Ensemble Methods

Bagging

Boosting

Stacking

Julia Lenc

Analytics Journey

Business Analytics (BA)

1. Intro: Business and Revenue models. KPIs
2. Business models translated into analytics
3. Techniques: Descriptive, Diagnostic, Predictive, Prescriptive

Diagnostic Techniques

1. Inference: hypotheses testing
2. Unsupervised Learning: clustering, dimensionality reduction, anomalies

Predictive Techniques

1. Supervised learning: overview
2. Preparation: data pre-processing
3. Foundations: model choice and evaluation
4. Regression: linear and non-linear
5. Classification: logistic regression, Naive Bayes, k-NNs
6. Time series: ARIMA, SARIMA, Exponential Smoothing
7. Non-linear: a. Decision Trees, b. SVM, c. (G)ARCH
8. Ensemble: bagging, boosting, stacking (this presentation!)
9. Neural Networks: FFNN, CNN, RNN, Transformers

Prescriptive Techniques

1. Optimization: Linear, Non-linear and Dynamic programming
2. Simulation: Monte Carlo, Discreet Events, System Dynamics
3. Probabilistic Sequence: Markov Chains, Markov Decision Processes
4. Reinforcement Learning: Q-Learning, Deep RL, Policy Gradient

Ensemble Learning

Intro

1. Trade-off 1: interpretability vs performance.
2. Trade-off 2: generalization vs accuracy.
3. What is ensemble learning (Bagging, Boosting, Stacking)?
4. When to use ensembles? Business applications.
5. When NOT to use ensembles? Risks behind complex models.

Core concepts: math intuition

1. Bagging (Bootstrap Aggregation).
2. Random Forest.
3. Boosting.
4. AdaBoost, Gradient Boosting, XGBoost.
5. Stacking.

Modeling steps + Python libraries and functions

1. SMART business question.
2. Data preparation.
3. EDA.
4. Model set up.
5. Training and evaluating initial model.
6. Tuning: n_estimators, max_depth, learning_rate, feature_importances
7. Interpretation: SHAP, feature importance.
8. Action! How to talk to non-tech partners.

Julia Lenc

Introduction to Ensembles

Julia Lenc

Interpretability vs Performance

- **Interpretability**: how easily humans can understand and explain how the model works. Focus: **which factors** impact predictions and **how.**
- **Performance**: how well a model makes accurate predictions on unseen data. Focus: **evaluation metrics** (e.g., Accuracy, Precision) on a test set.

Interpretability (simpler models) VS Performance (complex models)

Models from least to most complex

1.Baseline models

a. regression - linear and non-linear

b. classification - logistic regression, Naive Bayes, k-NNs

c. time series - ARIMA, SARIMA, Exponential Smoothing

↓

2. Advanced models

a. regression and classification - Decision Tree, SVM / SVR

b. time series - ARCH / GARCH

↓

3. Ensembles

a. bagging (most popular: Random Forest)

b. boosting (most popular: AdaBoost, Gradient Boosting, XGBoost)

c. stacking

↓

4. Neural Networks

Julia Lenc

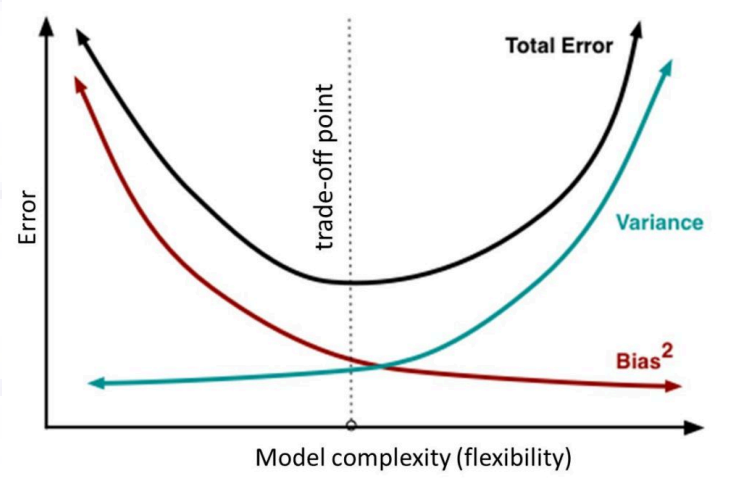
Accuracy vs Generalization

Bias → Accuracy: how closely a model captures the patterns of the data.

- **High Bias → risk of underfitting**; the model oversimplifies and may miss important patterns.
- **Low Bias → risk of overfitting**; the model learns the patterns effectively but can become prone to minor changes in data.

Variance → Generalization: model's sensitivity to fluctuations in the data.

- **High Variance → risk of overfitting**; the model fits the training data very closely but fails to generalize to new data.
- **Low Variance → risk of underfitting**: the model is more stable across datasets but might miss nuanced details.



Julia Lenc

What is ensemble learning?

Ensemble methods combine multiple individual "base" models to create a single, more powerful predictive model. *Individual models may be limited by bias or variance; ensembles use "the wisdom of crowds" to improve accuracy and robustness.*

Types:

1. **Bagging** *aka* Bootstrap Aggregating *helps to reduce variance. It mainly* **helps with overfitting***-prone models like decision tree. Bagging works by creating* **many versions of a base model** *on different random subsamples ("bootstrap" samples) of the data, and aggregating results, e.g., by* **voting** *or averaging.* **Random Forest** *is the most popular method.*

2. **Boosting** *helps to reduce bias. It* **helps with underfitting**, *when initial solution ("weak model") systematically misses patterns. Boosting builds* **models sequentially**, *so each new model focuses on "fixing" errors of the previous one. Due to its nature it is prone to overfitting! Popular algorithsm are* **AdaBoost, Gradient Boosting, XGBoost**.

3. **Stacking helps to reduce both bias and variance** *by combining various models (decision trees, regressions, SVMs, etc.); a* **meta-model** *then learns the "best way" to integrate their outputs. Very* **poweful** *but* **difficult to execute and to interpret**.

Julia Lenc

When to use ensembles?

Rule of thumb: performance (e.g., ensembles) is for execution, interpretability (e.g. decision tree) is for strategy.
Therefore, ensembles, as performance models should be used when performance is priority over interpretability... or, simply when you cannot achieve any reasonable level of performance with simpler models.

Fast Moving Consumer Goods
Interpretability - Marketing designs go-to-market campaign and needs to know how levels of media budget and its split impact profit.
Performance - Supply Chain Manager splits the forecast into weeks or days to secure stocks but avoid excessive inventory.

Banking
Interpretability - Credit risk officer wants to determine why a customer is likely to default on a loan (e.g., income instability, spending patterns).
Performance - Fraud detection model flags suspicious transactions in real time during online banking.

Telecom
Interpretability – Customer retention team wants to understand why a user is likely to churn (e.g., frequent dropped calls, high data costs).
Performance – Network anomaly detection model flags irregular traffic in real time to prevent outages or security breaches.

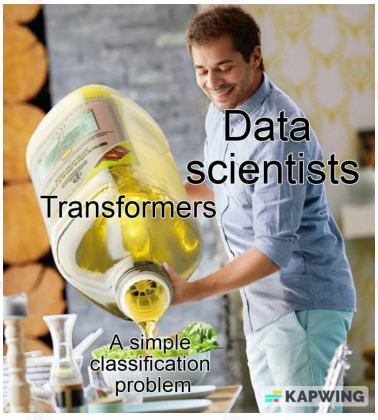
Julia Lenc

When NOT to use ensembles?

1. When **performance improves only slightly** compared to simpler models.

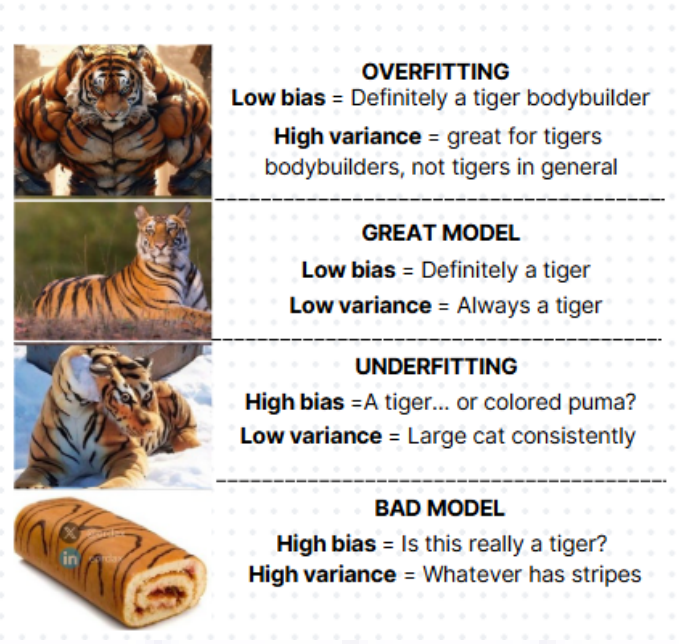
2. When your organization suspects that precision targeting may bring **reverse effects** and make target customers uncomfortable.

3. When you **don't have capabilities for precision targeting**, it's better to have interpretable insights for general population than very "fitted" insights for your (real or "wishful thinking") narrow prime prospect.



Case 1

Case 2



Case 3

Julia Lenc

Math behind Ensembles

Julia Lenc

Bagging

Step 0: you have already tried Decision Trees and they are not sufficient

Step 1: Bootstrat sampling.

- Your dataset has N examples (rows) and you decide to run T models on it.
- From your original sample you pick T samples (with replacement) of size N. Replacement means that some observations will be repeated - "red bubbles" and some will be ignored - "white bubbles": $D^{(t)} = \text{Bootstrap sample}, \quad t = 1, \ldots, T$

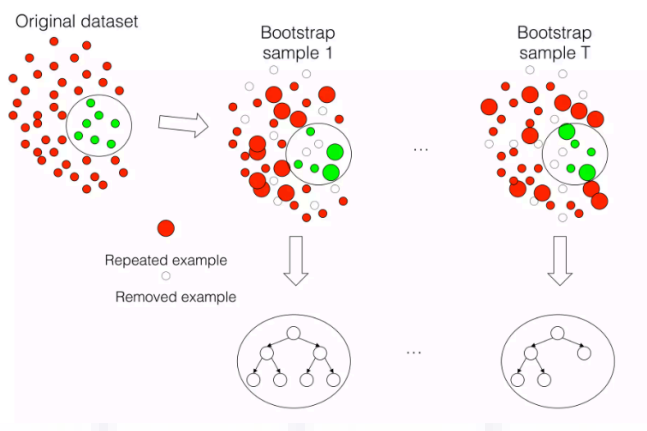
Step 2: Fit base models

- For each bootstrap sample you train a base model, e.g. Decision Tree.

  Train a model $h^{(t)}(x)$ on each $D^{(t)}$
- Each Tree will be different because it sees different part of data.

Step 3: Aggregate the outputs from all predictions

- Classification: majority vote $\hat{y}_{\text{bag}}(x) = \text{majority vote}[h^{(1)}(x), h^{(2)}(x), ..., h^{(T)}(x)]$
- Regression: average of the predictions $\hat{y}_{\text{bag}}(x) = \frac{1}{T}\sum_{t=1}^{T} h^{(t)}(x)$



Picture from: Gonzalo Martiney Munoz

Julia Lenc

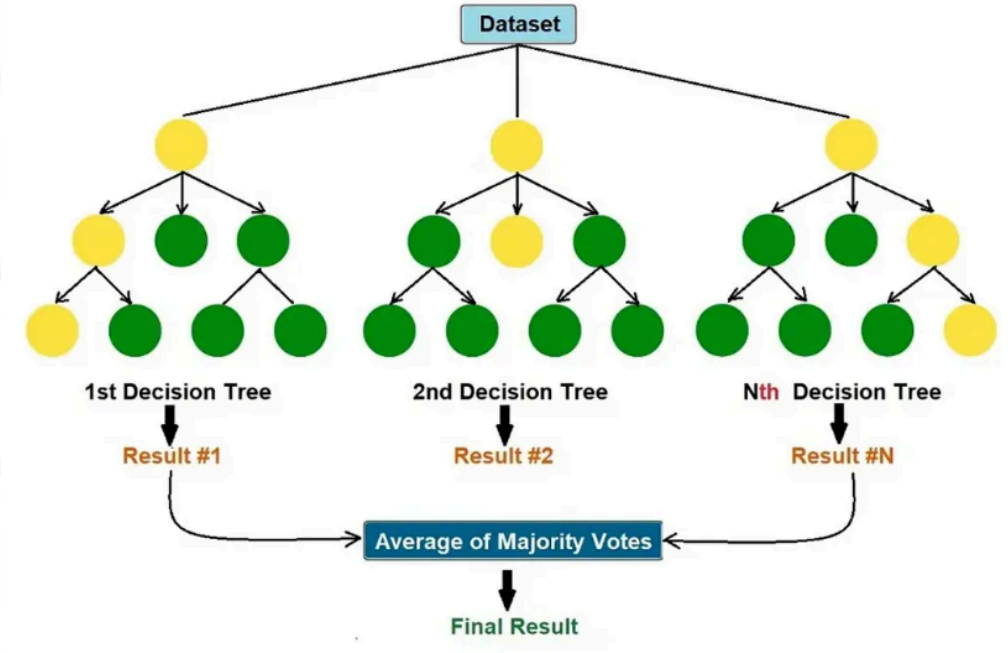
Random Forest

Random Forest is an improved version of Bagging. We do the same steps... but there is a **twist!**

When building each decision tree, instead of checking all possible features for the best split, we choose a **random subset of features** for each split.

Why this is so cool?

- In Bagging, trees can still look very similar **if some features are very strong**, so the **predictions might still be "correlated."**
- In Random Forest, by forcing each tree to look at different random subsets of features, the trees become even more different!
- This makes the final combined prediction even more reliable.



Julia Lenc

Picture from: Iyurek Kılıç

Boosting - Ada Boost example

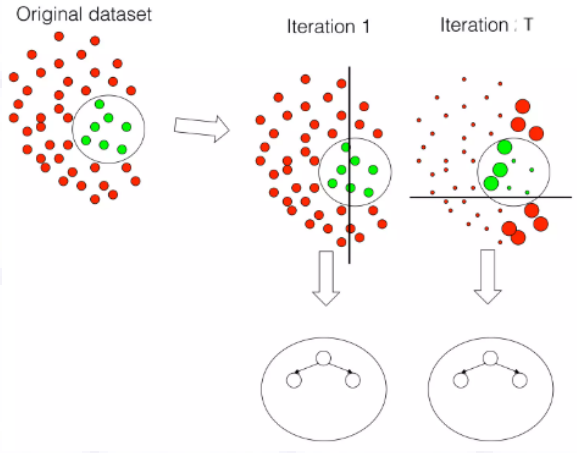
Step 0: you have already tried Decision Trees and they are not sufficient

Step 1: Model initialization

- Your dataset has N examples (rows). You will iterate T times: t=1,2,...,T.
- Before the first iterations, each has the same weight: $w_i^{(1)} = \frac{1}{N} \quad \text{for all } i = 1, \ldots, N$

Step 2: Boosting rounds - iterations

- You train the model ("wear learner") $h_t(x)$ T times. t = 1, 2... T
- Each time, the error is calculated: $\text{err}_t = \frac{\sum_{i=1}^{N} w_i^{(t)} \cdot I(h_t(x_i) \neq y_i)}{\sum_{i=1}^{N} w_i^{(t)}}$
- Each time, the model importance (alpha) is calculated: $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \text{err}_t}{\text{err}_t}\right)$
- The higher is alpha, the better this simple model performed (lower error) and the more influence this model will have on the final combined prediction. If a weak learner does really well, it gets a bigger voting power; if it does poorly (error close to 0.5 - random guess), its voting power is small.
- A higher alpha will also lead to stronger changes in the example weights for the next round. This means the algorithm will try even harder to adjust and fix the mistakes made on examples that were misclassified $w_i^{(t+1)} = w_i^{(t)} \cdot \exp(-\alpha_t y_i h_t(x_i))$
- Normalize the weights so they sum to 1.



Picture from: Gonzalo Martiney Munoz

Step 3: Final prediction

- The final prediction for a new sample x is a weighted vote from all rounds.
- Each tree/classifier's vote is weighted by how well it performed: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

When to stop? Defining T

- Performance no longer improves
- Error reaches pre-defined threshold
- Set in advance, e.g. maximum T = 1000

3 popular boosting methods

1. AdaBoost (Adaptive Boosting) – the simplest

- Main idea: train a series of simple models one after another, each time focusing more on the examples that were previously misclassified.
- How it works: **mistakes** made by earlier models are given more "attention" in the next round, so l**ater models try harder** on those examples.
- Good for: fast understanding, small-to-medium datasets.

2. Gradient Boosting – more flexible and powerful

- Main idea: models are added one-by-one, and each new one is trained to correct the errors (residuals) of the whole group so far.
- How it works: instead of just looking at which points were wrong, it uses a more general **idea of "how far off" the group's prediction is**, and tries to make improvements step by step, like using feedback to guide learning.
- Good for: more complicated problems and different types of data/tasks.

3. XGBoost (Extreme Gradient Boosting) – most advanced

- Main idea: **builds on gradient boosting, but is optimized** for speed, accuracy and ability to handle large datasets.
- How it works: adds lots of **engineering tricks for faster training** (parallelization, handling missing values, regularization to avoid overfitting).
- Good for: large datasets, when **top performance** is needed or if you want to show up in Kaggle competitions 😄

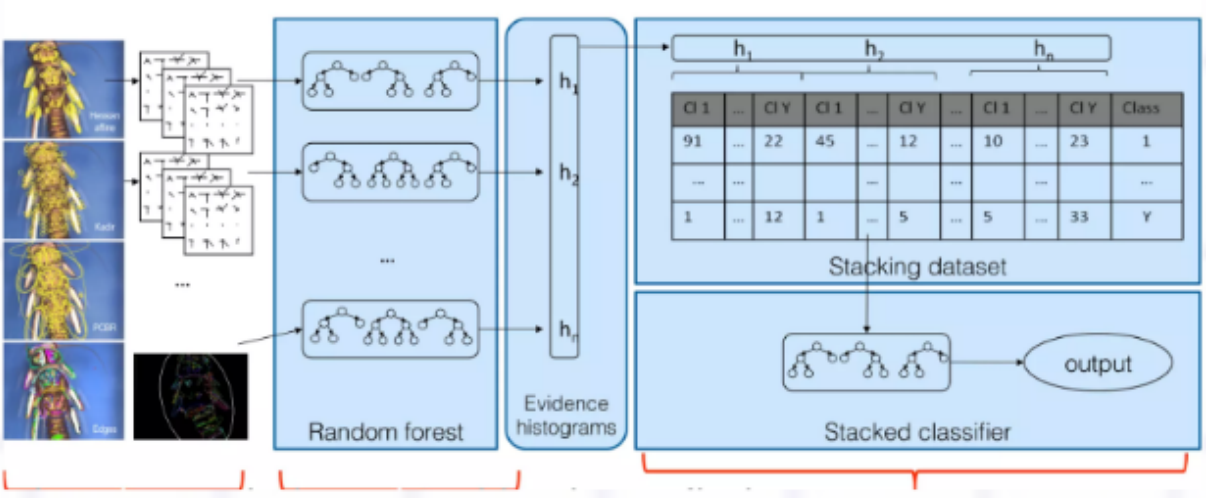
Julia Lenc

Stacking

Stacking is an advanced ensemble method that **combines the strengths** of both **bagging and boosting**. The key idea is to use multiple different models (like decision trees, random forests, SVMs, etc.), then combine their outputs using another model called a "meta-learner"

How does it work?

- **Stage 1: Several different models** (the "base models") are trained in parallel on the same dataset. For each data point, each model makes a prediction.
- **Stage 2:** The predictions from all the base models are collected and become the inputs for a new model - **the "meta-learner".** It learns how to best combine the base models' predictions for the final output.



Picture from: Gonzalo Martiney Munoz

Julia Lenc

How to :
set up
evaluate
tune
interpret

Julia Lenc

Stage 1: business question

Understand and formulate the business question as:

Specific: Define the problem clearly

- What is the business metric?
- What do we want to predict?

Measurable: Determine evaluation criteria (e.g., RMSE).

Achievable: Ensure data and resources match modeling goals.

Relevant: Validate that action will be taken (e.g., pricing)

Time-bound: Identify stakeholders and key deadlines.

Julia Lenc

Stage 2: data preparation

Step	Ensembles
Data cleansing	✅
Labeling	✅
Addressing imbalance	critical for Boosting
Transformation	if needed
Encoding	if needed
Feature selection	recommended
Dimensionality reduction	✅
Learn how	click here

The process: Stage 3

EDA

What to check?

1. Strong Predictors (Signal)

- Identify features with clear relationship to your target.
- Visualize (for classification): boxplots, violin plots, scatterplots by class.
- Visualize (for regression): scatterplots, line plots, correlation coefficients.

2. High-Cardinality Categorical Features

- Too many unique categories can create a flood of features.

  Consider combining rare categories or selecting the most informative levels.

3. Outliers

- Extreme values, although Ensembles tolerate them, can influence predictions.

4. Remove or combine very similar features (strong correlations).

5. Double check for class imbalance and feature noise - critical for Boosting!

Julia Lenc

EDA

EDA with pandas, numpy, matplotlib, seaborn, scikit-learn

1. Identify Strong Predictors (Signal)

Visualize feature-target relationships

- boxplot(), violinplot() ← seaborn, matplotlib
- scatterplot() ← seaborn, matplotlib
- corr(), corrplot() ← pandas, seaborn

2. Explore High-Cardinality Categorical Features

Check unique values

- nunique(), value_counts() ← pandas

Visualize categorical split

- countplot(), barplot() ← seaborn, matplotlib

3. Detect Outliers

Identify via visualization

- boxplot(), swarmplot() ← seaborn, matplotlib

Julia Lenc

EDA

EDA with pandas, numpy, matplotlib, seaborn, scikit-learn

4. Remove/Combine Highly Correlated Features (if needed)

Correlation matrix

- corr() ← pandas
- heatmap() ← seaborn

5. Double-check Class Imbalance & Feature Noise (Critical for Boosting)

Class imbalance

- value_counts() ← pandas
- Counter ← collections
- plot() ← matplotlib, seaborn

Noise, redundancy

- describe(), isnull().sum() ← pandas
- duplicated() ← pandas

Julia Lenc

Stage 4: Model set up

1. Model choice (use only if Decision Trees or SVM/SVR failed)

Try Bagging (Random Forest) if

- your goal is to reduce variance of unstable models (e.g., Decision Trees)
- your data is noisy
- use: RandomForestClassifier / RandomForestRegressor (scikit-learn)

Try Boosting if

- your goal is to reduce bias (simpler models underfit)
- AdaBoost is the 1st choice for fairly clean problems, small & medium dataset
- Gradient Boosting is the choice for larger datasets (faster) and mixed data
- XGBoost is the most advanced, for complex problems and Kaggle 🙂
- use: AdaBoostClassifier / HistGradientBoostingClassifier (scikit-learn), XGBClassifier (xgboost)

2. Key parameters for an initial run

All: n_estimators (start at 100), random_state

Random Forest: max_depth (try shallow/None), max_features

Boosting: learning_rate (start at 0.1), max_depth (start low, e.g. 1-3)

Use: scikit-learn, xgboost

Julia Lenc

The process: Stage 5

Training. Evaluation (classification)

Confusion matrix

	Predicted positives	Predicted negatives
Actual positives	True positives (TP)	False negatives (FN)
Actual negatives	False positives (FP)	True negatives (TN)

Core evaluations metrics

Accuracy: overall correctness. (TP + TN) / total predictions

Precision: % correctly predicted positives among all positives. TP / (TP + FP)
Critical if the cost of False Positive is high. Examples: insurance company pays false claim, bank gives a loan to a customer with bad records, cybersecurity (false alarms disrupt operations).

Recall: % correctly predicted positives among all corrects. TP / (TP + FN)
Critical if capturing Positives is more important than avoiding False Positives. Examples: fraudulent transactions detection, retail recommender system.

F1 score: model's performance. 2*(Precision * Recall) / (Precision + Recall)

The process: Stage 5

Training. Evaluation (regression)

n = number of data points

MAE (Mean Absolute Error): $\text{MAE} = \frac{1}{n}\sum_{i-1}^{n}|y_{\text{true},i} - y_{\text{pred},i}|$ Average absolute difference between the predicted and actual values. The most interpretable metric. Use when errors are acceptable as long as they cancel each other. Example: house prices.

MSE (Mean Squared Error): $\text{MSE} = \frac{1}{n}\sum_{i-1}^{n}(y_{\text{true},i} - y_{\text{pred},i})^2$ Average of the squared difference between the actual and predicted values. Penalizes for large errors. Use when relationship is deterministic, precision is critical, larger errors significantly impact conclusions. Example: R&D, engineering.

RMSE (Root Mean Squared Error): $\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i-1}^{n}(y_{\text{true},i} - y_{\text{pred},i})^2}$ like MSE, but expressed in the same units as the target variable. Use when larger errors must be penalized because they lead to system collapse. Example: supply chain, electricity demand forecasting.

DO NOT USE MAPE (Mean Average Percentage Error)!

If your data contains zeros or very small values (consider WMAPE or other alternatives).

Training and evaluation

1. Split the data into training and test sets

train_test_split ← sklearn.model_selection

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True)
```

2. Model training

fit() ← any sklearn estimator

```
model.fit(X_train, y_train)
```

3. Model prediction

predict() ← any sklearn estimator

```
y_pred = model.predict(X_test)
```

4. Model evaluation

Classification: accuracy_score, precision_score, recall_score, f1_score, confusion_matrix ← sklearn.metrics

Regression: mean_absolute_error, mean_squared_error, r2_score ← sklearn.metrics

```
from sklearn.metrics import mean_absolute_error, accuracy_score, ...
```

5. Overfitting / Underfitting signs

Overfitting: Train error <<<< Test error

Underfitting: Both errors high

***for XGBoost it's not scikit-learn but xgboost**

Julia Lenc

Stage 6: tuning

1. Hyperparameters tuning

1a. Manual tuning set via model initiation:

- n_estimators (number of trees/boosting rounds)
- max_depth (tree depth, controls complexity)
- learning_rate (step size for boosting, XGBoost/GBDT)
- Optional: min_samples_split, subsample, etc.

How to?

```
RandomForestClassifier(n_estimators=100, max_depth=3)

XGBClassifier(n_estimators=100, max_depth=3, learning_rate=0.1)
```

1b. Grid search for best parameters with GridSearch CV, e.g.:

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV


model = XGBClassifier()
param_grid = {'n_estimators': [50, 100], 'max_depth': [3, 6]}
grid = GridSearchCV(model, param_grid, cv=3)
grid.fit(X_train, y_train)
```

Julia Lenc

Stage 6: tuning

2. Overfitting remedies

- **Simpler models**:

  lower max_depth, fewer trees (n_estimators)

- **Feature selection**

  use model.feature_importances_ to extract features from model

- **Cross-validation** using GridSearchCV for parameter search

- **Early stopping, where available** (XGBoost, LightGBM, CatBoost, etc.)

Stage 7: interpretation

1. Feature importances

- **What it tells us:** which features are the most influential for predictions.
- **How to use:** identify key drivers, drop unimportant features, validate using domain knowledge.
- **How to set:** use built-in (e.g., .feature_importances_)

2. SHAP

- **What it tells us**: Not just which features matter, but how each feature pushes a prediction up or down for each example.
- **Global understanding**: aggregate SHAP reveals widespread drivers and their typical effect (e.g., positive vs. negative impact).
- **Local understanding**: for a single prediction, see exactly which features caused that result.
- **How to use:** explain individual decisions, debug unexpected results, ensure fairness.
- **How to set:** use shap.TreeExplainer(model).

3. Other methods:

- **Partial Dependence / ICE Plots** - how predictions change with features
- **Permutation Importance**: more reliable feature relevance measure
- **LIME**: Single-prediction explanations, model-agnostic

Stage 8: talk to the business

1. Talk decisions, NOT methods

- **Emphasize outputs.** "This model tells you which customers are most likely to churn" (NOT: ensembles are more advanced than SVM).
- **Use analogies, if asked how Ensembles work.** "Ensembles work like a panel of experts pooling their opinions to give a stronger answer".

BUT always **know what went into the model and why** you chose Ensembles. Especially Finance may get very curious. Be prepared!

2. Fully understand the business question, KPIs and language

- **Finance (banking)**: credit risk, fraud detection, P&L, ROI
- **Finance (manufacturing)**: P&L, ROI, pricing optimization, forecast
- **Marketing examples:** targeting, propensity score, segmentation, churn
- **Product supply:** inventory, stockout, overstock, out-of-stock

3. Demistify black box:

- **Explain trade-off and cosequences:** performance vs intepretability and make sure stakeholders are ready to compromise. Most importantly, make sure yourself that **Ensembles were chosen for the reason.**
- **Explain:** use the outputs of feature importances, SHAP, etc.

4. Use past success cases if available, e.g. conversion improvement, churn decrease, higher loan repayment.

Julia Lenc

Did you find it useful?

Save

Share

Follow

Analytics, Market Research, Machine Learning and AI

Julia Lenc