Machine Learning Practice

From Raw Data to Production Models

The Reality of ML Practice

Before you can type `model = LogisticRegression().fit(X, y)`:

- Feature Engineering
- Algorithm Selection
- Handling Overfitting
- Hyperparameter Tuning
- Performance Assessment

Reality: Most ML work happens before and after the model training line of code.

5.1 Feature Engineering

The Core Challenge

Product Manager: *"We need to predict customer retention. Here are 5 years of interaction logs."*

Your Task: Transform raw logs into a **labeled dataset** $\{(x_i, y_i)\}_{i=1}^{N}$

Feature Engineering: Converting raw data into informative feature vectors that enable learning algorithms to build predictive models.

What Makes a Good Feature?

Informative Features: Those that allow learning algorithms to predict labels accurately

High Predictive Power: Strong correlation with target variable

Examples for customer retention:

- Subscription price → economic commitment
- Login frequency → engagement level
- Session duration → user satisfaction
- Response time → system quality perception

Key Insight: Everything measurable can become a feature, but not everything should.

5.1.1 One-Hot Encoding

Problem: Many algorithms require numerical inputs, but data often contains categorical features

Solution: Convert categorical values into binary vectors

Example - Colors:

```
red    = [1, 0, 0]
yellow = [0, 1, 0]
green  = [0, 0, 1]
```

Why not use numbers? `red=1, yellow=2, green=3` implies ordering and distance relationships that don't exist.

5.1.2 Binning (Bucketing)

Purpose: Convert continuous features into categorical bins

When useful:

- Give learning algorithms "hints" about value ranges
- Reduce noise in continuous measurements
- Create interpretable thresholds

Example - Age binning:

```
Original: age = 7 years
Bins: [0–5], [6–10], [11–15]
Result: age_bin2 = 1, others = 0
```

Benefit: Algorithm learns "exact age doesn't matter within these ranges"

5.1.3 Normalization

Purpose: Scale features to standard range [0,1] or [-1,1]

Formula: $\bar{x}^{(j)} = \frac{x^{(j)} - \min^{(j)}}{\max^{(j)} - \min^{(j)}}$

Why normalize?

- **Faster learning:** Prevents large-scale features from dominating gradients
- **Numerical stability:** Avoids computer precision issues with very large/small numbers
- **Fair comparison:** Features contribute proportionally to distance calculations

Example: Feature range [350, 1450] → [0, 1]

5.1.4 Standardization (Z-score)

Purpose: Rescale features to have μ=0, σ=1 (standard normal distribution)

Formula: $\hat{x}^{(j)} = \frac{x^{(j)} - \mu^{(j)}}{\sigma^{(j)}}$

When to use standardization vs normalization:

- **Standardization:** Unsupervised learning, normally distributed data, presence of outliers
- **Normalization:** Most other cases, bounded feature ranges preferred

Modern reality: Most algorithms are robust to different scales, but preprocessing still helps.

5.1.5 Missing Data Strategies

Common approaches:

1. **Remove examples** with missing values (if dataset is large enough)
2. **Use algorithms** that handle missing values natively
3. **Data imputation** techniques

Imputation methods:

- Replace with **mean/median** of feature
- Replace with **out-of-range value** (e.g., -1 for [0,1] range)
- Replace with **middle value** (e.g., 0 for [-1,1] range)
- **Regression-based:** Use other features to predict missing values

5.1.6 Advanced Imputation

Regression-based imputation:

1. Treat missing feature as target variable
2. Use remaining features as predictors
3. Train regression model on complete examples
4. Predict missing values

Indicator variables:

- Add binary feature: "was original value missing?"
- Replace missing value with 0 or chosen constant
- Algorithm learns to handle missingness pattern

Critical: Use same imputation method for training and prediction data.

5.2 Algorithm Selection

Key Decision Factors

Explainability requirements:

- High accuracy "black boxes" (neural networks, ensembles)
- vs. Interpretable models (linear regression, decision trees)

Computational constraints:

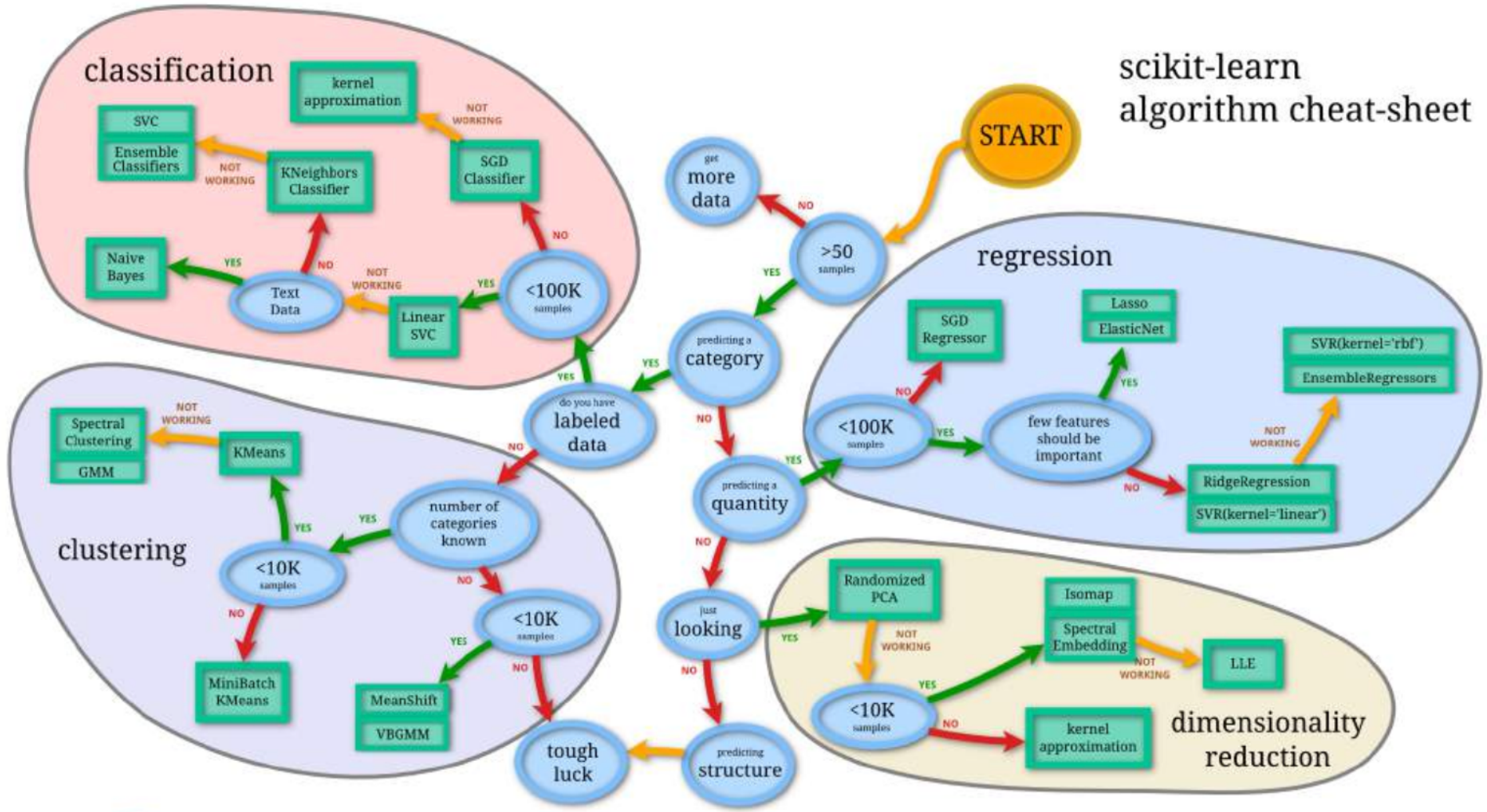
- In-memory vs. out-of-memory datasets
- Training speed requirements
- Prediction speed requirements

5.2 Algorithm Selection

Key Decision Factors

Data characteristics:

- Number of features and examples
- Categorical vs. numerical features
- Linear vs. non-linear relationships


scikit-learn
algorithm cheat-sheet
START
classification
kernel approximation
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
SGD Classifier
Naive Bayes
YES
NO
Text Data
NOT WORKING
Linear SVC
YES
<100K samples
NO
get more data
NO
>50 samples
YES
regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
predicting a category
YES
NO
do you have labeled data
YES
NO
<100K samples
YES
few features should be important
YES
NOT WORKING
RidgeRegression
NO
SVR(kernel='linear')
predicting a quantity
YES
NO
clustering
Spectral Clustering
NOT WORKING
GMM
KMeans
YES
number of categories known
YES
<10K samples
NO
NO
<10K samples
YES
NO
MiniBatch KMeans
MeanShift
VBGMM
just looking
YES
NO
Randomized PCA
NOT WORKING
Isomap
Spectral Embedding
NOT WORKING
LLE
YES
<10K samples
NO
kernel approximation
dimensionality reduction
tough luck
predicting structure

Training, Validation, and Test Sets

Three-way split necessity:

1. **Training set:** Build the model
2. **Validation set:** Choose algorithm and tune hyperparameters
3. **Test set:** Final assessment before production

Split proportions:

- Traditional: 70% / 15% / 15%
- Big data era: 95% / 2.5% / 2.5%

Why three sets? Prevent overfitting to evaluation data itself.

5.3 Underfitting vs. Overfitting

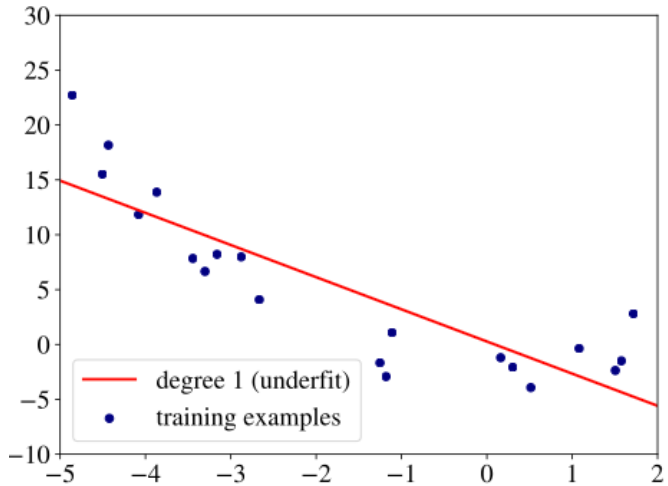
Underfitting (High Bias):

- Poor performance on training data
- Model too simple for the problem
- Features lack predictive power

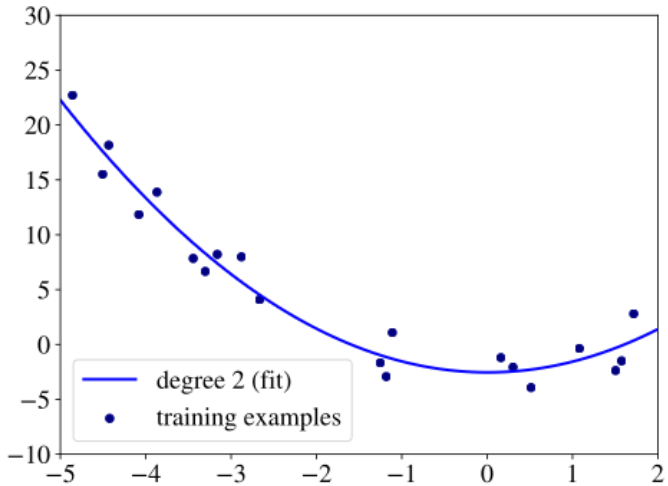
Overfitting (High Variance):

- Great training performance, poor validation/test performance
- Model too complex for available data
- Too many features, too few examples

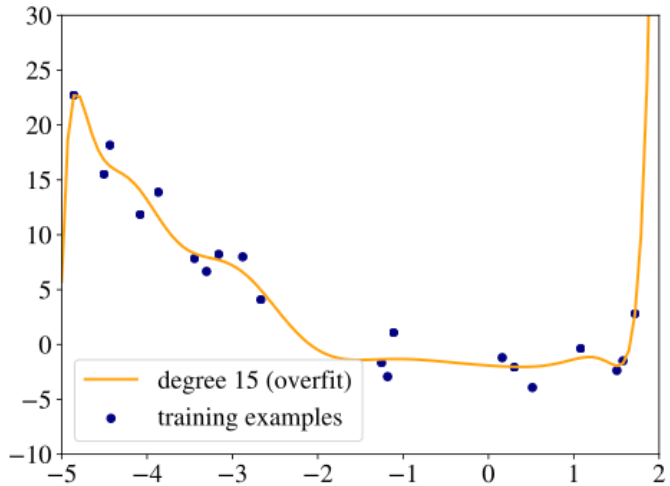
Goal: Find the sweet spot between underfitting and overfitting.



Underfitting



Good fit



Overfitting

Solutions to Underfitting

Increase model complexity:

- Use more sophisticated algorithms
- Add polynomial features
- Increase neural network depth/width

Improve features:

- Engineer more informative features
- Add domain-specific transformations
- Collect additional relevant data

Example: Predicting cancer with height, blood pressure, heart rate → clearly insufficient features

Solutions to Overfitting

Reduce model complexity:

- Simpler algorithms (linear vs. polynomial)
- Fewer neural network parameters
- Shallower decision trees

Data-based solutions:

- Collect more training examples
- Dimensionality reduction
- Feature selection

Regularization: Most widely used approach

5.5 Regularization

The Overfitting Problem

High-dimensional data + few examples = overfitting risk

Learning algorithm tries to fit training data perfectly by:

- Assigning non-zero weights to most features
- Learning noise and sampling artifacts
- Creating overly complex decision boundaries

Solution: Force algorithm to build simpler models through regularization.

L1 and L2 Regularization

L1 Regularization (Lasso):

$$\min_{w,b} C|w| + \frac{1}{N}\sum_{i=1}^{N}(f_{w,b}(x_i) - y_i)^2$$

L2 Regularization (Ridge):

$$\min_{w,b} C\|w\|^2 + \frac{1}{N}\sum_{i=1}^{N}(f_{w,b}(x_i) - y_i)^2$$

Hyperparameter C: Controls regularization strength

- C = 0: No regularization
- Large C: Heavy regularization (risk of underfitting)

L1 vs. L2 Properties

L1 Regularization:

- Creates **sparse models** (many weights = 0)
- Automatic **feature selection**
- Better for **interpretability**
- Non-differentiable at zero

L2 Regularization:

- Keeps all features with **small weights**
- Better **predictive performance**
- **Differentiable** everywhere, **computational advantages**

Elastic Net: Combines L1 and L2 regularization

5.6 Model Performance Assessment

Regression Assessment

Mean Squared Error (MSE):

- Compute MSE on training set
- Compute MSE on test set
- Compare the two values

Overfitting indicator: Test MSE >> Training MSE

Baseline comparison: Model should outperform mean prediction

Solutions: Regularization, hyperparameter tuning, simpler models

5.6 Model Performance Assessment

R² (Coefficient of Determination):

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

Symbol Definitions: y_i: Actual target value for example i , $\hat{y}_i$: Predicted value for example i (y-hat), $\bar{y}$: Mean of all actual target values (y-bar), SS_{res}: Sum of Squared Residuals (prediction errors), SS_{tot}: Total Sum of Squares (variance from mean)

R² Interpretation:

- **R² = 1:** Perfect predictions (all variance explained)
- **R² = 0:** Model performs as well as mean baseline
- **R² < 0:** Model performs worse than mean baseline

Classification Metrics

Core assessment tools:

- Confusion Matrix
- Accuracy
- Precision/Recall
- ROC Curve and AUC

Binary classification focus (extends to multiclass)

Key insight: Different metrics matter for different problems

5.6.1 Confusion Matrix

Spam Detection Example:

```
                  Predicted
                 spam  not_spam
Actual  spam      23      1      (TP=23, FN=1)
     not_spam     12    556      (FP=12, TN=556)
```

Definitions:

- **True Positives (TP):** Correctly identified spam
- **False Negatives (FN):** Missed spam
- **False Positives (FP):** Incorrectly flagged as spam
- **True Negatives (TN):** Correctly identified not spam

5.6.2 Precision and Recall

Precision: $\frac{TP}{TP+FP}$ (of predicted positives, how many were correct?)

Recall: $\frac{TP}{TP+FN}$ (of actual positives, how many were found?)

Document search analogy:

- **Precision:** Relevant documents / All returned documents
- **Recall:** Relevant documents returned / All relevant documents

Trade-off: Usually cannot maximize both simultaneously

Precision/Recall in Practice

Spam detection priorities:

- **High precision:** Avoid blocking legitimate emails
- **Lower recall acceptable:** Some spam in inbox is tolerable

Medical screening priorities:

- **High recall:** Don't miss diseases
- **Lower precision acceptable:** False alarms can be resolved

Tuning methods:

- Adjust class weights
- Modify decision thresholds
- Hyperparameter optimization

5.6.3 Accuracy

Definition: $\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$

When useful: All classification errors are equally costly

When problematic: Imbalanced classes or different error costs

Example: 99% non-spam emails → 99% accuracy by always predicting "not spam"

5.6.4 Cost-Sensitive Accuracy

Problem: Different types of errors have different costs

Solution: Assign costs to FP and FN, weight accordingly

Spam example:

- Cost of FP (blocking good email): High
- Cost of FN (missing spam): Low

Implementation: Multiply FP and FN counts by respective costs before calculating accuracy

5.6.5 ROC Curve and AUC

ROC Curve: Plot of True Positive Rate vs. False Positive Rate

True Positive Rate: $\frac{TP}{TP+FN}$ (same as Recall)

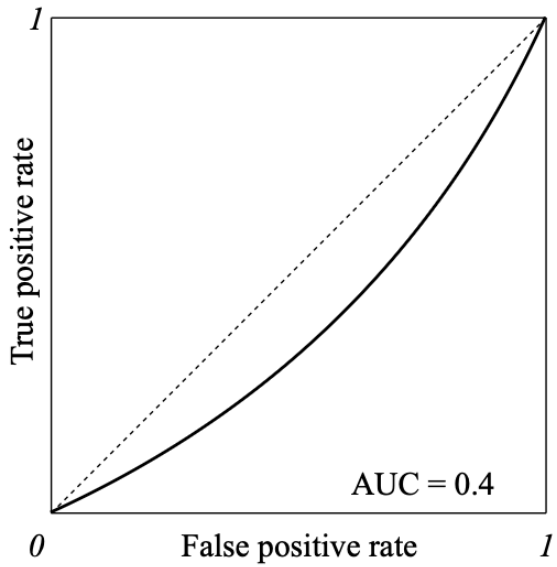
False Positive Rate: $\frac{FP}{FP+TN}$

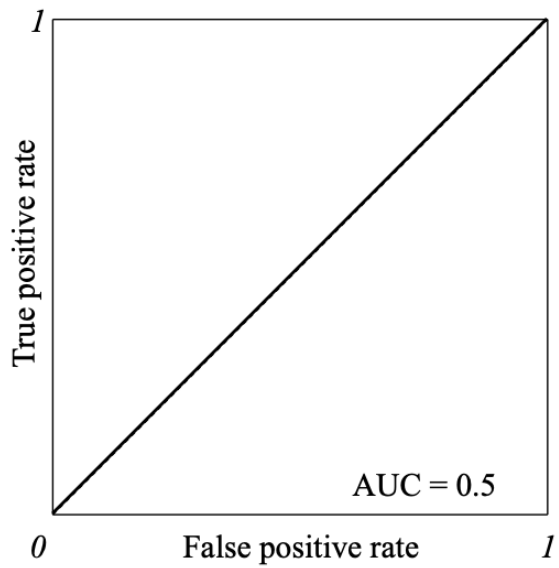
Construction: Vary decision threshold, plot (FPR, TPR) points

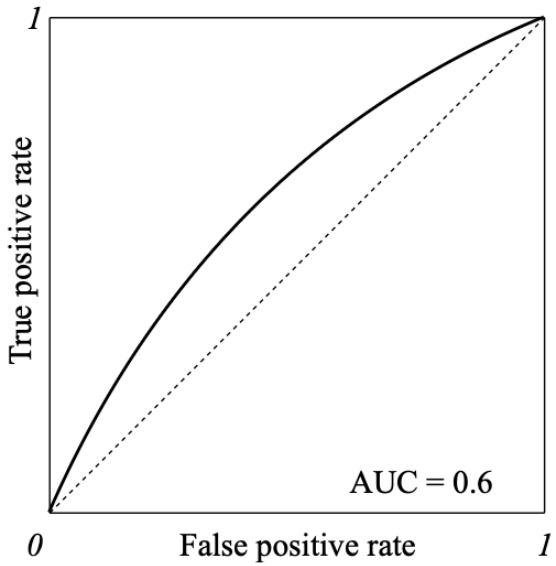
AUC (Area Under Curve): Single number summary of classifier performance

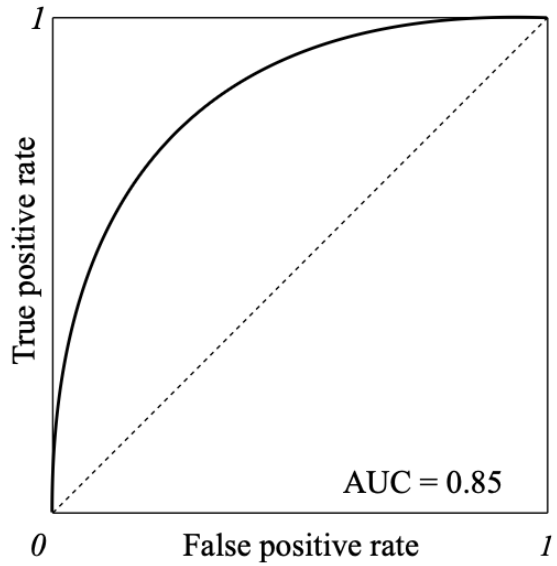
Perfect classifier: AUC = 1.0

Random classifier: AUC = 0.5


1
True positive rate
AUC = 0.4
0
False positive rate
1


1
True positive rate
AUC = 0.5
0
False positive rate
1


1
True positive rate
AUC = 0.6
0
False positive rate
1


1
True positive rate
AUC = 0.85
0
False positive rate
1

Summary: From Raw Data to Model

Essential workflow:

1. **Feature engineering** → Create informative representations
2. **Algorithm selection** → Match method to problem constraints
3. **Data splitting** → Enable unbiased evaluation
4. **Regularization** → Control model complexity
5. **Performance assessment** → Choose appropriate metrics

Key insight: Most ML success comes from careful data preparation and evaluation, not just algorithm choice.