Ensembles: bagging, stacking, blending

MISiS Mega Science, Spring Semester

Nikita Kazeev, Andrey Ustyuzhanin

March 2021

LAMBDA • HSE

Lecture overview

After the lecture, you will be able to use the ensemble methods to improve the model performance.

Lecture overview

After the lecture, you will be able to use the ensemble methods to improve the model performance.

- Unless you want an overengineered model to win at a competition, you usually don't want to do the steps from this lecture by hand

Lecture overview

After the lecture, you will be able to use the ensemble methods to improve the model performance.

- Unless you want an overengineered model to win at a competition, you usually don't want to do the steps from this lecture by hand
- ...but still might want to understand how to better tune the knobs of the pre-packaged model you'll use in practice

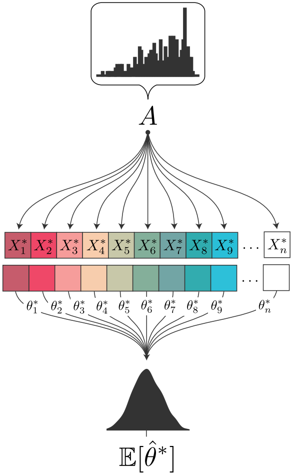
Bagging and Random Forests

Motivation

- The root of all evil in machine learning is the finite amount of data
- When a learning algorithm trains the model, it's forced between Scylla and Charybdis. Trust the data too much, and overfit. Trust the data too little, and underfit.
- What if we fight evil with evil and have many versions of the algorithm trained on different subsets of the dataset so that the biases cancel each other?

The bootstrapping procedure

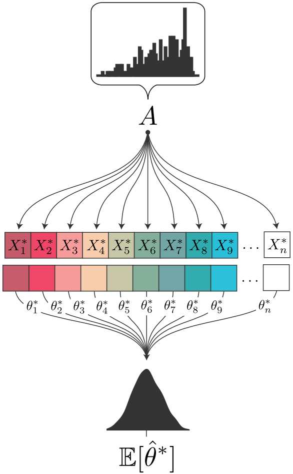
- Input: a sample $D = \{(x_i, y_i)\}$



Picture: http://www.drbunsen.org/bootstrap-in-picture

The bootstrapping procedure

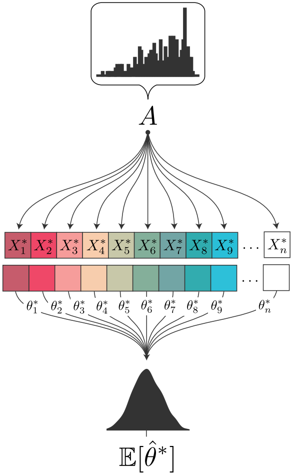
- Input: a sample $D = \{(x_i, y_i)\}$
- Bootstrapping: generate new samples $X_j^\star$ of (x_i, y_i) drawn from D uniformly at random with replacement (replicated (x_i, y_i) possible!)



Picture: http://www.drbunsen.org/bootstrap-in-picture

The bootstrapping procedure

- Input: a sample $D = \{(x_i, y_i)\}$
- Bootstrapping: generate new samples $X_j^\star$ of (x_i, y_i) drawn from D uniformly at random with replacement (replicated (x_i, y_i) possible!)
- **Bagging (bootstrap aggregating):**
  1. Generate N bootstrapped samples $X_1^\star, \ldots, X_n^\star$
  2. Learn n models $h_1, \ldots, h_n$
  3. Average predictions to obtain $h(x) = \frac{1}{n} \sum_{j=1}^{n} h_j(x)$
  4. Profit!



Picture: http://www.drbunsen.org/bootstrap-in-picture

The Random Forest algorithm

- Bagging over decision trees

The Random Forest algorithm

- Bagging over decision trees
- Reduce error via averaging over instances and features

The Random Forest algorithm

- Bagging over decision trees
- Reduce error via averaging over instances and features
- Input: a sample $D = \{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathcal{X} \equiv \mathbb{R}^d, y_i \in \mathcal{Y}$

The Random Forest algorithm

- Bagging over decision trees
- Reduce error via averaging over instances and features
- Input: a sample $D = \{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathcal{X} \equiv \mathbb{R}^d, y_i \in \mathcal{Y}$
- The algorithm iterates for $j = 1, \ldots, N$:

The Random Forest algorithm

- Bagging over decision trees
- Reduce error via averaging over instances and features
- Input: a sample $D = \{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathcal{X} \equiv \mathbb{R}^d, y_i \in \mathcal{Y}$
- The algorithm iterates for $j = 1, \ldots, N$:

1. Pick p random features out of d

The Random Forest algorithm

- Bagging over decision trees
- Reduce error via averaging over instances and features
- Input: a sample $D = \{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathcal{X} \equiv \mathbb{R}^d, y_i \in \mathcal{Y}$
- The algorithm iterates for $j = 1, \ldots, N$:

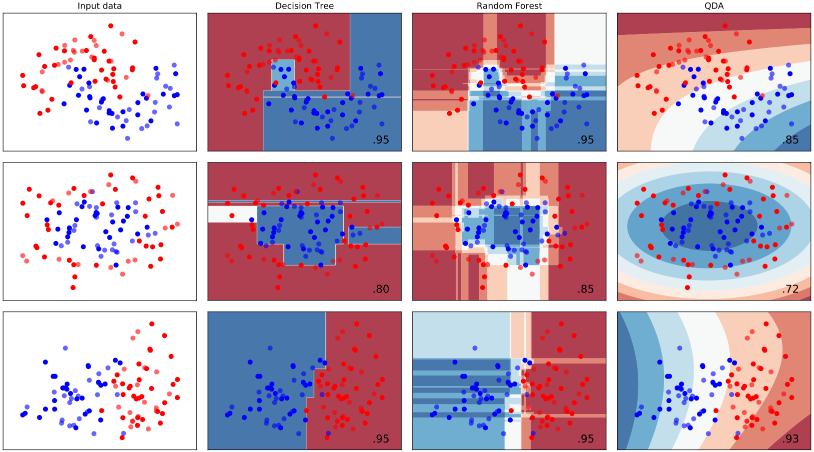
1. Pick p random features out of d
2. Bootstrap a sample $D_j = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in \mathcal{Y}$

The Random Forest algorithm

- Bagging over decision trees
- Reduce error via averaging over instances and features
- Input: a sample $D = \{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathcal{X} \equiv \mathbb{R}^d, y_i \in \mathcal{Y}$
- The algorithm iterates for $j = 1, \ldots, N$:

1. Pick p random features out of d
2. Bootstrap a sample $D_j = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \mathbb{R}^p, y_i \in \mathcal{Y}$
3. Learn a decision tree $h_j(\mathbf{x})$ using the bootstrapped D_j

Random Forest: synthetic examples



Random Forests Bias and Variance

Remember the bias-variance decomposition?

$$\mathrm{MSE}(\mathbf{x}) = \underbrace{\mathbb{E}_y\left[\left(y - \mathbb{E}[y \mid x]\right)^2\right]}_{\text{noise}} + \underbrace{\left(\mathbb{E}_D\left[f_D\left(\mathbf{x}\right)\right] - \mathbb{E}[y \mid x]\right)^2}_{\text{bias}} + \underbrace{\mathbb{E}_D\left[\left(f_D\left(\mathbf{x}\right) - \mathbb{E}_D\left[f_D\left(\mathbf{x}\right)\right]\right)^2\right]}_{\text{variance}}$$

Bagging and Bias

- Bias: not made any worse by bagging multiple hypotheses

$$\underbrace{\mathbb{E}_y\left[\left(\mathbb{E}_D\left[\frac{1}{N}\sum_{n=1}^{N}\tilde{f}_D(x)\right] - \mathbb{E}[y\,|\,x]\right)^2\right]}_{\text{bias of the ensemble}} = \mathbb{E}_y\left[\left(\frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_D[\tilde{f}_D(x)] - \mathbb{E}[y\,|\,x]\right)^2\right] =$$

$$= \underbrace{\mathbb{E}_y\left[\left(\mathbb{E}_D\left[\tilde{f}_D(x)\right] - \mathbb{E}[y\,|\,x]\right)^2\right]}_{\text{bias of the individual model}}$$

Bagging and Variance

- Variance: Let $F = \frac{1}{N}\sum_{n=1}^{N} \tilde{f}_n(\mathbf{x})$

$$\mathrm{Var}(F) = \frac{1}{N^2}\sum_{i,j}\mathrm{Cov}(\tilde{f}_i, \tilde{f}_j) = \frac{1}{N^2}\sum_{i}\left[\mathrm{Var}(\tilde{f}_i) + \sum_{j\neq i}\mathrm{Cov}(\tilde{f}_i, \tilde{f}_j)\right]$$

Bagging and Variance

- Variance: Let $F = \frac{1}{N}\sum_{n=1}^{N} \tilde{f}_n(\mathbf{x})$

$$\mathrm{Var}(F) = \frac{1}{N^2}\sum_{i,j}\mathrm{Cov}(\tilde{f}_i, \tilde{f}_j) = \frac{1}{N^2}\sum_{i}\left[\mathrm{Var}(\tilde{f}_i) + \sum_{j\neq i}\mathrm{Cov}(\tilde{f}_i, \tilde{f}_j)\right]$$

- All the models $\tilde{f}_i$ use the same algorithm, so

$$\mathrm{Var}(F) = \frac{1}{N}\mathrm{Var}(\tilde{f}) + \frac{1}{N^2}\sum_{i}\sum_{j\neq i}\mathrm{Cov}(\tilde{f}_i, \tilde{f}_j)$$

Bagging and Variance

- Variance: Let $F = \frac{1}{N}\sum_{n=1}^{N}\tilde{f}_n(\mathbf{x})$

$$\mathrm{Var}(F) = \frac{1}{N^2}\sum_{i,j}\mathrm{Cov}(\tilde{f}_i, \tilde{f}_j) = \frac{1}{N^2}\sum_{i}\left[\mathrm{Var}(\tilde{f}_i) + \sum_{j\neq i}\mathrm{Cov}(\tilde{f}_i, \tilde{f}_j)\right]$$

- All the models $\tilde{f}_i$ use the same algorithm, so

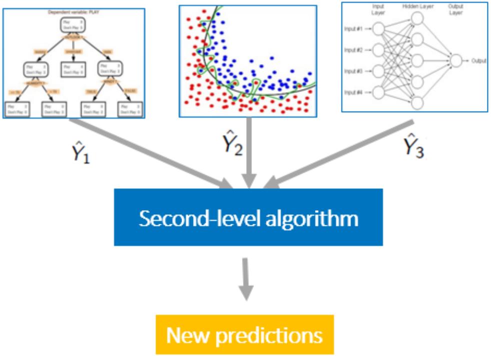
$$\mathrm{Var}(F) = \frac{1}{N}\mathrm{Var}(\tilde{f}) + \frac{1}{N^2}\sum_{i}\sum_{j\neq i}\mathrm{Cov}(\tilde{f}_i, \tilde{f}_j)$$

- Conclusion: Variance is N times lower for uncorrelated hypotheses, and is unchanged for fully-correlated.

Stacked generalisation

Motivation

What if I train an algorithm B that corrects the mistakes of algorithm A?



Picture: `https://blogs.sas.com`

Blending

- Partition the training dataset D into D_1 and D_2
- Train models $\tilde{\mathsf{f}}_\mathsf{i}(\mathbf{x})$ on D_1
- Compute predictions of $\mathsf{Z}_\mathsf{i} = \tilde{\mathsf{f}}_\mathsf{i}(\mathsf{D}_2)$
- Train the meta-model $\phi(\mathsf{Z}_1, \ldots, \mathsf{Z}_\mathsf{N}, \mathsf{D}_2)$ on the predictions obtained on the previous step and features

Blending

- Partition the training dataset D into D_1 and D_2
- Train models $\tilde{f}_i(\mathbf{x})$ on D_1
- Compute predictions of $Z_i = \tilde{f}_i(D_2)$
- Train the meta-model $\phi(Z_1, \ldots, Z_N, D_2)$ on the predictions obtained on the previous step and features
- Do you see a glaring issue with this approach?

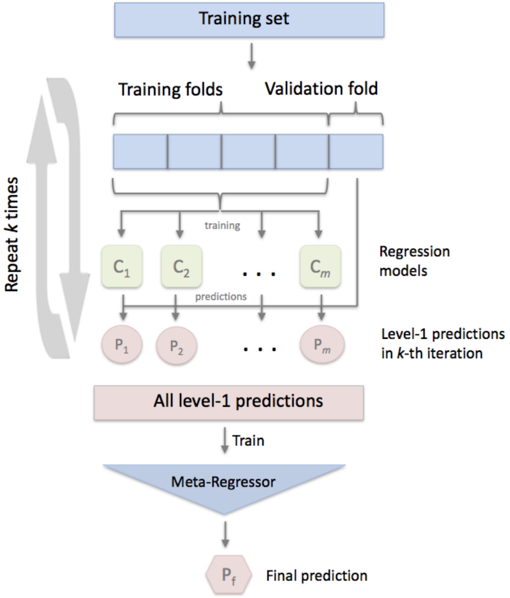
Blending

- Partition the training dataset D into D_1 and D_2
- Train models $\tilde{f}_i(\mathbf{x})$ on D_1
- Compute predictions of $Z_i = \tilde{f}_i(D_2)$
- Train the meta-model $\phi(Z_1, \ldots, Z_N, D_2)$ on the predictions obtained on the previous step and features
- Do you see a glaring issue with this approach?
- Both levels are trained on half of the dataset – unacceptable waste in the quest for 1% performance gain!

Stacking

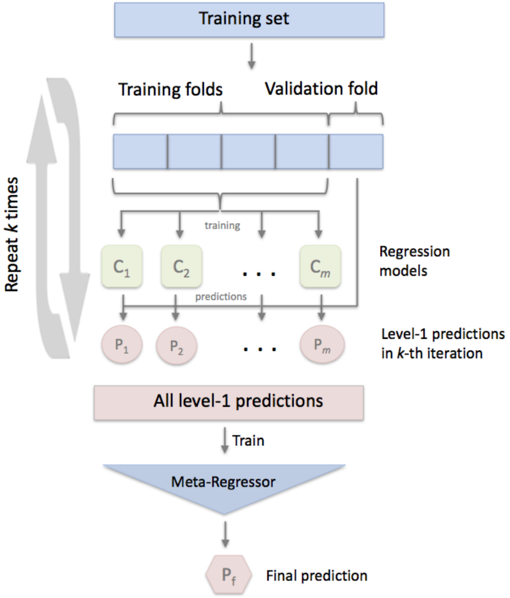
1. Partition train into k folds
2. Just like in cross-validation, k times train each level-1 model leaving one fold out; predict on the left-out fold

Picture: `https://rasbt.github.io/mlxtend/user_guide/regressor/StackingCVRegressor/`



Stacking

1. Partition train into k folds
2. Just like in cross-validation, k times train each level-1 model leaving one fold out; predict on the left-out fold
3. Fit the meta-model on all the level-1 predictions, optionally concatenated with features

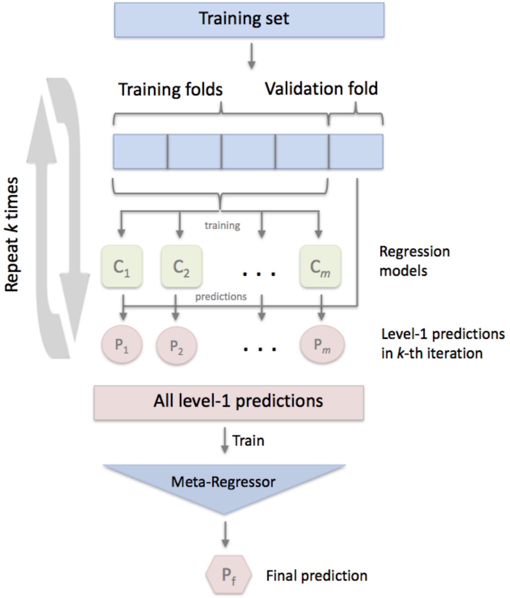


Picture: `https://rasbt.github.io/mlxtend/user_guide/regressor/StackingCVRegressor/`

Stacking

1. Partition train into k folds
2. Just like in cross-validation, k times train each level-1 model leaving one fold out; predict on the left-out fold
3. Fit the meta-model on all the level-1 predictions, optionally concatenated with features
4. For prediction, first evaluate the level-1 models, then the meta-model

Picture: `https://rasbt.github.io/mlxtend/user_guide/regressor/StackingCVRegressor/`



Summary

- Bootstrapping: a general statistical technique for computing sample functionals (and their variance)
- Bagging: meta-learner over arbitrary algorithms via bootstrap aggregation
- The Random Forest algorithm: Bagging over decision trees
- Stacking: train a learner on the outputs of other learners
- Blending: a simplified version of stacking

Thank you!

nikita.kazeev@cern.ch

anaderiRu

hse_lambda

Acknowledgements

These slides are based on the slides for for the previous edition of the MLHEP school by Alexey Artemov.