Introduction to Machine Learning

ML-Basics: Losses & Risk Minimization

HOW TO EVALUATE MODELS

OVERVIEW

No Free Lunch In machine learning, there’s something called the “No Free Lunch” theorem. In a nutshell, it states that no one algorithm works best for every problem, and it’s especially relevant for supervised learning (i.e. predictive modeling).
For example, you can’t say that neural networks are always better than decision trees or vice-versa. There are many factors at play, such as the size and structure of your dataset.
As a result, you should try many different algorithms for your problem, while using a hold-out “test set” of data to evaluate performance and select the winner. Hypothesis space + Risk + Optimization

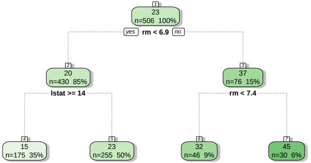
CART FUNCTIONALITY

SUPERVISED NON-PARAMETRIC WHITE-BOX FEATURE SELECTION

General idea Starting from a root node, ***classification & regression trees (CART)*** perform repeated **binary splits** of the data according to feature values, thereby subsequently dividing the input space $\mathcal{X}$ into T **rectangular partitions** Q_t.

Hypothesis space $\mathcal{H} = \left\{ f(\mathbf{x}) : f(\mathbf{x}) = \sum_{t=1}^{T} c_t \mathbb{I}(\mathbf{x} \in Q_t) \right\}$

- Pass observations along until each ends up in exactly one leaf node
- In each step, find the optimal feature-threshold combination to split by
- Assign response c_t to leaf node t



Full tree

CART FUNCTIONALITY

Empirical risk

- Empirical risk is calculated for each potential terminal node $\mathcal{N}_t$ of a split.
- In general, trees can handle any type of loss function. Typical choices are:
  - Classification (for g classes):
    - Using ***Brier score*** $\mathcal{R}(\mathcal{N}_t) = \sum_{(\mathbf{x},y)\in\mathcal{N}_t} \sum_{k=1}^{g} (\mathbb{I}(y = k) - \pi_k(\mathbf{x}))^2$
    - Using ***Bernoulli loss*** $\mathcal{R}(\mathcal{N}_t) = \sum_{(\mathbf{x},y)\in\mathcal{N}_t} \sum_{k=1}^{g} \mathbb{I}(y = k) \cdot \log(\pi_k(\mathbf{x}))$
  - Regression: Using ***quadratic loss*** $\mathcal{R}(\mathcal{N}_t) = \sum_{(\mathbf{x},y)\in\mathcal{N}_t} (y - c_t)^2$

Optimization **Exhaustive** search over all (randomly selected) split candidates in each node to minimize empirical risk in the child nodes (greedy optimization)

Hyperparameters **Complexity**, i.e., number of leaves T
$\rightarrow$ Controlled via tree depth, minimum number of observations per node, maximum number of leaves, minimum risk reduction per split, ...

CART **PRO'S & CON'S**

Advantages

- + **Easy** to understand, interpret & visualize
- + Automatic handling of **non-numerical** features
- + Built-in **feature selection**
- + Automatic handling of **missings**
- + **Interaction** effects between features easily possible, even of higher orders
- + **Fast** computation and good scalability
- + High **flexibility** (custom split criteria or leaf-node prediction rules)

Disadvantages

- - Rather **low accuracy** (at least, without bagging or boosting)
- - High **variance/instability**: strong dependence on training data
- - Therefore, poor generalization & risk of **overfitting**
- - Several steps required for modeling **linear** relationships
- - In presence of categorical features, **bias** towards features with **many categories**

Simple and good with feature selection, but not the best predictor

CART APPLICATION

For applications of CART, note the following:

Pruning / early stopping

Unless interrupted, splitting will go on until each leaf node contains a single observation (expensive + overfitting!)

→ Use **pruning** and **stopping criteria** to limit complexity

Implementation

R: package `rpart`
Python: `DecisionTreeClassifier` / `DecisionTreeRegressor` from package `scikit-learn`

Bagging / boosting

Since CART are instable predictors on their own, they are typically ensembled to form a ***random forest*** (***bagging***) or used in combination with ***boosting***.

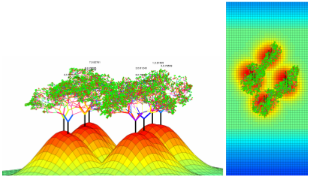
RANDOM FORESTS FUNCTIONALITY

SUPERVISED NON-PARAMETRIC BLACK-BOX FEATURE SELECTION

General idea Random forests are ***bagging ensembles***: they combine multiple CART (base learners) to form a strong learner. They use **complex** trees with low bias and compensate for single trees' high variance by aggregating M of them in a **decorrelated** manner.

Hypothesis space $\mathcal{H} = \left\{ f(\mathbf{x}) : f(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{T^{[m]}} c_t^{[m]} \mathbb{I}(\mathbf{x} \in Q_t^{[m]}) \right\}$

- Training on bootstrap samples of the data and only on a random subset of features to incur variability
- Aggregation via averaging (regression) or majority voting (classification)



RANDOM FORESTS **FUNCTIONALITY**

Empirical risk Applicable with **any** kind of loss function (just like tree base learners)
→ Computation of empirical risk for all potential child nodes in all trees

Optimization **Exhaustive** search over all (randomly selected) split candidates in each node of each tree to minimize empirical risk in the child nodes (greedy optimization)

Hyperparameters

- **Ensemble size**, i.e., number of trees
- **Complexity**, i.e., number of leaves T of each base learner
  → Controlled via tree depth, minimum number of observations per node, maximum number of leaves, minimum risk reduction per split, ...
- **Number of split candidates**, i.e., number of features to be considered as splitting variables at each split
  → Frequently used heuristics: $\lfloor \sqrt{p} \rfloor$ for classification and $\lfloor p/3 \rfloor$ for regression

RANDOM FORESTS PRO'S & CON'S

Advantages

+ Translation of most advantages of **trees** (e.g., inherent variable selection, handling of missing data)
+ Fairly good at **prediction**: improved accuracy through bagging
+ Inherent computation of **feature importance**
+ Quite **stable** wrt changes in the data
+ Good with **high-dimensional** data, even in presence of noisy covariates
+ Applicable to **unbalanced** data
+ Easy to **parallelize**
+ Rather easy to **tune**

Disadvantages

- Loss of single trees' **interpretability** – black-box method
- Hard to **visualize**
- Often suboptimal for **regression**
- Computationally demanding – both in **runtime** and **memory**
- Often still inferior in **performance** to other methods (e.g., boosting)
- Overfitting?

Fairly good predictor, but black-box method

RANDOM FORESTS APPLICATION

For applications of CART, note the following:

Pruning / early stopping

Unless interrupted, splitting will go on until each leaf node contains a single observation (expensive + overfitting!)

→ Use **pruning** and **stopping criteria** to limit complexity.

Implementation

R: package `ranger`
Python: `RandomForestClassifier` / `RandomForestRegressor` from package `scikit-learn`

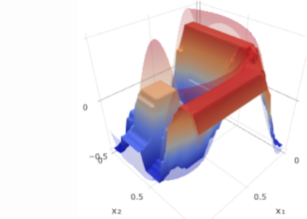
GRADIENT BOOSTING **FUNCTIONALITY**

SUPERVISED NON-PARAMETRIC BLACK-BOX FEATURE SELECTION

General idea Gradient boosting (GB) is an ***ensemble*** method that constructs a strong learner from weak base learners (frequently, CART).
As opposed to ***bagging***, however, base learners are assembled in a **sequential, stage-wise** manner: in each iteration, GB improves the current model by adding a new component that minimizes empirical risk. The final model is a weighted sum of base learners $b(\mathbf{x}, \theta^{[m]})$ with weights $\beta^{[m]}$.

Hypothesis space $\mathcal{H} = \left\{ f(\mathbf{x}) : f(\mathbf{x}) = \sum_{m=1}^{M} \beta^{[m]} b(\mathbf{x}, \theta^{[m]}) \right\}$

- Finding next additive component $\widehat{=}$ fitting base learner to current point-wise residuals
- One boosting iteration $\widehat{=}$ one approximate gradient step in function space
- Fitting of each base learner using information from previously added ones



GRADIENT BOOSTING **FUNCTIONALITY**

Empirical risk

- **Outer loss:** Loss used to compute pseudo-residuals – how large is the error of the current model fit?
  → Arbitrary **differentiable** loss function
- **Inner loss:** Loss used to fit next base learner component to current pseudo-residuals → Typically, ***quadratic loss*** (desirable optimization properties)

Optimization ***Functional gradient descent*** for outer, **arbitrary** procedure for inner optimization loop (depending on inner loss)

Hyperparameters

- **Ensemble size**, i.e., number of base learners
- **Learning rate**, i.e., impact of single base learner
- **Complexity** of base learners (depending on type used)