Ensemble methods

Victor Kitov

v.v.kitov@yandex.ru

Yandex School of Data Analysis

Ensemble learning

Definition 1

Ensemble learning - using multiple machine learning methods for a given problem and combining their outputs to obtain final result.

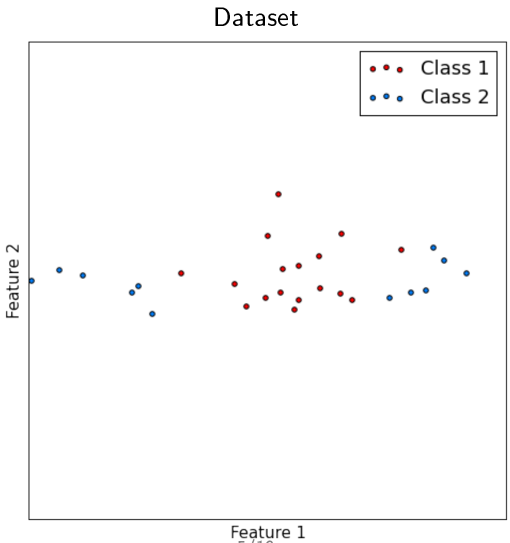
Synonyms: committee-based learning, multiple classifier systems.

Motivation of ensembles

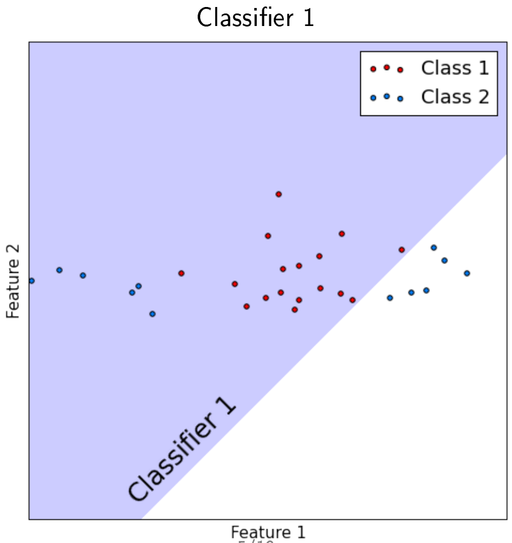
- Benefits for prediction:
  - increased accuracy
  - increased robustness.
- Justification: some predictors are compensating the errors of other predictors
- When to use:
  - existing model hypothesis space is too narrow to explain the true one (high model bias)
  - avoid local optima of optimization methods (high model variance)
  - too small dataset to figure out concretely the exact model hypothesis
- Frequently the task itself promotes usage of ensembles (such as computer security):
  - multiple sources of diverse information
  - different abstraction levels need to be united

Table of Contents

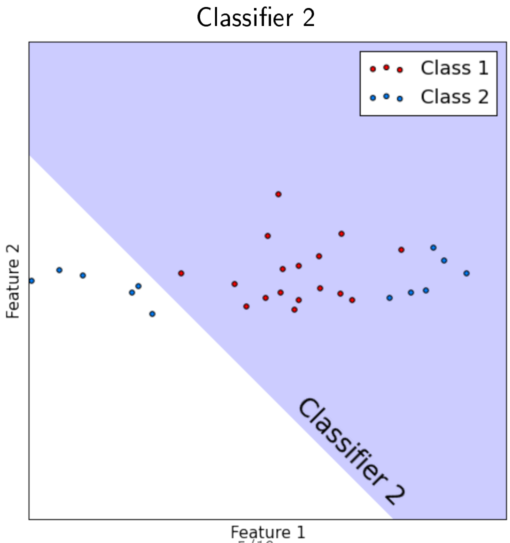
Motivation for classification



Motivation for classification

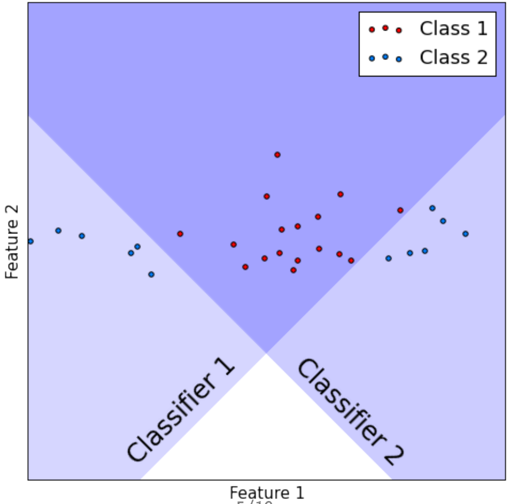


Motivation for classification

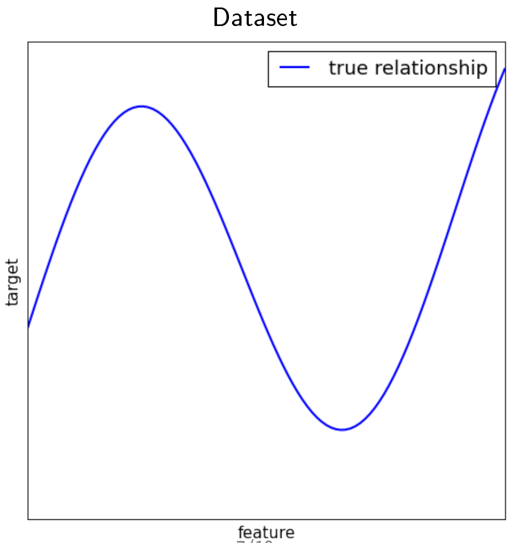


Motivation for classification

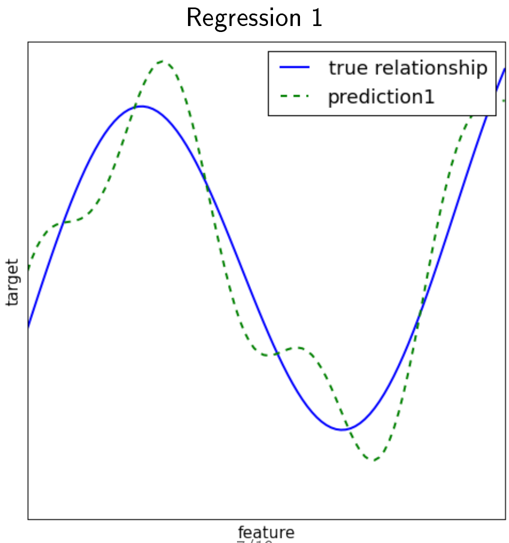
Classifier 1 and classifier 2 combined using AND rule



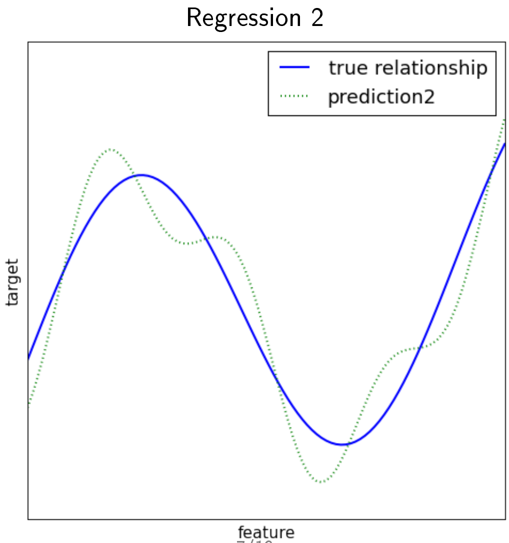
Motivation for regression



Motivation for regression



Motivation for regression



Motivation for regression

Regression 1 and regression 2 combined using averaging

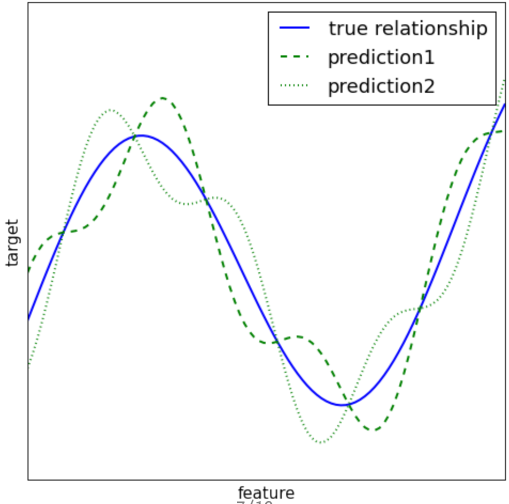


Table of Contents

2 Popular ensemble methods
- Bagging and random forest
- Fixed integration schemes for classification

Bagging & random subspaces

- Bagging
  - random selection of samples (with replacement)
  - *what is the probability that observation will not belong to bootstrap sample?*
  - *what is the limit of this probability with $N \to \infty$?*
- Random subspace method:
  - random selection of features (without replacement)
- We can apply both methods jointly

Random forests

Input: training dataset $TDS = \{(x_i, y_i), 1 = 1, 2, ...N\}$; the number of trees B and the size of feature subsets m.

1. for $b = 1, 2, ...B$:
    1. generate random training dataset TDS^b of size N by sampling (x_i, y_i) pairs from TDS with replacement.
    2. build a tree using TDS^b training dataset with feature selection for each node from random subset of features of size m (generated individually for each node) without replacement.
2. Evaluate the quality by assigning output to x_i, $i = 1, 2, ...n$ using majority vote (classification) or averaging (regression) among trees with $b \in \{b : (x_i, y_i) \notin T^b\}$

Output: B trees. Classification is done using majority vote and regression using averaging of B outputs.

Comments

- Random forests use random selection on both samples and features
- Left out samples are used for evaluation of model performance.
- Less interpretable than individual trees
- +: Parallel implementation
- -: different trees are not targeted to correct mistakes of each other
- Extra-Random trees: more bias and less variance by random sampling (feature,value) pairs.

Fixed combiner at class level

Output of base learner k

Exact class: ω_1 or ω_2.

Combiner predicts ω_1 if:

- all classifiers predict ω_1 (AND rule)
- at least one classifier predicts ω_1 (OR rule)
- at least k classifiers predict ω_1 (k-out-of-N)
- majority of classifiers predict ω_1 (majority vote)

Each classifier may be assigned a weight, based on its performance:

- weighted majority vote
- weighted k-out-of-N (based on score sum)

Fixed combiner - ranking level

Output of base learner k

Ranking of classes:

$$\omega_{k_1} \succeq \omega_{k_2} \succeq \ldots \succeq \omega_{k_C}$$

Ranking is equivalent to scoring of each class (with incomparable scoring between classifiers).

Definition 2

Let $B_k(i)$ be the count of classes scored below ω_i by classifier k. **Borda count $B(i)$ of class ω_i** is the total number of classes scored below ω_i by all classifiers:

$$B(i) = \sum_{k=1}^{K} B_k(i)$$

Combiner predicts ω_i where $i = \arg\max_i B(i)$

Fixed combiner at class probability level

Output of base learner k

Vectors of class probabilities:

$$[p^k(\omega_1),\ p^k(\omega_2),\ \dots\ p^k(\omega_C)]$$

Combiner predicts ω_i if $i = \arg\max_i F(p^1(\omega_i), p^2(\omega_i), \dots p^K(\omega_i))$

- F = mean or median.

Finding constant weights

Weighted averaging combiner

$$f(x) = \sum_{k=1}^{K} w_k f_k(x)$$

Naive fitting

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{N} \mathcal{L}(y_i, \sum_{k=1}^{K} w_k f_k(x_i))$$

will overfit. The mostly overfitted method will get the most weight.

Linear stacking

- Let training set $\{(x_i, y_i), i = 1, 2, ...N\}$ be split into M folds.
- Denote $fold(i)$ to be the fold, containing observation i
- Denote $f_k^{-fold(i)}$ be predictor k trained on all folds, except $fold(i)$.

Definition

Linear stacking (or stacked generalization) is weighted averaging combiner, where weights are found using

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{N} \mathcal{L}(y_i, \sum_{k=1}^{K} w_k f_k^{-fold(i)}(x_i))$$

Generalized stacking

Definition

Generalized stacking is prediction

$$f(x) = A_\theta \left(f_1(x), f_2(x), \dots f_K(x)\right),$$

where A is some general form predictor and θ is a vector of parameters, estimated by

$$\widehat{\theta} = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}\left(y_i,\ A_\theta\left(f_1^{-fold(i)}(x), f_2^{-fold(i)}(x), \dots f_K^{-fold(i)}(x)\right)\right)$$

- Stacking is the most general approach
- It is a winning strategy in most ML competitions.
- $f_i(x)$ may be:
  - class number (coded using one-hot encoding).
  - vector of class probabilities
  - any initial or generated feature