MADS-ML – Machine Learning

Ensemble Learning

Prof. Dr. Stephan Doerfel

FACHHOCHSCHULE KIEL
University of Applied Sciences

Moodle ()

Outline

Motivation

Voting

Bagging

Boosting

Why ensembles?

Why ensembles?

Idea Combine a set of weak learners into a stronger one.

Why ensembles?

Idea Combine a set of weak learners into a stronger one.

- Create a meta classifier.

Why ensembles?

Idea Combine a set of weak learners into a stronger one.

- Create a meta classifier.
- Use a strategy to combine the predictions of the base learners into a single (final) classification.

Outline

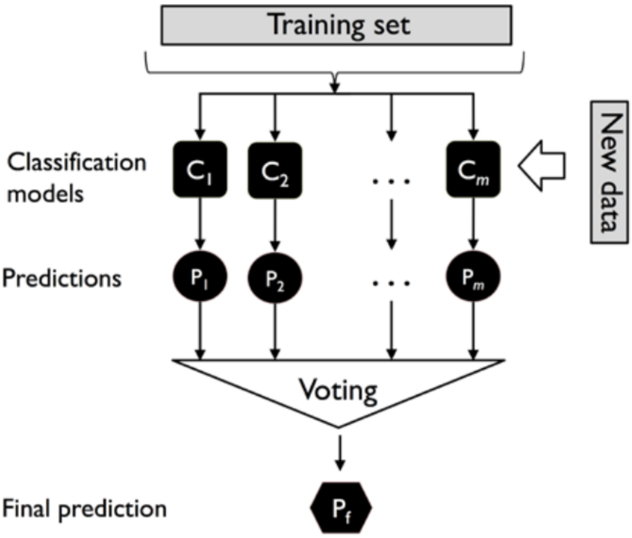
Motivation

Voting

Bagging

Boosting

Voting Schemes



Source: [1], p. 220

Voting Schemes

In Python: `sklearn.ensemble.VotingClassifier`

Voting Schemes

In Python: `sklearn.ensemble.VotingClassifier`

Hard voting (plurality voting)

Voting Schemes

In Python: `sklearn.ensemble.VotingClassifier`

Hard voting (plurality voting)

- Each classifier votes for the class it predicted with weight 1

Voting Schemes

In Python: `sklearn.ensemble.VotingClassifier`

Hard voting (plurality voting)

- Each classifier votes for the class it predicted with weight 1
- The class with the most votes wins

Voting Schemes

In Python: `sklearn.ensemble.VotingClassifier`

Hard voting (plurality voting)

- Each classifier votes for the class it predicted with weight 1
- The class with the most votes wins
- Ties are broken implementation-specific, e.g. ordered by class label.

Voting Schemes

In Python: `sklearn.ensemble.VotingClassifier`

Hard voting (plurality voting)

- Each classifier votes for the class it predicted with weight 1
- The class with the most votes wins
- Ties are broken implementation-specific, e.g. ordered by class label.

Soft voting

Voting Schemes

In Python: `sklearn.ensemble.VotingClassifier`

Hard voting (plurality voting)

- Each classifier votes for the class it predicted with weight 1
- The class with the most votes wins
- Ties are broken implementation-specific, e.g. ordered by class label.

Soft voting

- Each classifier computes a probability for each class.

Voting Schemes

In Python: `sklearn.ensemble.VotingClassifier`

Hard voting (plurality voting)

- Each classifier votes for the class it predicted with weight 1
- The class with the most votes wins
- Ties are broken implementation-specific, e.g. ordered by class label.

Soft voting

- Each classifier computes a probability for each class.
- Per class, probabilities are summed up.

Voting Schemes

In Python: `sklearn.ensemble.VotingClassifier`

Hard voting (plurality voting)

- Each classifier votes for the class it predicted with weight 1
- The class with the most votes wins
- Ties are broken implementation-specific, e.g. ordered by class label.

Soft voting

- Each classifier computes a probability for each class.
- Per class, probabilities are summed up.
- Class with the highest sum wins.

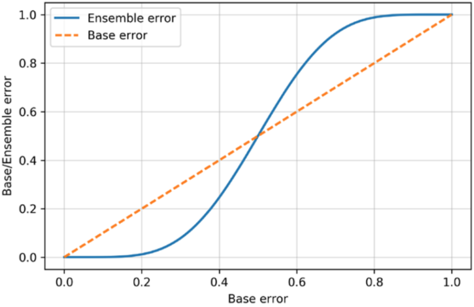
The choice of the base predictors (types and number) and the voting scheme are new hyperparameters.

Rationale

Why is a combination of weak base learners more successful than a single one?

Rationale

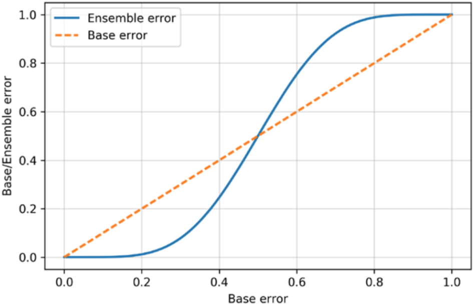
Why is a combination of weak base learners more successful than a single one?



Source: [1], p. 223

Rationale

Why is a combination of weak base learners more successful than a single one?

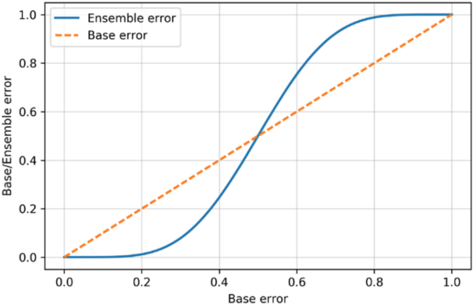


Source: [1], p. 223

- idealized binary classification – **independent** error probabilities

Rationale

Why is a combination of weak base learners more successful than a single one?



Source: [1], p. 223

- idealized binary classification – **independent** error probabilities
- error probability (missclassification) of the ensemble is lower than that of any base classifier if the base classifiers is better than random guessing

Use Cases

- combine weak methods into a stronger one

Use Cases

- combine weak methods into a stronger one
- combine complementary approaches on the same data.

Use Cases

- combine weak methods into a stronger one
- combine complementary approaches on the same data.
- combine different data sources (each requiring different algorithms)

Outline

Bagging

- use a voting scheme

Bagging

- use a voting scheme
- train each classifier on a different subset of the training data

Bagging

- use a voting scheme
- train each classifier on a different subset of the training data
- use random sampling with replacement to create the subsets.

Bagging

- use a voting scheme
- train each classifier on a different subset of the training data
- use random sampling with replacement to create the subsets.
- Python: `class sklearn.ensemble.BaggingClassifier`

Bagging

- use a voting scheme
- train each classifier on a different subset of the training data
- use random sampling with replacement to create the subsets.
- Python: `class sklearn.ensemble.BaggingClassifier`

Notebook 07_1_ensembles_wines, Cells 1–7

Algorithm Random Forest

- forest = ensemble of trees
- bagging: each tree learned on a different data subset
- additionally: feature bagging each tree may only choose from a randomly sampled subset of features (random subspace) → more variation among the trees
- main hyperparameters: number of trees + hyperparameters of decision trees
- interpretability of decision trees is lost
- less prone to overfitting
- in sklearn: `sklearn.ensemble.RandomForestClassifier`

Outline

Boosting

- train a sequence of classifiers

Boosting

- train a sequence of classifiers
- "focus" training of classifier i on missclassified instances of classifier $i - 1$

Boosting

- train a sequence of classifiers
- "focus" training of classifier i on missclassified instances of classifier $i - 1$
- warning: overfitting

Boosting

- train a sequence of classifiers
- "focus" training of classifier i on missclassified instances of classifier $i-1$
- warning: overfitting
- most popular implementation: AdaBoost

Boosting

- train a sequence of classifiers
- "focus" training of classifier i on missclassified instances of classifier $i-1$
- warning: overfitting
- most popular implementation: AdaBoost
- Python: `sklearn.ensemble.AdaBoostClassifier`

AdaBoost

- use a weight vector $\boldsymbol{w}$ to control the influence of individual instances

AdaBoost

- use a weight vector $\boldsymbol{w}$ to control the influence of individual instances
- $\boldsymbol{w}$ has one entry for each sample

AdaBoost

- use a weight vector $\boldsymbol{w}$ to control the influence of individual instances
- $\boldsymbol{w}$ has one entry for each sample
- initialize $\boldsymbol{w}^0$ with ones

AdaBoost

- use a weight vector $\boldsymbol{w}$ to control the influence of individual instances
- $\boldsymbol{w}$ has one entry for each sample
- initialize $\boldsymbol{w}^0$ with ones
- for i in $1, 2, \ldots, m$:

AdaBoost

- use a weight vector $\boldsymbol{w}$ to control the influence of individual instances
- $\boldsymbol{w}$ has one entry for each sample
- initialize $\boldsymbol{w}^0$ with ones
- for i in $1, 2, \ldots, m$:
  - learn classifier C^i with training data weighted with $\boldsymbol{w}^{i-1}$

AdaBoost

- use a weight vector $\boldsymbol{w}$ to control the influence of individual instances
- $\boldsymbol{w}$ has one entry for each sample
- initialize $\boldsymbol{w}^0$ with ones
- for i in $1, 2, \ldots, m$:
  - learn classifier C^i with training data weighted with $\boldsymbol{w}^{i-1}$
  - determine a coefficient based on the error rate of C^i

AdaBoost

- use a weight vector $\boldsymbol{w}$ to control the influence of individual instances
- $\boldsymbol{w}$ has one entry for each sample
- initialize $\boldsymbol{w}^0$ with ones
- for i in $1, 2, \dots, m$:
  - learn classifier C^i with training data weighted with $\boldsymbol{w}^{i-1}$
  - determine a coefficient based on the error rate of C^i
  - use the coefficient to increase weights of missclassified instances and decrease weights of correctly classified instances to yield $\boldsymbol{w}^i$

AdaBoost

- use a weight vector $\boldsymbol{w}$ to control the influence of individual instances
- $\boldsymbol{w}$ has one entry for each sample
- initialize $\boldsymbol{w}^0$ with ones
- for i in $1, 2, \ldots, m$:
  - learn classifier C^i with training data weighted with $\boldsymbol{w}^{i-1}$
  - determine a coefficient based on the error rate of C^i
  - use the coefficient to increase weights of missclassified instances and decrease weights of correctly classified instances to yield $\boldsymbol{w}^i$
- Compute final prediction using a weighted combination of the m classifiers (weights are combinations of the respective coefficients and a learning rate)

AdaBoost – Algorithm

- For training data X, y, initialize $\boldsymbol{w}^{(0)}$ with $|X|$ ones and normalize

AdaBoost – Algorithm

- For training data X, y, initialize $\boldsymbol{w}^{(0)}$ with $|X|$ ones and normalize
- for i in $1, 2, \ldots, m$:

AdaBoost – Algorithm

- For training data X, y, initialize $\boldsymbol{w}^{(0)}$ with $|X|$ ones and normalize
- for i in $1, 2, \ldots, m$:
  - train $C^{(i)}$ on X, y with weights $\boldsymbol{w}^{(i-1)}$ and compute $\hat{y}^{(i)}$

AdaBoost – Algorithm

- For training data X, y, initialize $\boldsymbol{w}^{(0)}$ with $|X|$ ones and normalize
- for i in $1, 2, \ldots, m$:
  - train $C^{(i)}$ on X, y with weights $\boldsymbol{w}^{(i-1)}$ and compute $\hat{y}^{(i)}$
  - error rate: $\epsilon^{(i)} := \sum_{x=1, y_x \neq \hat{y}_x^{(i)}}^{|X|} w_x^{(i-1)}$

AdaBoost – Algorithm

- For training data X, y, initialize $\boldsymbol{w}^{(0)}$ with $|X|$ ones and normalize
- for i in $1, 2, \ldots, m$:
  - train $C^{(i)}$ on X, y with weights $\boldsymbol{w}^{(i-1)}$ and compute $\hat{y}^{(i)}$
  - error rate: $\epsilon^{(i)} := \sum_{x=1, y_x \neq \hat{y}_x^{(i)}}^{|X|} w_x^{(i-1)}$
  - coefficient $\alpha^{(i)} := 0.5 \log \frac{1-\epsilon^{(i)}}{\epsilon^{(i)}}$

AdaBoost – Algorithm

- For training data X, y, initialize $\boldsymbol{w}^{(0)}$ with $|X|$ ones and normalize
- for i in $1, 2, \ldots, m$:
  - train $C^{(i)}$ on X, y with weights $\boldsymbol{w}^{(i-1)}$ and compute $\hat{y}^{(i)}$
  - error rate: $\epsilon^{(i)} := \sum_{x=1, y_x \neq \hat{y}_x^{(i)}}^{|X|} w_x^{(i-1)}$
  - coefficient $\alpha^{(i)} := 0.5 \log \frac{1-\epsilon^{(i)}}{\epsilon^{(i)}}$
  - update weights $w_x^{(i)} := w_x^{(i-1)} e^{-y_x \hat{y}_x^{(i)} \alpha^{(i)}}$ and normalize $\boldsymbol{w}^{(i)}$

AdaBoost – Algorithm

- For training data X, y, initialize $\boldsymbol{w}^{(0)}$ with $|X|$ ones and normalize
- for i in $1, 2, \ldots, m$:
  - train $C^{(i)}$ on X, y with weights $\boldsymbol{w}^{(i-1)}$ and compute $\hat{y}^{(i)}$
  - error rate: $\epsilon^{(i)} := \sum_{x=1, y_x \neq \hat{y}_x^{(i)}}^{|X|} w_x^{(i-1)}$
  - coefficient $\alpha^{(i)} := 0.5 \log \frac{1-\epsilon^{(i)}}{\epsilon^{(i)}}$
  - update weights $w_x^{(i)} := w_x^{(i-1)} e^{-y_x \hat{y}_x^{(i)} \alpha^{(i)}}$ and normalize $\boldsymbol{w}^{(i)}$

Note: If $C^{(i)}$ is better than guessing, then $\epsilon^{(i)} < 0.5$. Thus,
$\alpha^{(i)} = .5 \log \frac{1-\epsilon^{(i)}}{\epsilon^{(i)}} > 0$ and $e^{-y_x \hat{y}_x^i \alpha^i} \begin{cases} > 1 \text{ for } y_x \neq \hat{y}_x^i \\ < 1 \text{ for } y_x = \hat{y}_x^i \end{cases}$.

AdaBoost in Python

- `csklearn.ensemble.AdaBoostClassifier`

AdaBoost in Python

- `csklearn.ensemble.AdaBoostClassifier`
- parameters: estimator type, number of estimators

AdaBoost in Python

- `csklearn.ensemble.AdaBoostClassifier`
- parameters: estimator type, number of estimators
- additionally use a learning rate to control how much the i-th learner contributes to the result of the previou $i-1$ learners

AdaBoost in Python

- `csklearn.ensemble.AdaBoostClassifier`
- parameters: estimator type, number of estimators
- additionally use a learning rate to control how much the i-th learner contributes to the result of the previou $i-1$ learners

Notebook 07_1_ensembles_wines, Cells 8–10

References

S. Raschka and V. Mirjalili.
Python Machine Learning.
Packt Publishing Ltd., Livery Place 35 Livery Street Birmingham B3 2PB, UK, second edition, September 2017.