Predicting Health Indicators for Open Source Projects
(using Hyperparameter Optimization)

Tianpei Xia, Wei Fu, Rui Shu, Rishabh Agrawal, Tim Menzies

Department of Computer Science **North Carolina State University**

March 17, 2022 (v3)

The Problem: Which OSS Projects are Healthy?

- Open-source software (OSS) dominates the modern technology stack, making up over 80% of software in many products.
- This creates a critical question for managers, foundations, and commercial companies (like IBM or Red Hat): **“Which of these projects are any good?”** .
- We need reliable “project health indicators” to assess a project’s status and make decisions about:
  - Resource allocation.
  - Package management (what to include in a distro) .
  - Staff management and workload prediction.

What is “Project Health”?

- There is no single, consolidated definition of “project health”.
- However, researchers and practitioners agree that healthy projects are “vigorous” and “active”.
- Based on teleconferences with Linux developers, we identified 7 key activity indicators:
  1. Number of **contributors**
  2. Number of **commits**
  3. Number of **opened pull-requests**
  4. Number of **closed pull-requests**
  5. Number of **opened issues**
  6. Number of **closed issues**
  7. Project popularity (number of **“stars”**)

Do Developers *Actually* Care? (Yes)

We surveyed 112 core contributors from 68 projects to validate these indicators.

Finding: 6 of the 7 indicators are considered “Very Important” or “Somewhat Important” by most developers.

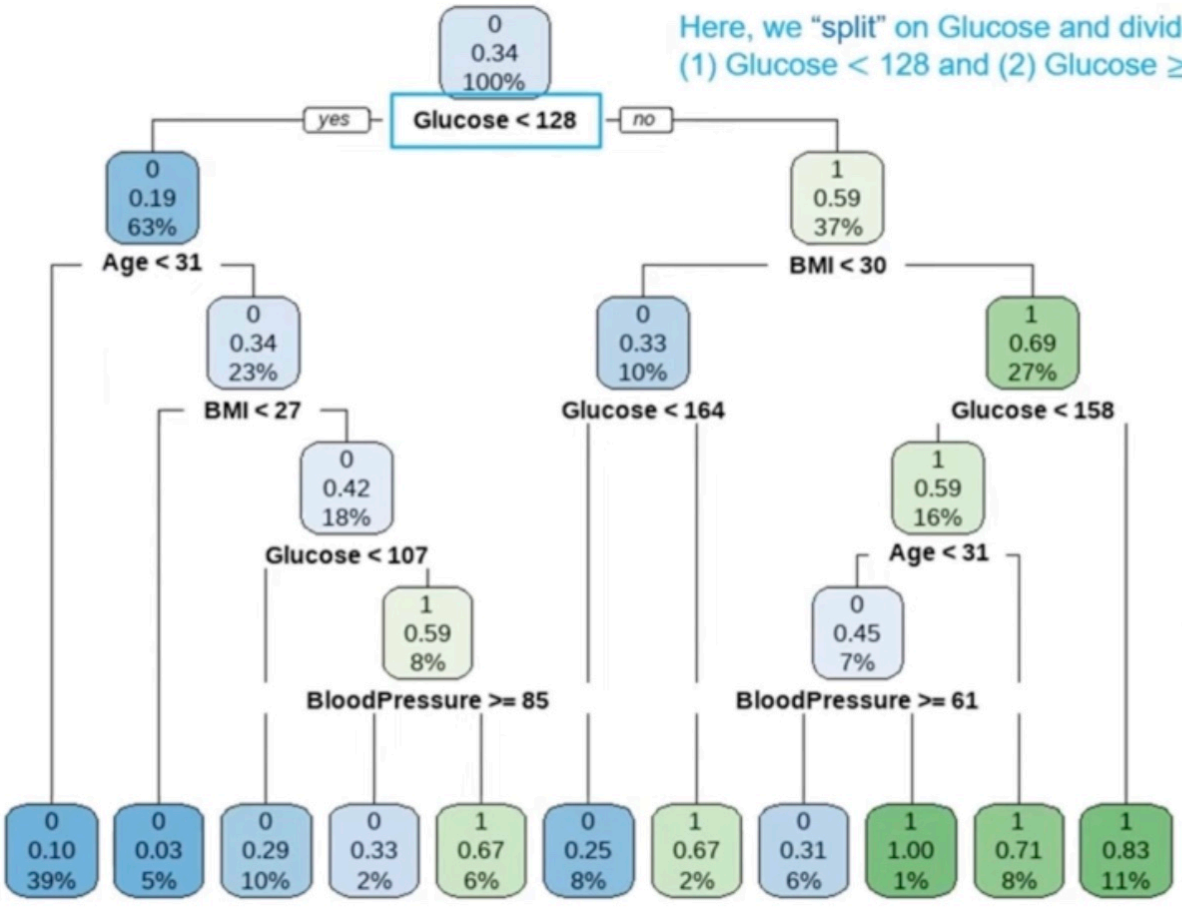
The Exception: “Stargazers” (stars). 52% of developers rated this as “Not Important”.

Table 1: Importance of 7 Indicators to Project Health (based on the survey).

	Very Important	Somewhat Important	Not Important
contributors	33%	46%	21%
commits	57%	31%	12%
opened pull-requests	31%	56%	13%
closed pull-requests	46%	34%	20%
opened issues	33%	59%	8%
closed issues	40%	53%	7%
stargazers	12%	36%	52%

The Core Problem: Prediction is Hard

- While we can *identify* these indicators, *predicting* them is difficult.
- **Traditional estimation algorithms** (KNN, SVR, Random Forest, Linear Regression, CART) make **many mistakes** and have **high error rates**.
- **Why?** Most prior work *neglects to tune* the control parameters (hyperparameters) of their learners, instead using default "off-the-shelf" settings.



Here, we "split" on Glucose and divide the *feature space* into:
(1) Glucose < 128 and (2) Glucose ≥ 128.

The Hypothesis: HPO is the Key

This paper's central hypothesis is that this high error rate can be **greatly reduced** using **Hyperparameter Optimization (HPO)**.

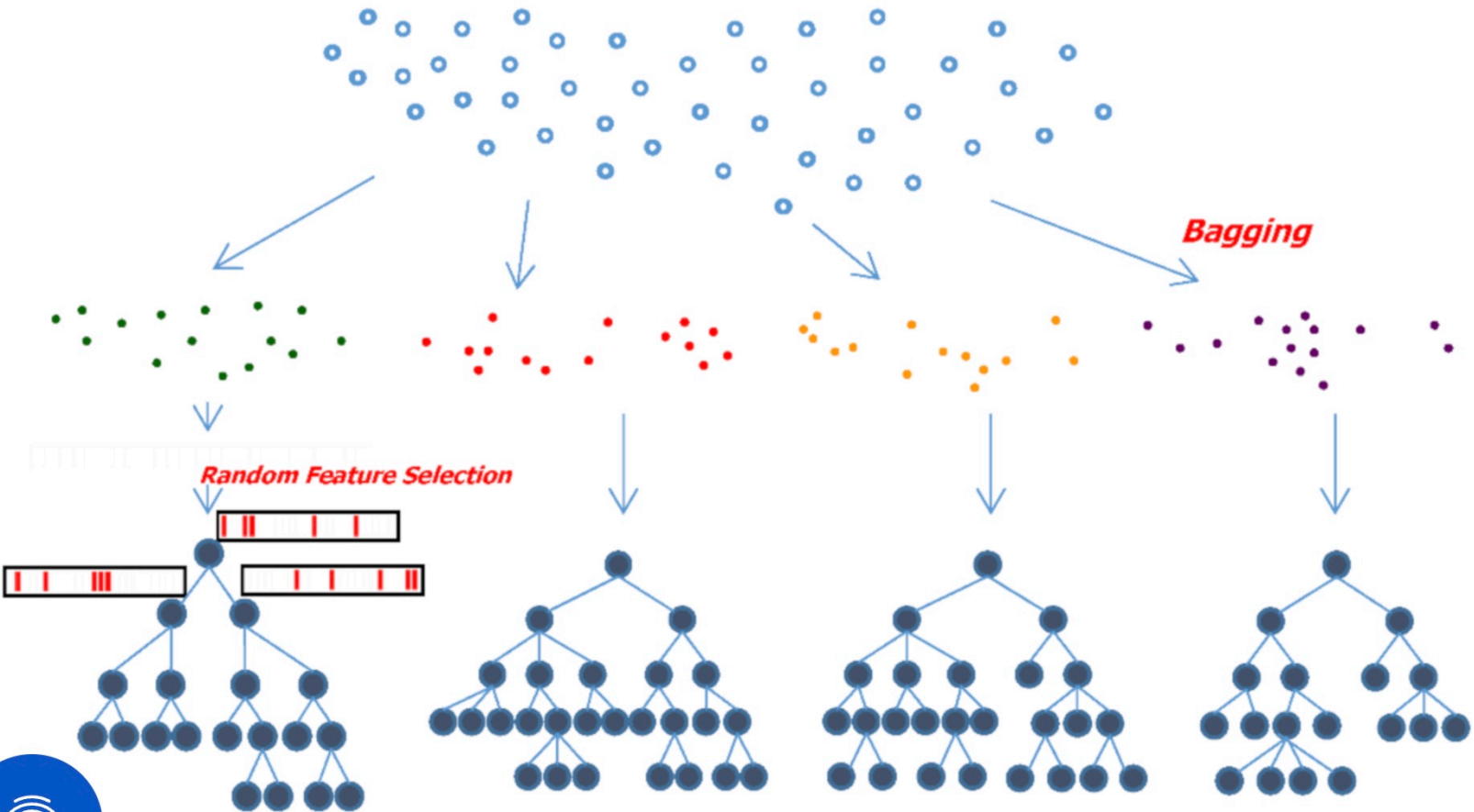
- Recent research in Software Engineering (SE) shows that HPO can *significantly* improve model performance.
- **Our Goal:** Apply HPO to the task of predicting project health indicators and see if it makes a difference.

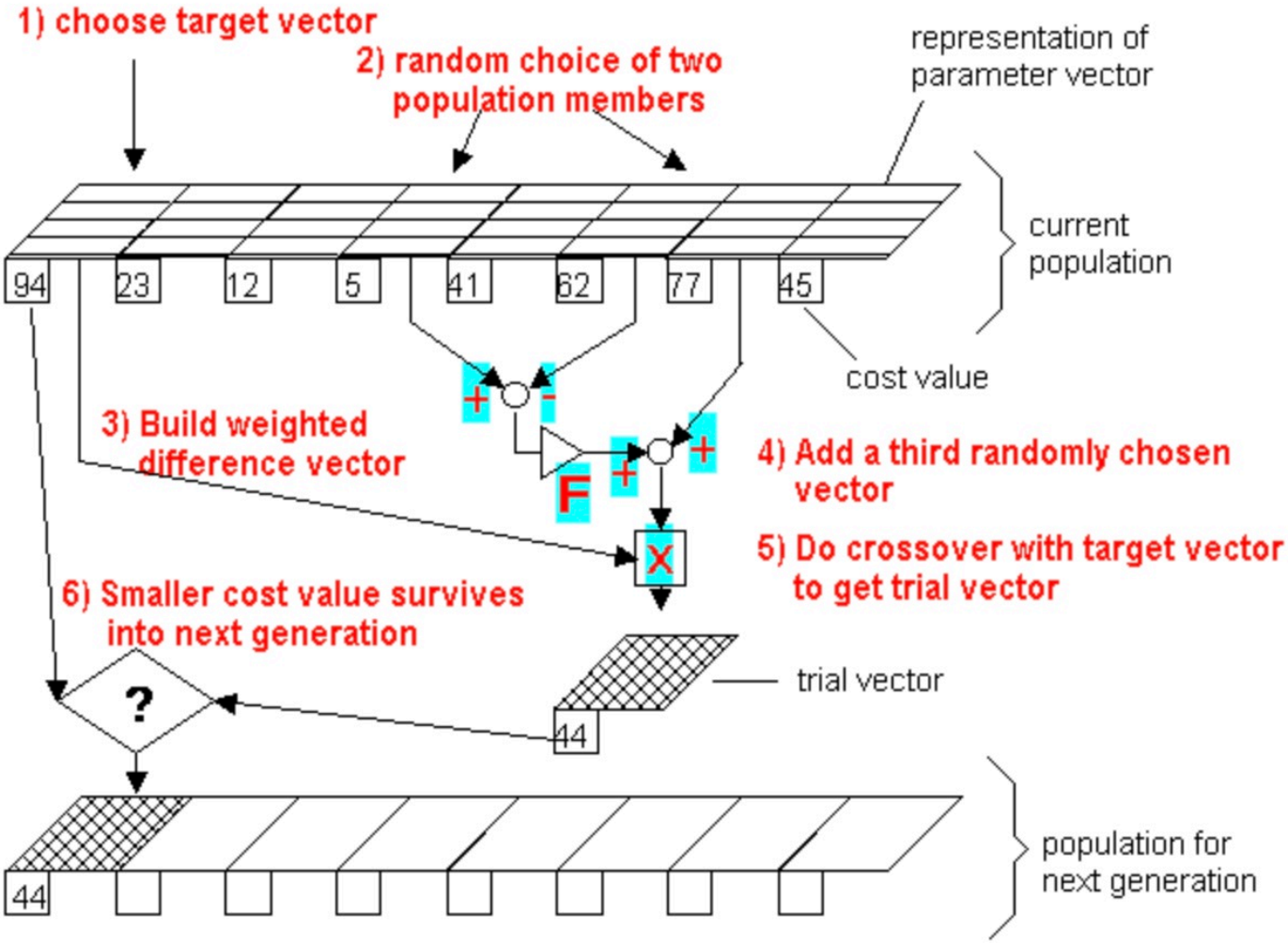
Research Questions

- **RQ1: How to find the trends of project health indicators?**
  - (i.e., Can we *accurately* predict them 1, 3, 6, and 12 months into the future?)
- **RQ2: What features matter the most in prediction?**
  - (i.e., Which indicators are the most important inputs for predicting other indicators?)
- **RQ3: How to improve the performance of health indicators prediction?**
  - (i.e., Do HPO methods *consistently* and *significantly* outperform untuned methods?)

Data & Methods (High Level)

- **Data:** 64,181 months of data from 1,159 GitHub projects (as of April 2020).
  - Projects were filtered to be active, collaborative, and engineered (e.g., removing "tutorials" or "resumes") .
- **Baselines (Untuned):** 5 popular learners with default settings .
  - `KNN`, `SVR`, `LNR`, `RFT`, `CART`
- **Optimized Methods (HPO):** 5 HPO techniques, mostly applied to CART (Regression Trees).
  - `RDCART` (Random Search)
  - `GSCART` (Grid Search)
  - `FLASH` (Sequential Model-Based Optimizer)
  - `DECART` (Differential Evolution)
  - `ASKL` (Auto-sklearn, a SOTA HPO system)


Bagging
Random Feature Selection


1) choose target vector
2) random choice of two population members
representation of parameter vector
current population
94
23
12
5
41
62
77
45
cost value
3) Build weighted difference vector
F
4) Add a third randomly chosen vector
5) Do crossover with target vector to get trial vector
6) Smaller cost value survives into next generation
?
trial vector
44
44
population for next generation

Methods (Low-Level): How We Predict

- **Sliding Window:** To predict the current month, we use the previous 6 months of data as input.
- **Train/Validate Split (for HPO):**
  - Use the first 5 months for **training** the model.
  - Use the 6th month for **validating** (i.e., finding the best hyperparameters).
- **Metrics:**
  - **MRE** (Magnitude of Relative Error): Lower is better. A 25% MRE is considered very good.
  - **SA** (Standardized Accuracy): Higher is better .
- **Statistics:** We use the Friedman test + Nemenyi Post-Hoc test to statistically rank and group all 10 learners .

RQ1: Can We Predict? (Yes, with HPO)

- **Untuned learners (Baselines) are very inaccurate.** LNR has a median MRE of 148% for commits; CART is 54% . These are often wrong by 2x or more.
- **HPO learners are very accurate.** GSCART, DECART, and ASKL achieve low median MREs, often under 25-30%.
- For predicting "contributors," DECART has a 21% MRE.

RQ1 (cont'd): Can We Predict the Future?

- Yes. We tested predictions for 1, 3, 6, and 12 months out .
- **Finding:** Prediction error *does not* increase much as we look further into the future.
- From 1 month to 12 months, the median MRE degrades by only ~17% and SA by ~20%.

Table 11: MRE and SA results with DECART, predicting for 1, 3, 6, 12 months into the future.

	Health Indicator	1 month	3 month	6 month	12 month
Median, MRE	commit	0.39	0.40	0.43	0.53
	contributor	0.21	0.21	0.23	0.26
	openPR	0.22	0.22	0.24	0.30
	closePR	0.33	0.33	0.35	0.40
	openISSUE	0.23	0.23	0.26	0.30
	closedISSUE	0.28	0.29	0.30	0.35
	median ratio change		101%	107%	117%
Median, SA	commit	0.41	0.40	0.38	0.30
	contributor	0.59	0.57	0.51	0.40
	openPR	0.53	0.52	0.48	0.38
	closePR	0.39	0.38	0.34	0.28
	openISSUE	0.51	0.50	0.44	0.36
	closedISSUE	0.43	0.43	0.39	0.32
	median ratio change		98%	91%	80%
IQR, MRE	commit	0.78	0.85	0.99	1.11
	contributor	0.46	0.49	0.57	0.66
	openPR	0.76	0.82	0.94	1.07
	closePR	0.71	0.76	0.90	1.02
	openISSUE	0.44	0.47	0.54	0.61
	closedISSUE	0.52	0.56	0.66	0.75
	median ratio change		108%	116%	113%
IQR, SA	commit	0.98	1.07	1.19	1.47
	contributor	0.89	0.97	1.10	1.35
	openPR	0.91	0.99	1.12	1.35
	closePR	1.02	1.12	1.28	1.57
	openISSUE	0.96	1.04	1.14	1.34
	closedISSUE	0.89	0.97	1.08	1.25
	median ratio change		109%	112%	122%

RQ1 (cont'd): Why This Matters (Case Study)

- This chart shows 14 real-time OS projects.
- **Zephyr** (purple) is growing rapidly in contributors.
- Other projects (e.g., RIOT, LiteOS) are stagnant.
- For a manager of a "flat" project, a prediction of contributor growth with only **21% MRE** is *extremely useful* for planning and resource management.

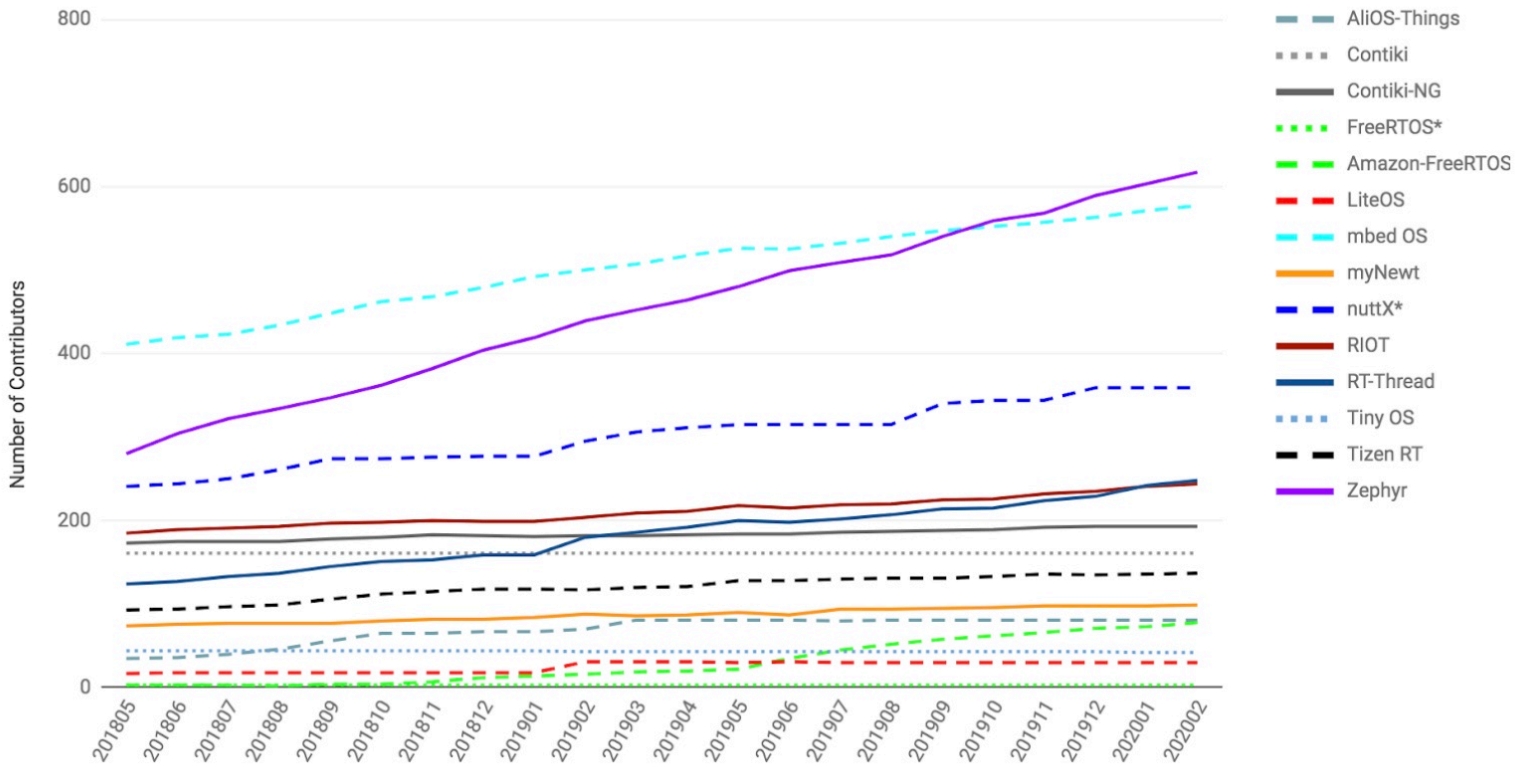


Fig. 3: The trends in number of contributors across 14 real-time OS projects since 2018. Zephyr (one in purple) has out-paced other similar software projects. (data source: The Apache Software Foundation)

RQ2: What Features Matter Most?

- We looked at the Gini importance (feature importance) inside the DECART trees to see what features were used most often .
- **Finding:** Some features are used more than others.
  - `openPR` (17% mean) and `closeISSUE` (19% mean) are highly used.
  - `closePR` (6% mean) is the *least used* feature.
- **But:** No feature is useless. `closePR` is still used 13% of the time when predicting `closeISSUE`.

Target	commit	contributor	openPR	closePR	openISSUE	closeISSUE	star	ISSUEcomment	mergedPR	fork
commit	n/a	7%	15%	4%	7%	22%	6%	19%	5%	15%
contributor	22%	n/a	4%	4%	20%	8%	23%	5%	8%	6%
openPR	10%	7%	n/a	5%	24%	29%	2%	4%	16%	3%
closePR	13%	12%	24%	n/a	9%	21%	7%	6%	4%	4%
openISSUE	3%	6%	16%	2%	n/a	16%	5%	5%	24%	23%
closedISSUE	9%	7%	24%	13%	8%	n/a	5%	21%	6%	7%
mean	11%	8%	17%	6%	14%	19%	8%	10%	11%	10%

Table 12: The mean scores of Gini importance in trees generated by DECART (observed percentages in 1,159 cases).

RQ2 (cont'd): The "So What?"

- While some features are more important than others, it's not a clear-cut case for feature selection.
- The data suggests that all the indicators are entangled.
- **Answer 2 (Conclusion):**
  - We identified the most and least important features.
  - But overall, the best practice is probably to **use all of these features**.

RQ3: How to Improve? (Use HPO)

- We compared the “win rates” of all 10 learners using the Friedman-Nemenyi statistical tests.
- A “win” means the learner was in the top-performing statistical group for a given project .
- **Finding:** The HPO methods *dominate* the untuned methods.

RQ3: How to Improve? (Use HPO)

- We compared the “win rates” of all 10 learners using the Friedman-Nemenyi statistical tests.
- A “win” means the learner was in the top-performing statistical group for a given project .
- **Finding:** The HPO methods *dominate* the untuned methods.

RQ3: HPO Methods Win (DECART & ASKL)

- **DECART** (simple HPO) and **ASKL** (complex SOTA) are the clear winners.
- They both have a median "win rate" of **75%** based on MRE.
- The untuned methods (like LNR, SVR) often have median win rates below 30%.
- **Answer 3:**
  - **Hyperparameter optimized methods generate better prediction performance than the other methods**.

RQ3: HPO Methods Win (DECART & ASKL)

- **DECART** (simple HPO) and **ASKL** (complex SOTA) are the clear winners.
- They both have a median "win rate" of **75%** based on MRE.
- The untuned methods (like LNR, SVR) often have median win rates below 30%.
- **Answer 3:**
  - **Hyperparameter optimized methods generate better prediction performance than the other methods**.

Discussion: But Isn't HPO *Slow*?

- This is a major concern. HPO is famously CPU-intensive.
- In other SE domains (like defect prediction), HPO can take **nearly three days** of CPU time *per dataset*.
- If this were true for our 1,159 projects, our method would be impractical.

Key Finding: NO! HPO is *FAST* Here.

- This was a surprise. For this problem, HPO is extremely fast.
- The mean runtime for HPO (like DECART) is **12-14 seconds *per project***.
- We ran `DECART` (simple) against `ASKL` (complex SOTA) on 20 projects.
  - `DECART` was **5.7 times faster** (324s vs 1843s).
  - ...and achieved the *same* prediction performance.

Why So Fast & Effective? (The "Secret")

This isn't a general HPO problem; it's a **specialized domain**.

1. **Tiny Datasets:** We are running 1,159 *small* optimizations, not one big one. Each project's dataset is tiny (at most 10 features x 60 months of data).
2. **Learner Idiosyncrasies:** Standard learners (like CART) rely on statistical properties from *large* data samples. They perform poorly on small data.
3. **HPO is the Fix:** HPO *adjusts the learner to the idiosyncrasies of the small dataset.*

This is why HPO is both **necessary** (it fixes the learner) and **fast** (the dataset is tiny).

Conclusion

1. **Yes, we can predict project health.** We can accurately predict 6 key health indicators for OSS projects 1, 3, 6, and 12 months into the future.
2. **Traditional learners fail.** Using "off-the-shelf" learners gives very high error rates.
3. **The key is Hyperparameter Optimization (HPO).** HPO is *vital* to tune the learners to the "idiosyncrasies" of these small, project-specific datasets.
4. **This process is fast and practical.** Unlike other domains, HPO here takes only *seconds per project*, not days. A simple HPO (DECART) is just as good as, and much faster than, a complex SOTA tool (ASKL).

Future Work

- **Private Repos:** Test these techniques on large, *private* "in-house" GitHub ecosystems (like those at IBM).
- **More Indicators:** Explore other indicators suggested by developers, such as the number of new/leaving contributors.
- **Other Learners:** Explore HPO on different learners (like Random Forest) or use deep learning (like LSTMs) now that we have a large data pipeline.

References

[Xia20] T. Xia, W. Fu, R. Shu, R. Agrawal, T. Menzies, "Predicting Health Indicators for Open Source Projects (using Hyperparameter Optimization)," *arXiv:2006.07240v3*, 2022.

[Bao19] L. Bao, X. Xia, D. Lo, G. C. Murphy, "A large scale study of long-time contributor prediction for GitHub projects," *IEEE Trans. Softw. Eng.*, 2019.

[Fu16] W. Fu, T. Menzies, X. Shen, "Tuning for software analytics: Is it really necessary?," *IST Journal*, 2016.

[Nair18] V. Nair, Z. Yu, T. Menzies, N. Siegmund, S. Apel, "Finding Faster Configurations using FLASH," *IEEE Trans. Softw. Eng.*, 2018.

[Feu19] M. Feurer, A. Klein, K. Eggensperger, et al., "Auto-sklearn: efficient and robust automated machine learning," *Automated Machine Learning*, 2019.

[Storn97] R. Storn, K. Price, "Differential evolution-a simple and efficient heuristic for global optimization over cont. spaces," *JoGO*, 1997.