample.

Feature leakage:

- Using next-month CPI to forecast this-month sales (future predictor)
- Target-encoding computed on the full dataset before splitting (statistical leakage)

Temporal leakage:

- Random train/test shuffle on a time series
- Fitting `StandardScaler` on the full series before splitting

Random train/test split is *correct* for i.i.d. data. It is **wrong** for time series. Always split in **chronological order**.

In production: leakage makes reported RMSE optimistic by $2\times$–$5\times$.

Socratic: a practitioner fits `StandardScaler` on the full series before splitting. What is the first production sign that something went wrong?

1. **No random shuffling** — always chronological order
2. **Validation after train, test after validation** — strict temporal ordering; no overlaps
3. **No future-dependent features** — any feature using time $t + k$ data cannot be known at time t
4. **Scale within each fold** — fit `StandardScaler` on train-fold only; transform val/test with train parameters
5. **Optional gap** — buffer between train end and val start prevents autocorrelation bleed-through

These rules are the time-series extensions of the bias-variance discipline.

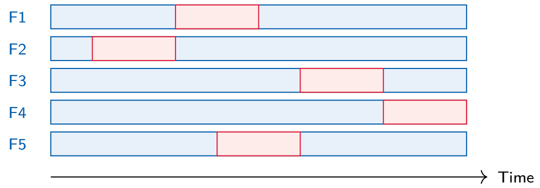
Break them and every accuracy number you report is invalid.

`train_test_split(shuffle=False)` respects ordering. Use `TimeSeriesSplit` for CV.

Cross-Validation for Time Series

Standard k-fold CV violates time ordering. Walk-forward CV is the fix.

Standard 5-fold (shuffled):



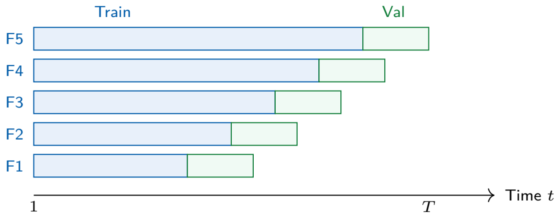
Wrong for time series: validation block may precede some training data $\Rightarrow$ temporal leakage.

Failure modes:

- Uses future data to predict the past
- Train and validation observations are not exchangeable under autocorrelation
- Validation RMSE is optimistically biased

Bergmeir et al. (2018): random k-fold gives biased CV estimates for autoregressive series. The bias grows with autocorrelation strength.

The correct analog: expanding-window validation introduced in L06, now applied to hyperparameter tuning.



```python
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5, gap=0)
for train_idx, val_idx in tscv.split(X):
    X_train, X_val = X[train_idx], X[val_idx]
```

gap: skip observations between train end and val start. Set gap=H when forecasting H steps ahead to prevent overlap.

General CV theory: (Arlot and Celisse 2010).

1. Choose candidate values: $\lambda \in \{0.01, 0.1, 1, 10, 100\}$
2. For each λ: run `TimeSeriesSplit` CV, compute mean validation RMSE
3. Select λ^* minimizing mean CV RMSE
4. **Refit on full train + val** with λ^*
5. Evaluate exactly once on held-out test set

Step 4 is critical: refit on all non-test data. Using the CV train-fold only wastes data.

5-fold `TimeSeriesSplit`:

$\lambda = 0.1 \Rightarrow$ CV RMSE $= 1{,}780$ (optimal)

$\lambda = 0.001$ (overfit): $1{,}890$

$\lambda = 100$ (underfit): $2{,}140$

Test RMSE with $\lambda^* = 0.1$: $1{,}810$ (close to CV RMSE, as expected).

Regularization Preview

When there are many predictors, unconstrained OLS overfits. Shrinkage is the solution. Full treatment: Lecture 8.

$$\hat{\beta} = \arg\min_{\beta} \underbrace{\sum_{t=1}^{T} (y_t - \beta^\top x_t)^2}_{\text{OLS loss}} + \lambda \cdot P(\beta)$$

Ridge: $P(\beta) = \|\beta\|_2^2$ **LASSO (Tibshirani 1996):** $P(\beta) = \|\beta\|_1$ **EN (Zou and Hastie 2005):** $\alpha\|\beta\|_1 + (1-\alpha)\|\beta\|_2^2$ (Hastie et al. 2009)

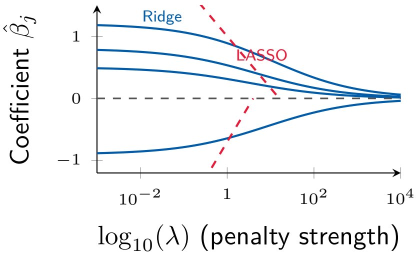
- **Ridge**: shrinks all coefficients smoothly; none reach exactly zero
- **LASSO**: sets some coefficients to exactly zero $\Rightarrow$ automatic variable selection
- **Elastic Net**: handles collinear predictors; ridge + LASSO combined

All three trade bias for variance via λ.

$\lambda = 0$: OLS (maximum variance).
$\lambda \to \infty$: intercept only (maximum bias).

Note: EN α is the LASSO/Ridge mixing ratio (not ETS smoothing α from L03).

Full derivation in L08.



As $\lambda \to \infty$: $\hat{\beta}_j \to 0$ (all predictors removed).

As $\lambda \to 0$: OLS solution.

LASSO paths hit zero at a finite λ — Ridge paths only approach zero asymptotically.

CV selects λ^* at the bias-variance minimum.

Illustrative paths. L08 derives the exact form from the subgradient conditions.

Feature Engineering Preview

ML models do not handle time series natively. We must create temporal structure explicitly.
Full treatment: Lecture 11.

A raw series $y_1, \ldots, y_T$ is not a feature matrix. We extract temporal patterns as columns.

Unlike ARIMA, ML algorithms assume i.i.d. rows: they cannot exploit autocorrelation internally. We must encode temporal structure as explicit columns.

Feature type	**Formula**	**Business meaning**
Lag	$x_t^{(k)} = y_{t-k}$	Sales k periods ago
Rolling mean	$\bar{y}_{t,w} = \frac{1}{w}\sum_{k=1}^{w} y_{t-k}$	Recent trend level
Rolling std	$s_{t,w} = \mathrm{std}(y_{t-w:t-1})$	Recent volatility
Month-of-year	$\mathbf{1}[\mathrm{month}(t) = m]$	Seasonal dummy
Trend counter	$t = 1, 2, \ldots, T$	Linear time trend

All features **must** use only $y_1, \ldots, y_{t-1}$ when predicting y_t. Any feature using y_t or later creates leakage. Apply `.shift(1)` before any rolling window.

Full treatment in L11: calendar features, Fourier terms, interaction features, and pipeline automation.

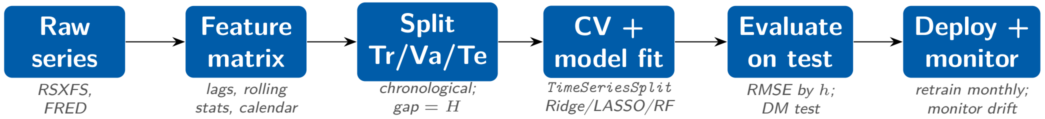
```
# Lag features: shift(1) avoids leakage
X['lag_1']  = y.shift(1)
X['lag_12'] = y.shift(12)

# Rolling mean of y_{t-1}...y_{t-3}
X['roll_3'] = y.shift(1).rolling(3).mean
    ()

# Calendar feature (always known)
X['month'] = y.index.month

# Drop rows with NaN from lags
X = X.dropna()
```

- `.shift(1)`: shifts series forward one period — ensures x_t uses y_{t-1}, not y_t
- `.rolling(3).mean()` after `.shift(1)`: mean of $y_{t-1}, y_{t-2}, y_{t-3}$
- `.index.month`: calendar feature; no leakage (calendar is always known in advance)

L11 wraps this logic in a `sklearn.Pipeline` that prevents leakage automatically at every CV fold.

The ML Forecasting Pipeline

End-to-end: features $\rightarrow$ split $\rightarrow$ fit $\rightarrow$ evaluate $\rightarrow$ deploy.



The `sklearn.Pipeline` object chains preprocessing (`StandardScaler`) + model into a single estimator. This ensures the scaler is fitted on train-fold only during CV — eliminating a common source of leakage (James et al. 2023).

ML **extends** classical forecasting for non-linearity, many predictors, and unknown structure.

$\mathrm{MSE} = \mathrm{Bias}^2 + \mathrm{Var} + \sigma^2$ governs every modeling decision.

Three-way train/val/test split with chronological ordering is non-negotiable.

Walk-forward CV (`TimeSeriesSplit`) replaces k-fold for time series.

Regularization (L08) and feature engineering (L11) are the main levers for controlling bias-variance.

Lecture roadmap:

Lecture	Topic
L08	LASSO, Ridge, EN
L09	RF, XGBoost
L10	LSTM, attention
L11	Feature pipeline
L12	Capstone cases

Lab 7: full pipeline from raw RSXFS to LASSO and Ridge with walk-forward CV.

Arlot, Sylvain and Alain Celisse (2010). “A Survey of Cross-Validation Procedures for Model Selection”. In: *Statistics Surveys* 4, pp. 40–79.

Bergmeir, Christoph, Rob J. Hyndman, and Bonsoo Koo (2018). “A Note on the Validity of Cross-Validation for Evaluating Autoregressive Time Series Prediction”. In: *Computational Statistics & Data Analysis* 120, pp. 70–83.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd. New York: Springer. URL: `https://hastie.su.domains/ElemStatLearn/`.

James, Gareth et al. (2023). *An Introduction to Statistical Learning with Applications in Python*. 1st. New York: Springer. URL: `https://www.statlearning.com/`.

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos (2020). “The M4 Competition: 100,000 Time Series and 61 Forecasting Methods”. In: *International Journal of Forecasting* 36.1, pp. 54–74.

- Tibshirani, Robert (1996). "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society: Series B* 58.1, pp. 267–288.
- Zou, Hui and Trevor Hastie (2005). "Regularization and Variable Selection via the Elastic Net". In: *Journal of the Royal Statistical Society: Series B* 67.2, pp. 301–320.