Machine Learning Lab Sessions 0-1 & 2

Discussion Questions and Answers

BFA3 – Université Paris-Dauphine
M2 Level

This document provides comprehensive answers to all discussion questions from the Machine Learning Lab sessions on Function Minimization, Linear Regression, Polynomial Regression, and Classification.

Session 0-1: Function Minimization & Linear Regression

Experiment 1: Impact of Step Size on Convex Functions

Q1. How does the number of iterations change as α increases? Why?

As α increases, the number of iterations generally decreases (up to a point). This is because larger step sizes allow the algorithm to move further toward the minimum in each iteration. With a small α (e.g., 0.01), the algorithm takes tiny steps and requires many iterations to reach the minimum. With a moderate α (e.g., 0.1), the algorithm converges more efficiently with fewer iterations. However, if α becomes too large (e.g., 1.0), the algorithm may overshoot the minimum repeatedly, causing oscillations and potentially requiring more iterations or even diverging. The optimal α balances convergence speed with stability.

Q2. What happens when α is too large? Can you explain this behavior mathematically?

When α is too large, the algorithm overshoots the minimum. Mathematically, consider the update rule $x_{k+1} = x_k - \alpha\nabla f(x_k)$. Near a minimum where the function is locally quadratic $f(x) \approx f(x^*) + (1/2)H(x-x^*)^2$, the gradient is approximately $\nabla f \approx H(x-x^*)$. If $\alpha > 2/H$ (where H is the Hessian), the correction $\alpha\nabla f$ can exceed twice the distance to the minimum, causing x_{k+1} to overshoot to the opposite side. This creates oscillations around the minimum. If α is even larger, these oscillations grow in magnitude, leading to divergence where the algorithm moves further from the minimum with each iteration.

Q3. For convex functions, does gradient descent always find the global minimum? Why or why not?

Yes, for convex functions, gradient descent will converge to the global minimum if the step size α is appropriately chosen. This is because convex functions have a unique property: every local minimum is also the global minimum. Mathematically, for a convex function, any point where $\nabla f(x) = 0$ is a global minimizer. Since gradient descent follows the negative gradient direction (the direction of steepest descent) and the function has no local minima that are not global minima, the algorithm cannot get stuck in suboptimal solutions. However, convergence requires: (1) α must be small enough to prevent divergence, (2) the function must be smooth (differentiable), and (3) sufficient iterations must be allowed. With these conditions met, gradient descent is guaranteed to find the global minimum of a convex function.

Experiment 2: Impact of Step Size on Non-Convex Functions

Q4. Does gradient descent always find the global minimum for egg-box functions? What does it find instead?

No, gradient descent does not always find the global minimum for egg-box (non-convex) functions. Instead, it typically finds a local minimum. Egg-box functions have multiple valleys and peaks, creating numerous local minima scattered throughout the domain. Gradient descent follows the path of steepest descent from its starting point, which leads it to the nearest local minimum. Once the algorithm reaches a local minimum where $\nabla f(x) = 0$, it stops because the gradient is zero, even though there may be deeper valleys (better minima) elsewhere in the domain. The algorithm has no mechanism to escape from local minima and explore other regions of the function. This is a fundamental limitation of gradient descent on non-convex functions.

Q5. Run the algorithm 5 times with the same α on an egg-box function. Do you always get the same result? Why or why not?

No, you will likely get different results each time, even with the same α. This occurs because the algorithm initializes at a random starting point within the domain for each run. Since egg-box functions have multiple local minima distributed across the domain, different starting points lead to different convergence destinations. For example, if one run starts in the left portion of the domain, it might converge to a local minimum in that region, while another run starting on the right side will converge to a different local minimum. The final result is highly dependent on the initial position. This sensitivity to initialization is a characteristic challenge of non-convex optimization and demonstrates why advanced techniques like random restarts, simulated annealing, or stochastic optimization methods are often employed for non-convex problems.

Q6. How does changing α affect which local minimum the algorithm finds?

The step size α affects both the convergence path and potentially which local minimum is found. A smaller α makes the algorithm take smaller, more cautious steps, causing it to descend into the nearest local minimum in a more direct path. The trajectory stays close to the steepest descent direction. A larger α causes the algorithm to take bigger steps, which can allow it to 'jump over' small local minima and potentially reach a different basin of attraction. However, if α is too large, the algorithm may oscillate or diverge entirely. The interaction between α and the initial position determines the final local minimum. With a moderate α, nearby starting points generally converge to the same local minimum, but with a large α, the algorithm's jumping behavior can lead to more variability in outcomes. Essentially, α influences the algorithm's exploration-exploitation trade-off during the descent process.

Experiment 3: Gradient Descent for Linear Regression

Q7. Is the OLS objective function convex? How can you tell from the plot?

Yes, the OLS (Ordinary Least Squares) objective function is convex. You can tell from the plot because it shows a bowl-shaped or parabolic curve with a single, well-defined minimum. Mathematically, for linear regression $L(a,b) = (1/n)\Sigma(y_i - ax_i - b)^2$, this is a quadratic function in the parameters a and b. The second derivative (Hessian matrix) of any quadratic function is constant and positive semi-definite (in fact, positive definite for non-degenerate data), which is the mathematical definition of a convex function. In the plot, you observe that: (1) there is exactly one minimum point, (2) the function increases as you move away from the minimum in any direction, and (3) there are no local minima or saddle points. This U-shaped or bowl-shaped appearance is the visual signature of convexity, ensuring that gradient descent will reliably converge to the unique global minimum.

Q8. Watch the regression line evolve as gradient descent runs. Describe how the line changes from iteration to iteration.

As gradient descent progresses, the regression line evolves systematically toward the optimal fit. Initially, the line is typically far from the data points because it starts with random or poor parameter values. In early iterations, you observe large changes: the line's slope adjusts significantly to align with the general trend of the data, and the intercept shifts to move the line closer to the data cloud. As the algorithm continues, the adjustments become more refined and smaller in magnitude. The line 'settles' into position, making minor corrections to both slope and intercept. Near convergence, the changes become nearly imperceptible as the gradient approaches zero. The line oscillates less and less around the optimal position. The evolution demonstrates the iterative refinement process: gradient descent continuously measures the error, computes which direction to adjust the parameters, and takes steps proportional to both the error magnitude and the step size α.

Q9. What value of α gives good convergence for the OLS problem? Is it the same as for the simple convex functions in Panel 1?

The optimal α for OLS regression is often different from simple convex functions in Panel 1, typically requiring a smaller value. While simple 1D convex functions might work well with $\alpha = 0.1$ or even 0.5, OLS problems often require α in the range of 0.01 to 0.1 for stable convergence. This difference arises from several factors: (1) Scale of gradients - the OLS gradient magnitude depends on the data scale and number of points, which can be much larger than simple functions; (2) Parameter coupling - in OLS, the slope and intercept parameters are coupled, meaning updates to one affect the optimal value of the other, requiring more careful steps; (3) Conditioning - the Hessian of the OLS objective depends on the spread and correlation structure of the data, affecting the optimal step size. The best α for a specific OLS problem can be found experimentally by testing values and observing convergence speed without oscillation. As a rule of thumb, if the loss initially increases or oscillates wildly, α is too large; if convergence is very slow (hundreds of iterations), α might be too small.

Synthesis Questions

Q10. Compare the number of iterations needed for gradient descent on: (a) A simple convex function (Panel 1), (b) The OLS objective function (Panel 2). What factors might explain the difference?

The number of iterations can vary significantly between these problems. Simple convex functions (Panel 1) often converge in 20-100 iterations with appropriate α, while OLS problems (Panel 2) may require similar or sometimes more iterations depending on several factors: (1) Dimensionality - Simple 1D functions optimize a single parameter, while OLS optimizes two coupled parameters (slope and intercept), increasing complexity; (2) Condition number - The ratio of the largest to smallest eigenvalue of the Hessian affects convergence rate. Well-conditioned simple functions converge faster, while OLS conditioning depends on data distribution; (3) Gradient magnitude - Initial gradients in OLS can be large if predictions are far from observations, requiring more iterations to descend; (4) Stopping criteria - The convergence threshold ε may interact differently with different function scales; (5) Starting point - Random initialization in OLS might be farther from the optimum than in bounded simple functions. Generally, problems with more parameters, worse conditioning, or larger initial errors require more iterations. The convergence rate also follows O(1/k) for general convex functions but can be faster for strongly convex functions with optimal α.

Q11. In practice, how would you choose a good value for α? Propose a strategy.

A practical strategy for choosing α involves multiple approaches: (1) **Grid search** - Test values on a logarithmic scale (e.g., 0.001, 0.01, 0.1, 1.0) and select the largest α that converges stably without oscillation. (2) **Learning rate schedule** - Start with a larger α for fast initial progress, then decrease it over time (e.g., $\alpha_k = \alpha_0/(1+k)$ where k is iteration number) to enable fine-tuning near the minimum. (3) **Line search** - For each iteration, choose α that minimizes $f(x_k - \alpha\nabla f(x_k))$ through backtracking or other line search methods. (4) **Adaptive methods** - Use algorithms like Adam, RMSprop, or AdaGrad that automatically adjust learning rates per parameter based on gradient history. (5) **Theoretical bounds** - For functions with known Lipschitz continuous gradients (constant L), use $\alpha < 2/L$ for guaranteed convergence. (6) **Monitor convergence** - Plot the loss vs. iteration; good α shows smooth, monotonic decrease. Oscillations suggest α is too large; very slow progress suggests it's too small. In practice, combining grid search with monitoring is most common: start with a reasonable range, observe behavior, and adjust accordingly.

Q12. What are the advantages of using gradient descent over the closed-form solution for linear regression? When would you prefer one over the other?

Advantages of gradient descent: (1) Scalability - Complexity O(kn) where k is iterations and n is data points, versus $O(n^3)$ for matrix inversion in closed-form; practical for large datasets. (2) Memory efficiency - Only stores gradients, not the full (X^TX) matrix which can be enormous. (3) Online learning - Can update parameters as new data arrives without recomputing from scratch. (4) Regularization - Easily incorporates L1/L2 regularization through modified gradients. (5) Extensibility - Generalizes to non-linear models, neural networks, and problems without closed-form solutions. **Advantages of closed-form solution:** (1) Exactness - Provides the exact optimal solution (subject to numerical precision), no approximation. (2) No hyperparameters - No need to tune α or

decide on convergence criteria. (3) Speed for small data - Faster for small to medium datasets ($n < 10,000$). (4) Guaranteed solution - Always provides the best solution in one computation. **When to prefer each:** Use closed-form for small datasets ($n < 10,000$), when you need exact solutions, for quick prototyping, or when computational resources are not constrained. Use gradient descent for large datasets ($n > 100,000$), in streaming/online settings, when memory is limited, for regularized models, or when building systems that will extend to more complex models.

Session 2: Polynomial Regression & Classification

Experiment 1: Quadratic Regression

Q1. Examine the 3D loss surface. Is it convex? What does this imply about the uniqueness of the optimal solution?

Yes, the 3D loss surface for quadratic regression is convex. You can observe this by noting that the surface forms a bowl or paraboloid shape without any local minima, saddle points, or valleys. Mathematically, the quadratic OLS objective $L(a,b) = (1/n)\Sigma(y_i - ax_i^2 - bx_i - c)^2$ is a quadratic function of parameters a and b, and its Hessian matrix is positive semi-definite (positive definite for non-degenerate data). The convexity of the loss surface has crucial implications: (1) **Unique global minimum** - There exists exactly one point (a*,b*) that minimizes the loss; (2) **No local minima** - Any local minimum is the global minimum, so gradient descent cannot get stuck; (3) **Guaranteed convergence** - With appropriate step size α, gradient descent will converge to the optimal solution from any starting point; (4) **Predictable behavior** - The optimization landscape is well-behaved and does not depend on initialization. This convexity property makes quadratic regression as reliable as linear regression from an optimization perspective.

Q2. Try fitting a Linear OLS model to the Quadratic dataset. Compare the final loss values. What do you observe?

When fitting a linear model to quadratic data, you observe that the final loss value is significantly higher than when fitting a quadratic model. This occurs because the linear model ($y = ax + b$) cannot capture the curvature in the data. Specifically: (1) **Systematic bias** - The linear model will show systematic errors: it underpredicts at the extremes and overpredicts (or vice versa) in the middle of the data range, creating a U-shaped or inverted U-shaped residual pattern; (2) **Higher MSE** - The mean squared error remains large even at convergence because the model is fundamentally misspecified; (3) **Model inadequacy** - No amount of optimization can overcome the structural limitation - the linear model lacks the flexibility to fit the parabolic pattern. For example, if the true relationship is $y = 2x^2 + 3x + 1$ and you fit $y = ax + b$, the best linear approximation might capture the average slope but misses the curvature entirely. The loss ratio (Linear MSE / Quadratic MSE) quantifies the improvement from using the more appropriate model. This demonstrates a fundamental principle in machine learning: model capacity must match the complexity of the data-generating process.

Q3. In the context of the volatility smile, why might we prefer a quadratic model even if the improvement in fit seems small?

In volatility smile modeling, even small improvements in fit have significant financial implications: (1) **Pricing accuracy** - Options pricing is extremely sensitive to implied volatility. A 1% error in volatility can translate to meaningful mispricings, especially for out-of-the-money options. Small absolute errors in fit can mean large monetary errors in derivative portfolios. (2) **Risk management** - Hedging strategies (Greeks computation) depend on the volatility surface. A linear model that misses curvature will produce incorrect deltas, gammas, and vegas, leading to improper hedge ratios and unintended

risk exposures. (3) **Arbitrage detection** - The shape of the volatility smile contains information about market expectations and risk-neutral probabilities. Capturing the quadratic curvature helps identify mispriced options and arbitrage opportunities. (4) **Regulatory compliance** - Financial regulations require accurate mark-to-market valuations. Using an inadequate model can lead to compliance issues and understated risk. (5) **Tail risk** - The curvature of the smile reflects market pricing of tail events (crashes, jumps). A linear model that ignores this systematically misprices tail risk, which can be catastrophic during market stress. Even if the R^2 improvement seems modest (e.g., from 0.95 to 0.98), the quadratic model's ability to capture the true structure is essential for sound financial decision-making.

Experiment 2: Model Mismatch

Q4. When fitting a quadratic model to linear data, what do you expect for the coefficient a (the x^2 term)?

When fitting a quadratic model $y = ax^2 + bx + c$ to truly linear data (where the true model is $y = mx + c$), you expect the coefficient a to be very close to zero. Mathematically, the quadratic model will find that the x^2 term does not improve the fit, so the optimization process drives a toward zero to minimize the loss. Specifically: (1) **Optimal value** - If the data is perfectly linear with no noise, $a^* = 0$ exactly, reducing the model to $y = bx + c$ (linear); (2) **With noise** - In the presence of random noise, a might be slightly non-zero due to random fluctuations, but it should be statistically insignificant and much smaller in magnitude than b; (3) **Interpretation** - This demonstrates that the quadratic model 'degrades gracefully' to a linear model when curvature is absent. The model has the flexibility to represent both linear and quadratic patterns. (4) **Statistical testing** - In practice, you would test whether a is significantly different from zero using a t-test. If not significant, you would prefer the simpler linear model by Occam's razor. This behavior illustrates an important principle: more complex models should reduce to simpler ones when the additional complexity is unsupported by the data.

Q5. Is there any danger in always using a more complex model (e.g., quadratic instead of linear)? Relate this to the concept of overfitting.

Yes, there are significant dangers in always using more complex models, primarily related to **overfitting**: (1) **Overfitting definition** - A model overfits when it captures noise and random fluctuations in the training data rather than the underlying true pattern. The model performs well on training data but poorly on new, unseen data. (2) **Why it occurs** - Complex models have more parameters and flexibility. With quadratic vs. linear, we add one parameter (a). With higher polynomials, we add even more. This flexibility allows the model to memorize training data, including noise. (3) **Generalization failure** - While a quadratic model fit to linear data might have slightly lower training loss (by fitting noise), it will likely have higher test loss on new data because the x^2 term captures spurious patterns. (4) **Interpretability** - More complex models are harder to interpret and explain. A simple linear relationship is easier to understand and trust than an unnecessary quadratic one. (5) **Computational cost** - Complex models require more computation for training and inference. (6) **Variance vs. Bias trade-off** - Simpler models have higher bias (may underfit) but lower variance (more stable predictions). Complex models have lower bias but higher variance (predictions vary more with different training sets). The principle of **parsimony (Occam's razor)** states: prefer the simplest model that adequately explains the data. In practice, use cross-validation to compare models and select based on test set performance, not just training loss.

Experiment 3: Classification with Logistic Regression

Q6. As gradient descent progresses, how does the decision boundary change? Does it stabilize?

As gradient descent progresses in logistic regression, the decision boundary (the line where P(y=1) = 0.5) evolves systematically: (1) **Initial positioning** - The boundary starts at a random or arbitrary position, often misclassifying many points; (2) **Early iterations** - The boundary moves rapidly, rotating and translating to better separate the two classes. You observe large changes in orientation and position as the gradient is large; (3) **Middle iterations** - The boundary continues adjusting but with smaller changes, fine-tuning its position to minimize classification errors. It begins to settle between the two clusters; (4) **Near convergence** - The boundary makes very small adjustments, eventually stabilizing at an optimal position that maximizes the margin between classes or minimizes the cross-entropy loss; (5) **Final state** - The boundary stabilizes when the gradient becomes negligibly small. For well-separated classes, it typically settles perpendicular to the line connecting cluster centers, positioned to maximize the likelihood of the observed labels. The stabilization indicates convergence: the parameters (w_1, w_2, b) have reached values where further small changes do not improve the loss significantly. However, note that for perfectly separable data, logistic regression weights can grow unbounded (the boundary keeps moving away to increase confidence), so convergence is defined by small gradient rather than parameter stability.

Q7. What happens if you set α too high (e.g., $\alpha = 5$)? Explain in terms of the loss function.

Setting α too high in logistic regression causes severe convergence problems: (1) **Loss increases** - Instead of decreasing monotonically, the loss may increase after updates. This happens because the update $x_{k+1} = x_k - \alpha\nabla L(x_k)$ overshoots the minimum. The large step carries the parameters far past the optimal point to a region with higher loss; (2) **Oscillation** - The algorithm oscillates wildly, jumping from one side of the optimal region to the other without settling. The decision boundary swings back and forth dramatically between iterations; (3) **Numerical instability** - Large parameter values can cause the sigmoid function $\sigma(z)$ to saturate at 0 or 1. When $z = w_1x_1 + w_2x_2 + b$ becomes very large in magnitude ($|z| > 500$), predictions become numerically extreme ($p \approx 0$ or $p \approx 1$), and gradients can explode or vanish; (4) **Loss explosion** - The cross-entropy loss $L = -(1/n)\Sigma[y \log(p) + (1-y)\log(1-p)]$ can approach infinity when predictions are confidently wrong ($p \to 0$ when y=1 or $p \to 1$ when y=0). A large α can cause such confident misclassifications; (5) **Divergence** - In extreme cases, parameters grow without bound and the algorithm completely fails to converge. The mathematical explanation is that α violates the Lipschitz constant of the gradient. For stable convergence, we need $\alpha < 2/L$ where L is the Lipschitz constant of ∇L. Exceeding this bound breaks the contraction property that guarantees convergence.

Q8. Generate a White Noise dataset and try to fit a Logistic Regression model. What happens? What does this tell you about trying to classify random data?

When fitting logistic regression to white noise (randomly labeled data with no true pattern), several instructive phenomena occur: (1) **No convergence** - The decision boundary wanders aimlessly without settling into a stable position. Since there's no true pattern to learn, the gradient points in arbitrary directions; (2) **Loss plateaus** - The

cross-entropy loss stabilizes around 0.693 (equal to -log(0.5)), which is the loss when the model predicts P(y=1) = 0.5 for all points. This is the best any model can do with purely random data; (3) **Random performance** - Classification accuracy hovers around 50%, no better than random guessing. The model cannot achieve better than chance performance because there is no signal to extract; (4) **Overfitting with complexity** - If we added more features or polynomial terms, the model might achieve higher training accuracy by memorizing noise, but test accuracy would remain 50%, demonstrating extreme overfitting. **Implications:** (a) **Signal matters** - Machine learning requires genuine patterns in data; algorithms cannot create predictive power from pure randomness; (b) **Baseline comparison** - Always compare model performance to random chance (50% for balanced binary classification) to verify that learning is actually occurring; (c) **Interpretability caution** - Even with random data, the model will produce a decision boundary and predictions. Without validation, one might mistakenly believe the model has learned something meaningful. This highlights the critical importance of test/validation sets and statistical significance testing.

Experiment 4: Financial Interpretation

Q9. In credit scoring, false negatives (predicting 'no default' when the borrower defaults) and false positives (predicting 'default' when the borrower repays) have different costs. How might you adjust the classification threshold (default is 0.5) to account for this asymmetry?

The classification threshold can be adjusted based on the relative costs of false positives (FP) and false negatives (FN): **Cost-sensitive threshold selection:** (1) **Quantify costs** - If a false negative (missed default) costs $10,000 on average (unpaid loan) and a false positive costs $500 (lost interest from rejected good borrower), the cost ratio is 20:1. (2) **Adjust threshold** - Lower the threshold below 0.5 to classify more borrowers as 'likely to default'. For example, use threshold = 0.3, meaning classify as default if P(default) ≥ 0.3. This increases sensitivity to defaults at the cost of more false alarms. (3) **Optimal threshold** - Mathematically, choose threshold t that minimizes expected cost: E[Cost] = P(y=1)·Cost(FN)·P(predict 0|y=1) + P(y=0)·Cost(FP)·P(predict 1|y=0). This can be computed using a confusion matrix at different thresholds. (4) **ROC analysis** - Plot the ROC curve (true positive rate vs. false positive rate) and select the threshold that provides the best trade-off given cost constraints. (5) **Business constraints** - In practice, financial institutions might set thresholds based on: (a) Risk appetite (conservative banks use lower thresholds), (b) Regulatory capital requirements (defaults affect capital ratios), (c) Portfolio targets (maintain certain approval rates), (d) Expected profit maximization considering both default losses and interest income. For asymmetric costs where FN >> FP, use threshold < 0.5. Where FP >> FN (rare), use threshold > 0.5. The key insight is that the 0.5 threshold is optimal only when both error types are equally costly, which is rarely true in finance.

Q10. For the volatility smile fitting problem: (a) What would happen if you used a higher-degree polynomial (cubic, quartic)? (b) What are the trade-offs between model complexity and out-of-sample performance?

(a) Higher-degree polynomials: Using cubic (x^3) or quartic (x^4) polynomials for volatility smile fitting provides more flexibility but introduces challenges: (1) **Better in-sample fit** - Higher-degree polynomials can capture more complex smile shapes (smirks, frowns) and might reduce training MSE; (2) **Overfitting risk** - With limited data points (typically 5-20 strike prices), cubic/quartic models can overfit, creating unrealistic oscillations between observed points; (3) **Extrapolation danger** - Polynomials of degree ≥ 3 can produce wildly incorrect predictions outside the data range. For out-of-the-money options beyond observed strikes, predictions might be nonsensical (negative implied volatilities); (4) **Instability** - Higher-degree models are sensitive to outliers and small data perturbations. A single misquoted option price can dramatically affect the entire fitted curve; (5) **Arbitrage violations** - Complex polynomial fits might violate no-arbitrage conditions (e.g., calendar spread arbitrage, butterfly arbitrage) by creating non-monotonic or convex-violated shapes in the volatility surface. **(b) Trade-offs:** (1) **Bias-variance trade-off** - Simpler models (quadratic) have higher bias (may miss complex patterns) but lower variance (stable, consistent predictions). Complex models (quartic) have lower bias but higher variance (predictions vary significantly with different data samples); (2) **Sample size requirements** - Complex models require more data. A good rule: need at least 5-10 observations per parameter. Cubic has 4 parameters, quartic has 5; (3) **Regularization need** - Complex models benefit from regularization (L1/L2 penalties) to prevent

overfitting; (4) **Out-of-sample performance** - This is measured by fitting on historical data and testing on recent data. Typically, quadratic or cubic with regularization performs best; quartic rarely improves out-of-sample MSE despite better in-sample fit. **Practical recommendation:** Start with quadratic, validate with cross-validation, and only increase complexity if out-of-sample performance genuinely improves. In practice, practitioners often use splines or SVI (stochastic volatility inspired) parameterizations rather than raw polynomials for better-behaved fits.

Q11. In a market regime classification problem, would you expect the decision boundary to remain stable over time? What implications does this have for model retraining frequency?

No, the decision boundary for market regime classification is unlikely to remain stable over time, with important implications: **Why instability occurs:** (1) **Regime shifts** - Market dynamics change due to policy changes (monetary policy, regulations), economic cycles (recessions, expansions), and structural shifts (technology adoption, globalization). The relationship between features (e.g., returns, volume) and regimes (bull/bear) evolves; (2) **Non-stationarity** - Financial time series are fundamentally non-stationary. Statistical properties (mean, variance, correlations) change over time, making historical patterns less predictive; (3) **Black swan events** - Crises (2008 financial crisis, COVID-19) fundamentally alter market behavior, rendering pre-crisis models obsolete; (4) **Market learning** - As trading strategies become well-known, they stop working (alpha decay). The signals that previously predicted regimes lose predictive power; (5) **Volatility clustering** - Market volatility itself is time-varying, affecting the salience of different features. **Implications for retraining:** (1) **Frequent retraining** - Models should be retrained regularly: (a) High-frequency trading: daily or even intraday, (b) Tactical allocation: weekly to monthly, (c) Strategic allocation: monthly to quarterly; (2) **Rolling windows** - Use rolling training windows (e.g., past 2-5 years) rather than all historical data to ensure recent patterns dominate; (3) **Online learning** - Implement online/incremental learning that continuously updates parameters as new data arrives; (4) **Model monitoring** - Track out-of-sample performance metrics continuously. Trigger retraining when performance degrades below a threshold; (5) **Ensemble approaches** - Maintain multiple models trained on different time periods and combine their predictions to hedge against regime change; (6) **Adaptive features** - Use features that adapt to changing market conditions (e.g., rolling volatility, dynamic correlations) rather than fixed look-back periods. **Practical strategy:** Combine scheduled retraining (e.g., monthly) with performance-based triggers (retrain if accuracy drops X%). Always maintain holdout periods and validate that retraining actually improves performance before deploying new models.