UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

WESLEY MATTEUS ARAÚJO DOS SANTOS

UNDERSTANDING THE TESTING CULTURE OF MACHINE LEARNING PROJECTS ON GITHUB

CAMPINA GRANDE - PB

2023

WESLEY MATTEUS ARAÚJO DOS SANTOS

UNDERSTANDING THE TESTING CULTURE OF MACHINE LEARNING PROJECTS ON GITHUB

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Everton Leandro Galdino Alves.

CAMPINA GRANDE - PB

2023

WESLEY MATTEUS ARAÚJO DOS SANTOS

UNDERSTANDING THE TESTING CULTURE OF MACHINE LEARNING PROJECTS ON GITHUB

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Everton Leandro Galdino Alves

Orientador – UASC/CEEI/UFCG

Professor Dr. Carlos Wilson Dantas de Almeida

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 14 de fevereiro de 2023.

CAMPINA GRANDE - PB

RESUMO

Nos últimos anos, o uso de aprendizado de máquina aumentou em diversas indústrias, mostrando seu notável potencial para resolver tanto problemas antigos como emergentes em uma escala nunca antes vista. No entanto, apesar dos esforços na produção de modelos novos e melhorados, bem como metodologias de treinamento mais confiáveis, pouco se sabe sobre como esses softwares estão sendo testados. Neste trabalho, investigamos a adoção de bibliotecas Python para, ou relacionadas, a testes automatizados em mais de 290 repositórios de aprendizado de máquina no Github. Nós também comparamos repositórios que usam e não usam essas ferramentas, em termos de métricas de qualidade, e estudamos sua cobertura de código. Como resultado, 28 bibliotecas usadas para fins de suporte a testes foram identificadas e 65,19% de todos os projetos adotaram pelo menos uma delas. Nós também encontramos que projetos de aprendizagem por reforço e de análise/visualização de dados têm as maiores adoções de testes automatizados, e que *unittest, pytest e doctest* são as bibliotecas mais utilizadas em nosso corpus. Além disso, descobrimos que metade dos projetos que usam pelo menos uma biblioteca de testes, tem menos *code smells* (48,28% em mediana) e, em média, eles têm menos vulnerabilidades (71,42%).

Understanding the testing culture of machine learning projects on Github

Wesley Santos*
wesley.santos@ccc.ufcg.edu.br
Federal University of Campina Grande
Campina Grande, Paraíba

Everton Alves
everton@computacao.ufcg.edu.br
Federal University of Campina Grande
Campina Grande, Paraíba

ABSTRACT

In the last few years, the use of machine learning has spiked in several industries, showing its remarkable potential for solving both old and emergent problems on a scale never seen before. However, despite the efforts on producing new and improved models, as well as more reliable training methodologies, little is known about how these softwares are being tested. In this paper, we investigate the adoption of Python libraries for or related to automated testing on more than 290 machine learning repositories on Github. We also compare repositories that do and do not use those tools, in terms of quality metrics, and study their code coverage. As a result, 28 libraries used for testing support purposes were identified and 65.19% of all projects adopted at least one of them. We also found that reinforcement learning and data analysis/visualization projects have the highest adoptions of automated testing, and that unittest, pytest and doctest are the most used libraries in our corpus. Furthermore, we found that half of the projects that use at least one testing library, have less code smells (48.28% in median) and, on average, they have less vulnerabilities (71.42%).

ACM Reference Format:
Wesley Santos and Everton Alves. 2023. Understanding the testing culture of machine learning projects on Github. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the last few years, the usage of machine learning has spiked in several industries, showing remarkable potential in solving both old and emergent problems. Of which, some widespread examples include self-driving cars [2], sales analysis [4], medical diagnosis [9], biometric recognition [11], and improved decision-making [10]. At the same time, these successful solutions were accompanied by the development of new tools, frameworks, and applications. Many of them have their source code available on platforms like Github [1].

Nonetheless, despite its success, as machine learning and artificial intelligence take bigger roles and responsibilities, concerns about their trustworthiness are enforced [12]. Models and their infrastructure are expected to be thoroughly evaluated and checked for correctness, efficiency, and fairness. And while the process of testing machine learning models is still at early stages, due to the fact that several problems arise from their non-deterministic and data-driven nature (e.g. the Oracle Problem [3]), it has seen growing interest lately.

However, the current research efforts on testing these systems do not seem to account for how their implementations and support modules are being evaluated in practice. That leads to a lack of understanding of what practices for testing and quality assurance are predominant in this context. Moreover, we lack information associated with these practices. For example, it is unclear if projects with different testing profiles (e.g. that do and do not employ automated tests) present markedly different code statistics, such as number of bugs, smells, and vulnerabilities, as well as, if those issues present different degrees of severity. All of these problems together amount to a shortage of actionable information that could guide future research and the development of new testing tools and/or algorithms.

In this paper, we perform an initial empirical study over test adoption on Python machine learning projects. We work only with Python projects since it is used by more than 70% of ML developers and data scientists [1], and previous works suggest that the literature on Python machine learning projects is scarce [6]. In our study, we collected more than 290 projects divided in 8 categories. Then, we identified the libraries being used for automated testing purposes. Afterwards, we compare the ones that use some form of testing with the ones that do not use it in terms of bugs, smells, among others normalized statistics. Finally, we analyze the code coverage of all the repositories that report it, amounting to 80 projects.

As a result, we found that reinforcement learning and data analysis/visualization projects are the ones that most adopt testing support libraries, ranging from 90.91% to 83.33%, respectively. We also identified that 28 libraries are being used for testing purposes. And from those, unittest, pytest and doctest are the ones primarily used. Furthermore, we found that half of the projects in our sample, that use at least one of the packages, have less code smells (48.28% in median) and, on average, they have less vulnerabilities (71.42%), normalizing by lines of code (LOC). And that the number of major bugs by LOC in some categories, such as computer vision, can be up to 98% less in projects that use some form of testing. At last, we established that 27.3% of projects publicly report their code coverage of 86.38%, on average.

The remainder of this paper is as follows. In section 2 we explore the methodology used to collect and process our corpus of projects.

[1] https://github.com

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In section 3 we discuss the analyzes that we made over all the statistics about test adoption. Afterwards, we show the limitations and threats to validity of our research in section 4. In section 5 we presented the related works. And, at last, we conclude and suggest future works in section 6.

2 MATERIALS AND METHODS

In this section, we first present the research questions that guide the study. Then, we explore the method to collect, process and analyze all the projects' used data.

2.1 Research Questions

Our goal with this work is to understand the testing culture in machine learning projects. We want to quantify how much of these projects use automated testing, which tools they use, and we want to determine if the existence of tests make a difference in code quality metrics, such as number of bugs and code smells.

RQ1 *Do Python machine learning (PML) projects apply automated testing?*

Software testing is an essential activity when trying to reduce the number of potential problems in code and it gives developers more confidence about how the software performs. Thereby, this question aims to understand the frequency in which machine learning projects are tested in practice. For that, we focus on the projects' source code and we do not differentiate the levels of the tests being employed (e.g. integration, unit, system, etc).

RQ2 *Which automated support tools are used for PML projects ?*

We also aim to investigate which libraries are being used for testing these projects, showing possible preferences and commonalities between them. As well as, potentially revealing test types that are more used than others.

RQ3 *How do projects that do not use automated testing tools compare to the ones that use regarding code quality ?*

Software testing generally is said to produce more reliable and higher quality code, so we are going to investigate the effect of test existence in terms of code quality metrics, such as number of smells, technical debt, bugs and their severity. This question aims to understand how different testing project profiles compare to each other, more specifically the ones that do and do not use automated testing packages.

RQ4 *Are PML projects well tested ?*

The goal of this research question is to assess how well the tests are exercising the code. For that, we are going to use code coverage. Lower values might indicate the need to write tests that explore more of these softwares, finding potential problems faster.

2.2 Data Collection

In our study, we collected the projects to compose our corpus from a well-known and actively maintained list of machine learning repositories on Github [2]. More specifically, we target the section dedicated for python projects. To fetch those projects, we used the Github API [3]. All scripts and additional analysis are available online [4]. Our study followed a methodology comprised of four steps: Project Filtering, Automated support mechanisms, Code quality statistics, and Code coverage analysis

2.2.1 Project filtering. Originally, there were 344 project entries divided into 8 categories: i) computer vision, related to tasks of image apprehension, processing, analysis, and/or understanding; ii) data analysis/visualization, related to visualization and processing of data; iii) general-purpose machine learning, composed of applications, frameworks, or libraries that do not fall in a specific category; iv) kaggle [5] competitions source code, a platform for practicing and competing on several machine learning scenarios; v) natural language processing (NLP), related to processing of natural languages; vi) miscellaneous scripts, smaller scripts and codebases simpler in nature; vii) neural networks, projects that implement or use models loosely inspired on the biological brain; and viii) reinforcement learning, related to projects that apply learning by reinforcement models to solve problems.

However, some of the entries did not point to the repository link. Therefore, the first step was to write a script to collect them. After that, we manually verified the results and removed entries pointing to the same project. As our focus is Python code, we decided to filter out all multi-language projects with less than 20% of the source code written in Python. We also avoid python notebook repositories, as some of the project analysis tools that we use in our study could not analyze them. That process resulted in the final amount of 293 projects.

2.2.2 Automated support mechanisms. We are interested in identifying the tools used for, or related to, automated testing on our corpus. With that in mind, considering as different projects use multiple strategies to require packages and some of the libraries are built-in (e.g. unittest and doctest), we built a script that processes each repository. For that, we searched all filenames containing "test" and looked for imports in each of their lines. At last, we collected the description of every possible tool identified on the PyPI website [6] (repository of software for the Python programming language) and manually filtered the ones related to the testing pipeline.

2.2.3 Code quality statistics. To compare projects that use and do not use automated testing packages. We used the Sonarqube tool [7], an automatic software that runs static analysis over the projects and reports several metrics [8], such as number of bugs, defined as a mistake that can lead to error or unexpected behavior in runtime;

[2] Machine learning project list: https://github.com/josephmisiti/awesome-machine-learning
[3] Github REST API: https://docs.github.com/en/rest
[4] Paper scripts: https://github.com/Wesley-M/tcc
[5] https://www.kaggle.com/
[6] https://pypi.org/
[7] https://www.sonarsource.com/products/sonarqube/
[8] Sonarqube metrics and their definitions: https://docs.sonarqube.org/latest/user-guide/metric-definitions/

Table 1: Amount of projects by coverage tracking service (CTS)

CTS	# of projects that adopt the CTS	%
codecov	49	16.72 %
coveralls	23	7.85 %
codeclimate	2	0.68 %
self-hosted	1	0.34 %

code smells, issues that make the code confusing and difficult to maintain; technical debt, measures the effort to solve all the code smells; reliability remediation effort, a measure of effort to solve all the bugs; duplicated lines density, the percentage of duplicated lines; number of vulnerabilities, a point in the code that's open to attack; and cognitive complexity, a measure that tries to capture how intuitive a unit of code is.

We also consider bug severity when comparing them. According to the Sonarqube's team [9], blocker bugs have a high probability to impact the behavior of the application in production; critical bugs either have a low probability of impacting the application behavior in production or it represents a security flaw; major bugs are flaws that can highly impact the developer's productivity; and minor bugs can slightly impact the developer's productivity.

With respect to sonarqube's measure process, both smells, bugs and vulnerabilities rely on a database of rules to be recognized. Each of those rules have an estimated time (in minutes) for them to be solved by developers. The sum of those times, in case of smells, gives the technical debt and, in case of bugs, the reliability remediation effort. The duplicated lines density, as the name suggests, is the ratio of duplicated lines by the total number of lines. Finally, we normalize statistics such as the number of bugs, to have a fair comparison regardless of project size.

2.2.4 Code coverage analysis. Several analysis tools can measure code coverage. However, configuring and running those tools in each project is error prone and time consuming. Therefore, we opted for collecting the coverage information that is publicly available in the repositories README.md files, amounting to 80 projects. We found several services that provide these README.md files with coverage badges, such as Codecov [10], Coveralls [11], Code Climate [12] and self-hosted. From 293 projects on our corpus, 49 (16.72%) report Codecov badges, 23 (7.85%) Coveralls, 2 (0.68%) Code Climate and 1 (0.34%) was self-hosted. This distribution can be seen on table 1.

3 RESULTS

In this section, we explore the results of our investigation. First, we analyze the adoption of automated testing by machine learning category. Then, we show the libraries that we identified as used for testing purposes, and compare projects that use at least one of those tools with the ones that do not. Finally, we analyze how much code is covered by these tests on the projects that publish it.

[9]Sonarqube issue definitions: https://docs.sonarqube.org/latest/user-guide/issues/
[10]https://about.codecov.io/
[11]https://coveralls.io/
[12]https://codeclimate.com/

Table 2: % of projects that adopt test support packages

Category	Proportion	%
Reinforcement Learning	10/11	90.91%
Data Analysis	40/48	83.33%
General Purpose	91/121	75.21%
Natural Language Processing	27/38	71.05%
Computer Vision	11/24	45.83%
Miscellaneous Scripts	9/21	42.86%
Neural Networks	3/8	37.5%
Kaggle Competition	0/22	0%
Overall	191/293	65.19%

3.1 Do Python machine learning (PML) projects apply automated testing ?

Table 2 shows the proportions and percentages of machine learning projects that apply some kind of automated tests by category. A project is considered to adopt automated tests if it imports at least one library related to testing in the source code. We found that 65.19% of all projects adopt some kind of testing mechanism. Moreover, only one project from Reinforcement Learning did not adopt any of the identified libraries, reaching a percentage of 90.91%, while kaggle's competition code shows no adoption of tests. Miscellaneous scripts and neural networks also display low rates of adoption, below 50%.

For the kaggle case, we believe that, as it is a competitive environment and the projects' goals are mainly short-term, the authors might drive more efforts to other code aspects (e.g. model accuracy, recall, etc) than testing. We also speculate that miscellaneous scripts suffer from related causes, as they are composed of smaller code bases. Regarding the high test adoption on Reinforcement Learning (90.91%), after manual investigation, we found that the majority of those repositories are popular libraries being used to power other projects. In those cases, an effective test suite often can help developers to better trust that the library is stable and reliable.

3.2 Which automated support tools are used for PML projects ?

After processing all the test files from our corpus, we identified 28 libraries related to automated testing. The next step was to collect each library usage frequency and ascertain the ones most used in different contexts, which we can see in Figure 1. As a result, we found that some general-purpose libraries, such as unittest, built-in on the language, as well as pytest, the most popular external library in our sample, are widely used. Nose is also one the most utilized, however it is an extension for unittest that is deprecated and does not receive new updates.

Meanwhile, libraries with more specific goals have lower adoption rates. There are several applications for these tools, but some examples include behavior mocking (e.g. requests_mock, pytest_mock, fakeredis, mongomock, etc), property-based testing (e.g. hypothesis), or even executable comments that can serve as documentation (e.g. doctest, examples).

A possible reason for the high prevalence and wide usage of unittest, pytest and doctest might be that Python developers are

Table 3: Percentage increase in normalized mean of bugs for projects that do not apply tests and apply them, respectively

Category	BLOCKER	MAJOR	MINOR	CRITICAL
Computer Vision	Infinite**	-98.03%	Infinite**	Undefined*
Data Analysis	42.36%	248.3%	95.64%	Infinite**
General-Purpose	11.03%	-65.47%	-73.15%	Infinite**
Kaggle	Infinite**	Infinite**	Undefined*	Undefined*
Miscellaneous Scripts	198.31%	2870.7%	-52.83%	Undefined*
NLP	114.05%	-68.62%	-49.88%	Infinite**
Neural Networks	Undefined*	Infinite**	-92.88%	Undefined*
Reinforcement Learning	Infinite**	122.04%	-94.2%	Infinite**

* There is no combination of category and bug severity present in the data.

** This combination does not exist in the projects without tests.

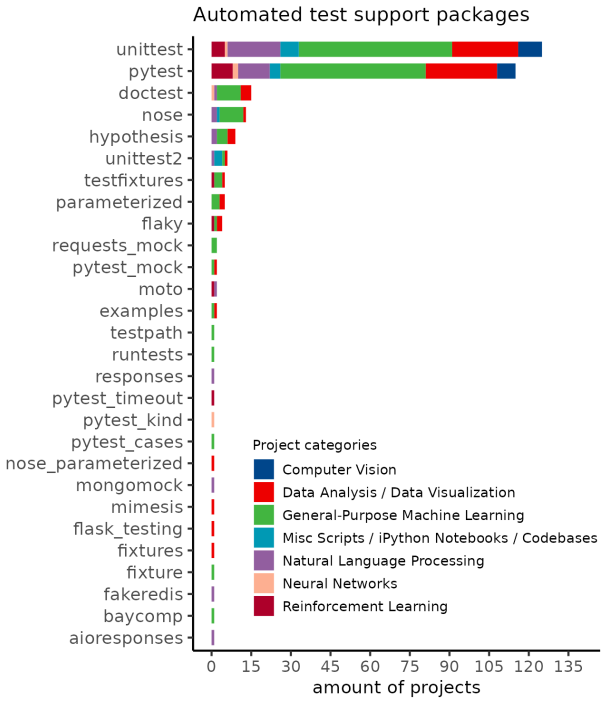

Figure 1: Amount of automated test support libraries by category.

more prone to apply unit tests on their machine learning projects, however unittest and pytest can potentially be used for integration tests as well. We can also see that in 5th place, there is the "hypothesis" library, which suggests that property-based testing is relatively popular, mostly in data analysis and general-purpose projects.

3.3 How do projects that do not use automated testing tools compare to the ones that use regarding code quality ?

Table 4 shows code statistics for both project testing profiles (projects that have tests and those who do not). Moreover, each analysis is broken down into minimum, mean, median and maximum values. A percentage increase of each and every statistic between projects that employ and do not employ test libraries is also shown on the table. Accordingly, we found that projects that do not use any test support library have fewer lines of code and cognitive complexity, with a percentage increase of over 1000% in complexity between them, on average.

In terms of bugs, even normalized by lines of code, we found more of those issues on projects that use tests (7.01% on average). Complementing that finding, Table 3 shows the percentage increase between the means of normalized number of bugs for projects that do not employ and employ tests, respectively. That can be expressed mathematically as:

- $PI_{m,n}$: Percentage increase between m and n
- P_j : Number of projects for category j
- LOC_i: Number of lines of code on project i
- $NumBugs_{i,j,k}$: Number of bugs on project i, of category j and severity k
- $NormBugMean_{j,k}$: Normalized mean number of bugs for category j and severity k
- $MeanWithTestAdoption_{j,k}$: $NormBugMean_{j,k}$ for projects with tests
- $MeanWithoutTestAdoption_{j,k}$: $NormBugMean_{j,k}$ for projects without tests

$$PI_{m,n} = (m - n)/n$$

$$NormBugMean_{j,k} = \frac{\sum_{i=1}^{P_j} \frac{NumBugs_{i,j,k}}{LOC_i}}{P_j}$$

So each cell value can be defined as:

$$PI(MeanWithTestAdoption_{j,k}, MeanWithoutTestAdoption_{j,k})$$

Positive values mean that there are more bugs per LOC, on average, in projects that adopt tests, and vice versa. Undefined values mean that there is no combination of project category and bug severity in both testing profiles. And infinite means that there

Table 4: Sonarqube code statistics from projects with and without automated test support packages

Characteristic	Without automated tests (N = 102)	With automated tests (N = 191)	Percentage Increase
Lines of code (LOC)			
Min	26.0000000	51.0000000	96.15%
Mean	8923.2772277	56567.9057592	533.94%
Median	1509.0000000	13193.0000000	774.29%
Max	549774.0000000	2235379.0000000	306.6%
Cognitive Complexity (CC)			
Min	2.0000000	2.0000000	0%
Mean	722.1980198	8250.2356021	1042.38%
Median	280.0000000	1620.0000000	478.57%
Max	9237.0000000	413988.0000000	4381.84%
Duplicated lines (%)			
Min	0.0000000	0.0000000	Undefined*
Mean	7.3871287	5.9094241	-20%
Median	1.8000000	2.7000000	50%
Max	76.5000000	69.7000000	-8.89%
Bugs (normalized by LOC)			
Min	0.0000000	0.0000000	Undefined*
Mean	0.0005430	0.0005825	7.27%
Median	0.0000000	0.0002870	Undefined*
Max	0.0089410	0.0073380	-17.93%
Reliability Remediation Effort in minutes (normalized by LOC)			
Min	0.0000000	0.0000000	Undefined*
Mean	0.0038931	0.0036804	-5.46%
Median	0.0000000	0.0016364	Undefined*
Max	0.0443076	0.0575428	29.87%
Smells (normalized by LOC)			
Min	0.0000000	0.0003950	Undefined*
Mean	0.0554352	0.0296518	-46.51%
Median	0.0400000	0.0206864	-48.28%
Max	0.2343173	0.1540606	-34.25%
Technical Debt in minutes (normalized by LOC)			
Min	0.0000000	0.0028881	Undefined*
Mean	0.3128514	0.1785655	-42.92%
Median	0.2560511	0.1403209	-45.2%
Max	1.1162362	0.9351202	-16.23%
Vulnerabilities (normalized by LOC)			
Min	0.0000000	0.0000000	Undefined*
Mean	0.0000353	0.0000101	-71.42%
Median	0.0000000	0.0000000	Undefined*
Max	0.0019881	0.0007346	-63.05%

* The difference can not be determined, given one of the values is 0.

are no bugs on the projects that do not apply tests, while there is at least one on the ones that apply.

In terms of blocker bugs, for all categories, we noticed that, on average, projects that adopt tests presented more of those bugs per LOC. We speculate that might happen because the projects that do not adopt tests are so much smaller than their counterparts, that it is harder to introduce serious unnoticed bugs. Regarding critical bugs, on our corpus there was not any combination in which the two means were bigger than 0, so all values were either undefined or infinite. Major bugs give mixed results, of which test adoption is preferable in computer vision (-98.03%), NLP (-68.62%) and general purpose (-65.47%) projects. Lastly, minor bugs appear to show more in projects without test adoption on almost all categories, except computer vision and data analysis.

Meanwhile, there is substantially less code smells on projects with tests, 48.28% on average. Which also happens with vulnerabilities and technical debt, 71.42% and 42.92%, respectively, on average. These findings could mean that machine learning code that is tested is easier to maintain and it is more secure than their non tested counterparts.

3.4 Are PML projects well tested ?

Table 5 shows that 80 projects (27.3% of all projects) publish test coverage on their README.md file, displaying 86.38%, on average. We also found that miscellaneous scripts, computer vision and general-purpose machine learning projects show, on average, the most code coverage, ranging from 94% and 91.67% to 89.35%, respectively. While neural networks present the least mean coverage overall (68%).

A possible reason for the high coverage of miscellaneous scripts is the small number of projects that actually report it, with just 1 (4.76%) project reporting its statistics. Meanwhile, general purpose projects have one of the largest adoptions of coverage reporting (33.06%) and one of the highest mean coverages (89.35%). We speculate that a reason for the low rate of coverage on neural networks might be attributed to the difficulty of testing their learning algorithms. In general, the test coverage is relatively high (86.38%), but only a small proportion of projects publish these statistics (27.3%).

4 LIMITATIONS

There are some threats to the validity of the results of our study, we made some concessions to enable the use of automated analysis. For the purpose of our research, we used only the section destined for python projects on the curated list of machine learning projects on Github that we referenced before. It could be argued that more languages should be considered, however, due to time constraints and complex requirements, we opted to focus on only one language. Moreover, Python is one of the most used programming languages by data scientists and ML developers.

Our initial corpus had 344 entries, however several of them contained problems, a few of them were duplicated and there were multi-language projects where Python played a minor role, which could add bias into our analysis. As well as python notebooks, which could not be analyzed by some of our tools. To mitigate those problems, we filtered out projects with less than 20% of Python code and the interactive notebooks as well. There were also entries that pointed to projects outside Github, so we built scripts to scrape the likely main repositories and manually verified the results.

For collecting all the libraries related to testing, we built scripts to heuristically search for imports, isolate possible library names, and check their descriptions and summaries on PyPI for mentions of "test". This could result in libraries that are not necessarily related to testing. So we also manually validated all the libraries, by reading their summaries. Besides that, one characteristic of our corpus is that there are considerably different proportions of projects by category, and some of them had only a few repositories, which can also bias the result.

Lastly, we only collected code coverage reported by the repositories on their README.md file, due to the wide range of strategies that the authors use to publish them, such as Codecov[13], Coveralls[14], Code Climate[15]. And even when a project apparently uses one of those solutions, the integration could be outdated or they could have used more than one of them in the past. To manually retrieve and validate the coverage of all projects would be potentially hard; and it could take too much time to collect a satisfactory sample. We are aware that it can bias the results to only consider reported values. But we believe that is an acceptable compromise for an initial analysis of a variety of projects.

5 RELATED WORKS

In this section we explore related works that deal with machine learning testing and open source project mining for software testing practices.

Zhang et al [12] conducted a comprehensive survey in 138 papers on machine learning testing research. They also defined machine learning testing, analyzed and reported research topic distribution, datasets, and trends on ML testing. As a result, they found that the majority of research is made on supervised machine learning (over 110 papers), and mostly focus on robustness and correctness, meanwhile only a small proportion deal with test interpretability, privacy, or efficiency. The authors also identified challenges, open problems, and promising research. While this paper focuses on machine learning testing and its challenges, our study is centered over how machine learning projects are being tested in practice, by mining and extracting test data over open source repositories.

Silva et al [5] investigate the adoption of testing on open source ecosystems, taking into account several programming languages and automated support mechanisms. After performing an empirical study over 184 popular Github repositories, they found that Go, PHP and Javascript are the ones that most adopt tests; and listed testing libraries, tools and frameworks most adopted for each language ecosystem. A test coverage analysis was also performed on 571 open source projects, from which Python was found to have, on average, 80.84% of code coverage. In comparison, our results show an average of 86.34% for python machine learning projects that publish their coverages on README.md files. The authors also reported a test adoption rate of 37.5% for Python, however we obtained a value of 65.19%. This might be due to their dependency recognition method ignoring built-in mechanisms, which we circumvent by processing each import statement on all test files.

Kochhar et al [8] performed a study on 627 Android open-source projects. They investigated, among other topics, the presence of test cases and measured code coverage. They also surveyed developers who have hosted their apps on Github to collect testing practices, where they question about tools that are used. As a result, they found automated tests at 14% of the projects. They also explored line and block code coverage on 41 projects with existing tests, and displayed challenges faced by developers while testing. Moreover, JUnit was established as the most adopted testing framework. And the mean line and block coverage were found to be 16.03% and 17.22%, respectively. In comparison, we deal with Python machine

[13]https://about.codecov.io/
[14]https://coveralls.io/
[15]https://codeclimate.com/

Table 5: Project coverage by category

Category	# of projects that publish coverage	%	Mean coverage (%)
Miscellaneous Scripts	1/21	4.76 %	94 %
Computer Vision	3/24	12.5 %	91.67 %
General-Purpose	40/121	33.06 %	89.35 %
Data Analysis	19/48	39.58 %	85.26 %
Natural Language Processing	9/38	23.68 %	83 %
Reinforcement Learning	2/11	18.18 %	81 %
Neural Networks	1/8	12.5 %	68 %
Overall	80/293	27.3 %	86.38 %

learning projects, where we found a much higher coverage, on average, of 86.38%.

At last, Kochhar et al [7] studies more than 20000 open source projects on Github. They explore topics such as correlation of tests and project metrics (e.g. bugs, project size, development team size, number of bug reports), and number of test cases by programming language. As a result, they found that 61.5% have at least one test case, and that C++, C and PHP have a higher number, on average, of test cases per project. They also established that Python has a mean of 41 test cases by project. Compared to our study, they allow for a wider project corpus, while we instead focus on machine learning projects written in Python, and explore which tools are used in source code. In our context, we also found an overall test adoption of 65.19%.

6 CONCLUSION

The use of machine learning in the last few years saw a spike in several industries. From that follows the importance of better testing those softwares. To try and guarantee more reliability, trustworthiness and fairness. However, we lack actionable data on how these applications are being tested to this day. That includes knowing test adoption statistics, which can give insights about the software testing maturity in this area, and what tools are being used, informing researchers and developers of what libraries to look for.

In this paper, we make four main contributions. Firstly, we show the test adoption for over 290 python machine learning projects divided in 8 categories. Counting the repositories that import at least one library related to testing. Secondly, we report 28 tools for testing purposes that we collected from our corpus, and their frequency usages per category, making clear which ones are more popular and their relative use. Thirdly, we compare projects with different testing profiles (e.g. that employ tests versus the ones that do not), in terms of code quality, to highlight the importance of software testing. And finally, we explore the code coverage of all the projects that publicly report them in the README.md file.

Our results show that 65.19% of the 293 projects adopt at least one testing support library. We established that unittest, pytest and doctest are the ones most often used, in this order. In addition, projects that adopt at least one automated testing support tool have less code smells (48.28% in median) and, on average, they have less vulnerabilities (71.42%). Major bugs can also be up to 98% less prevalent on these softwares, on average, in the case of computer vision. We also found an overall mean of 86.38% of test coverage and expanded those findings by category.

In our study we worked with an uneven distribution of projects per category and only collected coverage from the repositories README.md file. In the future, we could collect a more balanced sample and try to run coverage tools directly on the projects. Other opportunities would be to collect statistics on the test cases, such as their number, study which types of tests are being performed, explore the sonarqube's issue contents and the social aspect of those projects on Github (e.g. contributors and contribution guides). Lastly, other programming languages could be considered, to give a better picture of machine learning testing practices.

REFERENCES

[1] 2021. Developer Nation Pulse Report. https://www.developernation.net/developer-reports/dn21

[2] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M Paixao, Filipe Mutz, et al. 2021. Self-driving cars: A survey. *Expert Systems with Applications* 165 (2021), 113816.

[3] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2014. The oracle problem in software testing: A survey. *IEEE transactions on software engineering* 41, 5 (2014), 507–525.

[4] Sunitha Cheriyan, Shaniba Ibrahim, Saju Mohanan, and Susan Treesa. 2018. Intelligent sales prediction using machine learning techniques. In *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. IEEE, 53–58.

[5] Rômulo Martins da Silva, Cafer Cruz, Heleno de S. Campos, Leonardo GP Murta, and Vânia de Oliveira Neves. 2019. What is the adoption level of automated support for testing in open-source ecosystems?. In *Proceedings of the IV Brazilian Symposium on Systematic and Automated Software Testing*. 80–89.

[6] Danielle Gonzalez, Thomas Zimmermann, and Nachiappan Nagappan. 2020. The state of the ml-universe: 10 years of artificial intelligence & machine learning software development on github. In *Proceedings of the 17th International conference on mining software repositories*. 431–442.

[7] Pavneet Singh Kochhar, Tegawendé F Bissyandé, David Lo, and Lingxiao Jiang. 2013. An empirical study of adoption of software testing in open source projects. In *2013 13th International Conference on Quality Software*. IEEE, 103–112.

[8] Pavneet Singh Kochhar, Ferdian Thung, Nachiappan Nagappan, Thomas Zimmermann, and David Lo. 2015. Understanding the test automation culture of app developers. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 1–10.

[9] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. 2017. A survey on deep learning in medical image analysis. *Medical image analysis* 42 (2017), 60–88.

[10] Yuri Nieto, Vicente Gacía-Díaz, Carlos Montenegro, Claudio Camilo González, and Rubén González Crespo. 2019. Usage of machine learning for strategic decision making at higher educational institutions. *IEEE Access* 7 (2019), 75007–75017.

[11] Nicolas Ortiz, Ruben Dario Hernández, Robinson Jimenez, Mauricio Mauledeoux, and Oscar Avilés. 2018. Survey of biometric pattern recognition via machine learning techniques. *Contemporary Engineering Sciences* 11, 34 (2018), 1677–1694.

[12] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* 48, 1 (2020), 1–36.