Ensemble Methods

Andrea Passerini
andrea.passerini@unitn.it

Machine Learning

Ensemble Methods

The rationale

- Groups of individuals can make better decisions than individuals (if they don't all think the same)
- Combining multiple ML models can produce better predictions than those of any single model
- Training ensembles can (often) be parallelized, with substantial computational savings

Note

Models should be diverse enough for the ensemble to work

How to guarantee diversity?

Ensembling strategies

- **bagging**: diversify the training sets
- **stacking**: diversify the models being trained
- **boosting**: (progressively) diversify the importance of examples

Bagging: Bootstrap aggregation

In a nutshell

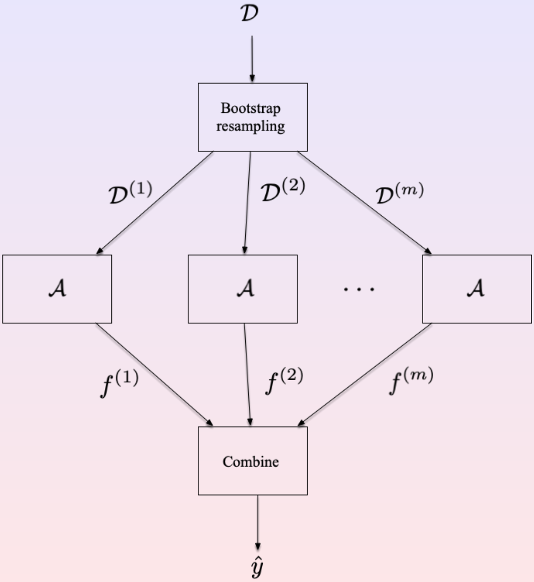
- Take a learning algorithm $\mathcal{A}$ (e.g. decision tree learning)
- Extract m different datasets $\mathcal{D}^{(i)}$ from the original training set $\mathcal{D}$
- Train one base model per dataset

$$f^{(i)} = \mathcal{A}(\mathcal{D}^{(i)})$$

- Combine predictions of different base models

$$\hat{y} = \textsc{Combine}(f^{(1)}(x), \ldots, f^{(m)}(x))$$

Bagging: Bootstrap aggregation



Bagging: Bootstrap aggregation

Extracting datasets: boostrap resampling

- Simply partitioning $\mathcal{D}$ into m subsets produces very small training sets
- *Bootstrap resampling* extracts $N = |\mathcal{D}|$ samples from $\mathcal{D}$ *with replacement* (i.e., same example can be selected multiple times)
- Repeating the procedure m times to get diverse datasets of the same size as $\mathcal{D}$

Bagging: Bootstrap aggregation

Diversity of Datasets

- Each example has probability $(1 - 1/N)$ of not being selected at each draw
- Each example has probability $(1 - 1/N)^N$ of not being selected after N draws
- For large enough N, this is 37%, i.e., 37% of the dataset are not part of a given training set (*out-of-bag instances*, oob)
- oob instances can be used to estimate test performance of the base model

Bagging: Bootstrap aggregation

Combination strategies

Majority voting: predict the class with most votes (classification)

$$\hat{y} = argmax_y \sum_{i=1}^{m} \delta(y, \hat{y}^{(i)})$$

Soft voting: predict the class with largest sum of predicted probability (classification). Assumes base classifier outputs a confidence/probability

$$\hat{y} = argmax_y \sum_{i=1}^{m} f_y^{(i)}(x)$$

Mean: predict the mean (or median) of the base model outputs (regression)

$$\hat{y} = \sum_{i=1}^{m} f^{(i)}(x)$$

Bagging example: Random forests

Bagging + model decorrelation

- Use decision trees as the base models
- Even if $\mathcal{D}^{(i)} \neq \mathcal{D}^{(j)}$, the learned DTs can be too much correlated
- Introduce additional stochasticity in the DT training process
- At each node, choose the best feature to split from a random selection of the set of features (instead of using them all)

Stacking: stacked generalization

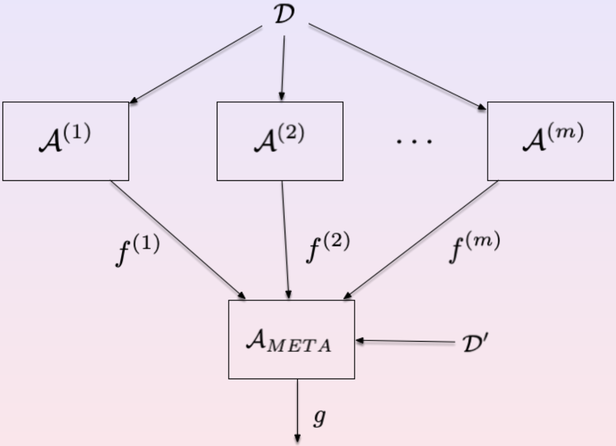
In a nutshell

- Train m base models $f^{(i)}$ on $\mathcal{D}$ with m *different* learning algorithms $\mathcal{A}^{(i)}$
$$f^{(i)} = \mathcal{A}^{(i)}(\mathcal{D})$$
- Use a *meta-learner* to learn to combine base models (e.g., linear combination)
$$g = \mathcal{A}_{META}([f^{(1)}, \dots, f^{(m)}], \mathcal{D}')$$
- Use the meta-model to output ensemble predictions
$$\hat{y} = g\left([f^{(1)}(x), \dots, f^{(m)}(x)]\right)$$

Note

The meta-model should be trained on a dataset $\mathcal{D}' \neq \mathcal{D}$, or it will learn to focus on the best performing model on $\mathcal{D}$.

Stacking: stacked generalization



Boosting

In a nutshell

- Take a learner $\mathcal{A}$ and train it on $\mathcal{D}$
- Reweight examples in $\mathcal{D}$ based on their accuracy according to the trained model (harder examples get larger weights)
- Train $\mathcal{A}$ again on the reweighted dataset
- Repeat the procedure m times
- Combine the learned models into the final model

Boosting weak learners

From weak learners to strong learners

- A *weak learner* learns models with accuracy slightly better than random.
- A weak learner is easy to implement and train.
- Applying boosting with weak learners as the base learning algorithm allows to turn them into *strong learners*
- This can be easier than designing a strong learner directly.

Boosting example: AdaBoost

procedure ADABOOST
  $\mathbf{d}^{(0)} \leftarrow \langle \frac{1}{N}, \dots, \frac{1}{N} \rangle$ ▷ initialize importance weights uniformly
  for $i = 1, \dots, m$ **do**
    $f^{(i)} \leftarrow \mathcal{A}(\mathcal{D}, \mathbf{d}^{(i-1)})$ ▷ train i^{th} model on weighted data
    $\hat{y}_n \leftarrow f^{(i)}(x_n), \forall n$ ▷ collect model predictions
    $\hat{\epsilon}^{(i)} \leftarrow \sum_n d_n^{(i-1)} \mathbb{1}[y_n \neq \hat{y}_n]$ ▷ compute weighted training error
    $\alpha^{(i)} \leftarrow \frac{1}{2} \log \left(\frac{1-\hat{\epsilon}^{(i)}}{\hat{\epsilon}^{(i)}} \right)$ ▷ compute adaptive parameter
    $d_n^{(i)} \leftarrow \frac{1}{Z} d_n^{(i-1)} \exp(-\alpha^{(i)} y_n \hat{y}_n), \forall n$ ▷ re-weight examples
  end for
  return $f(\mathbf{x}) = \mathrm{sgn}(\sum_i \alpha^{(i)} f^{(i)}(\mathbf{x}))$ ▷ return ensemble model
end procedure

AdaBoost

$$d_n^{(i)} \leftarrow \frac{1}{Z} d_n^{(i-1)} \exp(-\alpha^{(i)} y_n \hat{y}_n)$$

Example re-weighting

- correctly classified examples $y_n \hat{y}_n = +1$ are multiplicatively downweighted
- incorrectly classified examples $y_n \hat{y}_n = -1$ are multiplicatively upweighted
- Z is a normalization constant making weights sum to one (importance as probability distribution over examples)

Role of adaptive parameter (1)

- Let $\mathcal{D}$ have 80 positive and 20 negative examples
- Assume the first classifier $f^{(1)}$ simply returns the majority class (i.e., $f^{(1)}(\mathbf{x}_n) = +1$ for all n)
- It's weighted error rate is 0.2 (all negative examples are incorrectly predicted)
- The adaptive parameter is

$$\alpha^{(1)} = \frac{1}{2} \log \left(\frac{1 - \hat{\epsilon}^{(i)}}{\hat{\epsilon}^{(i)}} \right) = 1/2 \log(4)$$

- The multiplicative weight for positive (correct) examples is $\exp(-\alpha^{(i)} y_n \hat{y}_n) = \exp(-1/2 \log(4)) = 1/2$.
- The multiplicative weight for negative (incorrect) examples is $\exp(-\alpha^{(i)} y_n \hat{y}_n) = \exp(1/2 \log(4)) = 2$.

Role of adaptive parameter (2)

- The normalization Z (ignoring $d_n^{(i-1)}$ for simplicity) is $80 * 1/2 + 20 * 2 = 80$
- the normalized weights for positive and negative examples are $1/2 * 1/80 = 1/160$ and $2 * 1/80 = 1/40$
- The overall weight of positive and negative examples is now $1/160 * 80 = 1/2$ and $1/40 * 20 = 1/2$
- The reweighted dataset is thus fully balanced, and a majority class predictor is not a weak learner any more (accuracy exactly 50%).

References

- H. Daume. *A Course in Machine Learning*, http://ciml.info/
- K. Murphy, Probabilistic Machine Learning: An Introduction, The MIT Press, 2021 (online version at https://probml.github.io/pml-book/book1.html)