Advanced Symbolic Regression Algorithms

Ing. Jan Žegklitz
zegkljan@fel.cvut.cz
Supervisor: Ing. Petr Pošík, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering
Dept. of Cybernetics

May 12, 2016

Outline

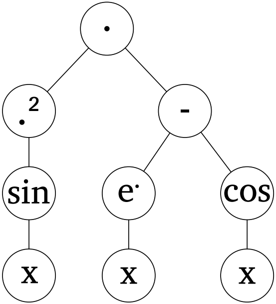
Symbolic Regression

- A unique kind of regression analysis.
- The models have a form of mathematical expression.
    - Preferably as small and as simple as possible.

Symbolic Regression

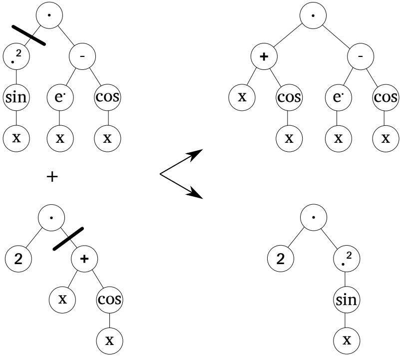
- Originally a subfield of Genetic Programming
  - Individuals encoded as parse trees.

$$\hat{y} = \sin^2 x \cdot (e^x - \cos x)$$



Symbolic Regression

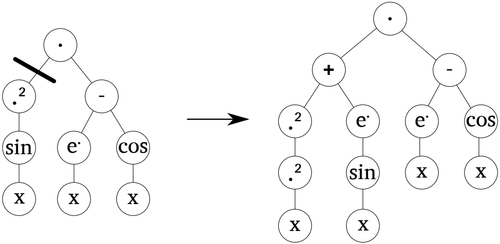
- Originally a subfield of Genetic Programming
    - Individuals encoded as parse trees.
    - Exploration through genetic operators of **crossover** and mutation.



Symbolic Regression

Jan Žegklitz
zegkljan
@fel.cvut.cz

- Originally a subfield of Genetic Programming
  - Individuals encoded as parse trees.
  - Exploration through genetic operators of crossover and **mutation**.



Symbolic Regression

- Originally a subfield of Genetic Programming
    - Individuals encoded as parse trees.
    - Exploration through genetic operators of crossover and mutation.
    - Exploitation through selection mechanism that promotes the better solutions.

State-of-the-Art in Symbolic Regression

- Classical, “vanilla” GP is often slow for SR tasks.
- Global trend – combine GP with linear regression
- Three selected algorithms
    - MGGP
    - EFS
    - FFX

MGGP
Multi-Gene Genetic Programming[1,2]

---

[1] Mark Hinchliffe et al. "Modelling Chemical Process Systems Using a Multi-Gene Genetic Programming Algorithm". In: *Late Breaking Paper, GP'96*. Stanford, USA, 1996, pp. 56–65.

[2] Dominic P Searson. "GPTIPS2: an open-source software platform for symbolic datamining". In: *Springer Handbook of Genetic Programming Applications*. Ed. by A H Gandomi, A H Alavi, and C Ryan. In press. 2015.

State-of-the-Art in Symbolic Regression

Multi-Gene Genetic Programming

MGGP = GP + multiple genes + linear regression

State-of-the-Art in Symbolic Regression

Multi-Gene Genetic Programming

MGGP = **GP** + multiple genes + linear regression

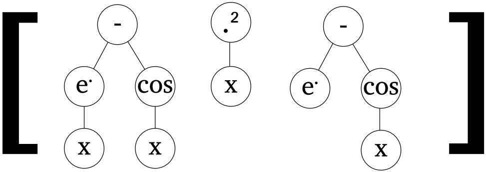
- Classical GP-style representation – parse trees.

State-of-the-Art in Symbolic Regression

Multi-Gene Genetic Programming

MGGP = GP + **multiple genes** + linear regression

- Classical GP-style representation – parse trees.
- Individual = more than one tree.

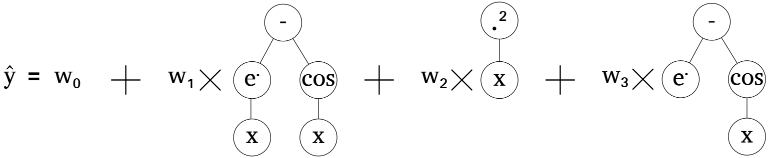


State-of-the-Art in Symbolic Regression

Multi-Gene Genetic Programming

MGGP = GP + **multiple genes** + linear regression

- Classical GP-style representation – parse trees.
- Individual = more than one tree.
    - Combined via linear combination.

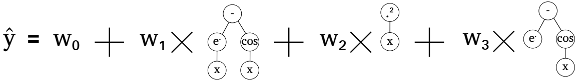


State-of-the-Art in Symbolic Regression

Multi-Gene Genetic Programming

MGGP = GP + multiple genes + **linear regression**

- Classical GP-style representation – parse trees.
- Individual = more than one tree.
    - Combined via linear combination.
- Coefficients determined via linear regression

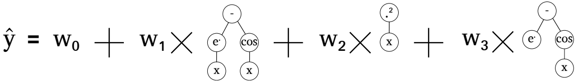


State-of-the-Art in Symbolic Regression

Multi-Gene Genetic Programming

MGGP = GP + multiple genes + **linear regression**

- Classical GP-style representation – parse trees.
- Individual = more than one tree.
    - Combined via linear combination.
- Coefficients determined via linear regression

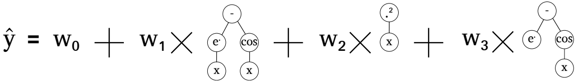


$$\hat{\mathbf{y}} = w_0 + w_1 g_1(\mathbf{x}) + w_2 g_2(\mathbf{x}) + \cdots + w_n g_n(\mathbf{x})$$

State-of-the-Art in Symbolic Regression
Multi-Gene Genetic Programming

MGGP = GP + multiple genes + **linear regression**

- Classical GP-style representation – parse trees.
- Individual = more than one tree.
  - Combined via linear combination.
- Coefficients determined via linear regression



$$\hat{y} = w_0 + w_1 g_1(\mathbf{x}) + w_2 g_2(\mathbf{x}) + \cdots + w_n g_n(\mathbf{x})$$

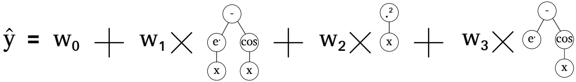
$$\hat{\mathbf{y}} = [\, \mathbf{1} \quad \mathbf{g_1} \quad \mathbf{g_2} \ \cdots \ \mathbf{g_n} \,] \cdot \mathbf{w}$$

State-of-the-Art in Symbolic Regression

Multi-Gene Genetic Programming

MGGP = GP + multiple genes + **linear regression**

- Classical GP-style representation – parse trees.
- Individual = more than one tree.
  - Combined via linear combination.
- Coefficients determined via linear regression



$$\hat{y} = w_0 + w_1 g_1(\mathbf{x}) + w_2 g_2(\mathbf{x}) + \cdots + w_n g_n(\mathbf{x})$$

$$\hat{\mathbf{y}} = [\,\mathbf{1} \quad \mathbf{g_1} \quad \mathbf{g_2} \; \cdots \; \mathbf{g_n}\,] \cdot \mathbf{w}$$

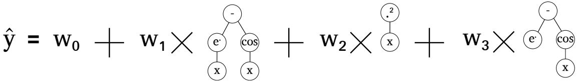
$$\hat{\mathbf{y}} = \mathbf{G}\mathbf{w}$$

State-of-the-Art in Symbolic Regression

Multi-Gene Genetic Programming

MGGP = GP + multiple genes + **linear regression**

- Classical GP-style representation – parse trees.
- Individual = more than one tree.
    - Combined via linear combination.
- Coefficients determined via linear regression



$$\hat{\mathbf{y}} = w_0 + w_1 g_1(\mathbf{x}) + w_2 g_2(\mathbf{x}) + \cdots + w_n g_n(\mathbf{x})$$

$$\hat{\mathbf{y}} = [\,1 \quad \mathbf{g}_1 \quad \mathbf{g}_2 \ \cdots \ \mathbf{g}_n\,] \cdot \mathbf{w}$$

$$\hat{\mathbf{y}} = \mathbf{G}\mathbf{w} \quad \rightarrow \quad \mathbf{w} = (\mathbf{G}^\top \mathbf{G})^{-1}\mathbf{G}^\top \mathbf{y}$$

State-of-the-Art in Symbolic Regression
Multi-Gene Genetic Programming

Notes

- New crossover operator – high-level crossover.
    - Exchanges whole genes between individuals.
- Maximum number of genes is limited.
- Maximum size/depth of a gene is usually also limited.
- No need to search for the linear parts – they are computed optimally w.r.t. the available genes.

EFS
Evolutionary Feature Synthesis[3]

---

[3] Ignacio Arnaldo, Una-May O'Reilly, and Kalyan Veeramachaneni. "Building Predictive Models via Feature Synthesis". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO '15. Madrid, Spain: ACM, 2015, pp. 983–990. ISBN: 978-1-4503-3472-3. DOI: 10.1145/2739480.2754693.

State-of-the-Art in Symbolic Regression

Evolutionary Feature Synthesis

- Very recent – presented in 2015.
- Main idea: population does not consist of models but of features that **collectively** form a single model.
  - Via linear combination.
- Features evolved in a GP-like fashion.
- LASSO for solving the linear regression and as a selection mechanism.

Outline of the algorithm

1. Initial population contains p original features, i.e. $p = \#dimensions$.
   - Always present in the population.
2. Run LASSO to get the linear combination, store the model if it is the best so far.
3. Compose new features and append them to the population.
4. Run LASSO (again), use the information from it to score the features.
5. Keep only q best composed features.
6. If no improvement for N generations or timeout → stop, else go to step 2.

Outline of the algorithm

1. Initial population contains p original features, i.e. $p = \#\mathit{dimensions}$.
    - Always present in the population.
2. Run LASSO to get the linear combination, store the model if it is the best so far.
3. Compose new features and append them to the population.
4. Run LASSO (again), use the information from it to score the features.
5. Keep only q best composed features.
6. If no improvement for N generations or timeout → stop, else go to step 2.

State-of-the-Art in Symbolic Regression

Evolutionary Feature Synthesis

Outline of the algorithm

1. Initial population contains p original features, i.e. $p = \#\mathit{dimensions}$.
   - Always present in the population.
2. Run LASSO to get the linear combination, store the model if it is the best so far.
3. Compose new features and append them to the population.
4. Run LASSO (again), use the information from it to score the features.
5. Keep only q best composed features.
6. If no improvement for N generations or timeout → stop, else go to step 2.

State-of-the-Art in Symbolic Regression

Evolutionary Feature Synthesis

Outline of the algorithm

1. Initial population contains p original features, i.e. $p = \#dimensions$.
    - Always present in the population.
2. Run LASSO to get the linear combination, store the model if it is the best so far.
3. Compose new features and append them to the population.
4. Run LASSO (again), use the information from it to score the features.
5. Keep only q best composed features.
6. If no improvement for N generations or timeout → stop, else go to step 2.

Advanced SR Algorithms
Jan Žegklitz
zegkljan @fel.cvut.cz

Outline of the algorithm

1. Initial population contains p original features, i.e. $p = \#dimensions$.
    - Always present in the population.
2. Run LASSO to get the linear combination, store the model if it is the best so far.
3. Compose new features and append them to the population.
4. Run LASSO (again), use the information from it to score the features.
5. Keep only q best composed features.
6. If no improvement for N generations or timeout → stop, else go to step 2.

Outline of the algorithm

1. Initial population contains p original features, i.e. $p = \#dimensions$.
    - Always present in the population.
2. Run LASSO to get the linear combination, store the model if it is the best so far.
3. Compose new features and append them to the population.
4. Run LASSO (again), use the information from it to score the features.
5. Keep only q best composed features.
6. If no improvement for N generations or timeout → stop, else go to step 2.

State-of-the-Art in Symbolic Regression

Evolutionary Feature Synthesis

Outline of the algorithm

1. Initial population contains p original features, i.e. $p = \#dimensions$.
    - Always present in the population.
2. Run LASSO to get the linear combination, store the model if it is the best so far.
3. Compose new features and append them to the population.
4. Run LASSO (again), use the information from it to score the features.
5. Keep only q best composed features.
6. If no improvement for N generations or timeout → stop, else go to step 2.

Composition of new features

- Create μ new features (i.e. to the following μ-times):
    1. Select 1 resp. 2 feature(s) from population.
        - Using tournament selection based on *feature importance*.
    2. Apply an unary resp. binary operator on the selected feature(s).
    3. Compute Pearson correlation coefficient r between the child and the parent (output values).
    4. If $r > t$ (preset threshold), discard the child.

Composition of new features

- Create μ new features (i.e. to the following μ-times):
  1. Select 1 resp. 2 feature(s) from population.
     - Using tournament selection based on *feature importance*.
  2. Apply an unary resp. binary operator on the selected feature(s).
  3. Compute Pearson correlation coefficient r between the child and the parent (output values).
  4. If $r > t$ (preset threshold), discard the child.

State-of-the-Art in Symbolic Regression

Evolutionary Feature Synthesis

Composition of new features

- Create μ new features (i.e. to the following μ-times):
    1. Select 1 resp. 2 feature(s) from population.
        - Using tournament selection based on *feature importance*.
    2. Apply an unary resp. binary operator on the selected feature(s).
    3. Compute Pearson correlation coefficient r between the child and the parent (output values).
    4. If $r > t$ (preset threshold), discard the child.

Composition of new features

- Create μ new features (i.e. to the following μ-times):
  1. Select 1 resp. 2 feature(s) from population.
     - Using tournament selection based on *feature importance*.
  2. Apply an unary resp. binary operator on the selected feature(s).
  3. Compute Pearson correlation coefficient r between the child and the parent (output values).
  4. If $r > t$ (preset threshold), discard the child.

Composition of new features

- Create μ new features (i.e. to the following μ-times):
  1. Select 1 resp. 2 feature(s) from population.
     - Using tournament selection based on *feature importance*.
  2. Apply an unary resp. binary operator on the selected feature(s).
  3. Compute Pearson correlation coefficient r between the child and the parent (output values).
  4. If $r > t$ (preset threshold), discard the child.

Scoring of features

1. Run LASSO on the extended population.
    - Using pathwise coordinate descent, obtain weights β^λ where $\lambda \in \Gamma$ are decreasing regularization coefficients.
2. Estimate importance of each feature h_j:

$$importance(h_j) = \sum_{\lambda_i \in \Gamma} score(j, \beta^{\lambda_i})$$

$$score(j, \beta^{\lambda_i}) = \begin{cases} R^2_{\lambda_i} & \text{if} \beta^{\lambda_i}_j \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Scoring of features

1. Run LASSO on the extended population.
    - Using pathwise coordinate descent, obtain weights β^{λ} where $\lambda \in \Gamma$ are decreasing regularization coefficients.
2. Estimate importance of each feature h_j:

$$importance(h_j) = \sum_{\lambda_i \in \Gamma} score(j, \beta^{\lambda_i})$$

$$score(j, \beta^{\lambda_i}) = \begin{cases} R^2_{\lambda_i} & \text{if} \beta_j^{\lambda_i} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

State-of-the-Art in Symbolic Regression

Evolutionary Feature Synthesis

Scoring of features

1. Run LASSO on the extended population.
    - Using pathwise coordinate descent, obtain weights β^λ where $\lambda \in \Gamma$ are decreasing regularization coefficients.
2. Estimate importance of each feature h_j:

$$importance(h_j) = \sum_{\lambda_i \in \Gamma} score(j, \beta^{\lambda_i})$$

$$score(j, \beta^{\lambda_i}) = \begin{cases} R^2_{\lambda_i} & \text{if} \beta_j^{\lambda_i} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Notes

- No tree representation, only the output values → **very fast**.
    - Basically just $N \times (p + q)$ matrix.
- Is at least as good as pure LASSO fit on the original variables.

State-of-the-Art in Symbolic Regression

Evolutionary Feature Synthesis

Notes

- No tree representation, only the output values → **very fast**.
  - Basically just $N \times (p + q)$ matrix.
- Is at least as good as pure LASSO fit on the original variables.

Notes

- No tree representation, only the output values → **very fast**.
  - Basically just $N \times (p + q)$ matrix.
- Is at least as good as pure LASSO fit on the original variables.

FFX
Fast Function eXtraction[4]

---

[4] Trent McConaghy. "FFX: Fast, Scalable, Deterministic Symbolic Regression Technology". English. In: *Genetic Programming Theory and Practice IX*. ed. by Rick Riolo, Ekaterina Vladislavleva, and Jason H. Moore. Genetic and Evolutionary Computation. Springer New York, 2011, pp. 235–260. ISBN: 978-1-4614-1769-9. DOI: `10.1007/978-1-4614-1770-5_13`.

State-of-the-Art in Symbolic Regression

Fast Function Extraction

- Non-evolutionary, deterministic algorithm.
- Main idea: exhaustively generate **a lot** of basis functions, reduce them using regularized regression.

Outline of the algorithm

1. Generate univariate bases.
   - x^n, $op(x^n)$ (n is from a predefined set of constants, op is a function from a predefined set of functions)
2. Generate interacting-variable bases.
   - All products of two univariate bases except those of kind $op(\cdot) \cdot op(\cdot)$.
3. Run pathwise regularized learning (elastic net) on all the bases with decreasing regularizaiton coefficient.
   - Stop when at most N_{max} bases have nonzero coefficient.
4. Non-dominated filtering of all acquired models.
   - Tradeoff between accuracy and # of bases.

Outline of the algorithm

1. Generate univariate bases.
    - $x^n, op(x^n)$ (n is from a predefined set of constants, op is a function from a predefined set of functions)
2. Generate interacting-variable bases.
    - All products of two univariate bases except those of kind $op(\cdot) \cdot op(\cdot)$.
3. Run pathwise regularized learning (elastic net) on all the bases with decreasing regularizaiton coefficient.
    - Stop when at most N_{max} bases have nonzero coefficient.
4. Non-dominated filtering of all acquired models.
    - Tradeoff between accuracy and # of bases.

State-of-the-Art in Symbolic Regression

Fast Function Extraction

Outline of the algorithm

1. Generate univariate bases.
   - $x^n, op(x^n)$ (n is from a predefined set of constants, op is a function from a predefined set of functions)
2. Generate interacting-variable bases.
   - All products of two univariate bases except those of kind $op(\cdot) \cdot op(\cdot)$.
3. Run pathwise regularized learning (elastic net) on all the bases with decreasing regularizaiton coefficient.
   - Stop when at most N_{max} bases have nonzero coefficient.
4. Non-dominated filtering of all acquired models.
   - Tradeoff between accuracy and # of bases.

State-of-the-Art in Symbolic Regression

Fast Function Extraction

Outline of the algorithm

1. Generate univariate bases.
   - $x^n, op(x^n)$ (n is from a predefined set of constants, op is a function from a predefined set of functions)
2. Generate interacting-variable bases.
   - All products of two univariate bases except those of kind $op(\cdot) \cdot op(\cdot)$.
3. Run pathwise regularized learning (elastic net) on all the bases with decreasing regularizaiton coefficient.
   - Stop when at most N_{max} bases have nonzero coefficient.
4. Non-dominated filtering of all acquired models.
   - Tradeoff between accuracy and # of bases.

State-of-the-Art in Symbolic Regression

Fast Function Extraction

Outline of the algorithm

1. Generate univariate bases.
    - $x^n, op(x^n)$ (n is from a predefined set of constants, op is a function from a predefined set of functions)
2. Generate interacting-variable bases.
    - All products of two univariate bases except those of kind $op(\cdot) \cdot op(\cdot)$.
3. Run pathwise regularized learning (elastic net) on all the bases with decreasing regularizaiton coefficient.
    - Stop when at most N_{max} bases have nonzero coefficient.
4. Non-dominated filtering of all acquired models.
    - Tradeoff between accuracy and # of bases.

State-of-the-Art in Symbolic Regression

Fast Function Extraction

Notes

- Deterministic.
- Complexity: $O(N \cdot n^4)$, N is number of training samples, n is number of variables.
    - Can be reduced to $O(N \cdot n^2)$ by slight modification of basis generation.
- Uses *hinge functions*: $\max(0, x - thr)$ and $\max(0, thr - x)$, thr is a threshold, several values uniformly distributed in the range of training data variables.
- In between steps 2 and 3 a “rational functions trick” is applied.
    - Enables to fit rational functions with linear regression.
- The whole algorithm is run multiple times with different capabilites turned on and off – non-dominated front is produced from all of these runs.

Notes

- Deterministic.
- Complexity: $O(N \cdot n^4)$, N is number of training samples, n is number of variables.
  - Can be reduced to $O(N \cdot n^2)$ by slight modification of basis generation.
- Uses *hinge functions*: $\max(0, x - thr)$ and $\max(0, thr - x)$, thr is a threshold, several values uniformly distributed in the range of training data variables.
- In between steps 2 and 3 a "rational functions trick" is applied.
  - Enables to fit rational functions with linear regression.
- The whole algorithm is run multiple times with different capabilites turned on and off – non-dominated front is produced from all of these runs.

Notes

- Deterministic.
- Complexity: $O(N \cdot n^4)$, N is number of training samples, n is number of variables.
    - Can be reduced to $O(N \cdot n^2)$ by slight modification of basis generation.
- Uses *hinge functions*: $\max(0, x - thr)$ and $\max(0, thr - x)$, thr is a threshold, several values uniformly distributed in the range of training data variables.
- In between steps 2 and 3 a “rational functions trick” is applied.
    - Enables to fit rational functions with linear regression.
- The whole algorithm is run multiple times with different capabilites turned on and off – non-dominated front is produced from all of these runs.

State-of-the-Art in Symbolic Regression
Fast Function Extraction

Notes

- Deterministic.
- Complexity: $O(N \cdot n^4)$, N is number of training samples, n is number of variables.
    - Can be reduced to $O(N \cdot n^2)$ by slight modification of basis generation.
- Uses *hinge functions*: $\max(0, x - thr)$ and $\max(0, thr - x)$, thr is a threshold, several values uniformly distributed in the range of training data variables.
- In between steps 2 and 3 a “rational functions trick” is applied.
    - Enables to fit rational functions with linear regression.
- The whole algorithm is run multiple times with different capabilites turned on and off – non-dominated front is produced from all of these runs.

State-of-the-Art in Symbolic Regression

Fast Function Extraction

Notes

- Deterministic.
- Complexity: $O(N \cdot n^4)$, N is number of training samples, n is number of variables.
    - Can be reduced to $O(N \cdot n^2)$ by slight modification of basis generation.
- Uses *hinge functions*: $\max(0, x - thr)$ and $\max(0, thr - x)$, thr is a threshold, several values uniformly distributed in the range of training data variables.
- In between steps 2 and 3 a "rational functions trick" is applied.
    - Enables to fit rational functions with linear regression.
- The whole algorithm is run multiple times with different capabilites turned on and off – non-dominated front is produced from all of these runs.

State-of-the-Art in Symbolic Regression

Fast Function Extraction

Notes

- Deterministic.
- Complexity: $O(N \cdot n^4)$, N is number of training samples, n is number of variables.
    - Can be reduced to $O(N \cdot n^2)$ by slight modification of basis generation.
- Uses *hinge functions*: $\max(0, x - thr)$ and $\max(0, thr - x)$, thr is a threshold, several values uniformly distributed in the range of training data variables.
- In between steps 2 and 3 a "rational functions trick" is applied.
    - Enables to fit rational functions with linear regression.
- The whole algorithm is run multiple times with different capabilites turned on and off – non-dominated front is produced from all of these runs.

Improvements

- Linear regression helps a lot but does not deal with the inside of the models.
- Inner constants are usually tuned “blindly”.
    - Some kind of (random) manipulation + selection.

Backpropagation of Error

Improvements

Backpropagation of Error

Weighted trees

- Introduce inner constants (weights) to **every** non-terminal node:

  $$f_n(x) \to f_n(x^*) = f_n(w_1 x + w_2)$$

  where f_n is a function of some non-terminal node.
- Similarly for functions of multiple arguments – a and b for each of them.
  - In $+$ use only multiplicative constants.
  - In $\cdot$ use only additive constants.
- Example:

  $$f(\mathbf{x}) = \sin^2(x_1 \cdot x_2)$$

  becomes

  $$f(\mathbf{x}) = (w_1 \cdot \sin(w_3(x_1 + w_5) \cdot (x_2 + w_6) + w_4) + w_2)^2$$

Weighted trees

- Introduce inner constants (weights) to **every** non-terminal node:

  $$f_n(x) \to f_n(x^*) = f_n(w_1 x + w_2)$$

  where f_n is a function of some non-terminal node.
- Similarly for functions of multiple arguments – a and b for each of them.
  - In $+$ use only multiplicative constants.
  - In $\cdot$ use only additive constants.
- Example:

  $$f(\mathbf{x}) = \sin^2(x_1 \cdot x_2)$$

  becomes

  $$f(\mathbf{x}) = (w_1 \cdot \sin(w_3(x_1 + w_5) \cdot (x_2 + w_6) + w_4) + w_2)^2$$

Improvements

Backpropagation of Error

Weighted trees

- Introduce inner constants (weights) to **every** non-terminal node:

  $$f_n(x) \to f_n(x^*) = f_n(w_1 x + w_2)$$

  where f_n is a function of some non-terminal node.
- Similarly for functions of multiple arguments – a and b for each of them.
  - In $+$ use only multiplicative constants.
  - In $\cdot$ use only additive constants.
- Example:

  $$f(\mathbf{x}) = \sin^2(x_1 \cdot x_2)$$

  becomes

  $$f(\mathbf{x}) = (w_1 \cdot \sin(w_3(x_1 + w_5) \cdot (x_2 + w_6) + w_4) + w_2)^2$$

Improvements

Backpropagation of Error

Weighted trees

- Introduce inner constants (weights) to **every** non-terminal node:

  $$f_n(x) \to f_n(x^*) = f_n(w_1 x + w_2)$$

  where f_n is a function of some non-terminal node.
- Similarly for functions of multiple arguments – a and b for each of them.
  - In $+$ use only multiplicative constants.
  - In $\cdot$ use only additive constants.
- Example:

  $$f(\mathbf{x}) = \sin^2(x_1 \cdot x_2)$$

  becomes

  $$f(\mathbf{x}) = (w_1 \cdot \sin(w_3(x_1 + w_5) \cdot (x_2 + w_6) + w_4) + w_2)^2$$

Backpropagation of error

- Use the Backpropagation algorithm from ANNs to determine $\frac{\partial C}{\partial w_i}$ where C is some cost function.
- For root node $f_{root}(x^*) = f_{root}(w_1 x + w_2)$:

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial f_{root}}{\partial x^*}$$

- For non-root node $f(x^*) = f(w_1 x + w_2)$

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial w_2^{parent}} \cdot w_1^{parent} \cdot \frac{\partial f}{\partial x^*}$$

- For any node $f(x^*) = f(w_1 x + w_2)$

$$\frac{\partial C}{\partial w_1} = x \cdot \frac{\partial C}{\partial w_2}$$

Improvements

Backpropagation of Error

Backpropagation of error

- Use the Backpropagation algorithm from ANNs to determine $\frac{\partial C}{\partial w_i}$ where C is some cost function.
- For root node $f_{root}(x^*) = f_{root}(w_1 x + w_2)$:

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial f_{root}}{\partial x^*}$$

- For non-root node $f(x^*) = f(w_1 x + w_2)$

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial w_2^{parent}} \cdot w_1^{parent} \cdot \frac{\partial f}{\partial x^*}$$

- For any node $f(x^*) = f(w_1 x + w_2)$

$$\frac{\partial C}{\partial w_1} = x \cdot \frac{\partial C}{\partial w_2}$$

Backpropagation of error

- Use the Backpropagation algorithm from ANNs to determine $\frac{\partial C}{\partial w_i}$ where C is some cost function.
- For root node $f_{root}(x^*) = f_{root}(w_1 x + w_2)$:

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial f_{root}}{\partial x^*}$$

- For non-root node $f(x^*) = f(w_1 x + w_2)$

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial w_2^{parent}} \cdot w_1^{parent} \cdot \frac{\partial f}{\partial x^*}$$

- For any node $f(x^*) = f(w_1 x + w_2)$

$$\frac{\partial C}{\partial w_1} = x \cdot \frac{\partial C}{\partial w_2}$$

Backpropagation of error

- Use the Backpropagation algorithm from ANNs to determine $\frac{\partial C}{\partial w_i}$ where C is some cost function.
- For root node $f_{root}(x^*) = f_{root}(w_1 x + w_2)$:

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial f_{root}}{\partial x^*}$$

- For non-root node $f(x^*) = f(w_1 x + w_2)$

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial w_2^{parent}} \cdot w_1^{parent} \cdot \frac{\partial f}{\partial x^*}$$

- For any node $f(x^*) = f(w_1 x + w_2)$

$$\frac{\partial C}{\partial w_1} = x \cdot \frac{\partial C}{\partial w_2}$$

Improvements
Backpropagation of Error

Backpropagation of error

- Use the Backpropagation algorithm from ANNs to determine $\frac{\partial C}{\partial w_i}$ where C is some cost function.
- For root node $f_{root}(x^*) = f_{root}(w_1 x + w_2)$:

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial f_{root}}{\partial x^*}$$

- For non-root node $f(x^*) = f(w_1 x + w_2)$

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial w_2^{parent}} \cdot w_1^{parent} \cdot \frac{\partial f}{\partial x^*}$$

- For any node $f(x^*) = f(w_1 x + w_2)$

$$\frac{\partial C}{\partial w_1} = x \cdot \frac{\partial C}{\partial w_2}$$

Improvements
Backpropagation of Error

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
  1. Evaluate the tree (forward pass).
  2. Fit the outputs via LR (incl. an additive constant).
  3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
  4. Update weights.
  5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, "insert" its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node's $\frac{\partial C}{\partial w_2}$.

Improvements

Backpropagation of Error

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
  1. Evaluate the tree (forward pass).
  2. Fit the outputs via LR (incl. an additive constant).
  3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
  4. Update weights.
  5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, "insert" its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node's $\frac{\partial C}{\partial w_2}$.

Improvements

Backpropagation of Error

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
  1. Evaluate the tree (forward pass).
  2. Fit the outputs via LR (incl. an additive constant).
  3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
  4. Update weights.
  5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, “insert” its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node’s $\frac{\partial C}{\partial w_2}$.

Improvements

Backpropagation of Error

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
  1. Evaluate the tree (forward pass).
  2. Fit the outputs via LR (incl. an additive constant).
  3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
  4. Update weights.
  5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, "insert" its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node's $\frac{\partial C}{\partial w_2}$.

Improvements
Backpropagation of Error

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
  1. Evaluate the tree (forward pass).
  2. Fit the outputs via LR (incl. an additive constant).
  3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
  4. Update weights.
  5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, “insert” its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node's $\frac{\partial C}{\partial w_2}$.

Improvements

Backpropagation of Error

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
  1. Evaluate the tree (forward pass).
  2. Fit the outputs via LR (incl. an additive constant).
  3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
  4. Update weights.
  5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, "insert" its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node's $\frac{\partial C}{\partial w_2}$.

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
  1. Evaluate the tree (forward pass).
  2. Fit the outputs via LR (incl. an additive constant).
  3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
  4. Update weights.
  5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, "insert" its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node's $\frac{\partial C}{\partial w_2}$.

Improvements

Backpropagation of Error

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
  1. Evaluate the tree (forward pass).
  2. Fit the outputs via LR (incl. an additive constant).
  3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
  4. Update weights.
  5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, "insert" its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node's $\frac{\partial C}{\partial w_2}$.

Improvements

Backpropagation of Error

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
    1. Evaluate the tree (forward pass).
    2. Fit the outputs via LR (incl. an additive constant).
    3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
    4. Update weights.
    5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, "insert" its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node's $\frac{\partial C}{\partial w_2}$.

Improvements

Backpropagation of Error

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
    1. Evaluate the tree (forward pass).
    2. Fit the outputs via LR (incl. an additive constant).
    3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
    4. Update weights.
    5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, "insert" its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node's $\frac{\partial C}{\partial w_2}$.

Improvements

Backpropagation of Error

Weighting the root node

- The output of the root node is not weighted.
- Linear regression to the rescue!
- Follow this pattern:
    1. Evaluate the tree (forward pass).
    2. Fit the outputs via LR (incl. an additive constant).
    3. Determine $\frac{\partial C}{\partial w_i}$ by backpropagation.
    4. Update weights.
    5. Repeat as desired.
- Can easily work with multiple genes, i.e. in an MGGP fashion.
- Just consider the linear combination as a function applied to the output of each tree, "insert" its derivative in between $\frac{\partial C}{\partial \hat{y}}$ and $\frac{\partial f_{root}}{\partial x^*}$ when computing each of the root node's $\frac{\partial C}{\partial w_2}$.

Improvements

Backpropagation of Error

Updating the weights

- Once $\frac{\partial C}{\partial w_i}$ are known, any updating algorithm can be used.
- Rprop family of updating algorithms.
  - Only $\operatorname{sgn}\left(\frac{\partial C}{\partial w_i}\right)$ are used instead of the values.
    - Numerically much safer as beasts like $\mathrm{e}^{\mathrm{e}^{\mathrm{e}^{x}}}$ can easily appear.
  - Keeps and adaptively changes magnitude of update step per-weight.
  - Easy to implement.

Improvements
Backpropagation of Error

Open issues

- Backpropagation and update is very computationally demanding.
- Number of update steps.
  - The less backpropagation there is the faster the model evaluation and fitting is.
  - The more backpropagation there is the more accurate models are evolved.
  - Make the number of steps dependent on tree size?
- Which bases to update?
  - Dependent on the coefficient form the linear fit?
- Which nodes (weights) to update?
  - All?
  - Up to some depth?
  - Dependent on the magnitude of $\frac{\partial C}{\partial w_i}$?

Improvements

Backpropagation of Error

Open issues

- Backpropagation and update is very computationally demanding.
- Number of update steps.
  - The less backpropagation there is the faster the model evaluation and fitting is.
  - The more backpropagation there is the more accurate models are evolved.
  - Make the number of steps dependent on tree size?
- Which bases to update?
  - Dependent on the coefficient form the linear fit?
- Which nodes (weights) to update?
  - All?
  - Up to some depth?
  - Dependent on the magnitude of $\frac{\partial C}{\partial w_i}$?

Improvements

Backpropagation of Error

Open issues

- Backpropagation and update is very computationally demanding.
- Number of update steps.
  - The less backpropagation there is the faster the model evaluation and fitting is.
  - The more backpropagation there is the more accurate models are evolved.
  - Make the number of steps dependent on tree size?
- Which bases to update?
  - Dependent on the coefficient form the linear fit?
- Which nodes (weights) to update?
  - All?
  - Up to some depth?
  - Dependent on the magnitude of $\frac{\partial C}{\partial w_i}$?

Improvements

Backpropagation of Error

Open issues

- Backpropagation and update is very computationally demanding.
- Number of update steps.
  - The less backpropagation there is the faster the model evaluation and fitting is.
  - The more backpropagation there is the more accurate models are evolved.
  - Make the number of steps dependent on tree size?
- Which bases to update?
  - Dependent on the coefficient form the linear fit?
- Which nodes (weights) to update?
  - All?
  - Up to some depth?
  - Dependent on the magnitude of $\frac{\partial C}{\partial w_i}$?

Improvements

Backpropagation of Error

Open issues

- Backpropagation and update is very computationally demanding.
- Number of update steps.
    - The less backpropagation there is the faster the model evaluation and fitting is.
    - The more backpropagation there is the more accurate models are evolved.
    - Make the number of steps dependent on tree size?
- Which bases to update?
    - Dependent on the coefficient form the linear fit?
- Which nodes (weights) to update?
    - All?
    - Up to some depth?
    - Dependent on the magnitude of $\frac{\partial C}{\partial w_i}$?

Improvements

Backpropagation of Error

Open issues

- Redundancy.
    - Some weighs can be redundant.
    - Is it a problem?
- Models become (a little bit) messier.

Improvements

Backpropagation of Error

Open issues

- Redundancy.
    - Some weighs can be redundant.
    - Is it a problem?
- Models become (a little bit) messier.

Improvements
Feature Space Transformation

Feature Space Transformation

Improvements

Feature Space Transformation

- Do not use the “vanilla” feature space but some different, transformed space.
  - Make it easier for the genetics.
- Replace each variable leaf node $f_i(\mathbf{x}) = x_i$ by

$$f_i(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}_i + w_{i,0}$$

with *some* $\mathbf{w}$ and $w_{i,0}$.
- Each leaf node becomes some distinct affine transformation of $\mathbf{x}$.
- Learn the coefficients by backpropagation as well!
  - I.e. backpropagate one level more.

Improvements

Feature Space Transformation

- Do not use the “vanilla” feature space but some different, transformed space.
  - Make it easier for the genetics.
- Replace each variable leaf node $f_i(\mathbf{x}) = x_i$ by

$$f_i(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}_i + w_{i,0}$$

with *some* $\mathbf{w}$ and $w_{i,0}$.

- Each leaf node becomes some distinct affine transformation of x.
- Learn the coefficients by backpropagation as well!
  - I.e. backpropagate one level more.

Improvements

Feature Space Transformation

- Do not use the “vanilla” feature space but some different, transformed space.
    - Make it easier for the genetics.
- Replace each variable leaf node $f_i(\mathbf{x}) = x_i$ by

  $$f_i(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}_i + w_{i,0}$$

  with *some* $\mathbf{w}$ and $w_{i,0}$.
- Each leaf node becomes some distinct affine transformation of $\mathbf{x}$.
- Learn the coefficients by backpropagation as well!
    - I.e. backpropagate one level more.

Improvements

Feature Space Transformation

- Do not use the "vanilla" feature space but some different, transformed space.
  - Make it easier for the genetics.
- Replace each variable leaf node $f_i(\mathbf{x}) = x_i$ by

$$f_i(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}_i + w_{i,0}$$

with *some* $\mathbf{w}$ and $w_{i,0}$.

- Each leaf node becomes some distinct affine transformation of $\mathbf{x}$.
- Learn the coefficients by backpropagation as well!
  - I.e. backpropagate one level more.

Improvements

Feature Space Transformation

- Do not use the "vanilla" feature space but some different, transformed space.
    - Make it easier for the genetics.
- Replace each variable leaf node $f_i(\mathbf{x}) = x_i$ by

$$f_i(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}_i + w_{i,0}$$

with *some* $\mathbf{w}$ and $w_{i,0}$.

- Each leaf node becomes some distinct affine transformation of $\mathbf{x}$.
- Learn the coefficients by backpropagation as well!
    - I.e. backpropagate one level more.

Improvements

Feature Space Transformation

WIP, Open issues

- Not a nice transformation of the feature space – each leaf is a different one.
- Unify the transformations for each "index" of the feature space.
    - All leaves would use the same $\mathbf{w}_i$ and $w_{i,0}$ for each i.
    - Effectively moves from trees to DAGs.
- That would produce a single affine transformation of the feature space.
- Should we include the original features as well?

Improvements
Feature Space Transformation

WIP, Open issues

- Not a nice transformation of the feature space – each leaf is a different one.
- Unify the transformations for each “index” of the feature space.
    - All leaves would use the same $\mathbf{w}_i$ and $w_{i,0}$ for each i.
    - Effectively moves from trees to DAGs.
- That would produce a single affine transformation of the feature space.
- Should we include the original features as well?

Improvements

Feature Space Transformation

WIP, Open issues

- Not a nice transformation of the feature space – each leaf is a different one.
- Unify the transformations for each "index" of the feature space.
    - All leaves would use the same $\mathbf{w}_i$ and $w_{i,0}$ for each i.
    - Effectively moves from trees to DAGs.
- That would produce a single affine transformation of the feature space.
- Should we include the original features as well?

Improvements

Feature Space Transformation

WIP, Open issues

- Not a nice transformation of the feature space – each leaf is a different one.
- Unify the transformations for each "index" of the feature space.
  - All leaves would use the same $\mathbf{w}_i$ and $w_{i,0}$ for each i.
  - Effectively moves from trees to DAGs.
- That would produce a single affine transformation of the feature space.
- Should we include the original features as well?

Improvements

Feature Space Transformation

WIP, Open issues

- Not a nice transformation of the feature space – each leaf is a different one.
- Unify the transformations for each “index” of the feature space.
  - All leaves would use the same $\mathbf{w}_i$ and $w_{i,0}$ for each i.
  - Effectively moves from trees to DAGs.
- That would produce a single affine transformation of the feature space.
- Should we include the original features as well?

Boosting

Improvements

Boosting

Possibilites for boosting

- "High-level" boosting – combine multiple runs of an algorithm.
  - Repeatedly run a SR algorithm of choice, sequentially producing models.
  - Combine models in a boosting fashion.
- "Mid-level" boosting – combine models inside an algorithm.
  - Evolve single-gene models, like in original GP.
  - For each solution, at some point, fix the evolved model and start a new one (new gene).
  - Keep adding genes as evolution progresses.
- Boosting with local optimization

Improvements

Boosting

Possibilites for boosting

- "High-level" boosting – combine multiple runs of an algorithm.
    - Repeatedly run a SR algorithm of choice, sequentially producing models.
    - Combine models in a boosting fashion.
- "Mid-level" boosting – combine models inside an algorithm.
    - Evolve single-gene models, like in original GP.
    - For each solution, at some point, fix the evolved model and start a new one (new gene).
    - Keep adding genes as evolution progresses.
- Boosting with local optimization

Improvements

Boosting

Possibilites for boosting

- “High-level” boosting – combine multiple runs of an algorithm.
    - Repeatedly run a SR algorithm of choice, sequentially producing models.
    - Combine models in a boosting fashion.
- “Mid-level” boosting – combine models inside an algorithm.
    - Evolve single-gene models, like in original GP.
    - For each solution, at some point, fix the evolved model and start a new one (new gene).
    - Keep adding genes as evolution progresses.
- Boosting with local optimization

Improvements

Boosting

Possibilites for boosting

- "High-level" boosting – combine multiple runs of an algorithm.
  - Repeatedly run a SR algorithm of choice, sequentially producing models.
  - Combine models in a boosting fashion.
- "Mid-level" boosting – combine models inside an algorithm.
  - Evolve single-gene models, like in original GP.
  - For each solution, at some point, fix the evolved model and start a new one (new gene).
  - Keep adding genes as evolution progresses.
- Boosting with local optimization

Boosting with local optimization

- Multiple bases + local optimization = problem.
- Which bases should be optimized?
- Linear coefficients do not necessarily indicate a potential for improvement.
  - A really bad base may have big coefficient just because the others are just not optimized enough.
  - A really good base may have a low (or zero) coefficient just because the non-linear guts are just a little bit off.
- Use boosting as a replacement for LR.
- Optimize one base at a time, sequentially.
  - But which base should come fist?

Improvements

Boosting

Boosting with local optimization

- Multiple bases + local optimization = problem.
- Which bases should be optimized?
- Linear coefficients do not necessarily indicate a potential for improvement.
    - A really bad base may have big coefficient just because the others are just not optimized enough.
    - A really good base may have a low (or zero) coefficient just because the non-linear guts are just a little bit off.
- Use boosting as a replacement for LR.
- Optimize one base at a time, sequentially.
    - But which base should come fist?

Boosting with local optimization

- Multiple bases + local optimization = problem.
- Which bases should be optimized?
- Linear coefficients do not necessarily indicate a potential for improvement.
    - A really bad base may have big coefficient just because the others are just not optimized enough.
    - A really good base may have a low (or zero) coefficient just because the non-linear guts are just a little bit off.
- Use boosting as a replacement for LR.
- Optimize one base at a time, sequentially.
    - But which base should come fist?

Improvements
Boosting

Boosting with local optimization

- Multiple bases + local optimization = problem.
- Which bases should be optimized?
- Linear coefficients do not necessarily indicate a potential for improvement.
  - A really bad base may have big coefficient just because the others are just not optimized enough.
  - A really good base may have a low (or zero) coefficient just because the non-linear guts are just a little bit off.
- Use boosting as a replacement for LR.
- Optimize one base at a time, sequentially.
  - But which base should come fist?

Improvements

Boosting

Boosting with local optimization

- Multiple bases + local optimization = problem.
- Which bases should be optimized?
- Linear coefficients do not necessarily indicate a potential for improvement.
    - A really bad base may have big coefficient just because the others are just not optimized enough.
    - A really good base may have a low (or zero) coefficient just because the non-linear guts are just a little bit off.
- Use boosting as a replacement for LR.
- Optimize one base at a time, sequentially.
    - But which base should come fist?

Improvements
Boosting

Boosting with local optimization

- Multiple bases + local optimization = problem.
- Which bases should be optimized?
- Linear coefficients do not necessarily indicate a potential for improvement.
  - A really bad base may have big coefficient just because the others are just not optimized enough.
  - A really good base may have a low (or zero) coefficient just because the non-linear guts are just a little bit off.
- Use boosting as a replacement for LR.
- Optimize one base at a time, sequentially.
  - But which base should come fist?

Summary

- Symbolic Regression – derive mathematical expressions out of data.
- State-of-the-Art in SR
  - MGGP – multiple bases per individual, combined via LR
  - EFS – population of features instead of population of models + LASSO
  - FFX – deterministic, exhaustive generation of bases, Elastic net.
- Improvements
  - Backpropagation to tune the inside of the models.
  - Transformation of feature space.
  - Boosting.

Q&A

Thanks for your attention.

Questions?