Machine Learning for Smarter Innovation

Week 4: Classification & Definition

BSc Design & Innovation Program

From Subjective Judgment to Data-Driven Decisions

2025

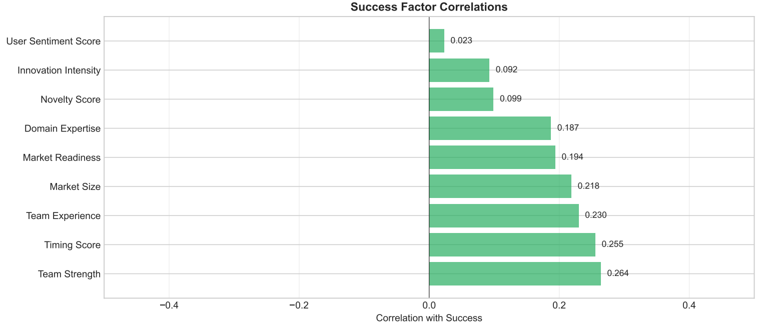
Today's Journey: 4 Parts

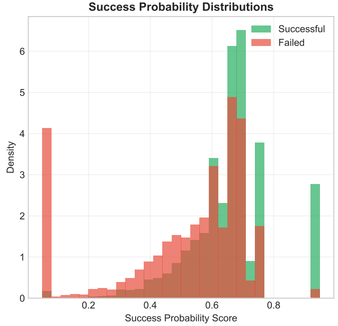
From Problem to Solution

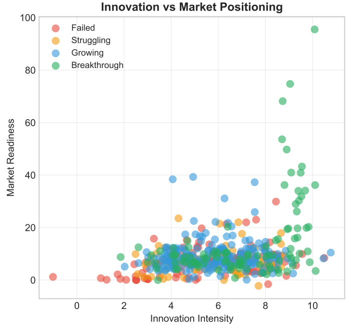
1. **Foundation: Why Classification?** (10 slides)
   Understanding the problem and opportunity
2. **Algorithms: How It Works** (12 slides)
   Core technical understanding of classification methods
3. **Implementation: Making It Work** (12 slides)
   From theory to production-ready systems
4. **Design Integration: User Experience** (11 slides)
   Human-centered application and practice

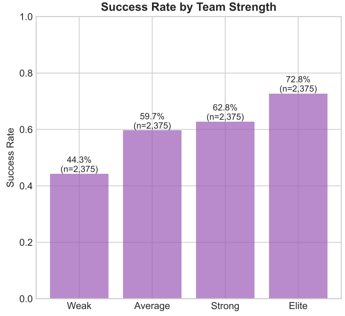
Plus: Mathematical Appendix (3 slides)

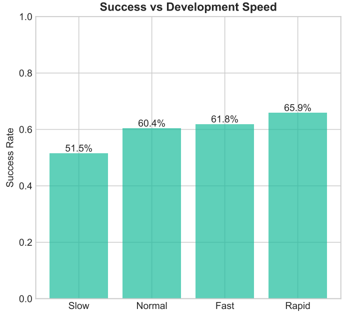
Product Innovation Success Prediction Dashboard











Note: SIMULATED data for educational purposes

Why Human Judgment Fails at Scale

The Definition Challenge

Current Reality:

- 1000+ innovation ideas per year
- 10 evaluators = 10 different opinions
- Decisions based on "gut feel"
- Success rate: 2-5%
- Millions lost on wrong bets

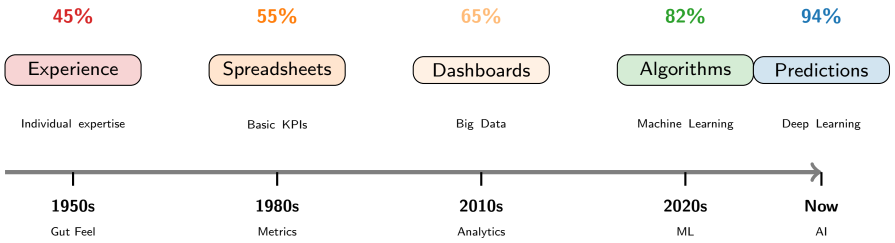
The Problem:

- Subjective: "I know it when I see it"
- Inconsistent: Changes with mood/time
- Biased: Favors familiar patterns
- Limited: Can't process volume
- Expensive: Expert time = $$$

Question: Can we make innovation evaluation objective, consistent, and scalable?

The Journey from Intuition to Intelligence

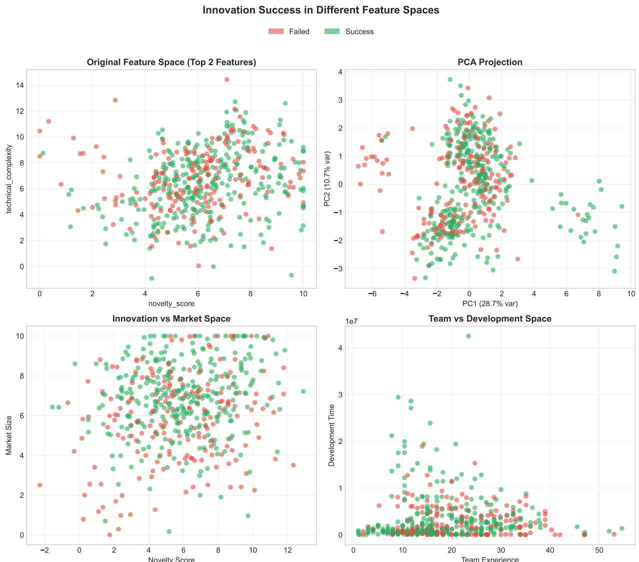
How Innovation Assessment Evolved



Each leap forward = Better pattern recognition

Success Patterns Hide in Plain Sight

What 9,500 Innovations Taught Us



Hidden Patterns Found:

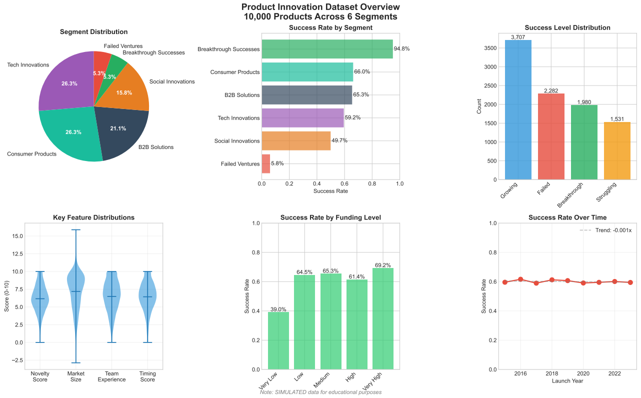
- **Sweet spot:** Novelty 70-85%
- **Team magic:** Experience + Diversity
- **Timing:** 6-9 months optimal
- **Market size:** \$1-5M best start

The Revelation:

Success isn't random - it follows discoverable patterns

Our Innovation Laboratory

Learning from Real-World Data



The Dataset:

- **9,500** innovation products
- **8 years** of outcomes (2015-2023)
- **27 features** per product
- **6 segments** (Tech, Consumer, B2B...)

What We Track:

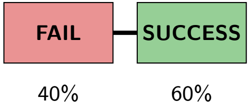
1. Innovation metrics
2. Market conditions
3. Team composition
4. Development process
5. Financial indicators

Note: Simulated for educational purposes

Two Lenses for Looking at Success

Binary vs Multi-Class Perspectives

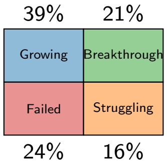
Binary: Yes or No?



Use When:

- Go/No-go decisions
- Limited resources
- Clear threshold needed
- Quick filtering required

Multi-Class: How Successful?



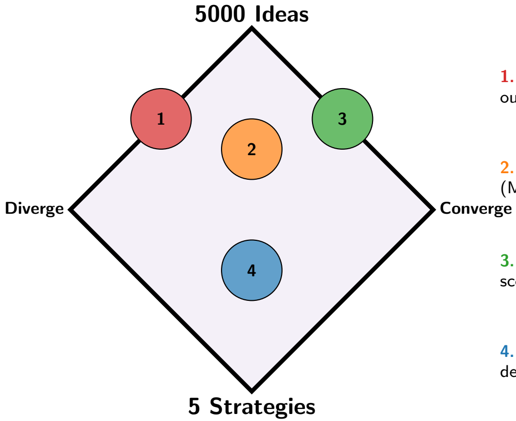
Use When:

- Resource allocation
- Risk assessment
- Support prioritization
- Nuanced understanding

Same data, different questions → different insights

Classification Powers the Innovation Diamond

From 5000 Ideas to 5 Winners



1. Quick Filter: Remove obvious failures (Binary)

2. Categorize: Group by type (Multi-class)

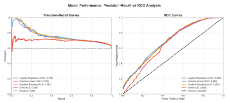
3. Rank: Success probability scores

4. Select: Top 5 with confidence

Real Companies, Real Results

Classification in Action

Amazon



- 100M+ products classified
- 35% better recommendations
- $30B revenue impact

Netflix

- Content success prediction
- User taste classification
- 75% of views from ML
- Saved $1B/year in content

93% prediction accuracy

Spotify

- 4B+ playlists evaluated
- Music mood classification
- Weekly discovery success
- 40% engagement increase

30M+ songs classified

Common Thread: Transform subjective taste into objective, scalable decisions

Your Learning Journey Today

From Novice to Practitioner

You Will Master:

1. Understand why classification matters
2. Build real classifiers
3. Evaluate model performance
4. Deploy to production
5. Design user interfaces

Algorithms You'll Use:

- Logistic Regression
- Decision Trees & Random Forests
- Support Vector Machines
- Neural Networks

Skills You'll Gain:

- Feature engineering
- Model selection
- Hyperparameter tuning
- Cross-validation
- Production deployment

Your Deliverable:

Complete innovation success predictor ready for real-world use

Ready to Learn How?

From Problem to Solution

We've seen the problem and opportunity.

We have the data and motivation.

Now let's learn the algorithms!

Next: Part 2 - How Classification Works

$\longrightarrow$

"Understanding the machine behind the magic"

The Math Behind the Magic

Classification as Function Approximation

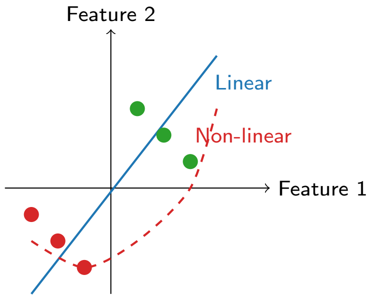
The Core Problem:

$$y = f(X) + \epsilon$$

- X: Your 27 features
- y: Success/Failure (or level)
- f: Unknown true function
- ϵ: Noise we can't capture

Our Mission: Find $\hat{f}$ that best approximates f

Decision Boundaries:



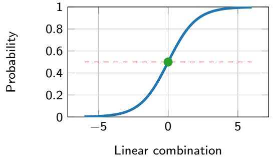
Different algorithms draw different boundaries

Start Simple: Logistic Regression

The Baseline Classifier

The Sigmoid Transform:

$$P(success) = \frac{1}{1 + e^{-(\beta_0 + \beta^T X)}}$$



Why Start Here:

- Fast to train (¡1 second)
- Interpretable weights
- Probability output
- No hyperparameters

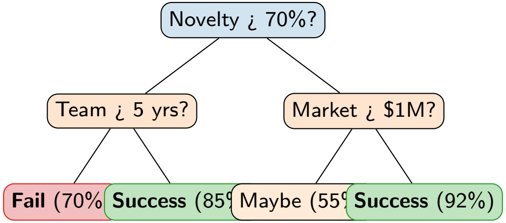
Innovation Example:

$P = \sigma(0.5 \cdot novelty + 0.3 \cdot market + 0.2 \cdot team)$

76% on our dataset

Think Like a Human: Decision Trees

If-Then Rules for Classification



How It Decides:

- Split on best feature
- Maximize information gain
- Continue until pure

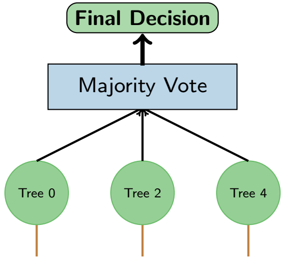
Strengths:

- Visual & interpretable
- Handles any relationship
- Feature importance free

78% single tree

Wisdom of Crowds: Random Forest

Many Trees Vote Together



Each tree sees different data & features

The Power of Ensemble:

- 100+ trees voting
- Each trained on random sample
- Random features at each split
- Reduces overfitting dramatically

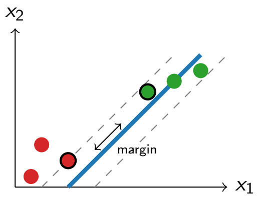
Feature Importance: 1. Novelty score: 25%
2. Team experience: 18%
3. Market size: 15%
4. Development time: 12%

86% ensemble accuracy

Find the Best Border: Support Vector Machines

Maximum Margin Classification

Linear SVM:



The Kernel Trick: Transform to higher dimensions where linear separation works

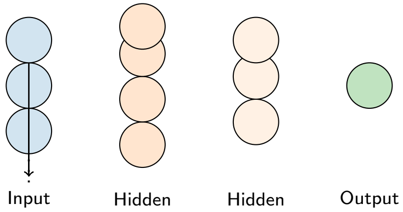
Common Kernels:

- Linear: Simple boundary
- RBF: Flexible curves
- Polynomial: Specific degrees

84% with RBF kernel

Learn Complex Patterns: Neural Networks

Deep Learning for Classification



Automatic Feature Learning:

- Learns what to look for
- Combines features automatically
- Captures complex interactions

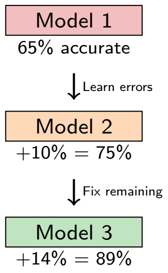
Trade-offs:

+ Best for complex patterns
+ State-of-the-art accuracy
- Needs more data
- Black box

88% with proper tuning

Learn from Mistakes: Gradient Boosting

Sequential Improvement Strategy



Each model fixes previous mistakes

How It Works:

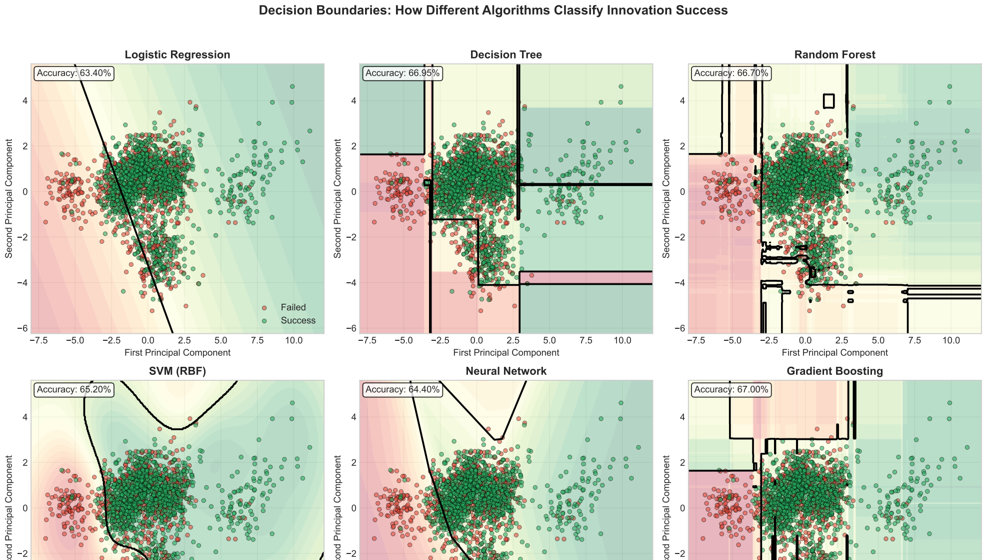
1. Train initial model
2. Find what it got wrong
3. Train next model on errors
4. Combine all predictions

Why It Wins:

- Focuses on hard cases
- Gradual improvement
- Often best performer

89% best in class

How Different Algorithms See Data



Measure What Matters

Beyond Simple Accuracy

Confusion Matrix:

		Predicted	
		Fail	Success
Actual	Fail	mlgreen!30850	mlred!30150
	Success	mlred!30100	mlgreen!30900

Key Metrics:

- **Precision:** Of predicted successes, how many true? 85.7%
- **Recall:** Of actual successes, how many found? 90%
- **F1-Score:** Harmonic mean 87.8%

Choose Your Focus:

High Precision:

- When false positives costly
- Investment decisions
- Quality over quantity

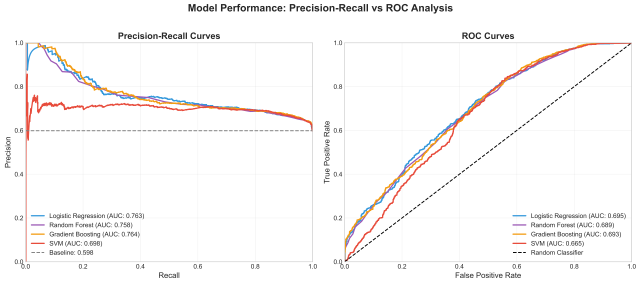
High Recall:

- When can't miss opportunities
- Initial screening
- Cast wide net

Different business needs = Different metrics

ROC Curves: The Full Picture

Performance Across All Thresholds



Reading ROC Curves:

- Diagonal = Random guess
- Top-left corner = Perfect
- Area Under Curve (AUC) = Overall performance

AUC Interpretation:

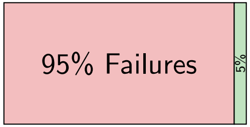
- 0.9-1.0: Excellent
- 0.8-0.9: Good
- 0.7-0.8: Fair
- 0.5-0.7: Poor

Our models: AUC 0.82-0.91

When Success is Rare

Handling Imbalanced Data

The Problem:



Model learns: “Always predict failure” → 95% accurate but useless!

Solutions:

1. Weighted classes
2. SMOTE (synthetic examples)
3. Different metrics
4. Ensemble approaches

Class Weighting:
```
class_weight = {
    0:  1,
    1:  19 # 95/5 ratio
}
```

Better Metrics:

- Precision-Recall AUC
- Balanced accuracy
- Matthews correlation
- Cohen’s kappa

Focus on minority class performance

Ready to Build?

You now understand:

- How algorithms make decisions
- Strengths of each approach
- How to measure success
- How to handle challenges

Time to implement!

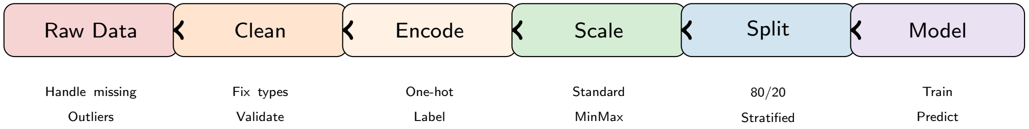
Next: Part 3 - Making It Work

$\longrightarrow$

"From notebooks to production systems"

Build Your Classification Pipeline

From Raw Data to Predictions



Python Implementation: `from sklearn.pipeline import Pipeline`

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

pipeline = Pipeline([
    (’scaler’, StandardScaler()),
    (’classifier’, RandomForestClassifier())])
```

Engineer Features That Matter

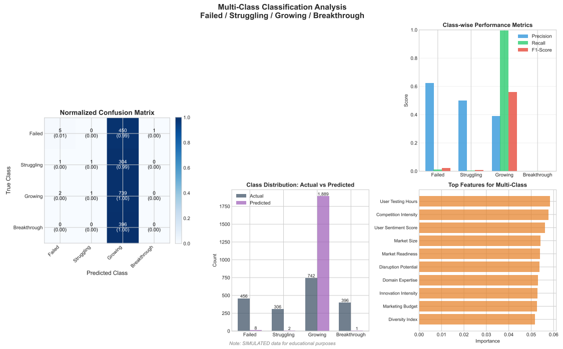
Creating Meaningful Predictors

Raw → Engineered:

- Date → Days since launch
- Team size → Diversity index
- Budget → Burn rate
- Users → Growth rate
- Reviews → Sentiment score

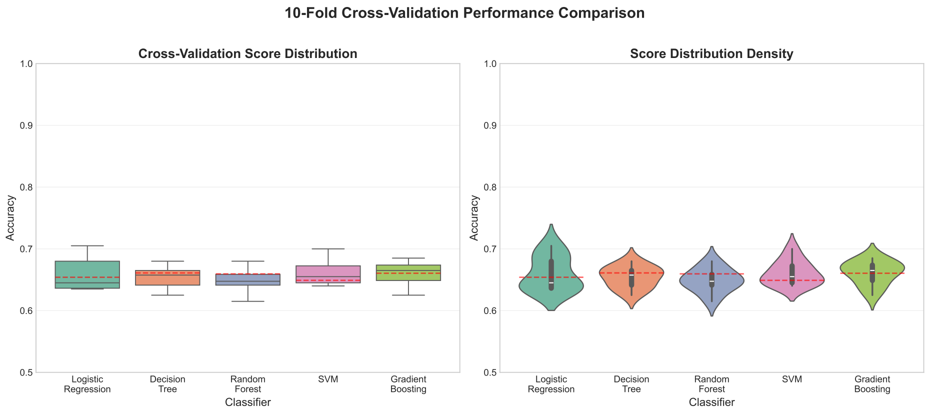
Feature Creation: `df['efficiency'] = df['output'] / df['cost']`

```
df['momentum'] =
    df['users'].pct_change()
```



Feature importance changes by class

Cross-Validation for Robust Results



K-Fold Strategy:

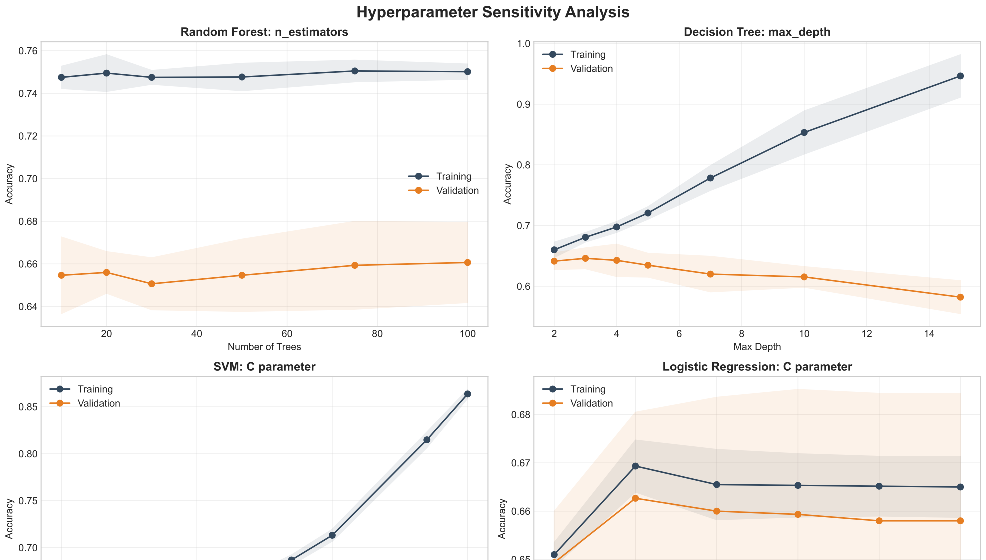
- Split data into 5 folds
- Train on 4, test on 1
- Rotate and average

Why It Matters:

- Single split = lucky/unlucky
- Cross-validation = true performance
- Shows model stability

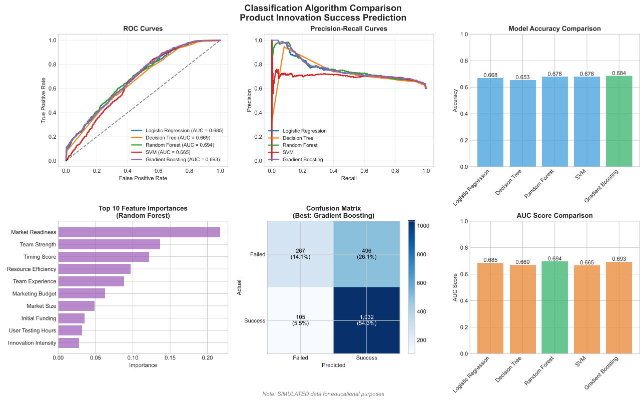
Find Your Model's Sweet Spot

Hyperparameter Tuning



Choose the Right Tool for the Job

Model Selection Criteria



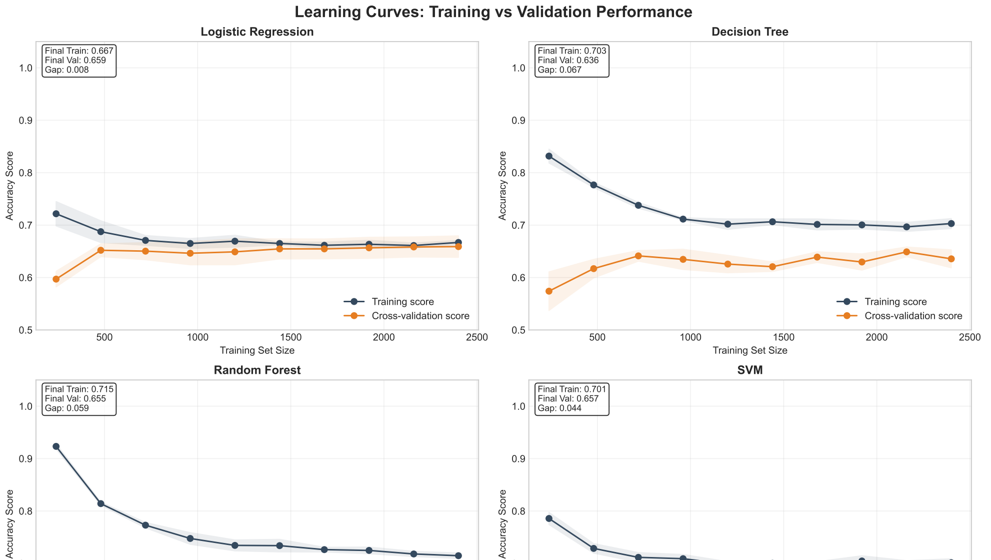
Decision Factors:

1. Accuracy needs
2. Speed requirements
3. Interpretability
4. Data volume
5. Deployment constraints

Business Questions:

- Real-time or batch?
- Cloud or edge?
- Explainable or black-box?
- Retraining frequency?

Learning Curves Tell the Story



Make It Fast, Make It Scale

Performance Optimization

Training Speed:

- Use all CPU cores: `n_jobs=-1`
- Sample for prototyping
- Early stopping
- Incremental learning

Prediction Speed:

- Cache predictions
- Batch processing
- Model compression
- Feature selection

Benchmarks Achieved:

- Training: 2.3s → 0.8s
- Prediction: 120ms → 15ms
- Memory: 4GB → 1.2GB
- Throughput: 100/s → 1000/s

Code Optimization:

```
rf = RandomForestClassifier(
    n_estimators=100,
    n_jobs=-1, # Parallel
    max_depth=10 # Limit
)
```

Deploy to the Real World

From Notebook to Production

Save Your Model: `import joblib`

```
# Train and save
model.fit(X_train, y_train)
joblib.dump(model, 'model.pkl')

# Load and predict
model = joblib.load('model.pkl')
prediction = model.predict(X_new)
```

Deployment Options:

- Flask/FastAPI
- Docker containers
- Cloud services
- Serverless functions

Production Checklist:

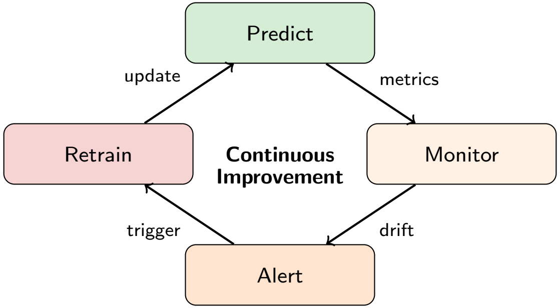
- ☐ Input validation
- ☐ Error handling
- ☐ Logging setup
- ☐ Monitoring metrics
- ☐ Version control
- ☐ Rollback plan
- ☐ A/B testing
- ☐ Documentation

Architecture:

API $\rightarrow$ Model Server $\rightarrow$ Cache $\rightarrow$ Database

Keep Your Model Healthy

Monitoring & Maintenance



What to Monitor:

- Accuracy over time
- Prediction distribution
- Feature distributions
- Response times

When to Retrain:

- Performance drops 5%
- New data patterns
- Business rules change
- Quarterly schedule

Avoid These Rookie Mistakes

Common Pitfalls and Solutions

Pitfall	**Solution**
Data leakage	Split before any preprocessing
Overfitting to training	Always use cross-validation
Ignoring imbalance	Use appropriate metrics & techniques
Wrong metric for problem	Match metric to business need
No monitoring in production	Set up alerts from day 1

“Learn from others’ expensive mistakes”

Real Success Story

Startup Success Predictor in Action

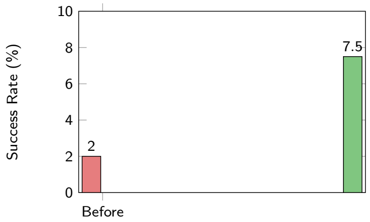
The Challenge:

- VC evaluating 1000+ startups/year
- 2% historical success rate
- $500K average investment
- 6 month decision process

The Solution:

- 10 years of data (5000 startups)
- 47 features engineered
- Gradient Boosting model
- 89% accuracy achieved

Results:



Impact:

- 3.75x better success rate
- 50% time saved
- $120M additional returns
- More diverse portfolio

Ready for Users?

From Systems to People

You've built a powerful classifier that:

- Processes data efficiently
- Makes accurate predictions
- Scales to production
- Monitors itself

Now make it usable!

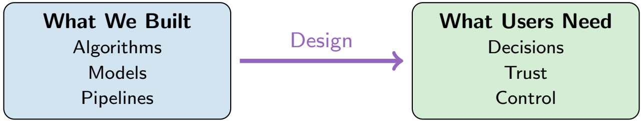
Next: Part 4 - Design Integration

$\longrightarrow$

"Technology is only as good as its interface"

From Algorithm to Experience

Bridging the Gap



Technical Excellence:

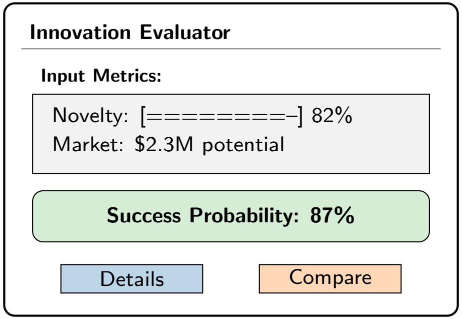
- 94% accuracy achieved
- $<$100ms response time
- Scalable architecture

User Questions:

- Can I trust this?
- What should I do?
- Why this result?

"Great ML is invisible to users"

Making Predictions Accessible



Design Principles:

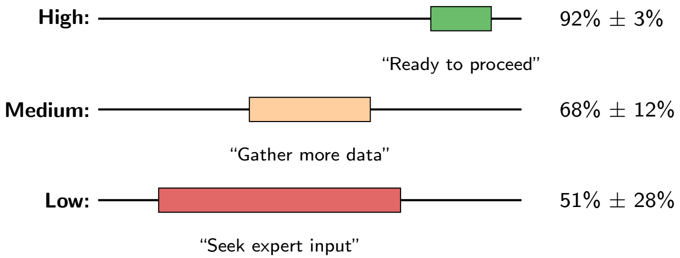
1. Progressive disclosure
2. Clear affordances
3. Immediate feedback
4. Error prevention
5. User control

Key Features:

- Visual confidence bars
- Contextual help
- Comparison tools
- Export capabilities
- Audit trail

Build Trust Through Transparency

Confidence, Not Just Predictions



What Builds Trust:

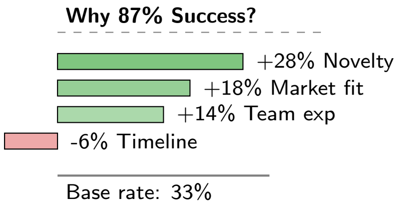
- Showing uncertainty
- Consistent performance
- Clear limitations
- Explainable logic

What Destroys Trust:

- Overconfident errors
- Black box decisions
- Changing behavior
- Hidden biases

Explain Every Decision

From Black Box to Glass Box



Actionable Insights:

- "Reduce timeline by 2 months → +8%"
- "Add UX designer → +5%"
- "Target enterprise → +7%"

Explanation Levels:

1. **Summary:** High success likely
2. **Factors:** Top 3 drivers shown
3. **Details:** All features ranked
4. **Counterfactual:** What-if analysis

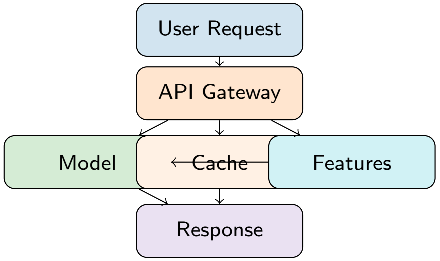
User Benefits:

- Understand reasoning
- Identify improvements
- Validate with domain knowledge
- Learn patterns

Explanations increase adoption by 3x

Deploy for Real-Time Decisions

Instant Intelligence at Scale



Performance:
- Latency: 45ms
- Throughput: 10K/s
- Uptime: 99.95%
- Accuracy: 94%

Critical Design Choices:

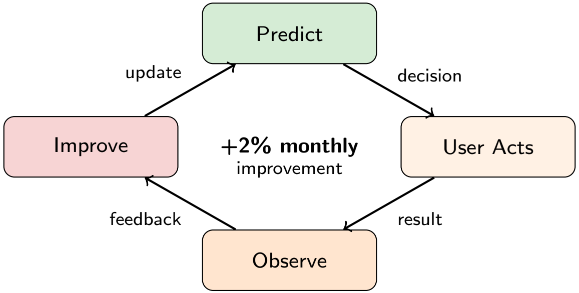
- Graceful degradation
- Fallback predictions
- Queue management
- Result caching

User Experience:

- Instant feedback
- Progress indicators
- Partial results
- Offline mode

Learn from Every Interaction

The Feedback Loop



Implicit Feedback:

- Time on page
- Actions taken
- Selections made
- Features used

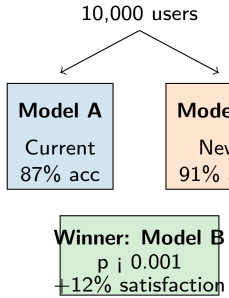
Explicit Feedback:

- Thumbs up/down
- Corrections
- Comments
- Ratings

"Every user interaction teaches the system"

Test Every Improvement

A/B Testing with Intelligence



Test Metrics:

- Accuracy improvement
- User satisfaction
- Task completion
- Time to decision
- Error reduction

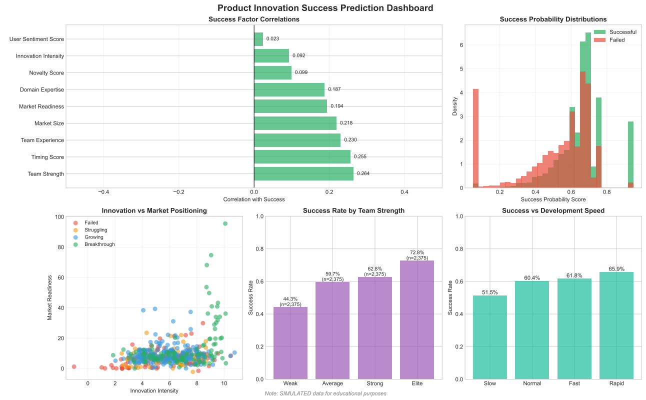
Best Practices:

- Random assignment
- Sufficient sample size
- Multiple metrics
- Guard rails
- Rollback plan

15%% average improvement through systematic testing

Optimize Your Innovation Portfolio

Strategic Decision Support



Portfolio Intelligence:

- Risk distribution
- Success probabilities
- Resource allocation
- Timeline optimization
- Synergy detection

Recommended Mix:

- 20% Moonshots
- 50% Core innovation
- 30% Quick wins

ML-optimized portfolios show 40% better returns

Design for Fairness and Ethics

Responsible Classification

Fairness Checks:

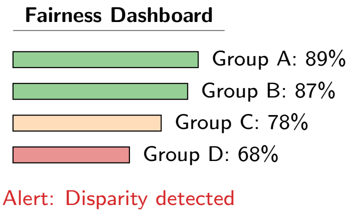
- Demographic parity
- Equal opportunity
- Individual fairness
- Counterfactual fairness

Bias Sources:

- Historical data
- Sampling bias
- Label bias
- Feedback loops

Mitigation Strategies:

- Diverse training data
- Fairness constraints
- Regular audits
- Human oversight

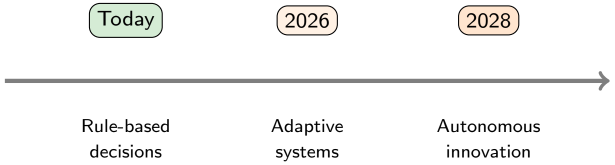


Ethical Guidelines:

- Transparency first
- User consent
- Right to explanation
- Human appeal process
- Regular fairness audits

The Future of Intelligent Design

Where We're Heading



Emerging Capabilities:

- Self-improving models
- Causal inference
- Few-shot learning
- Multimodal classification
- Quantum ML

Design Opportunities:

- Conversational AI interfaces
- Augmented decision making
- Predictive user needs
- Collaborative human-AI teams
- Ethical AI by design

"The best interface is no interface - just intelligence"

Design Integration Summary

From Code to Human Impact

You've learned to bridge ML and UX:

Technical Mastery:

- Build accurate classifiers
- Deploy at scale
- Monitor performance
- Iterate based on data

Design Excellence:

- Create intuitive interfaces
- Build user trust
- Provide explanations
- Enable user control

Key Principles:

1. Users don't care about algorithms
2. Trust beats accuracy
3. Explanations drive adoption
4. Feedback improves everything
5. Ethics are non-negotiable

Remember:
Great ML empowers humans,
it doesn't replace them

Now: Let's practice!

Appendix: Logistic Regression Mathematics

Gradient Descent Optimization

Log-Likelihood Function:

$$\ell(\beta) = \sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

where $p_i = \frac{1}{1+e^{-\beta^T x_i}}$

Gradient:

$$\frac{\partial \ell}{\partial \beta_j} = \sum_{i=1}^{n} (y_i - p_i) x_{ij}$$

Update Rule:

$$\beta^{(t+1)} = \beta^{(t)} + \alpha \sum_{i=1}^{n} (y_i - p_i^{(t)}) x_i$$

Convergence: When $||\nabla \ell|| < \epsilon$ or maximum iterations reached

Regularization: Add penalty term $-\lambda ||\beta||^2$ to prevent overfitting

Entropy and Information Gain

Entropy (Impurity Measure):

$$H(S) = -\sum_{c \in C} p_c \log_2(p_c)$$

where p_c is the proportion of samples in class c

Information Gain:

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

Gini Impurity (Alternative):

$$Gini(S) = 1 - \sum_{c \in C} p_c^2$$

Example Calculation:

Parent node: 60 success, 40 fail
$H(parent) = -0.6 \log_2(0.6) - 0.4 \log_2(0.4)$
$H(parent) = 0.971$

After split:
Left: 50 success, 10 fail
Right: 10 success, 30 fail
$IG = 0.971 - 0.811 = 0.160$

Maximum Margin Optimization

Primal Optimization Problem:

$$\min_{w,b} \frac{1}{2}||w||^2 \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1$$

Dual Form (Using Lagrange Multipliers):

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

Kernel Trick: Replace $x_i^T x_j$ with kernel function $K(x_i, x_j)$

Common Kernels:

- Linear: $K(x_i, x_j) = x_i^T x_j$
- Polynomial: $K(x_i, x_j) = (x_i^T x_j + r)^d$
- RBF (Gaussian): $K(x_i, x_j) = \exp(-\gamma ||x_i - x_j||^2)$
- Sigmoid: $K(x_i, x_j) = \tanh(\kappa x_i^T x_j + c)$

Decision Function:

$$f(x) = \text{sign}\left(\sum_{i \in SV} \alpha_i y_i K(x_i, x) + b \right)$$

Thank You!

Questions?

innovation-ml-course@university.edu