Week 4: Supervised Learning & Classification Algorithms

Heart Failure Survival Analysis

MDST Project

Winter 2026

Outline

Quick Recap: Week 3 - Unsupervised Learning

What We Did:

1. Normalized data using Z-score
2. Applied PCA for dimensionality reduction
3. Performed K-Means clustering
4. Performed Hierarchical clustering

Key Insights:

- Clusters didn't perfectly separate death events
- Some natural groupings emerged
- PCA showed most variance in 2-3 components

This Week: Use target labels to train predictive models!

Supervised vs. Unsupervised

Unsupervised (Week 3)	**Supervised (Week 4)**
No target variable	Target variable (DEATH_EVENT)
Find patterns/groupings	Predict outcomes
PCA, Clustering	Classification, Regression

Classification: Predict categories (0 or 1: survived or died)

Regression: Predict continuous values (not used here)

The Machine Learning Workflow

1. **Load & Explore Data:** Understand features and target
2. **Train/Test Split:** Avoid data leakage!
3. **Normalize:** Put features on same scale
4. **Train Model:** Fit algorithm to training data
5. **Evaluate:** Test on unseen data
6. **Compare:** Which model is best?

Critical Principle: Train on training set, evaluate on test set. Never train on test data!

Why Split Data?

Problem: If we train and test on the same data, the model can memorize instead of learning.

Without Splitting:

- Train accuracy: 99%
- Test accuracy: 60%
- Model has "overfitted"

With Splitting:

- Train: 70% of data
- Test: 30% of data
- True performance on unseen data

Stratification: Keep same class distribution in both splits (important for imbalanced data)

Data Split in Practice

Our Dataset:

- 299 total samples
- 203 survived (68%)
- 96 died (32%)

70/30 Split with Stratification:

- Training: 209 samples (68% survived, 32% died)
- Test: 90 samples (68% survived, 32% died)
- **Important:** Use training set statistics to normalize test set!

Z-Score Normalization (Reminder)

Why: Distance-based algorithms need features on same scale

$$z = \frac{x - \mu}{\sigma}$$

Important: Compute μ and σ on **training data only**, then apply to test data

Code Pattern:

1. Split data into X_train, X_test, y_train, y_test
2. Compute normalization from X_train
3. Apply to both X_train **and** X_test
4. Train models on normalized X_train
5. Evaluate on normalized X_test

Logistic Regression

Despite the name: This is a **classification** algorithm

How It Works:

- Models probability of each class using sigmoid curve
- Finds a **linear** decision boundary
- Outputs probability scores (0 to 1)

Advantages:

- Fast to train
- Interpretable coefficients
- Good baseline model

Disadvantages:

- Can't capture non-linear patterns
- Assumes features are roughly linearly related to outcome

Random Forest

Ensemble Method: Combines many decision trees

How It Works:

1. Creates multiple random subsets of training data (sampling with replacement)
2. Trains a decision tree on each subset
3. Each tree votes on the prediction
4. Majority vote = final prediction

Advantages:

- Captures non-linear relationships
- Generally more accurate than single trees
- Provides feature importance scores
- Less prone to overfitting

Disadvantages:

- Less interpretable than logistic regression
- Slower to train

Support Vector Machines (SVM)

Goal: Find the best boundary that separates classes with maximum margin

Kernels:

- **Linear:** Simple, straight-line boundary
- **RBF (Radial Basis Function):** Non-linear, flexible boundary

Advantages:

- Works well with non-linear data (RBF kernel)
- Effective in high-dimensional spaces
- Robust to outliers

Disadvantages:

- Slower to train than RF
- Less interpretable
- Requires careful kernel selection

K-Nearest Neighbors (KNN)

Simple Idea: Classify based on nearest neighbors

How It Works:

1. For a new sample, find k nearest training samples
2. Majority class among neighbors = prediction

Advantages:

- Very simple to understand
- No training phase (lazy learner)

Disadvantages:

- Slow prediction time
- Sensitive to feature scaling
- Needs careful k selection

Default k=5, but should be tuned

Classification Metrics

Confusion Matrix:

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Common Metrics:

- **Accuracy:** (TP + TN) / Total [NOT great for imbalanced data]
- **Precision:** TP / (TP + FP) [Of predicted positives, how many are correct?]
- **Recall:** TP / (TP + FN) [Of actual positives, how many did we find?]
- **F1-Score:** Harmonic mean of precision and recall

For imbalanced data: Use F1, precision, or recall instead of accuracy

ROC-AUC Curve

ROC: Receiver Operating Characteristic

What It Shows:

- X-axis: False Positive Rate (FPR)
- Y-axis: True Positive Rate (TPR)
- Diagonal line = random classifier (AUC = 0.5)
- Curve above diagonal = good classifier (AUC ¿ 0.5)

AUC Score:

- 0.5 = random guessing
- 0.7-0.8 = good
- 0.8-0.9 = very good
- ¿0.9 = excellent

Advantage: Independent of class threshold, good for imbalanced data

Comparing Models

Standard Practice: Train multiple models and compare on test set

Models to Try:

1. Logistic Regression (baseline)
2. Random Forest
3. SVM (Linear and RBF kernels)
4. KNN (with different k values)

Evaluation Strategy:

1. Train on identical training sets
2. Evaluate on identical test sets
3. Compare: Accuracy, Precision, Recall, F1, ROC-AUC
4. Pick best overall model (or best for your use case)

Question: Does highest accuracy = best model? Not always! Consider precision vs recall tradeoffs.

Key Insights for This Dataset

Finding from Research (Chicco & Jurman 2020):

- Random Forest with all 13 features: 85% accuracy
- Random Forest with only 2 features (ejection_fraction + serum_creatinine): 82% accuracy

Lesson: More features $\neq$ better model!

This Week:

1. Build multiple classifiers
2. Evaluate performance
3. Understand when each model excels
4. No need to optimize everything—focus on learning!

Step-by-Step Implementation

1. **Load & Prepare Data**
   - Load CSV, drop the 'time' column
   - Separate features (X) and target (y)
2. **Train/Test Split**
   - 70% train, 30% test, stratified
3. **Normalize**
   - Z-score normalization on training stats
4. **Train Models**
   - Logistic Regression, Random Forest, SVM, KNN
5. **Evaluate**
   - Accuracy, Precision, Recall, F1, ROC-AUC
6. **Compare**
   - Which model performs best?

Common Mistakes to Avoid

1. **Data Leakage**
   - Using test data statistics for normalization [BAD]
   - Training on test data [BAD]
2. **Overfitting**
   - Model memorizes training data
   - Train accuracy 99%, test accuracy 60%
3. **Wrong Metrics for Imbalanced Data**
   - Using accuracy when data is imbalanced (68% vs 32%)
   - A model that always predicts '0' gets 68% accuracy!
4. **Forgetting to Normalize**
   - Some algorithms (SVM, KNN) require normalized features
5. **Not Comparing to Baseline**
   - Always train a simple model first (logistic regression)

This Week's Learning Goals

1. Understand supervised learning vs unsupervised
2. Properly split data and avoid data leakage
3. Train multiple classification algorithms
4. Evaluate models with appropriate metrics
5. Compare models and understand trade-offs

Outcome: You'll build a machine learning pipeline to predict heart failure outcomes and understand which algorithms work best for this problem.

Next Steps: Explore hyperparameter tuning, cross-validation, and feature selection in future work.