Simpler Software Analytics: When? When not?

Wei Fu
From Last Exam

Why study simplicity? cost, speed

When this won’t work? ε-Dominance

What’s the difference between SE/general data mining? under-exploited simplicities

Software analytics **should** be easier.
Software analytics **can** be easier.
But it can be **very hard** to show it can be easier.
And, sometimes, it can be **too easy**.

**My Thesis:**

**Future work:**

When to be simpler.
My Thesis

Software analytics should be easier.

Software analytics can be easier. [Fu et al. IST 2016]

But it can be very hard to show it can be easier. [Fu et al. FSE 2017 A]

And, sometimes, it can be too easy. [Fu et al. FSE 2017 B]
Software Analytics

Effort estimation

Software Defect Prediction

Many Others.....
A Typical Software Analytics Framework

- Data
- Learner
- Model

Training process
Data Mining Static Code Attributes to Learn Defect Predictors
Tim Menzies, Member, IEEE, Jeremy Greenwald, and Art Frank

Abstract—The value of using static code attributes to learn defect predictors has been widely debated. Prior work has explored issues like the merits of "McCabe versus Halstead versus lines of code counts" for generating defect predictors. We show here that such debates are irrelevant since how the attributes are used to build predictors is much more important than which particular attributes are used. Also, contrary to prior pessimism, we show that such defect predictors are demonstrably useful and, on the data studied here, yield predictions with a mean probability of detection of 71 percent and mean false alarm rates of 29 percent. These predictors would be useful for prioritizing a resource-bound exploration of code that has yet to be inspected.

Index Terms—Data mining defect prediction, McCabe, Halstead, artificial intelligence, empirical, naive Bayes.

Decision Tree, Naive Bayes
Data Mining Static Code Attributes to Learn Defect Predictors

Tim Menzies, Member, IEEE, Jeremy Greenwald, and Art Frank

Abstract—The value of using static code attributes to learn defect predictions has been widely debated. Prior work has explored issues like the merits of "McCabe versus Halstead versus lines of code counts" for generating defect predictors. We show here that such debates are irrelevant since the attributes are used to build predictors is much more important than which particular attributes are used. Also, contrary to prior pessimism, we show that such defect predictors are demonstrably useful and, on the data studied here, yield predictors with a mean probability of detection of 71 percent and mean false alarm rates of 20 percent. These predictors would be useful for prioritising a resource-bound explosion of code that has yet to be inspected.

Index Terms—Data mining defect prediction, McCabe, Halstead, artificial intelligence, empirical, naive Bayes.

Decision Tree, Naive Bayes

Deep Learning
Simpler or more complex software analytics?
My Thesis

Software analytics **should** be easier.

Software analytics **can** be easier. [Fu et al, IST 2016]

But it can be **very hard** to show it can be easier. [Fu et al, FSE 2017 A]

And, sometimes, it can be **too easy**. [Fu et al, FSE 2017 B]
Why Easier: Cost & Speed

Dr. Mark Harman@UCL
FSE’13: Wang et al\cite{Wang13}
Wait Years of CPU time

Dr. Tien N. Nguyen@UTDallas
ASE’15: Lam et al\cite{Lam15}
Wait Weeks of CPU time

Dr. Sung Kim@HKUST
FSE’16: Gu et al\cite{Gu16}
Wait 10 Days of GPU time

Dr. Tim Menzies@NCSU
FSE’17: Fu et al\cite{Fu17}
Wait 10 minutes of CPU time
Dr. Devanbu@UC Davis

FSE’17: Hellendoorn et al [Hellendoorn17]

Simpler, faster methods, complex DL is not always the best.

Why Easier: Cost & Speed

Are Deep Neural Networks the Best Choice for Modeling Source Code?

Vincent J. Hellendoorn
Computer Science Dept., UC Davis
Davis, CA, USA 95616
vhellendoorn@ucdavis.edu

Premkumar Devanbu
Computer Science Dept., UC Davis
Davis, CA, USA 95616
ptdevanbu@ucdavis.edu

ABSTRACT
Current statistical language modeling techniques, including deep-learning based models, have proven to be quite effective for source code. We argue here that the special properties of source code can be exploited for further improvements. In this work, we enhance established language modeling approaches to handle the special challenges of modeling source code, such as: frequent changes, larger, changing vocabularies, deeply nested scopes, etc. We present a fast, nested language modeling toolkit specifically designed for software, with the ability to add & remove text, and mix & swap out many models. Specifically, we improve upon prior cache-modeling work and present a model with a much more expansive, multi-level notion of locality that we show to be well-suited for modeling software. We present results on varying corpora in comparison with traditional N-gram, as well as RNN, and LSTM deep-learning language models, and release all our source code for public use. Our evaluations suggest that carefully adapting N-gram models for source code can yield performance that surpasses even RNN and LSTM based deep-learning models.

Statistical models from NLP, estimated over the large volumes of code available in GitHub, have led to a wide range of applications in software engineering. High-performance language models are widely used to improve performance on NLP-related tasks, such as translation, speech-recognition, and query completion; similarly, better language models for source code are known to improve performance in tasks such as code completion [15]. Developing models that can address (and exploit) the special properties of source code is central to this enterprise.

Language models for NLP have been developed over decades, and are highly refined; however, many of the design decisions baked-into modern NLP language models are finely-wrought to exploit properties of natural language corpora. These properties aren’t always relevant to source code, so that adapting NLP models to the special features of source code can be helpful. We discuss 3 important issues and their modeling implications in detail below.

Unlimited Vocabulary Code and NL can both have an unbounded vocabulary; however, in NL corpora, the vocabulary usually saturates quickly when scanning through a large NL corpus, pretty soon, one rarely encounters new words. New proper nouns (people
Why Easier: Cost

Local hardware:

- **AlphaGo**: 1920 CPUs and 280 GPUs*, $3000 electric bill per game

---

* [https://en.wikipedia.org/wiki/AlphaGo](https://en.wikipedia.org/wiki/AlphaGo)
My Thesis

Software analytics should be easier.

Software analytics can be easier. [Fu et al, IST 2016]

But it can be very hard to show it can be easier. [Fu et al, FSE 2017 A]

And, sometimes, it can be too easy. [Fu et al, FSE 2017 B]
Tuning is Ignored in SE

- DE: 2%
- Grid Search: 4%
- Manually Tuning: 2%
- Just Mention Tuning: 14%
- Never Mention Tuning: 78%

Literature Review On Defect Prediction*

Tuning Defect Predictors

Training process

Data -> Learner -> DE tuner -> Model
Differential Evolution

Population = Pick N options at random # e.g. N = 10

M times repeat : # e.g. M = 5

for Parent in Population:

- Select a, b, c = three other items in population.
- Candidate = a + f*(b-c) # ish
- if Candidate “better”, replace Parent.
Tuning Defect Predictors

![Graph of precision and F metrics for different methods across sorted datasets.](image-url)
# Time Cost of Tuning Defect Predictors

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Tuned_CART precision</th>
<th>Tuned_CART F</th>
<th>Naive_CART precision</th>
<th>Naive_CART F</th>
<th>Tuned_RanFst precision</th>
<th>Tuned_RanFst F</th>
<th>Naive_RanFst precision</th>
<th>Naive_RanFst F</th>
</tr>
</thead>
<tbody>
<tr>
<td>antV0</td>
<td>5.08</td>
<td>3.52</td>
<td>0.08</td>
<td>0.08</td>
<td>9.78</td>
<td>9.89</td>
<td>0.20</td>
<td>0.17</td>
</tr>
<tr>
<td>antV1</td>
<td>6.52</td>
<td>6.18</td>
<td>0.12</td>
<td>0.12</td>
<td>14.13</td>
<td>13.39</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>antV2</td>
<td>9.00</td>
<td>8.79</td>
<td>0.24</td>
<td>0.18</td>
<td>16.75</td>
<td>27.56</td>
<td>0.44</td>
<td>0.36</td>
</tr>
<tr>
<td>camelV0</td>
<td>12.68</td>
<td>17.00</td>
<td>0.24</td>
<td>0.28</td>
<td>28.49</td>
<td>22.52</td>
<td>0.34</td>
<td>0.41</td>
</tr>
<tr>
<td>camelV1</td>
<td>17.13</td>
<td>31.92</td>
<td>0.27</td>
<td>0.28</td>
<td>33.96</td>
<td>37.00</td>
<td>0.77</td>
<td>0.85</td>
</tr>
<tr>
<td>ivy</td>
<td>4.26</td>
<td>4.72</td>
<td>0.07</td>
<td>0.08</td>
<td>8.89</td>
<td>10.39</td>
<td>0.19</td>
<td>0.21</td>
</tr>
<tr>
<td>jeditV0</td>
<td>8.69</td>
<td>7.9</td>
<td>0.11</td>
<td>0.10</td>
<td>18.40</td>
<td>14.32</td>
<td>0.32</td>
<td>0.37</td>
</tr>
<tr>
<td>jeditV1</td>
<td>9.05</td>
<td>8.13</td>
<td>0.12</td>
<td>0.10</td>
<td>17.93</td>
<td>17.42</td>
<td>0.36</td>
<td>0.34</td>
</tr>
<tr>
<td>jeditV2</td>
<td>7.90</td>
<td>10.34</td>
<td>0.14</td>
<td>0.15</td>
<td>27.34</td>
<td>20.20</td>
<td>0.38</td>
<td>0.40</td>
</tr>
<tr>
<td>log4j</td>
<td>2.60</td>
<td>2.92</td>
<td>0.06</td>
<td>0.08</td>
<td>9.69</td>
<td>7.67</td>
<td>0.15</td>
<td>0.17</td>
</tr>
<tr>
<td>lucene</td>
<td>6.07</td>
<td>6.89</td>
<td>0.10</td>
<td>0.12</td>
<td>9.77</td>
<td>13.06</td>
<td>0.25</td>
<td>0.35</td>
</tr>
<tr>
<td>poiV0</td>
<td>7.42</td>
<td>7.80</td>
<td>0.09</td>
<td>0.10</td>
<td>25.86</td>
<td>19.29</td>
<td>0.28</td>
<td>0.32</td>
</tr>
<tr>
<td>poiV1</td>
<td>9.31</td>
<td>7.62</td>
<td>0.13</td>
<td>0.14</td>
<td>12.67</td>
<td>27.23</td>
<td>0.29</td>
<td>0.36</td>
</tr>
<tr>
<td>synapse</td>
<td>3.88</td>
<td>4.87</td>
<td>0.07</td>
<td>0.08</td>
<td>8.13</td>
<td>13.29</td>
<td>0.19</td>
<td>0.17</td>
</tr>
<tr>
<td>velocity</td>
<td>4.27</td>
<td>5.51</td>
<td>0.07</td>
<td>0.10</td>
<td>15.18</td>
<td>11.58</td>
<td>0.21</td>
<td>0.27</td>
</tr>
<tr>
<td>xercesV0</td>
<td>0.17</td>
<td>7.47</td>
<td>0.10</td>
<td>0.11</td>
<td>18.17</td>
<td>17.31</td>
<td>0.22</td>
<td>0.28</td>
</tr>
<tr>
<td>xercesV1</td>
<td>10.47</td>
<td>11.07</td>
<td>0.16</td>
<td>0.19</td>
<td>18.27</td>
<td>25.27</td>
<td>0.40</td>
<td>0.46</td>
</tr>
</tbody>
</table>

x62

x45

Fu et al. IST journal '16
My Thesis

Software analytics should be easier.

Software analytics can be easier. [Fu et al, IST 2016]

But it can be very hard to show it can be easier. [Fu et al, FSE 2017 A]

And, sometimes, it can be too easy. [Fu et al, FSE 2017 B]
Our Objective

Data → Complex Method (e.g., Deep Learning) → Model

Training process

Fu et al. Deep Learning, FSE’17
Our Objective

Data

Simple method

Model

Training process
Wei Fu, and Tim Menzies. "Easy over hard: a case study on deep learning."
Deep Learning in SE

- From 2015 to 2017, 11 DL paper in SE
- 4 Papers mentioned training cost
- None compares DL costs with competitor methods
How Hard Can It Be?

- Baseline methods are not well described
- No Data, No DL Code
- Did not report DL costs

Fu et al. Deep Learning, FSE’17
Given two questions from stack overflow, are they duplicate, direct link, indirect link or isolated?

Fu et al. Deep Learning, FSE'17
Comparison

**ASE’16(Xu et al.)**
- Baseline: SVM
- Proposed: CNN

**FSE’17(Fu et al.)**
- Baseline: CNN
- Proposed: SVM+DE
What I Did Over One Month

- Collect data from Stack Overflow (60 GB)
- Pre-process data
- Follow Xu et al, replicate their experiment
Xu et al. Baseline Method

Data

SVM

Model

Training process

Fu et al. Deep Learning, FSE’17
Successfully Reproduce Xu’s Baseline
Xu et al’s Complex Method: CNN

Data

Convolutional Neural Networks

Model

Training process
Typical CNN Architectures

- **AlexNet-8**
- **VGG-16**
- **Resnet-34**
- **Resnet-152**

*SqlML Azure Websites, 2017/09/12/convolutional-neural-network/*
Xu et al’s CNN Architecture

**CONV:** a dot product

**RELU:** $\max(0, x)$

**POOL:** downsampling
Simple Method: Tuning SVM With DE
More Details

Train Word2Vec

100,000 KU texts

Train

Word2Vec

Parameter Tuning

DE

Parameters

Train

Evaluate

SVM

Tuning KU vectors

Train

Best Tunings

New Training KU vectors

SVM

Testing KU vectors

Predict

Results

Training KU pairs

Lookup

Train Learner

Testing KU pairs

Lookup

Test Learner
Easy Over Hard: 
Simplicity = Better results

 Fu et al. Deep Learning, FSE’17
Easy Over Hard: Less Runtime

![Bar chart comparing DE+SVM and CNN runtime]

- DE+SVM: 10 minutes
- CNN: 840 minutes
My Thesis

Software analytics should be easier.
Software analytics can be easier. [Fu et al, IST 2016]
But it can be very hard to show it can be easier. [Fu et al, FSE 2017 A]
And, sometimes, it can be too easy. [Fu et al, FSE 2017 B]
Effort-Aware Just-in-Time Defect Prediction: Simple Unsupervised Models Could Be Better Than Supervised Models

Yibiao Yang, Yuming Zhou, Jinping Liu, Yangyang Zhao, Hongmin Lu, Lei Xu, Baowen Xu, and Hareton Leung

1Department of Computer Science and Technology, Nanjing University, China
2Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

ABSTRACT
Unsupervised models do not require the defect data to build the prediction models and hence incur a low building cost and gain a wide application range. Consequently, it would be more desirable for practitioners to apply unsupervised models in effort-aware just-in-time (JIT) defect prediction if they can predict defect-inducing changes well. However, little is currently known on their prediction effectiveness in this context. We aim to investigate the predictive power of simple unsupervised models in effort-aware JIT defect prediction, especially compared with the state-of-the-art supervised models in the recent literature. We first use the most commonly used change metrics to build simple unsupervised models. Then, we compare these unsupervised models with the state-of-the-art supervised models under cross-validation, time-wise cross-validation, and source-project prediction settings to determine whether they are of practical value. The experimental results, from open-source software systems, show that many simple unsupervised models perform better than the state-of-the-art supervised models in effort-aware JIT defect prediction.

FSE’16
A Typical Software Analytics Framework

Supervised

Data

Learner

Model

Training process
Yang et al: Unsupervised Framework

Testing Data

Testing Process

Results

More Details Over Here

Build 12 unsupervised models, on testing data:

<table>
<thead>
<tr>
<th>NF</th>
<th>NS</th>
<th>LT</th>
<th>FIX</th>
<th>ND</th>
<th>NDEV</th>
<th>EXP</th>
<th>REXP</th>
<th>SEXP</th>
<th>NUC</th>
<th>AGE</th>
<th>Entropy</th>
<th>LOC</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>23</td>
<td>2</td>
<td>12</td>
<td>4</td>
<td>2</td>
<td>8</td>
<td>0.3</td>
<td>32</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>24</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>13</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>0.4</td>
<td>42</td>
<td>?</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>89</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0.6</td>
<td>18</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>34</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>0.2</td>
<td>103</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>537</td>
<td>0</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>22</td>
<td>9</td>
<td>7</td>
<td>12</td>
<td>0.3</td>
<td>20</td>
<td>?</td>
</tr>
</tbody>
</table>

Predicted as “Defective”
Our comments

- Reported averaged results across all projects
- How to apply 12 unsupervised learners in practice
A Typical Software Analytics Framework

Data

Learner

Model

Training process

Fu et al. Defect Prediction, FSE'17
OneWay
OneWay is not “the Way”

OneWay:

“The alternative way, maybe not the best way!”

--Wei
OneWay Framework

Data

12 learners

Select best

Model

Training process

Fu et al. Defect Prediction, FSE'17
Performance Measure

- Recall
- Popt
- F1
- Precision

\[ P_{opt}(m) = 1 - \frac{S(\text{optimal}) - S(m)}{S(\text{optimal}) - S(\text{worst})} \]  
(Larger = Better)
Our Result Format

Recall

Report results on a project-by-project basis.
Research Questions

• All unsupervised predictors better than supervised?

• Is it beneficial to use supervised data?

• OneWay better than standard supervised predictors?
Research Questions

• All unsupervised predictors better than supervised?

• Is it beneficial to use supervised data?

• OneWay better than standard supervised predictors?
RQ1: All Unsupervised Predictors Better?
RQ1: All Unsupervised Predictors Better?
Research Questions

- All unsupervised predictors better than supervised?
- Is it beneficial to use supervised data?
- OneWay better than standard supervised predictors?
RQ2: Is It Beneficial to Use Supervised Data?
RQ2: Is It Beneficial to Use Supervised Data?
Research Questions

• All unsupervised predictors better than supervised?

• Is it beneficial to use supervised data?

• OneWay better than standard supervised predictors?
RQ3: *OneWay* Better than Standard Supervised Predictors?
Software analytics should be easier.
Software analytics can be easier. [Fu et al, IST 2016]
But it can be very hard to show it can be easier. [Fu et al, FSE 2017 A]
And, sometimes, it can be too easy. [Fu et al, FSE 2017 B]
When to be simpler?
“Many Roads Lead to Rome”

**Similar learners:**

- Lessman et al.[Lessman’08]: 17/22 defect predictors are indistinguishable.
- Gohtra et al.[Gohtra’15]: 32 defect predictors can be clustered into 4 groups.

*If learners have a “result space” (recall vs false alarm):*

- What “shape” of results spaces leads to “many roads”?
- Can we reverse engineer from that space a much simpler defect predictor?

Fu et al. ε-Dominance, submitted to FSE’18
Deb’s principle of $\varepsilon$-Dominance

If there exists some $\varepsilon$ value below which it is useless or impossible to distinguish results, then \textit{it is superfluous to explore anything less than} $\varepsilon$

\[ \epsilon = 0.2 \]
DART: Fast-and-Frugal Tree (FFT)

We used \( d=4 \), \( 2^d=16 \) trees to explore the results space.

1. if \( \text{cob} \leq 4 \) then false
2. else if \( rfc > 32 \) then true
3. else if \( \text{dam} > 0 \) then true
4. else if \( \text{amc} < 32.25 \) then true
5. else false

We used \( d=4 \), \( 2^d=16 \) trees to explore the results space.
RQ1: DARTS Better than Established Learners?

<table>
<thead>
<tr>
<th>Goal</th>
<th>Data</th>
<th>DART</th>
<th>SL</th>
<th>NB</th>
<th>EM</th>
<th>SMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>dis2heaven: (less is better)</td>
<td>log4j</td>
<td>23</td>
<td>53</td>
<td>51</td>
<td>56</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>jedit</td>
<td>31</td>
<td>40</td>
<td>41</td>
<td>34</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>lucene</td>
<td>33</td>
<td>40</td>
<td>44</td>
<td>44</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>poi</td>
<td>35</td>
<td>36</td>
<td>57</td>
<td>70</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>ivy</td>
<td>35</td>
<td>50</td>
<td>40</td>
<td>71</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>velocity</td>
<td>37</td>
<td>61</td>
<td>40</td>
<td>49</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>synapse</td>
<td>38</td>
<td>51</td>
<td>39</td>
<td>34</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>xalan</td>
<td>39</td>
<td>55</td>
<td>55</td>
<td>70</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>camel</td>
<td>41</td>
<td>60</td>
<td>52</td>
<td>44</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>xerces</td>
<td>42</td>
<td>68</td>
<td>60</td>
<td>50</td>
<td>69</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goal</th>
<th>Data</th>
<th>DART</th>
<th>SL</th>
<th>NB</th>
<th>EM</th>
<th>SMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>pip: (more is better)</td>
<td>ivy</td>
<td>28</td>
<td>17</td>
<td>9</td>
<td>28</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>jedit</td>
<td>39</td>
<td>10</td>
<td>9</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>synapse</td>
<td>43</td>
<td>26</td>
<td>24</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>camel</td>
<td>53</td>
<td>15</td>
<td>17</td>
<td>16</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>log4j</td>
<td>56</td>
<td>19</td>
<td>22</td>
<td>16</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>velocity</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>24</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>poi</td>
<td>73</td>
<td>51</td>
<td>19</td>
<td>33</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>lucene</td>
<td>81</td>
<td>43</td>
<td>27</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>xerces</td>
<td>90</td>
<td>4</td>
<td>9</td>
<td>15</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>xalan</td>
<td>99</td>
<td>11</td>
<td>15</td>
<td>100</td>
<td>51</td>
</tr>
</tbody>
</table>
### RQ2: DARTS Better than Goal-Savvy Learners?

<table>
<thead>
<tr>
<th>Data</th>
<th>$\text{dis2heaven}$ (less is better)</th>
<th>$P_{opt}$ (more is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DART</td>
<td>Tuning RF</td>
</tr>
<tr>
<td>ivy</td>
<td>35</td>
<td>56</td>
</tr>
<tr>
<td>jedit</td>
<td>31</td>
<td>35</td>
</tr>
<tr>
<td>synapse</td>
<td>38</td>
<td>57</td>
</tr>
<tr>
<td>camel</td>
<td>41</td>
<td>70</td>
</tr>
<tr>
<td>log4j</td>
<td>23</td>
<td>51</td>
</tr>
<tr>
<td>velocity</td>
<td>37</td>
<td>53</td>
</tr>
<tr>
<td>poi</td>
<td>34.8</td>
<td>27</td>
</tr>
<tr>
<td>lucene</td>
<td>33</td>
<td>35</td>
</tr>
<tr>
<td>xerces</td>
<td>42</td>
<td>70</td>
</tr>
<tr>
<td>xalan</td>
<td>38.7</td>
<td>36</td>
</tr>
</tbody>
</table>
RQ3: DARTS Better than Data-Savvy Learners?

<table>
<thead>
<tr>
<th>Goal</th>
<th>Data</th>
<th>DART</th>
<th>KNN</th>
<th>SMO</th>
<th>NB</th>
<th>RF</th>
<th>SL</th>
<th>DT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>log4j</td>
<td>23</td>
<td>45</td>
<td>44</td>
<td>50</td>
<td>44</td>
<td>40</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>jedit</td>
<td>31</td>
<td>45</td>
<td>52</td>
<td>41</td>
<td>39</td>
<td>44</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>lucene</td>
<td>33</td>
<td>37</td>
<td>45</td>
<td>44</td>
<td>41</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>poi</td>
<td>35</td>
<td>38</td>
<td>52</td>
<td>52</td>
<td>39</td>
<td>46</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>ivy</td>
<td>35</td>
<td>37</td>
<td>46</td>
<td>36</td>
<td>39</td>
<td>37</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>velocity</td>
<td>37</td>
<td>56</td>
<td>64</td>
<td>40</td>
<td>44</td>
<td>61</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>synapse</td>
<td>38</td>
<td>36</td>
<td>47</td>
<td>36</td>
<td>42</td>
<td>37</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>xalan</td>
<td>39</td>
<td>20</td>
<td>35</td>
<td>45</td>
<td>25</td>
<td>71</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>camel</td>
<td>41</td>
<td>45</td>
<td>62</td>
<td>47</td>
<td>35</td>
<td>53</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>xerces</td>
<td>42</td>
<td>45</td>
<td>67</td>
<td>52</td>
<td>52</td>
<td>53</td>
<td>53</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goal</th>
<th>Data</th>
<th>DART</th>
<th>KNN</th>
<th>SMO</th>
<th>NB</th>
<th>RF</th>
<th>SL</th>
<th>DT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ivy</td>
<td>28</td>
<td>26</td>
<td>27</td>
<td>10</td>
<td>27</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>jedit</td>
<td>39</td>
<td>3</td>
<td>17</td>
<td>6</td>
<td>10</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>synapse</td>
<td>43</td>
<td>39</td>
<td>38</td>
<td>27</td>
<td>36</td>
<td>36</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>camel</td>
<td>52.9</td>
<td>53</td>
<td>53</td>
<td>21</td>
<td>52</td>
<td>53</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>log4j</td>
<td>56</td>
<td>27</td>
<td>50</td>
<td>24</td>
<td>33</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>velocity</td>
<td>64</td>
<td>56</td>
<td>64</td>
<td>64</td>
<td>57</td>
<td>65</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>poi</td>
<td>73</td>
<td>67</td>
<td>69</td>
<td>26</td>
<td>72</td>
<td>72</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>lucene</td>
<td>81</td>
<td>45</td>
<td>49</td>
<td>27</td>
<td>49</td>
<td>42</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>xerces</td>
<td>90</td>
<td>73</td>
<td>63</td>
<td>20</td>
<td>50</td>
<td>77</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>xalan</td>
<td>99</td>
<td>99</td>
<td>98</td>
<td>24</td>
<td>93</td>
<td>100</td>
<td>88</td>
</tr>
</tbody>
</table>

Conclusion

Current Ways to Build State-of-the-art Predictive Models

Data Preprocessing:
- SMOTE
- SMOTUNED
- .......

Hyper-Parameter Tuning

Data Selection

Train Learner

Build a Simple Scout (e.g., DART) to Explore the Results Space

Our Approach
Future of Future Work

- Apply $\epsilon$-Dominance to other software analytics tasks.
  - Text Mining
  - Issue closing time prediction
- Determine $\epsilon$ threshold
- Other criteria to simplify software analytics.
From Last Exam

Why study simplicity? *cost, speed*

When this won’t work? *ε-Dominance*

What’s the difference between SE/general data mining? *under-exploited simplicities*
Thank You!