Hierarchical Gradient Smoothing for Probability Estimation Trees

Presenter: Dr He Zhang
Date: 12th May, 2020
OUTLINE

1. Motivation and related work
   - Where we can do better?
2. Methodology
   - The HGS algorithm
3. Experiment results
   - How does our model work?
4. Conclusion and discussion
   - Take-home messages
1. Motivation and related work
   ➢ Where we can do better?

2. Methodology
   ➢ The HGS algorithm

3. Experiment results
   ➢ How does our model work?

4. Conclusion and discussion
   ➢ Take-home messages
Motivation: Why do we still need single decision tree?

Can we make a single decision tree more accurate while still being efficient and highly interpretable?
Motivation:
What is probability estimation trees?

Decision Tree:
- output a discrete predicted class label

Probability Estimation Trees (PETs)
- output a class probability distribution

PETs are more preferred
- because accurate class probability estimates can show us the reliability of the prediction.

However, poor estimates at leaf nodes:
- high variance: data sparsity
- high bias: towards 0 and 1

Need to be improved.
Motivation: Why doing hierarchical smoothing for trees?

Hierarchical Smoothing / estimation idea:
To make each node a function of the data at the node and the estimate at the parent.

\[ p(\text{disease} | \text{has gene & male}) \sim p(\text{disease} | \text{has gene}) \]

\[ p(\text{disease} | \text{has gene}) \sim p(\text{disease}) \]

\[ P_{\text{MLE}} = \frac{0}{1} = 0\% \]

\[ P_{\text{Laplace}} = \frac{0+1}{1+2} = 33.3\% \]

\[ P_{\text{M-estimation}} = \frac{0+0.5}{1+1} = 25\% \]

Probability smoothing: make the estimates less extreme.

None of them use the fact that 91% of the patients with that gene have the disease!
Related Work:
Existing hierarchical smoothing methods

- Hierarchical Dirichlet Process (HDP) smoothing

\[
\theta_{4,y} = \frac{n_{4,y} + c_2 \theta_{2,y}}{N_4 + c_2},
\]

\[
\theta_{2,y} = \frac{n_{2,y} + c_1 \theta_{1,y}}{N_2 + c_1},
\]

\[
\theta_{1,y} = \frac{n_{1,y} + c_0 \theta_{0,y}}{N_1 + c_0}.
\]

Need sampling for the parameters

Recursive and Slow !!!

- M-branch smoothing has the same smoothing idea, but different parameter optimization method.
OUTLINE

1. Motivation and related work
   ➢ Where we can do better jobs?

2. Methodology
   ➢ The HGS algorithm

3. Experiment results
   ➢ How does our model work?

4. Conclusion and discussion
   ➢ Take-home messages
Methodology:
Hierarchical Gradient Smoothing (HGS) algorithm

How to make hierarchical smoothing faster?

- replace recursive smoothing with one-time smoothing

- replace sampling by gradient descent

Methodology:
Hierarchical Gradient Smoothing (HGS) algorithm

\[
\theta_{7,y} = \frac{n_{7,y} + \alpha_3 \cdot \theta_{3,y} + \alpha_1 \cdot \theta_{1,y}}{N_7 + \alpha_3 + \alpha_1}
\]

\[
\hat{\theta}_{HGS} = \frac{n_l,k + \alpha \sum_{p \in \text{anc}(l)} \alpha_p \hat{\theta}_{p,k}}{n_{l,} + \alpha \sum_{p \in \text{anc}(l)} \alpha_p}
\]

parameters: \( \hat{\alpha} \)

No sampling, only gradient descent
Yeah, faster !!!
Methodology:
How to optimize the weight parameters?

1. Propose a cost function for HGS

2. Calculate the gradient for each parameter

3. Conduct a standard gradient descent to optimize the parameters

\[ \text{LOOCV}(\alpha) = \frac{1}{N} \sum_{l \in \mathcal{L}} \sum_{k \in \mathcal{K}} n_{l,k} \cdot \log \left( \frac{1}{\theta_{l,k}^{\text{LOO}}} \right) \]

\[
\frac{\partial}{\partial \alpha_p} \text{LOOCV}(\alpha) = \frac{1}{N \ln 2} \sum_{l \in \text{des}(p)} \sum_{k \in \mathcal{K}} \beta_{l,k}
\]

where

\[
\beta_{l,k} = \frac{n_{l,k} \cdot \left( \theta_{l,k}^{\text{LOO}} - \theta_{p,k}^{\text{LOO}} \right)}{\left( n_{l,.} - 1 + \sum_{p \in \text{anc}(l)} \alpha_p \right) \cdot \theta_{l,k}^{\text{LOO}}}
\]
Methodology:

Let us talk more about the LOOCV cost function

• Leave-one-out cross validation (LOOCV):
  - a special case of k-folds cross-validation, where k equals to the number of examples.
• Incremental LOOCV: make LOOCV fast
  - it train on the full dataset, then delete the examples in one-fold, test on that fold, and insert the example back. The delete-test-insert phase is repeated for each of the folds.
• Now, if we remove a data example with true class $y$ at leaf node $l$ as the test example, the estimates becomes

\[
\theta_{l,k}^{\text{LOO}} = \frac{n_{l,k} - 1 + \sum_{p \in \text{anc}(l)} \alpha_p \theta_{p,k}^{\text{LOO}}}{n_{l,\cdot} - 1 + \sum_{p \in \text{anc}(l)} \alpha_p}
\]

  - Then, we conduct LOOCV on all the leaf nodes, and get the LOOCV cost function

\[
\text{LOOCV}(\alpha) = \frac{1}{N} \sum_{l \in \mathcal{L}} \sum_{k \in \mathcal{K}} n_{l,k} \cdot \log \left( \frac{1}{\theta_{l,k}^{\text{LOO}}} \right)
\]
OUTLINE

1. Motivation and related work
   - Where we can do better jobs?

2. Methodology
   - The HGS algorithm

3. Experiment results
   - How does our model work?

4. Conclusion and discussion
   - Take-home messages
• **Datasets:** 143 datasets from the UCI repository

| Data size | Count |
|-----------|-------|
| > 10,000  | 20    |
| > 1,000   | 52    |
| <1,000    | 71    |

• **Evaluation Metrics:**

| Metrics      | Description                                                                 |
|--------------|-----------------------------------------------------------------------------|
| RMSE         | measure the accuracy of class probability estimates                        |
| 0-1 Loss     | measure the accuracy of classification                                      |
| Win-Draw-Loss| when comparing two different models.                                        |

• **Evaluation Method:** 10-fold cross validation
1. HGS compared with other smoothing methods

Table 1: Win-Draw-Loss results (The boldface values are significant).

| Methods          | RMSE  | 0-1 Loss |
|------------------|-------|----------|
| HGS vs. MLE      | 108-2-33 | 69-22-52 |
| HGS vs. Laplace  | 111-4-28 | 68-22-53 |
| HGS vs. M-estimation | 98-4-41  | 66-23-54 |
| HGS vs. M-branch | 96-3-44  | 59-32-52 |
| HGS vs. HDP      | 92-1-50  | 64-21-58 |

- **Significantly better on class probability estimates:** RMSE
- **Better on classification accuracy:** 0-1 loss.

HGS is more accurate than other methods.
2. Efficiency of HGS

### Table 2: Averaged results on 143 datasets.

| Methods    | RMSE   | 0-1 Loss | Runtime |
|------------|--------|----------|---------|
| MLE        | 0.2596 | 0.2093   | 1.1     |
| Laplace    | 0.2499 | 0.2093   | 1.1     |
| M-estimation| 0.2485 | 0.2068   | 1.1     |
| HDP        | 0.2436 | 0.2078   | 4.9     |
| M-branch   | 0.2428 | 0.2062   | 9.3     |
| HGS        | **0.2410** | **0.2059** | **1.1** |

- HGS has similar runtime of single-layer methods.
- 5 times faster than HDP.
- 9 times faster than M-branch.

HGS is much more efficient than other hierarchical methods.
3. Training time comparison on different data sizes

![Graph showing training time comparison between HGS and HDP](chart)

- HGS faster than HDP on different data sizes.

Fig. 3: Training time comparison according to log data size.
4. Random Forest with HGS smoothing

- HGS is helpful for small forest.
- Single C4.5 + HGS $\approx$ RF_7 trees
  - Can be learnt and tested much faster
  - Has higher interpretability
  - Applicable for online learning applications

Fig. 4: HGS Smoothing on Random Forest in RMSE
OUTLINE

1. Motivation and related work
   ➢ Where we can do better jobs?
2. Methodology
   ➢ The HGS algorithm
3. Experiment results
   ➢ How does our model work?
4. Conclusion
   ➢ Take-home messages
Conclusion

- HGS is an order of magnitude faster than M-branch and HDP smoothing.

- HGS is significantly better on class probability estimation and better on classification accuracy.

- HGS is generally as good as or superior to random forest with 7 trees and almost as good with 10 trees, and thus suitable in online contexts.

- Random Forest does not need much smoothing because smoothing harms the diversity of random forest.

The implementation, dataset and raw results can be found on Github: https://github.com/icesky0125/DecisionTreeSmoothing
Thank you