SEN608 · WEEK 8

Statistical Machine Learning

KNN · Decision Trees · Bagging · Random Forest · Boosting · XGBoost

Reference: Bruce, Bruce & Gedeck (2026), AI-Assisted Statistics for Data Scientists, 3rd ed., Chapter 6

Assoc. Prof. Dr. Onur Polat

Hacettepe University · Graduate School of Informatics · Data Analytics and Statistics

Fall 2026-2027 · M.Sc. in Systems Engineering

Week 8 — Agenda

Today's topics

01

K-Nearest Neighbors (KNN)

Similarity-based classification

02

Decision Trees (CART)

Recursive partitioning algorithm

03

Bagging and Random Forest

Ensemble learning with bootstrap

04

Boosting and XGBoost

Sequential error correction

SECTION 1

K-Nearest Neighbors (KNN)

"Similar records belong to similar classes" — simple, intuitive, powerful

KNN — Core Idea

For each new record, find the K nearest records

Algorithm (3 steps):

- 1. Find K records with similar predictor values
-
- 2. Classification: assign the majority class among those K to the new record
-
- 3. Regression: predict the average of those K neighbors

Key Terms

Neighbor: a record with similar predictor values to another record

Distance metric: function that summarises the distance between two records

Standardization: subtract mean, divide by standard deviation (z-score)

K: the number of neighbors used in the nearest-neighbor calculation

Note: KNN has no "model" — all training data is kept in memory (lazy learner). Computation happens at prediction time.

LendingClub Loan Example

Target: Predict whether a new loan will be paid off or default. Variables: dti (debt-to-income) and payment_inc_ratio (payment-to-income). With K=20 on 200 loans, the new loan has 9 "default" and 11 "paid off" neighbors → prediction: paid off.

Distance Metrics

How do we measure "how close" two records are?

Euclidean Distance

$$d(x,u) = \sqrt{[(x_1-u_1)^2 + (x_2-u_2)^2 + \ldots + (x_p-u_p)^2]}$$

"As the crow flies" straight-line distance. Default choice. Suitable for continuous numeric variables.

Manhattan Distance

$$d(x,u) = |x_1-u_1| + |x_2-u_2| + \ldots + |x_p-u_p|$$

"City block" distance traveled in one direction at a time. Useful when similarity is defined as point-to-point travel time.

Mahalanobis Distance

Accounts for correlation between variables: if two variables are highly correlated, Mahalanobis treats them essentially as one variable for distance purposes. Equivalent to Euclidean distance between principal components. Higher computational cost — uses the covariance matrix.

⚠ Large-scale variables dominate distance. E.g., income/loan amount (in thousands) will overwhelm ratio variables (1-100). → Standardization is required.

One-Hot Encoding and Standardization

Making variables usable for KNN

One-Hot Encoding

Converts categorical (string) variables into binary dummy variables. "home_ownership" → 4 columns (mortgage/own/rent/other). The term comes from digital-circuit terminology: only one bit is allowed to be positive.

Standardization (z-score)

$$z = (x - \bar{x}) / s$$

"Standard deviations away from the mean". Brings all variables to a similar scale — no variable dominates the model simply because of its unit of measurement.

Effect of standardization — LendingClub example

In unstandardized data, revol_bal (in dollars, thousands) dominates the distance — the 5 nearest neighbors have very close revol_bal to the new record but the other variables play almost no role. After standardization, all variables contribute.

🔑 **Standardization is mandatory for KNN, PCA, and clustering algorithms.**

Choosing K — Bias-Variance Trade-off

Small K: overfitting · Large K: oversmoothing

If K is too small (e.g. K=1)

⚠ **Overfitting**

- Memorizes everything, including noise
- Poor performance on new data
- High variance: model changes drastically with different training sets

If K is too large

⚠ **Oversmoothing**

- Local structure is lost
- KNN loses its main advantage (local flexibility)
- High bias: model too simple to capture the true relationship

How do we find the best K?

- Determined by accuracy on holdout/validation data (cross-validation)
- Typical range: K = 1 to 20
- To avoid ties, choose an **odd number**
- High SNR data (handwriting recognition): small K preferred
- Low SNR data (loan default): large K preferred

KNN as a Feature Engine

KNN not as a standalone model but as an aid to others

Staged KNN:

1. Run KNN on data → derive class or probability for each record
2. Add this prediction as a **new feature** to the data
3. In the second stage, run another classifier (tree, logistic regression)

Real-estate example — "comps" (comparable sales)

Real-estate agents price a home by looking at recent sales of similar homes — this is essentially a manual KNN. Location, square footage, structure type, lot size, bed/bath count — the K-nearest home average price. This value can be added as a new feature to a larger model.

⚠ *Multicollinearity concern? No — the KNN-derived information is highly local (derived from a few nearby records), so it is additional information and not redundant.*

SECTION 2

Decision Trees (CART)

Breiman (1984) — if-then-else rules for classification and regression

Decision Trees — Core Idea

Classification and Regression Trees (CART)

CART / Decision Tree properties:

- "If-then-else" rules — interpretable, easy to implement
- Capture non-linear relationships and complex interactions
- Unlike KNN, the rules are expressed in a human-readable form
- Developed by Breiman, Friedman, Olshen, Stone (1984)
- **Foundation of Random Forest and Boosted Trees**

Key Terms

Recursive partitioning: repeatedly split the data into homogeneous subgroups
Split value: the predictor value that divides records in two
Node: graphical representation of a split value in the tree
Leaf: end of a rule — the classification rule itself
Impurity: extent of class mixture in a partition
Pruning: cutting back branches to reduce overfitting

Simple Example — LendingClub Loan Tree

"If borrower_score ≥ 0.575 → paid off (70% confidence); else, if borrower_score ≥ 0.375 and payment_inc_ratio < 10.42 → paid off..."

Tree is traversed top-down (root → leaf). Each node is a rule. Each leaf is a prediction for records matching that path.

Recursive Partitioning Algorithm

How does the tree grow?

For each partition A, find the best two-way split:

```
1. For each predictor variable Xj:
   a. For every value sj of Xj:
       i. Split records into Xj < sj and Xj ≥ sj
      ii. Measure class homogeneity in each sub-partition
   b. Pick the sj that maximizes homogeneity
2. Across all variables, pick the best (Xj, sj) pair
```

Recursion:

1. A ← the full dataset
2. Split A with the partitioning algorithm into A_1 and A_2
3. Repeat on A_1 and A_2
4. Stop when no further split sufficiently improves homogeneity

Measuring Homogeneity / Impurity

Gini impurity and Entropy

Accuracy is a poor impurity metric. Two common alternatives:

Gini Impurity

$$I(A) = p(1 - p)$$

- p = proportion of class 1 in partition
- Max: p=0.5 → I=0.25 (most mixed)
- Min: p=0 or p=1 → I=0 (pure)
- Fast to compute

Entropy

$$I(A) = -p \cdot \log_2(p) - (1-p) \cdot \log_2(1-p)$$

- From information theory
- Higher values than Gini for moderate purity
- Logarithms → slightly slower
- Similar behavior in practice

How the algorithm uses impurity

For each candidate split, impurity is measured in both sub-partitions → a weighted average is computed → the split with the lowest weighted average is selected. This maximizes "information gain".

Stopping Tree Growth

Three strategies against overfitting

1. Minimum Leaf Size

`min_samples_leaf (default: 1)`
`min_samples_split (default: 2)`

Stop splitting when the leaf would have fewer records than the threshold. Prevents small noise-filled leaves.

2. Maximum Depth

`max_depth (default: None)`

Allow the tree to grow to a certain depth only. Hard cutoff — typically 5-15 is enough.

3. Complexity Penalty

`ccp_alpha (default: 0)`

Do not split if the impurity drop is insufficient. Cost-complexity pruning: add a complexity penalty to the tree — larger α → simpler tree.

The optimal value is typically determined by cross-validation (GridSearchCV).

Regression and the Use of Trees

Continuous targets + the strengths of trees

Predicting a Continuous Value (regression)

Same algorithm, different metric: instead of impurity, use squared deviations from the mean (MSE). Performance is measured with RMSE. In scikit-learn: DecisionTreeRegressor.

Attractive aspects of trees:

✓ Data exploration tool

Visualize which variables matter and how they relate. Capture non-linear relationships.

✓ Communicative power

Rules are easy to explain to non-technical audiences. Ideal for "selling" a data-mining project to management.

⚠ Weakness of a single tree

A single decision tree is usually weak in predictive power. Random Forest and Boosted Trees retain the strength of trees while greatly improving accuracy — but the rule-based communicative power of a single tree is lost.

SECTION 3

Bagging and Random Forest

"Wisdom of the crowd" — the average of many models beats one model

Ensemble Learning

Galton's ox and the wisdom of the crowd

1906 — Sir Francis Galton at a county fair in England

A contest asked people to guess the weight of an ox on exhibit. 800 guesses were collected. Although individual guesses varied widely, both the mean and median came out within 1% of the ox's true weight. James Surowiecki discusses this phenomenon in his book "The Wisdom of Crowds" (2004).

Ensemble Modeling — Basic Algorithm

1. Build a baseline model with the given data
2. Build an additional model (different algorithm or sample)
3. For each record, **average / take majority vote** of predictions
4. *"Wisdom of the crowd" → better prediction than any single model!*

Bagging (Bootstrap Aggregation)

Breiman (1996) — general ensemble algorithm

Algorithm:

1. Initialize M (number of models) and n (records per sample, n < N). Set m = 1.
2. Take a bootstrap resample (with replacement) of n records from the training data → (Y_m, X_m) = the "bag"
3. Train a model $f_m(X)$ on (Y_m, X_m)
4. m = m + 1. If m ≤ M, go to step 2.

Bagged estimate:

$$f(X) = (1/M) \cdot [f_1(X) + f_2(X) + \ldots + f_M(X)]$$

Reduces variance (lowers overfitting). Especially effective for high-variance algorithms like decision trees.

Random Forest

Bagging + random features = Breiman (2001)

Adds one extra "twist" to standard bagging:

1. Draw a bootstrap sub-sample from records
2. For the first split, randomly (without replacement) sample $p < P$ variables from the P variables
3. Among the sampled p variables, find the best (X_j, s_j) pair — evaluate all possible split points
4. Pick (X_j, s_j) that maximizes homogeneity
5. For the next split, go back to step 2 (p variables are **re-sampled** at every split)
6. Continue until the tree is fully grown
7. Go back to step 1 → new bootstrap sample, new tree

💡 **Rule of thumb for p (variables sampled per split): $p \approx \sqrt{P}$**

```
Python with scikit-learn: RandomForestClassifier(n_estimators=500, max_features="sqrt", n_jobs=-1, oob_score=True, random_state=42)
```

Out-of-Bag (OOB) Error and Variable Importance

Two powerful diagnostic tools of RF

Out-of-Bag Error

- Bootstrap sample covers ~63% of data
- 37% is "out-of-bag" for each tree
- Compute error on OOB records
- ≈ cross-validation, but for free!
- Enabled with oob_score=True

Variable Importance

Quantitatively rank how much each variable contributes to the model. In a single tree, this is visible — in RF it is lost, but recovered by two methods.

Two measurement methods:

① Accuracy Decrease (Permutation)

How much does accuracy drop if we randomly permute one variable's values? Accuracy is measured on OOB data → cross-validated. More reliable but computationally expensive.

② Gini Decrease

Average Gini-impurity decrease across all splits on that variable. Byproduct of the algorithm → fast. Accessed via feature_importances_. Based on training data → less reliable.

Random Forest Hyperparameters

"Knobs" — critical for preventing overfitting

`n_estimators`

Default: 500

Number of trees. More trees → more stable but slower. 100-500 typical. Increase until OOB plateaus.

`max_features`

Default: "sqrt"

Variables evaluated per split. Default: sqrt(n_features) for classification. Low values de-correlate trees.

`min_samples_leaf`

Default: 1 → 10

Minimum records at a terminal node. Default in Python = 1 (overfitting risk!). Try 5-20.

`max_leaf_nodes`

Default: None

Maximum leaves per tree. Default unlimited → large trees. Lower = simpler model.

SECTION 4

Boosting and XGBoost

"Each successive model tries to reduce the previous model's error"

Boosting — Core Idea

A sequence of error-correcting models

Bagging = Honda Accord

🚗 Reliable and steady
- Parallel tree training
- Minimal tuning
- OOB diagnostics
- Low overfitting risk

Boosting = Porsche

🏎️ Powerful but requires care
- Sequential tree training
- Many hyperparameters
- Cross-validation mandatory
- High overfitting risk — regularization needed

Boosting variants:

- **AdaBoost (Freund & Schapire, 1997):** increase the weight of misclassified records
- **Gradient Boosting (Friedman, 2001):** minimize a cost function; fit models to pseudo-residuals
- **Stochastic Gradient Boosting:** sample both records and variables in each round — most widely used
- **XGBoost:** the most common and fastest implementation of stochastic gradient boosting (Chen & Guestrin, 2016)

AdaBoost Algorithm

Freund & Schapire — the most intuitive boosting variant

1. Initialize M (max number of models), m = 1. Observation weights: $w_i = 1/N$. Ensemble $F_0 = 0$.
2. Using the weights, train f_m to minimize the weighted error e_m.
3. Add model to ensemble: $F_m = F_{m-1} + \alpha_m \cdot f_m$, where $\alpha_m = \frac{1}{2} \cdot \log((1-e_m)/e_m)$
4. Update weights: increase weight of misclassified observations (by α_m).
5. m = m + 1. If m ≤ M, go to step 2.

Boosted estimate: $F = \alpha_1 \cdot f_1 + \alpha_2 \cdot f_2 + \ldots + \alpha_M \cdot f_M$

Increasing weights on misclassified observations → algorithm "trains harder" on its weak side. Lower-error models get larger α_m → heavier votes.

Gradient Boosting and XGBoost

AdaBoost reformulated as an optimization problem

Gradient Boosting (Friedman, 2001)

Rather than updating weights, poses the problem as the minimization of a cost function. Each round fits a model to the pseudo-residuals of the previous models — "trains harder" on records with large residuals. Stochastic Gradient Boosting adds record and variable sampling at each round.

XGBoost — eXtreme Gradient Boosting (Chen & Guestrin, 2016)

- **Public domain, most widely used boosting software**
- Computationally efficient, multi-language support (Python, R, Scala)
- Critical parameters:
  - `subsample:` fraction of records to sample each round (random-forest-like)
  - `eta / learning_rate:` shrinkage applied to α_m — low value (0.1) reduces overfitting
  - `max_depth:` tree depth (default 3-6 — shallower than RF)
  - `reg_lambda, reg_alpha:` L2 and L1 regularization

Overfitting and Regularization

XGBoost's chronic disease: overfitting

⚠ XGBoost overfits by default

LendingClub example: default XGBoost on all variables → training error 13.3%, test error 35.3%! Large gap → model has memorized the training data, fails to generalize. Unlike Random Forest, XGBoost with default settings is NOT reliable.

Regularization — add a cost-function penalty

L2 (Ridge) — reg_lambda

Penalty on sum of squared coefficients. $\lambda \cdot (b_1^2 + \ldots + b_p^2)$. Shrinks large coefficients. Euclidean-distance based.

L1 (Lasso) — reg_alpha

Penalty on sum of absolute coefficients. $\alpha \cdot (|b_1| + \ldots + |b_p|)$. Drives some coefficients to exact zero → feature selection. Manhattan-distance based.

💡 **Example: lambda=1000 on LendingClub → training error 31%, test error 33%. The training-test gap is nearly closed.**

Hyperparameters and Cross-Validation

Taming XGBoost with grid search + CV

`eta / learning_rate`	Shrinkage factor (0-1). Default: 0.3 (R) / 0.1 (Python). Try 0.05-0.1 for noisy data. Lower → more rounds needed.
`n_estimators / nrounds`	Boosting rounds. Must be increased if eta is low. More rounds do no harm if overfitting-prevention is in place.
`max_depth`	Tree depth. Default 6 (R) / 3 (Python). Shallower than RF — avoids complex interactions in noisy data.
`subsample, colsample_bytree`	Fraction of records / variables sampled per round. Default: 1.0 (all). 0.5-0.8 reduces overfitting.
`reg_lambda, reg_alpha`	L2 and L1 regularization. Controls overfitting. Python defaults: reg_lambda=1, reg_alpha=0. R: both 0.

K-fold CV for Hyperparameter Selection

LendingClub example

Procedure:

1. Randomly split the data into K folds (e.g. 5)
2. For each fold: train on K-1 folds, test on the held-out fold → fold error
3. Mean error across K folds → "out-of-sample" accuracy for that hyperparameter set
4. Try all parameter combinations → pick the set with the lowest mean error
5. Retrain the final model on the full data with these optimal parameters

Grid-search result (LendingClub, 5-fold):

```
tree_depth=3, learn_rate=0.5 → 33.34% (best)
tree_depth=6, learn_rate=0.1 → 33.51%
tree_depth=3, learn_rate=0.9 → 33.67%
tree_depth=12, learn_rate=0.9 → 38.52% (worst)
```

→ Shallow trees + low learning rate give more accurate AND more stable results.

Comparison of the Four Methods

Which method when?

Method	Speed	Interpretability	Overfit Risk	Tuning Difficulty	Typical Use
KNN	Slow (prediction)	Low	High (small K)	Low (K + scale)	Small data, feature engine
Decision Tree	Fast	HIGH ✓	Very high	Medium	Exploration + communication
Random Forest	Medium	Medium	Low	Low	Production, balanced
XGBoost	Medium (parallel)	Low	High — reg. required	HIGH	Highest accuracy

👂 *Practical advice: Start with Random Forest for prototyping (robust, minimal tuning). Switch to XGBoost + CV for the last 1-2% improvement. If interpretability is required, use a single decision tree or SHAP values.*

Week 8 — Summary

Four pillars of Statistical Machine Learning

KNN

- Assign the class of similar records
- Standardization is mandatory
- Powerful as a feature engine

Decision Trees

- Recursive partitioning, Gini/Entropy
- Interpretable rules
- Pruning for overfit control

Random Forest

- Bagging + random features
- OOB error = free CV
- Permutation vs Gini importance

XGBoost

- Stochastic gradient boosting
- L1/L2 regularization critical
- CV-based hyperparameter search

🎯 Preparation for next week

- Assignment applying this week's four methods on LendingClub (or your own data) with sklearn will be posted.
- Next week: Unsupervised Learning (Clustering, PCA) — unsupervised methods.
- Reading: Bruce, Bruce & Gedeck (2026), Chapter 6, pp. 249-297.