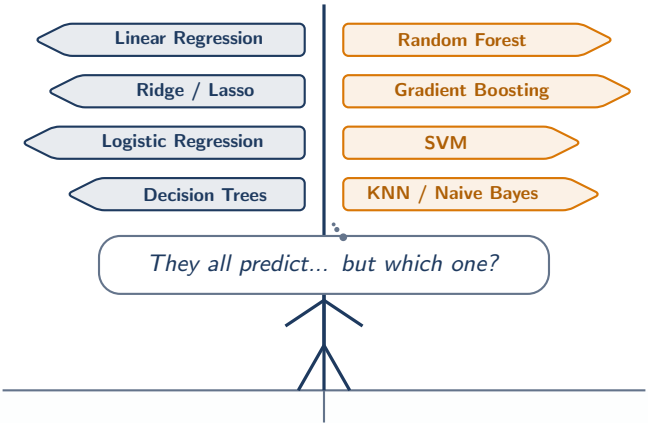
Supervised Learning: From Linear Regression to Gradient Boosting

Data Science with Python – BSc Course (Supplementary Lecture)

Data Science Program

BSc Course

90 Minutes



The answer depends on your data, not your preference.

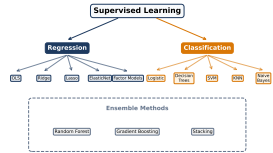
What Does the Supervised Learning Landscape Look Like?

Supervised learning = learn a function $f : X \to y$ from labeled examples.

Two branches:

- **Regression** – predict a number (e.g., stock return)
- **Classification** – predict a category (e.g., default / no-default)

Ensemble methods (strategies that combine base learners) cross both branches.



Read the chart:

- The tree splits supervised learning into regression (blue) and classification (amber)
- Ensemble methods (dashed border) combine base learners – a strategy, not a single algorithm
- Simplest methods on the left; complexity increases rightward

---

Every method on this map gets at least one slide. Start simple, add complexity.

Why Does Supervised Learning Dominate Industry ML?

88% of organisations report regular AI use in at least one business function (McKinsey State of AI 2024).

Where supervised learning runs in production:

1. **Credit scoring** – predict probability of default for each applicant
2. **Fraud detection** – flag suspicious transactions in real time
3. **Churn prediction** – identify customers likely to leave
4. **Algorithmic trading** – predict next-period returns from features

The most deployed models are **logistic regression** and **gradient boosting** (tree-based ensemble). Supervised learning remains the plurality paradigm for structured, tabular production workloads.

Source: itransition.com/McKinsey State of AI 2024. If you master supervised learning, you can solve most real-world ML problems.

What Will You Be Able to Do After This Lecture?

By the end of this 90-minute session, you will be able to:

1. **Compare** regression and classification methods on the bias-variance spectrum
2. **Select** an appropriate algorithm given data characteristics (size, features, task type)
3. **Evaluate** model performance using correct metrics (MSE, R-squared, AUC, F1)
4. **Explain** model predictions using SHAP (SHapley Additive exPlanations) values
5. **Avoid** common pitfalls: data leakage (test information bleeding into training), snooping bias (reporting only the best result), and wrong cross-validation

Narrative thread: the **bias-variance tradeoff** connects every method. Simple models have high bias; complex models have high variance. Every technique in this lecture is a strategy for navigating that tradeoff.

These five verbs map to Bloom's taxonomy levels: understand, apply, analyse, evaluate, create.

How Much Will This Stock Return Next Month?

Situation: You have 5 years of daily returns for 200 stocks and dozens of features (P/E ratio, momentum, volatility, sector).

Complication: You need to predict next-month returns to rank stocks for a portfolio. Eyeballing 200 scatterplots is not a strategy.

Question: Can we build a function $\hat{y} = f(X)$ that predicts returns from features?

That function is a **supervised learning model**. The features are X (input variables), the return is y (target variable), and "supervised" means we train on historical data where we know both X and y.

Regression predicts a continuous number. **Classification** (later) predicts a category.

This problem – predicting stock returns from features – is the central task of quantitative investing.

Why Is Linear Regression the First Model You Should Try?

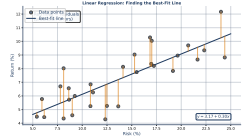
Model: $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$

OLS (Ordinary Least Squares) minimises $\sum(y_i - \hat{y}_i)^2$.

Closed-form solution: $\hat{\beta} = (X^\top X)^{-1} X^\top y$

Example: Predict monthly return from P/E ratio.
$\hat{y} = 0.012 - 0.0003 \times \text{P/E}$.
A stock with P/E = 20 gets
$\hat{y} = 0.012 - 0.006 = 0.006$ (0.6%/month).

As you learned in L21: OLS is 200 years old (Gauss, 1809) and still the starting point.



Read the chart:

- Horizontal axis = feature, vertical axis = target
- The line minimises squared vertical distances from points to line
- Points far from the line are poorly predicted – high residuals (prediction errors)

Linear regression is the foundation – learn it cold before adding complexity.

What Happens When You Add More Features?

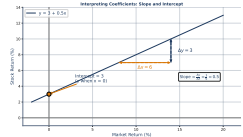
Adding features (P/E, momentum, volatility, sector) usually improves fit on **training data**.

But more features = more parameters β_j (regression coefficients) = more risk of **overfitting** (memorising noise instead of learning patterns).

Rule of thumb: need 10–20 observations per feature to avoid instability.

Coefficient interpretation: β_j = change in $\hat{y}$ for a one-unit change in x_j, holding other features constant.

Caution: scale matters – always standardise features before comparing coefficient magnitudes.



Read the chart:

- Each bar shows the magnitude and sign of one β_j
- Positive coefficients increase the prediction; negative decrease it
- The largest bars are the most influential features

More features is not always better. The next slide shows what happens when you add too many.

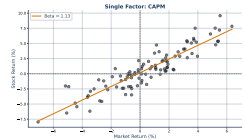
Can Regression Explain Why Stocks Move Together?

CAPM (Capital Asset Pricing Model):
$R_i - R_f = \alpha + \beta_{\text{mkt}}(R_m - R_f) + \epsilon$

- β_{mkt} (market beta) = market sensitivity – how much the stock moves with the market
- α (Jensen's alpha) = excess return not explained by the market (skill? mispricing?)
- ϵ (residual) = unexplained noise

Example: Apple has $\beta_{\text{mkt}} = 1.2$ – it amplifies market moves by 20%.

Fama-French adds size (SMB) and value (HML) factors: regression with 3 features instead of 1. As covered in L24.



Read the chart:

- Horizontal = market excess return, vertical = stock excess return
- Slope of the fitted line is β_{mkt} – steeper = more sensitive
- The intercept is the average α

CAPM is a regression with one feature. Fama-French is a regression with three. Factor models are supervised learning in disguise.

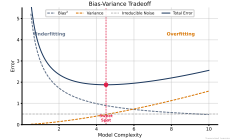
What Is the Central Dilemma of Machine Learning?

Bias = error from wrong assumptions (model too simple, misses patterns).
Variance = error from sensitivity to training data (model too complex, memorises noise).

Total error = Bias^2 + Variance + irreducible noise

- Simple models (OLS): high bias, low variance
- Complex models (deep trees): low bias, high variance
- **The sweet spot is in the middle**

Regularisation, ensembles, and cross-validation help you find that sweet spot.



Read the chart:

- Horizontal = model complexity, vertical = error
- Blue dashed = bias (decreases), amber dashed = variance (increases)
- Solid navy = total error with a minimum – the optimal complexity

---

Every method choice in this lecture is a bet on the bias-variance tradeoff.

How Does Ridge Regression Tame Overfitting?

OLS problem: large coefficients β_j, unstable predictions.

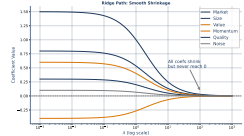
Ridge adds an L_2 penalty (sum of squared coefficients): Minimise $\sum(y_i - \hat{y}_i)^2 + \lambda \sum \beta_j^2$

λ (regularisation strength) controls the penalty:

- $\lambda = 0$: plain OLS
- $\lambda \to \infty$: all $\beta_j \to 0$

Coefficients **shrink toward zero but never reach exactly zero**.

On the bias-variance curve: Ridge moves you left – adds a bit of bias, removes a lot of variance. As covered in L22.



Read the chart:

- Horizontal = λ (log scale), vertical = coefficient value
- As λ increases, all coefficients shrink toward zero
- No coefficient hits exactly zero – Ridge keeps all features

Ridge is the go-to when you have many correlated features and no reason to exclude any.

How Does Lasso Perform Automatic Feature Selection?

Lasso uses an L_1 penalty (sum of absolute values):
Minimise $\sum(y_i - \hat{y}_i)^2 + \lambda \sum |\beta_j|$

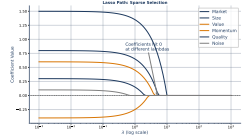
The key difference from Ridge: Lasso sets some coefficients **exactly to zero**.

This is **automatic feature selection** – the model decides which features matter.

Example: 200 stock features, Lasso keeps only 12. The remaining 188 have $\beta_j = 0$.

When to use Lasso: when you suspect many features are irrelevant.

As explored in L22: Lasso answers "which features matter?" Ridge answers "how much should each feature contribute?"



Read the chart:

- Same axes as Ridge – λ vs coefficient value
- As λ increases, coefficients hit exactly zero one by one
- Only a sparse set of features survive at high λ

Lasso adds bias by zeroing out features, but the variance reduction from a simpler model is often worth it.

What If You Need Both Shrinkage and Sparsity?

ElasticNet combines both penalties:
$\lambda_1 \sum |\beta_j| + \lambda_2 \sum \beta_j^2$

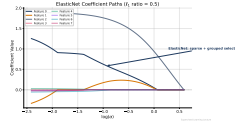
Mixing parameter $\alpha \in [0, 1]$:

- $\alpha = 1$: pure Lasso (L_1 only)
- $\alpha = 0$: pure Ridge (L_2 only)

Note: this α is the ElasticNet mixing parameter, not the CAPM intercept from Slide 9.

Advantage: when features are correlated (e.g., P/E and P/B ratios), Lasso arbitrarily picks one; ElasticNet keeps both.

sklearn: `ElasticNet(alpha=0.1, l1_ratio=0.5)`
`l1_ratio` is the mixing parameter.



Read the chart:

- Coefficient paths under ElasticNet as regularisation increases
- Some coefficients reach zero (like Lasso) while correlated groups stay together (like Ridge)
- The mixing parameter controls the balance

In practice, ElasticNet is the safest default for regularised regression – it never does worse than either penalty alone.

How Do You Know If a Regression Model Is Any Good?

MSE (Mean Squared Error) $= \frac{1}{n}\sum(y_i - \hat{y}_i)^2$
Penalises large errors heavily.

RMSE (Root MSE) $= \sqrt{\text{MSE}}$ – same units as y.

MAE (Mean Absolute Error) $= \frac{1}{n}\sum|y_i - \hat{y}_i|$
Robust to outliers.

$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$ – proportion of variance explained.

Example: Predicting monthly returns. RMSE $= 0.03$ means typical error is 3 percentage points. $R^2 = 0.05$ means 5% of variance explained – *typical for return prediction!* As covered in L23.

Read the chart:

- Blue dots = actual values; the line = model prediction
- Vertical segments show residuals (prediction errors)
- Squaring these segments and averaging gives MSE

In finance, $R^2 = 0.05$ is actually good. In engineering, $R^2 < 0.90$ is often useless. Context matters.

How Do You Know If Your Model Will Work on New Data?

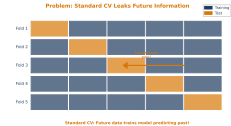
Train/test split: hold out 20%, train on 80%, evaluate on held-out data.

K-fold CV (cross-validation): rotate the test fold 5 or 10 times, average scores.

Warning for finance: standard K-fold **shuffles** data, mixing past and future. Use `TimeSeriesSplit` instead.

Walk-forward validation: retrain as new data arrives, matching deployment reality.

Lopez de Prado (2018) introduced **Purged K-Fold CV**: removes training observations whose labels overlap with test labels, then adds an embargo window. As covered in L23.



Read the chart:

- Training folds (blue) and validation fold (amber) rotate through the data
- Each fold takes a turn as the test set
- For time series: training must always precede validation

For time series data, always validate forward in time. Shuffled CV pretends you can see the future.

How Do You Decide: Approve or Reject a Loan?

Situation: A bank has 100,000 loan applications. Each has features (income, employment length, credit score, debt-to-income ratio).

Complication: A wrong approval costs $50,000 (default loss). A wrong rejection costs a good customer and future revenue.

Question: Can we predict the **probability of default** for each applicant?

This is a **classification problem**: the target $y \in \{0, 1\}$ is categorical (default / no-default), not continuous.

The ML framework (features, training, validation) is **identical** to regression – only the output changes from a number to a category.

Classification replaces "how much?" with "which one?" – the most common task in production ML.

How Do You Turn a Line into a Probability?

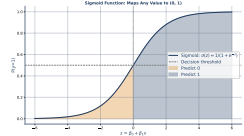
Model: $p = \sigma(\beta^\top x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\ldots)}}$

p (predicted probability) $\in (0, 1)$, not a class label.

Decision rule: predict class 1 if $p > 0.5$ (threshold is adjustable).

Example: Credit scoring. $p = 0.82$ means 82% estimated probability of default. The bank rejects.

XGBoost achieves AUC (Area Under the ROC Curve) 0.89–0.913 vs logistic AUC 0.76 in head-to-head credit scoring benchmarks – but banks still deploy logistic regression because regulators can inspect every coefficient. As you will explore in L25.



Read the chart:

- Horizontal = linear predictor $\beta^\top x$, vertical = probability
- The S-shaped sigmoid squeezes any score into $(0, 1)$
- Near 0, probability changes rapidly (the decision zone)

Logistic regression is linear regression with a sigmoid wrapper. If you understand OLS, you understand 80% of logistic regression.

Where Does the Model Draw the Line Between Classes?

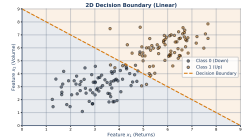
A **decision boundary** is the surface where $p = 0.5$ – the model is maximally uncertain.

Logistic regression: boundary is always a straight line (or hyperplane in higher dimensions).

This limits what logistic regression can learn – it **cannot capture curved or complex boundaries**.

More complex models (trees, SVM, KNN) can draw **nonlinear boundaries**.

Key insight: the shape of the decision boundary determines what the model can learn. Linear = simple but limited. Nonlinear = flexible but harder to interpret.



Read the chart:

- Two features on the axes; coloured points = two classes
- The straight line is the logistic regression boundary
- Points on the wrong side are misclassified

Logistic regression draws a straight line. Trees, forests, and SVMs can draw curves.

What If Your Model Could Explain Itself in Plain English?

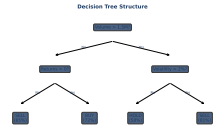
A **decision tree** splits data recursively: "Is income > $50K? If yes, go left. Is credit score > 700? If yes, approve."

Each split maximises **information gain** (or minimises impurity).

Result: a tree of if-then rules any human can read.

Finance use: fraud detection, credit approval – regulators can inspect the logic. Regulators issued 80 AML (Anti-Money Laundering) fines worth $263 million in H1 2024 (31% jump YoY), increasing pressure for explainable decision systems.

Problem: deep trees overfit badly. As covered in L26.



Read the chart:

- Each node shows a feature and threshold
- Follow branches from root to leaf for a prediction
- Leaf nodes show the predicted class

A decision tree is the only model a loan officer can explain in plain English to a customer.

How Do Trees Choose the Best Split?

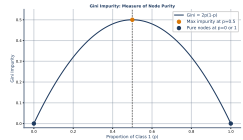
Gini impurity (probability of misclassification):
$G = 1 - \sum_k p_k^2$

Entropy (Shannon information content):
$H = -\sum_k p_k \log_2 p_k$

Both measure "purity" of a node – pure nodes (all one class) have $G = 0$, $H = 0$.

Example: A node with 70% default, 30% non-default:
$G = 1 - 0.7^2 - 0.3^2 = 1 - 0.49 - 0.09 = 0.42$

In practice, Gini and Entropy give nearly identical trees. sklearn uses Gini by default. As covered in L26.



Read the chart:

- Horizontal = proportion of class 1 (p), vertical = impurity
- Both peak at $p = 0.5$ (maximum uncertainty)
- The tree picks the split that reduces impurity the most

Focus on the concept: trees greedily pick the split that makes child nodes purest. The formula choice rarely matters.

What Happens When Your Model Memorises Instead of Learns?

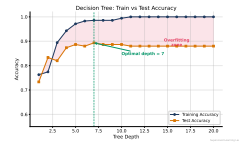
- A depth-1 tree (stump): too simple, high bias, low variance
- A depth-20 tree: memorises training data, training accuracy = 100%, test accuracy drops

The **overfitting gap** between train and test accuracy widens with depth.

Fix options:

1. Limit `max_depth`
2. Require `min_samples_leaf`
3. Prune after growing

Better fix: use an **ensemble** of shallow trees – random forests or gradient boosting.



Read the chart:

- Horizontal = tree depth, vertical = accuracy
- Training accuracy (navy) rises and plateaus near 100%
- Test accuracy (amber) peaks around depth 5–8, then drops

The gap between training and test accuracy is the overfitting gap. The optimal depth is where test accuracy peaks.

Why Is a Forest Smarter Than a Single Tree?

Grow B trees (e.g., $B = 100$) on bootstrap samples (**bagging**); each tree uses a random subset of features.

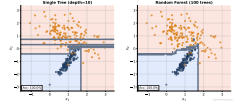
Average predictions (regression) or majority vote (classification).

Individual trees: high variance, low bias. Averaging **reduces variance** without increasing bias.

Key hyperparameters:

- `n_estimators`: number of trees
- `max_features`: features per split (`'sqrt'` is default)

sklearn:
`RandomForestClassifier(n_estimators=100)`



Read the chart:

- Left region: single tree boundary (jagged, overfits)
- Right region: 100-tree forest boundary (smoother)
- The forest averages away noise from individual trees

A forest is a committee of mediocre trees that together make excellent decisions.

How Does Boosting Learn from Its Own Mistakes?

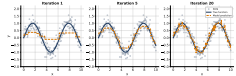
Build trees one at a time; each tree corrects the errors of the ensemble so far.

Tree t fits the **residuals** (mistakes) left by trees $1, \ldots, t-1$.

Learning rate η (step size) controls the contribution of each tree: slow learning = better generalisation.

Bagging (forests) reduces variance. Boosting reduces bias. Both are ensembles, different strategies.

Result: gradient boosting often achieves lower total error than random forests on tabular data.



Read the chart:

- Three stages: iteration 1 (weak fit), 5 (better), 20 (accurate)
- The model converges toward the true function
- Each tree fixes a specific error left by the previous ensemble

Boosting is greedy: it chases the biggest remaining errors. Powerful, but risks overfitting if you boost too long.

What Makes Gradient Boosting the Default for Tabular Data?

XGBoost = gradient boosting + regularisation + engineering optimisations.

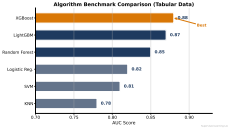
Key innovations:

1. L_1/L_2 regularisation on leaf weights
2. Sparsity-aware split finding
3. Parallel tree construction
4. Cache-optimised block structure

17 of 29 winning Kaggle solutions used XGBoost at publication time (Chen & Guestrin, KDD 2016).

Alternatives: **LightGBM** (faster), **CatBoost** (handles categoricals natively).

Grinsztajn et al. (2022, NeurIPS): tree-based models outperform deep learning on 45 tabular datasets.



Read the chart:

- Algorithms ranked by average AUC on benchmark datasets
- XGBoost/LGBM consistently rank near the top
- Logistic regression and KNN trail behind

Source: Grinsztajn et al. (2022), Chen & Guestrin (2016). For tabular data, gradient boosting is the default until proven otherwise.

What Is the Widest Street You Can Draw Between Two Crowds?

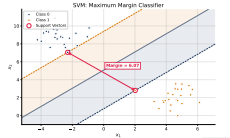
SVM (Support Vector Machine) finds the decision boundary that **maximises the margin** (the gap between classes).

Support vectors = the few data points closest to the boundary (the "hardest" examples). Only these determine the boundary.

Regularisation parameter C controls the tradeoff:

- Large C = narrow margin (low bias, high variance)
- Small C = wide margin (high bias, low variance)

Finance use: credit classification, sentiment analysis on small text datasets. SVM resists overfitting on small, high-dimensional data.



Read the chart:

- Two classes separated by a line with shaded margin
- Circled points are support vectors
- SVM picks the line that makes the margin widest

SVM's popularity peaked in the 2000s. Today it is niche but still useful for small, high-dimensional datasets.

Can You Separate Circles with a Straight Line?

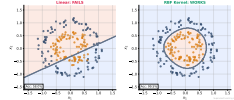
Problem: if classes are not linearly separable, a straight line fails.

Solution: map data to a higher-dimensional space where a linear boundary works.

The **kernel trick** does this implicitly – you never compute the high-dimensional coordinates directly.

Common kernels: RBF (Radial Basis Function / Gaussian), polynomial, sigmoid.

Example: Two concentric circles. Linear SVM fails. RBF kernel separates them easily by implicitly projecting into a higher dimension.



Read the chart:

- Left region: linear SVM fails on circular data
- Right region: RBF kernel SVM draws a curved boundary
- The kernel trick lets SVM handle nonlinear data

The kernel trick is mathematically elegant but adds a hyperparameter (gamma). Too large = overfitting; too small = underfitting.

Can You Predict Without Building a Model at All?

K-Nearest Neighbors (KNN): to predict for a new point, find the K closest training points, take their majority vote (classification) or average (regression).

No training step – all computation happens at prediction time ("lazy learner").

K controls the bias-variance tradeoff:

- $K = 1$: high variance (memorises)
- $K = n$: high bias (predicts majority class)

Limitation: $O(n \cdot d)$ per prediction – stores entire training set, prohibitive at scale. Curse of dimensionality causes distance metrics to lose discriminative power in high-dimensional spaces.

Read the chart:

- Three panels: K=1 (jagged), K=5 (balanced), K=15 (smooth)
- K=1 overfits; K=15 may underfit
- K=5 is the sweet spot – found via cross-validation

KNN: "tell me who your neighbors are, and I'll tell you what you are." Simple, but does not scale.

Why Does a "Wrong" Assumption Often Predict Well?

Bayes' rule: $P(y|X) \propto P(X|y)\,P(y)$

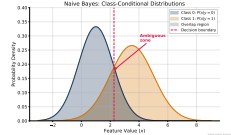
"Naive" assumption: features are **conditionally independent** given the class.

This is almost always wrong – but the model often works anyway!

Strengths: extremely fast, works with tiny datasets, excellent for text classification.

Finance use: news sentiment classification, spam filtering, real-time alert categorisation.

NB achieves **90.5%** accuracy on text classification; transformers reach 94.5% but at **50x inference cost** (IJC 2024). Hybrid NB+DL achieves 95% of BERT accuracy at 1/50th the cost.



Read the chart:

- Two overlapping distributions: $P(\text{feature}|\text{default})$ and $P(\text{feature}|\text{no default})$
- The classifier assigns the class with higher probability
- Where distributions overlap is the "ambiguous zone"

Naive Bayes assumes independence, which is wrong – but it often predicts well anyway. Speed and simplicity over accuracy.

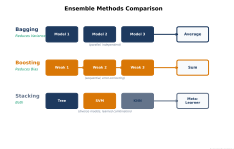
How Do Bagging, Boosting, and Stacking Compare?

Bagging (e.g., Random Forest): train models independently on bootstrap samples, then average.
⇒ **Reduces variance.**

Boosting (e.g., XGBoost): train models sequentially, each correcting previous errors.
⇒ **Reduces bias.**

Stacking: train diverse models, then train a meta-model on their predictions.
⇒ Can reduce both, but adds complexity.

Key principle: all three work because diverse errors cancel out. Ensemble diversity is the central driver of generalisation improvement (Fan et al., 2025).



Read the chart:

- Three rows: bagging (parallel), boosting (sequential), stacking (two-level)
- Arrows show data flow differences
- Labels show which error component each reduces

When in doubt: gradient boosting for tabular data, random forest for a quick baseline.

Why Is Accuracy Not Enough for Classification?

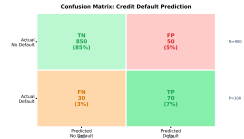
Confusion matrix: TP (True Positive), TN (True Negative), FP (False Positive), FN (False Negative).

Precision $= \frac{\text{TP}}{\text{TP+FP}}$ – "of predicted positives, how many are correct?"

Recall $= \frac{\text{TP}}{\text{TP+FN}}$ – "of actual positives, how many are caught?"

F1 $= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision+Recall}}$ – harmonic mean.

AUC-ROC (Area Under the Receiver Operating Characteristic curve) = threshold-independent measure. As covered in L27.



Read the chart:

- A 2×2 grid: TP, FP, FN, TN in four quadrants
- Precision reads across the predicted-positive row
- Accuracy = (TP+TN)/total, but misleading when classes are imbalanced

---

In fraud detection, you want high recall (catch all fraud) even at the cost of some false alarms. The choice is a business decision.

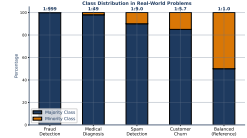
When Is 99% Accuracy Actually Terrible?

If 1% of transactions are fraud, a model that predicts "not fraud" for everything gets **99% accuracy** – and catches zero fraud.

Solutions:

1. `class_weight='balanced'` in sklearn
2. **SMOTE** (Synthetic Minority Oversampling Technique): creates synthetic minority samples by interpolating between existing ones
3. Threshold tuning: lower the decision threshold from 0.5 to catch more positives

Finance example: credit default prediction. 3% default rate. Without rebalancing, the model ignores defaults entirely. As covered in L28.



Read the chart:

- Two bars: majority class dominates (97%), minority is tiny (3%)
- A naive model predicts majority class for everything
- After SMOTE or weighting, the model detects the minority class

A 99%-accuracy model that misses all fraud is worse than a 95%-accuracy model that catches 80% of fraud.

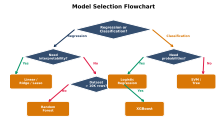
Which Algorithm Should You Use for Your Dataset?

Start with the simplest model that could work:

1. Logistic regression (classification) or linear regression (regression)
2. Check if it beats a "dumb" baseline (majority class, mean prediction)
3. If yes: try regularisation (Ridge/Lasso). If no: try random forest.
4. Still not good enough? Try gradient boosting (XGBoost/LGBM).

L_2-regularised logistic regression matched more complex models on **55% of small-sample datasets** (AMLB 2022).

SVM: only if small and high-dimensional. **KNN:** only for small data. **NB:** only for text or when speed is critical.



Read the chart:

- A decision flowchart: diamonds = questions, rectangles = recommendations
- Start at "What is your task?" – branch to regression or classification
- Follow arrows based on data size, feature count, and interpretability needs

Start simple, validate, iterate. The best model is the one that generalises, not the one that sounds fanciest.

How Do You Explain a Model's Decision to a Regulator?

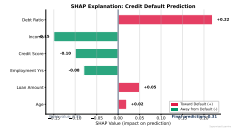
Feature importance (trees): how much each feature reduces impurity across all splits.

SHAP values (SHapley Additive exPlanations, any model): each feature's contribution to a *specific* prediction, based on cooperative game theory.

Example: Credit denial. SHAP says:

- Debt ratio pushed p up by $+0.22$
- Income pushed p down by -0.15
- Age had no effect ($\phi_j \approx 0$, where ϕ_j is the SHAP value for feature j)

EU AI Act: penalties up to **6% of global annual revenue** for unexplainable AI in high-risk applications.



Read the chart:

- Waterfall: each bar shows one feature's SHAP contribution
- Red bars push toward default; blue bars push away
- Base value + all SHAP values = final prediction

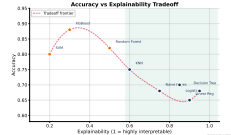
SHAP answers "why did the model predict THIS for THIS customer?" – not just "which features are generally important."

Is There Always a Tradeoff Between Accuracy and Explainability?

- **Linear/logistic:** highly interpretable, moderate accuracy
- **Decision trees:** interpretable, limited accuracy
- **Random forests:** accurate, 100 trees = 100 explanations
- **XGBoost:** most accurate on tabular data (AUC 0.89–0.913 for credit scoring), least interpretable natively

SHAP/LIME partially bridge the gap but add complexity.

Finance reality: regulated industries (banking, insurance) often require interpretable models even if they are less accurate. The "best" model is the most accurate model *you can explain*.



Read the chart:

- Scatter: horizontal = explainability, vertical = accuracy
- Logistic regression: very explainable, moderate accuracy
- XGBoost: high accuracy, low native explainability

In regulated industries, the "best" model is not the most accurate – it is the most accurate model you can explain.

What Does a Complete Supervised Learning Pipeline Look Like?

1. **Data** – collect features and labels
2. **Preprocess** – impute, scale, encode (inside a pipeline!)
3. **Feature eng.** – rolling averages, financial ratios
4. **Model** – start simple, validate, try ensembles
5. **Tune** – `GridSearchCV` or `RandomizedSearchCV`
6. **Evaluate** – test set or walk-forward (never training data)
7. **Deploy** – `joblib`, monitor for concept drift

Prevent **data leakage** at every step. As covered in L32.

Read the chart:

- Horizontal flow: Data → Preprocess → Model → Evaluate → Deploy
- Feedback arrow from Evaluate back to Model (iterate)
- Warning icon between steps: never fit on test data

The pipeline is more important than the algorithm. A good pipeline with logistic regression often beats a bad pipeline with XGBoost.

Do Neural Networks Beat Trees on Tabular Data?

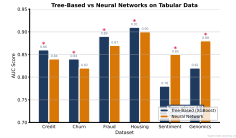
Grinsztajn et al. (2022, NeurIPS): tree-based models (RF, XGBoost) outperform deep learning on 45 medium-sized tabular datasets, even after hyperparameter tuning.

Three reasons trees win:

1. DL not robust to uninformative features
2. DL does not preserve data orientation
3. Tabular data has irregular decision boundaries that trees handle naturally via axis-aligned splits

DL excels on images, text, audio, sequences – data with spatial/temporal structure.

AutoML can deliver 90–95% of expert-tuned performance at roughly 10% of the effort (Nature Sci. Reports, 2025).



Read the chart:

- Trees (navy) vs neural networks (amber) across benchmarks
- Trees match or beat neural nets on most tabular datasets
- Neural nets win on a few high-volume datasets (asterisked)

Source: Grinsztajn et al. (2022). For tabular data, start with XGBoost. Reserve neural networks for images, text, and sequences.

Why Do Backtesting Stars Fail in Live Trading?

Data leakage: scaler fit on the full dataset including test set. *Fix:* put scaling inside a Pipeline.

Snooping bias: testing many models and reporting only the best score. *Fix:* holdout or nested CV.

Lookahead bias: using future information to predict the past (shuffled CV on time series). *Fix:* `TimeSeriesSplit` with an embargo window (Lopez de Prado, 2018).

Target leakage: a feature caused by the target (e.g., "loan_status" when predicting default). *Fix:* domain knowledge review.

92% accuracy in notebook, 61% in production = data leakage guaranteed.

Read the chart:

- Two pipelines: "Leaky" (red) scales before splitting; "Clean" (green) scales inside CV
- Leaky pipeline: inflated 92% accuracy
- Clean pipeline: honest 68% accuracy

If your model seems too good to be true, it probably is. Check for leakage before celebrating.

Is Supervised Learning Always the Right Answer?

No. Five scenarios where supervised learning is not the right tool:

1. **No labels available** – Use unsupervised learning (clustering, PCA) or self-supervised learning
2. **Labels are expensive to obtain** – Consider active learning (query the most informative examples) or semi-supervised approaches
3. **The relationship is non-stationary** – Financial regimes change; a model trained on bull markets fails in bear markets. Consider regime-switching models or frequent retraining.
4. **The cost of a wrong prediction is catastrophic** – Consider rule-based systems with human oversight rather than fully automated ML
5. **The data is too small** – With fewer than 50 observations, most ML models will overfit. Use domain expertise instead.

Knowing when NOT to use a tool is as important as knowing how to use it.

Regression Formula Reference

Models

OLS:
$\hat{\beta} = (X^\top X)^{-1} X^\top y$

Ridge (L_2):
$\min \sum (y_i - \hat{y}_i)^2 + \lambda \sum \beta_j^2$

Lasso (L_1):
$\min \sum (y_i - \hat{y}_i)^2 + \lambda \sum |\beta_j|$

ElasticNet:
$\min \sum (y_i - \hat{y}_i)^2 + \lambda_1 \sum |\beta_j| + \lambda_2 \sum \beta_j^2$

CAPM:
$R_i - R_f = \alpha + \beta_{\text{mkt}}(R_m - R_f) + \epsilon$

Metrics

$\text{MSE} = \frac{1}{n} \sum (y_i - \hat{y}_i)^2$

$\text{RMSE} = \sqrt{\text{MSE}}$

$\text{MAE} = \frac{1}{n} \sum |y_i - \hat{y}_i|$

$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$

where $\text{SS}_{\text{res}} = \sum (y_i - \hat{y}_i)^2$,
$\text{SS}_{\text{tot}} = \sum (y_i - \bar{y})^2$

Key Concepts

Bias-Variance:
$\text{Error} = \text{Bias}^2 + \text{Variance} + \sigma_\epsilon^2$

K-Fold CV:
$\text{CV}_K = \frac{1}{K} \sum_{k=1}^{K} \text{Error}_k$

Walk-Forward (finance):
Train on $[1, t]$, test on $[t{+}1, t{+}h]$, slide window forward.

Regularisation Effect:
$\lambda \uparrow \Rightarrow$ bias $\uparrow$, variance $\downarrow$

Print this slide. You will need it for the exam.

Classification Formula Reference

Models

Logistic Regression:
$p = \sigma(\beta^\top x) = \frac{1}{1+e^{-\beta^\top x}}$

Gini Impurity:
$G = 1 - \sum_k p_k^2$

Entropy:
$H = -\sum_k p_k \log_2 p_k$

Bayes' Rule:
$P(y|X) = \frac{P(X|y)\,P(y)}{P(X)}$

Metrics

Precision $= \frac{\text{TP}}{\text{TP+FP}}$

Recall $= \frac{\text{TP}}{\text{TP+FN}}$

F1 $= 2 \cdot \frac{P \cdot R}{P+R}$

Accuracy $= \frac{\text{TP+TN}}{N}$

AUC = Area under ROC curve (plots TPR vs FPR at all thresholds)

Advanced

SVM Margin:
margin $= \frac{2}{\|\mathbf{w}\|}$

Maximise margin $\Leftrightarrow$ minimise $\|\mathbf{w}\|^2$

SHAP Value:
$\phi_j = \sum_S \frac{|S|!(p-|S|-1)!}{p!}[f(S \cup j) - f(S)]$

$\sum_j \phi_j = f(x) - E[f(X)]$

"Each feature's marginal contribution, averaged over all orderings."

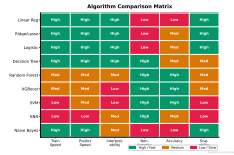
Print this slide. Use alongside the regression formula sheet.

How Do All Eleven Algorithms Compare Side by Side?

Read the chart:

- Rows = algorithms, columns = criteria (speed, interpretability, nonlinearity, scalability)
- Green = strength, amber = moderate, red = weakness
- No single algorithm wins on every criterion – choose based on your priorities

*SVM prediction speed depends on the number of support vectors.



No single algorithm dominates. The right choice depends on data size, interpretability needs, and deployment constraints.

Which Five ML Myths Can Derail Your Project?

Myth 1: "More data always helps."
Reality: More data helps only if it is representative. Biased data at scale = biased models at scale.

Myth 2: "The most accurate model is the best model."
Reality: In regulated industries, the most *interpretable* model that meets accuracy thresholds wins.

Myth 3: "Deep learning beats everything."
Reality: For tabular data, tree-based models consistently match or beat DL (Grinsztajn et al., 2022).

Myth 4: "Cross-validation gives the true error."
Reality: CV *estimates* the expected error; the variance of this estimate can be large for small datasets.

Myth 5: "Feature importance tells you causation."
Reality: Feature importance (including SHAP) shows correlation, not causation.

If you remember one thing from this slide: importance is not causation.

Can You Answer These Five Quick Checks?

Questions:

1. Name the loss function that OLS minimises.
2. What does Lasso do that Ridge does not?
3. Why is shuffled K-fold wrong for time series?
4. What does the "naive" in Naive Bayes mean?
5. When does a 99%-accuracy model fail?

Can You Answer These Five Quick Checks?

Questions:

1. Name the loss function that OLS minimises.
2. What does Lasso do that Ridge does not?
3. Why is shuffled K-fold wrong for time series?
4. What does the "naive" in Naive Bayes mean?
5. When does a 99%-accuracy model fail?

Answers:

1. Sum of squared residuals: $\sum(y_i - \hat{y}_i)^2$
2. Sets coefficients exactly to zero (automatic feature selection)
3. Future data leaks into training folds (lookahead bias)
4. Assumes features are conditionally independent given the class
5. When classes are heavily imbalanced (e.g., 1% fraud rate)

Scoring:

- 5/5: You understood the lecture deeply.
- 3–4/5: Review the sections you missed.
- 0–2/5: Re-read L21–L28 before the exam.

Cover the right column. Answer each question before revealing. Active recall beats passive review.

What Should You Read and Practice Next?

Textbooks (ranked by difficulty):

1. James et al., *Introduction to Statistical Learning* (ISLR) – free online, BSc level
2. Hastie et al., *Elements of Statistical Learning* (ESL) – free online, graduate level
3. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2022) – practical, sklearn-focused
4. Bishop & Bishop, *Deep Learning: Foundations and Concepts* (Springer, 2024) – bestselling ML book of 2024–2025

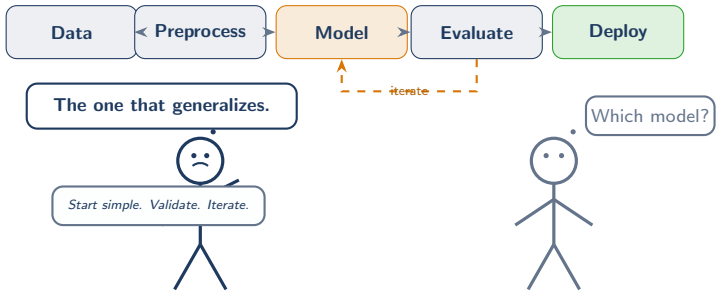
Finance-specific: de Prado, *Advances in Financial Machine Learning* (2018) – purged CV, backtest overfitting

Online: scikit-learn.org documentation, fast.ai courses (free)

This course: L21–L28 deep-dives, L29–L32 for unsupervised, L33–L36 for deep learning

Practice: Kaggle tabular competitions (start with "Titanic" or "House Prices")

The best way to learn ML is to build something. Pick a dataset, pick a method, and iterate.



Supervised Learning: From Linear Regression to Gradient Boosting