Machine Learning I

Chapter 08 - Ensemble Learning, Random Forests, Boosting

Prof. Dr. Sandra Eisenreich

December 14 2023

Hochschule Landshut

Introduction

Ensemble learning is not a model itself, but a method to increase the performance of several estimators (for all kinds of prediction tasks).

Basic idea:

- Bagging/Pasting: like using the "Publikumsjoker" at "Wer wird Millionär": You train a set of several different estimators on the same task and let them make predictions, then you take a combination of the individual predictions. This way the final prediction is much better than each individual prediciton.
- Boosting: Like answering a mulitple choice questionnaire with a bunch of people in turns: A answers all questions, then B tries to correct A's mistakes, C tries to correct B's remaining mistakes etc...

Ensembles and Bias-Variance Tradeoff

Why does it work?

- Bagging/Pasting - ? variance - ? bias
- Boosting - ? bias -? variance

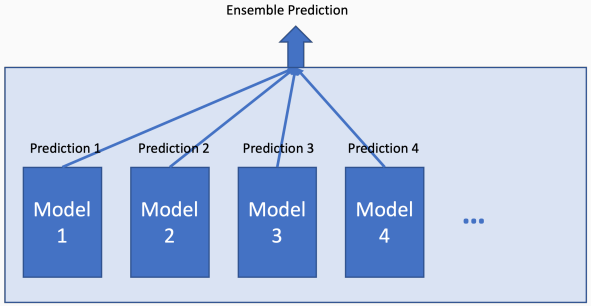
Ensemble Learning: Bagging, Pasting and Randomness

Variants and when to use?

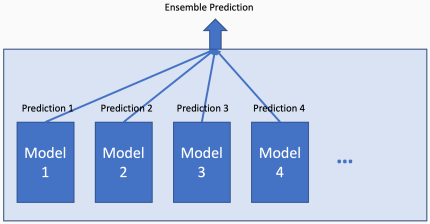
Variants:

- use different types of models to combine different views
- use the same type of model (e.g. trees) with high variance, i.e. small changes in the training data change the model substantially, and only change the training set slightly.

Advantage: Parallel training of models is possible!



Aggregation methods



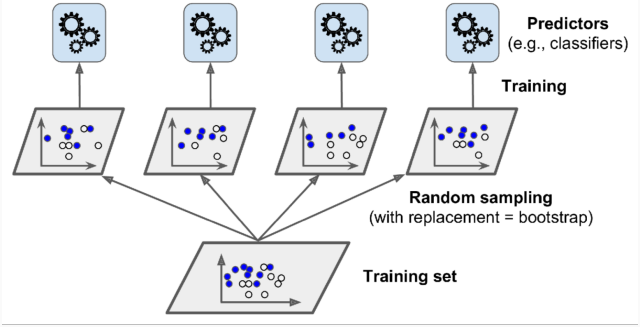
How do you aggregate the individual predictions of the estimators?

For Classification: ?

For Regression: ?

Bagging and Pasting

- Sampling with replacement is called bootstrap sampling.
- An ensemble of predictors trained with the same training algorithm on different training datasets generated from a common dataset using bootstrap sampling is called bagging (short for bootstrap aggregating).
- For sampling without replacement, training an ensemble like above is called pasting.



Bagging with Scikit-Learn

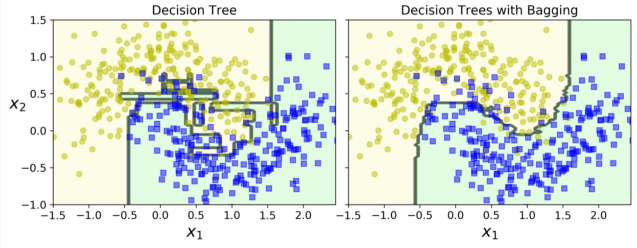
Scikit-Learn offers the `BaggingClassifier` API for bagging and pasting and the `BaggingRegressor` API for regression. The BaggingClassifier automatically performes soft voting if the base classifier can estimate probabilities.

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf=BaggingClassifier(
        DecisionTreeClassifier(), n_estimators=500,
        max_samples=100, bootstrap=True)
bag_clf.fit(X_train, y_train)
```

Bagging/pasting vs. single model

A single Decision Tree (left) versus a bagging ensemble of 500 trees (=Random Forest)(right)



Bagging vs. Pasting

Bagging is more frequently used than Pasting:

- $\oplus$: greater diversity in the training sets (samples can occur several times) than Pasting $\rightarrow$ better generalization, usually better results
- $\ominus$: each predictor only sees about 63% (*) of the training samples (possibly repeatedly).
- $\oplus$: one can use the unseen samples=out-of-bag (oob) instances, which are different 37% for each predictor, for testing. (set `oob_score=True` in Scikit-Learn and call it via `.oob_score_`)

Reason for (*): ?

Random Patches and Random Subspaces

Instead of sampling instances to get different training sets for the same model, one could also create different training sets by not sampling instances, but instead?

- Random patches method = sampling ?
- Random subspaces method= sampling ?

Feature sampling with Scikit-Learn:

- either set `bootstrap_features` to True, or:
- set `max_features` value less than 1.0 (the fraction of features considered for a split at each node).

Random Forests

Overview

Remember the advantages of trees:

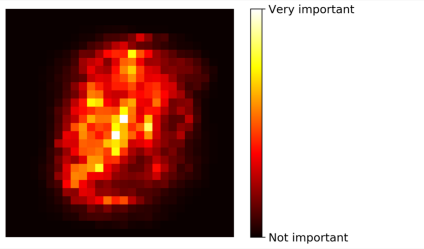
- easy and fast to train, quick inference
- feature selection
- works on regression and classification
- flexible
- explainable

Disadvantage: high variance, tendency to overfit. $\Rightarrow$ perfect for bagging/pasting ensemble: gets rid of the disadvantage while keeping most advantages (you only lose explainability).

A Random Forest is an ensemble of Decision Trees, trained via the bagging (or sometimes pasting) method.

Feature Importance

The feature importance of a random forest is given by the average of feature importances of the trees in the forest $\text{Importance}_i = \sum_T \text{Importance}_i(T)$.



Imporance of the individual pixels in the MNIST classification example.

Random Forests with Scikit-Learn

To get a Random Forest, you can use BaggingClassifier (typically with `max_samples`= size of the training set, because you already only see 63% of data) with a DecisionTreeClassifier. Better: Use RandomForestClassifier class, which is optimized for DecisionTrees.

Example:

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes = 16, n_jobs=-1)
rnd_clf.fit(X_train, y_train)
```

Feature importance: Scikit-Learn computes this automatically for each feature after training and scales the results so that the sum of all importances is equal to 1. Access the result using the `feature_imortances_` variable. Access the probability vector using `.predict_proba()`.

Boosting

Idea and when to use

Idea: each predictor in the ensemble is trained *sequentially to learn from (and correct!) the mistakes of the previous predictors*. This way, the ensemble can boost the performance of each individual predictor even more and turn weak learners into strong learners.

Advantage: ?

Disadvantage: ?

There are different ways of boosting, like AdaBoost and Gradient Boosting.

General principle: forward stagewise additive modeling

Suppose we want to train an estimator (usually a decision tree!) with data $(\mathbf{x}^{(i)}, y^{(i)})$ using a boosting method:

- First, we train one estimator F_1.
- Then, at each stage $1 \leq k$, we improve our estimator F_k by adding a new estimator h_k which improves the mistakes F_k makes:

$$F_{k+1} := F_k + \beta_k h_k.$$

How do we choose h_k?
?
This process is called forward stagewise additive modeling.

Example: Forward stagewise additive modeling with MSE

Now suppose we have a regression task and L=MSE, and $\beta_k = 1$:

$$\text{MSE} \quad = \quad ?$$

where $\text{res}_k^{(i)} = y^{(i)} - F_k(\mathbf{x}^{(i)})$ is the residual (i.e. the error) of F_k.
Training h to minimize MSE means that h is trained to fit ?

$\Rightarrow$ Forward stagewise additive modeling iteratively fits the "correction" h_k to the residuals.

Least Squares Boosting for Regression

Forward stagewise additive modeling with MSE cost function for a regression task is called Least Squares Boosting: it iteratively trains predictors h_k by minimizing the loss function

$$\frac{1}{m}\sum_{i=1}^{m}(\text{res}_k^{(i)} - h_k(\mathbf{x}^{(i)}))^2$$

as follows:

- Train a first predictor F_1 (e.g. a tree or any regression predictor) with the MSE loss function.
- For $k = 1, 2, \ldots,$ number-of-iterations:
  - train a predictor h_k of the same kind to minimize the new loss function: $\frac{1}{m}\sum_{i=1}^{m}(\text{res}_k^{(i)} - h_k(\mathbf{x}^{(i)}))^2$
  - compute the new predictions $F_{k+1}(\mathbf{x}^{(i)}) = (F_k + h_k)(\mathbf{x}^{(i)})$ and residuals $\text{res}_{k+1}^{(i)} := y^{(i)} - F_{k+1}(\mathbf{x}^{(i)})$

Motivation: Gradient Boosting

Question: How can we generalize Least Squares Boosting to other loss functions L than MSE?

Idea: Describe the loss function for the "correction functions" h_k in general terms of the loss function L for Least Squares Boosting and generalize this.

How? Note the following: If we compute the derivative of the MSE-loss function for the kth predictor F_k with respect to $\hat{y}^{(i)} := F_k(\mathbf{x}^{(i)})$, we get:

$$\partial_{\hat{y}^{(i)}} \text{MSE}(y^{(i)}, \hat{y}^{(i)}) \quad = \quad ?$$

$\Rightarrow$ fitting h_k to the residuals $\text{res}_k^{(i)}$ = fitting it to ?

Now we just replace the cost function MSE with other cost functions to get a generalization. This is the so-called Gradient Boosting.

Gradient Boosting in general

Let L be a loss function (e.g. MSE, for regression or classification). Gradient Boosting for loss L is forward stagewise additive modeling

$$F_k = \sum_{i=1}^{k} \beta_i h_i,$$

where the first estimator $h_1 = F_1$ is trained to minimize L, and each subsequent h_{k+1} is trained minimize the loss:

$$\tilde{L}_{k+1} := \sum_{i=1}^{m} \left(h_k(\mathbf{x}^{(i)}) - g_k^{(i)} \right), \text{ where } g_k^{(i)} = \partial_{F_k(\mathbf{x}^{(i)})} L(y^{(i)}, F_k(\mathbf{x}^{(i)}))$$

In other words: Each subsequent weak learner is trained to fit the gradients of the loss of the predictor up to now.

XGBoost

Note: In practice, one almost always takes trees as the underlying weak learners. This is called gradient tree boosting or gradient boosted tree.

The most frequently used boosting algorithm these days is XGBoost, it is among the best performing models on Kaggle for various competitions.

XGBoost (Extreme Gradient Boosting) is a regularized version of Gradient Tree Boosting: At each optimization step k, add a regularization term

$$\Omega_k = \gamma J_{k-1} + \frac{1}{2}\lambda \sum_{i=1}^{J_{k-1}} w_k^{(i)}$$

where J_k is the number of leaves of the tree to be trained, and $w_k^{(i)}$ is the predicted output for its ith leaf, regularizing the number of leaves and the size of the outputs.

Gradient Boosting with Scikit-Learn

```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradienBoostingRegressor()
gbrt.fit(X,y)
```

XGBoost is avaliable in the python library XGBoost:

```
import xgboost

xgb_reg = xgboost.XGBRegressor()
xgb_reg.fit(X_train, y_train)
y_pred = xgb_reg.predict(X_val)
```

Hyperparameters for Gradient Boosting

Scikit-Learn's GradientBoostingRegressor class (using decision trees as base estimators) has the following hyperparameters:

- Hyperparameters to control the growth of Decision Trees (eg. `max_depth`, `min_samples_leaf` and hyperparameters to control ensemble training, e.g. number of trees (`n_estimators`)
- The `learning_rate` hyperparameter scales β. If the value is low, you will need more trees in the ensemble to fit the training set, but the predictions will usually generalize better.
- the `subsample` hyperparameter specifies the fraction of training instances to be used for training each tree, eg if $= 25\%$, then each tree is trained on a randomly selected 25 percent of the training instances. speeds up trainig. This is called Stochastic Gradient Boosting.
- The `loss` hyperparameter can change the cost function.

The original boosting algorithm: AdaBoost for binary classification

A different algorithm (not Gradient Boosting) was the first boosting algorithm:

AdaBoost is forward stagewise additive modeling

$$F_{k+1} = F_k + \beta_k h_k$$

for binary classification (with labels $y^{(i)} \in \{-1, 1\}$) with the loss function:

$$L_{k+1} = \sum_i \exp\left(-y^{(i)} F_{k+1}(\mathbf{x}^{(i)})\right) :$$

If an instance $\mathbf{x}^{(i)}$ is classified correctly, $y^{(i)} F_{k+1}(\mathbf{x}^{(i)}) = 1$, and the summand for instance i is $= e^{-1}$; if it is misclassified, the summand for instance i is $= e > e^{-1}$, so minimizing this loss makes sure instances are classified correctly.

It is one of the most popular and powerful models for binary classification!

Complexity of ensemble methods

- **Bagging and Pasting**: ?
- **Random Forest:** The complexity of one CART in the ensemble is $O(d \cdot n \cdot m \log_2 m)$ (d= maximal depth), if you sample $\tilde{n}$ of the n features for each tree (e.g. random patches or subspaces method), it is $O(\tilde{n} \cdot m \log_2 m)$. The complexity of a Random Forest is ?
- **Boosting:** ?

AdaBoost and SAMME with Scikit-Learn

AdaBoost for binary classification with Scikit-Learn:

```
from sklearn.ensemble import AdaBoostClassifier


ada_clf = AdaBoostClassifier(
        DecisionTreeClassifier(max_depth=1), n_estimators=200,
        learning_rate=0.5)
ada_clf.fit(X_train, y_train)
```

There is a multiclass classification version of AdaBoost called SAMME. To use SAMME, modify the above code simply by choosing the hyperparameter `algorithm="SAMME.R"` in the AdaBoostClassifier.

Chapter 08 Summary

Chapter 08 Summary - Pasting and Bagging, Random Forests

- ML Ensembles are a set of different estimators whose predictions are aggregated for a final prediction as follows:
  - for hard classification use a majority-vote (hard voting)
  - for soft classification take the mean of the predicted probability vectors ()soft voting)
  - for Regression: take the mean of the outputs.
- Instead of using different models in an ensemble, one could use one model and train it several times on different training sets. These different training sets can be gotten from the original training set by
  - sampling without replacement $\Rightarrow$ pasting
  - bootstrap sampling: Sampling with replacement $\Rightarrow$ bagging Advantage: leads to diversity in training sets and often better estimators; Disadvantage: each estimator only sees around 63% of training data. However, out-of-bag (oob) instances can be used for testing.
  - Random subspaces method= taking all instances, but taking only a subset of the features
  - Random patches method = sampling both features and instances
- A Random Forest is an ensemble of Decision Trees, trained via the bagging (or sometimes pasting) method. It can give feature importance as the average of feature importances for the individual trees.

Chapter 08 Summary - Boosting

- Boosting is an ensemble method, where you train each predictor in the ensemble *sequentially to learn from (and correct!) the mistakes of the previous predictors*. It uses forward stagewise additive modeling:
-
  - First, we train one estimator F_1.
  - Then, at each stage $1 \leq k$, we improve our estimator F_k by adding a new estimator h_k which improves the mistakes F_k makes:

$$F_{k+1} := F_k + \beta_k h_k.$$

- Forward stagewise additive modeling with MSE cost function is called Least Squares Boosting: it iteratively trains models h_k by minimizing

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^{m} (\text{res}_k^{(i)} - h_k(\mathbf{x}^{(i)}))^2$$

  A generalization of Least Squares Boosting is Gradient Boosting.

Chapter 08 Summary - Gradient Boosting

- Let L be a loss function (e.g. MSE, for regression or classification). Gradient Boosting for loss L is forward stagewise additive modeling, where the first estimator F_1 is trained to minimize L, and each subsequent h_{k+1} is trained minimize the loss:

$$\tilde{L}_{k+1} := \sum_{i=1}^{m} \left(h_k(\mathbf{x}^{(i)}) - g_k^{(i)} \right), \text{ where } g_k^{(i)} = \partial_{F_k(\mathbf{x}^{(i)})} L(y^{(i)}, F_k(\mathbf{x}^{(i)}))$$

  In other words: Each subsequent weak learner is trained to fit the gradients of the loss of the predictor up to now.

- XGBoost (Extreme Gradient Boosting) is a regularized version of Gradient Tree Boosting: At each optimization step k, add a regularization term

$$\Omega_k = \gamma J_{k-1} + \frac{1}{2} \lambda \sum_{i=1}^{J_{k-1}} w_k^{(i)}$$

  where J_k is the number of leaves of the tree to be trained, and $w_k^{(i)}$ is the predicted output for its ith leaf, regularizing the number of leaves and the size of the outputs.

Chapter 08 Summary - AdaBoost

- AdaBoost is forward stagewise additive modeling

$$F_{k+1} = F_k + \beta_k h_k$$

for binary classification (with labels $y^{(i)} \in \{-1, 1\}$) with the loss function:

$$L_{k+1} = \sum_i \exp\Big(-y^{(i)} F_{k+1}(\mathbf{x}^{(i)})\Big)$$

SAMME is a generalization of AdaBoost for multiclass classification.