Software Analytics for Continuous Integration

Guilherme Ferreira
Institute One
Address One
gferrei@ncsu.edu

Timothy Menzies
Institute Two
Address Two
menzies@ncsu.edu

ABSTRACT

By using continuous integration services, one can automate the process of building a system and making it run against existing test suites. However, as it can take a substantial amount of time for a system to be built and ran against test sets, it would be advantageous to know ahead of time whether a build is going to pass or fail. In this paper we show that it's possible to predict, with over 90% precision and accuracy, the status of a build using only commit data, such as commit churn and lines of code added. Moreover, we also show that can be achieved in a relative short amount of time, making just-in-time build prediction a feasible option. With build prediction, it's possible for developers to know ahead of time whether their build is likely to pass or not, thus saving time and resources.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

1. INTRODUCTION

Continuous Integration, commonly referred to simply as CI or build, has become an integral part of building medium and large-sized software systems [9]. It first emerged as a part of Extreme Programming (XP) [1] to allow for a decrease in integration time. The practice is now being used by major companies and small projects alike, allowing teams to automatically run a suite of tests and commands to the latest version of the code as soon as it is committed. By automatically and frequently merging developer's working copies of the code with the main branch of a version control system, using a CI process increases the quality and decreases the overall risk of a software project [8].

Although Continuous Integration has become a necessary an important part in today's agile development practices [8], not much research has been done in the area. With that, we set out to address the following research questions:

- **RQ1:** Is it possible to predict a build status using only static commit data?

  By using only static git data (such as code churn and lines added) to predict the result of the build, we are able to build a model that can predict the build result *before* the build starts to run, and therefore before the commit gets submitted to the remote repository. With that, developers have the chance to fix their code before pushing it, thus decreasing the likelihood of faulty code on the remote repository. Also, by avoiding pushing a faulty code, the developer saves resources and time on the continuous integration server.

- **RQ2:** Is it possible to use transfer-learning by treating the entire dataset in a bag-of-words fashion [29], thus training the model on both a project's and other projects' instances?

  If so, can we also obtain the same results using just the project data, without transfer learning? Although that might seem obvious for larger projects, the same should not be implied for smaller projects with a small number of builds.

- **RQ3:** We also take into account the total time it took to build and run the models for the prediction. By doing so we expect to find learners that can be built and run in a relatively small amount of time. With that, we assert the possibility of just-in-time prediction, so that we can make predictions as soon as a developer commits their code to their local repository version.

The remainder of the paper is organized as follows: Section 2 provides an overview of the related work in the literature, while Section 3 describes the methodology for the experiments. Section 4 goes on to describe briefly each learner, and Section 5 provides the results of our experiments. In Section 6 we present possible threats to the validity of this paper, and conclude and give directions for future work in Section 7.

2. RELATED WORK

Even though continuous integration is being widely used in agile development, not much research work has been done on the subject, especially not combining it with software analytics. Stolberg et al. [27] started testing the impact of continuous integration tools on testing during agile development. Staahl et al. [26] described how continuous integration tools are used in different industry software development scenarios and described their models. Holck et al. [16] described how continuous integration practices can impact quality assurance. Karlesky et al. [17] described how agile development and continuous integration are used in a embedded software development world.

3. METHODOLOGY

1. Dataset

The dataset used for our experiments was the TravisTorrent dataset [2]. It was constructed by getting the data from the Travis CI API. For each build, they combined already available attributes, such as build number nad build result, with an analysis of the build log (such as how many tests were run, which test failed, âĂę) and repository and commit data from GitHub (such as latency between pushing and building), acquired through GHTorrent [12].

The dataset contains over 2,640,000 builds spread over more than 1,000 different projects. From the 17,313,330 active OSS repositories on GITHUB in August, 2015, the data set contains a deep analysis of the project source code, process and dependency status of 1,300 projects. To do this, they restricted their project space using established filtering criteria to all non-fork, nontoy, somewhat popular (> 10 watchers on GITHUB) projects with a history of TRAVIS CI use (> 50 builds) in Ruby (898) or Java (402). Both languages are very popular on GITHUB (2nd and 3rd, respectively) [4]. Then, they extracted and analyzed build information from TRAVIS CI build logs and the GHTORRENT database for each TRAVIS CI build in its history. Well-known projects in the TravisTorrent data set include all 691,184 builds from RUBY ON RAILS, GOOGLE GUAVA and GUICE, CHEF, RSPEC, CHECKSTYLE, ASCIIDOCTOR, RUBY and TRAVIS.

Figure 1 shows a sample of the data fields from TravisTorrent.

2. Data preprocessing

The dataset originally had 32 features from Github and 23 from Travis CI. As we are trying to predict the result of the build, weâĂŹll use as the class variable the feature *build result*, which means weather the build failed or passed.

As many features are not useful to us, we exclude them from our analysis. Features as *Project name, branch name, language, github and travis IDâĂŹs, etc.* have no predictive power for our purposes. After that, we were left with 26 features from Github and 13 from Travis CI, including some important ones as *source churn, test churn, files modified, tests failed, build duration, etc..*

As we previously stated, we will be using only the features that can be extracted at development time, such as the ones obtained from Github, thus arriving at our final dataset for the purposes of our experiments.

3. Library used

In order to compare different classification techniques, we decided to use the open-source machine learning library scikit learn [22]. We leave the tuning of the parameters of each individual learner as a future study.

4. Per-Project prediction

For prediction the build result for an individual project, we chose the two biggest projects in number of builds and the two smallest ones:

- Rails - Build count: 430948
- Chef - Build count: 189039
- Intellij-elixir - Build count: 56
- Gitlab-ci-runner - Build count: 56

4. CLASSIFICATION TECHNIQUES

In this section, we briefly explain the eight families of classification techniques that are used in our study. Table I provides an overview of each technique.

A. Statistical Techniques

Statistical techniques are based on a probability model [18]. These techniques are used to find patterns in datasets and build diverse predictive models [3]. Instead of simple classification, statistical techniques report the probability of an instance belonging to each individual class (i.e., defective or not) [18].

In this paper, we study the Naive Bayes and Simple Logistic statistical techniques. Naive Bayes is a probability-based technique that assumes that all of the predictors are independent of each other. Simple Logistic is a generalized linear regression model that uses a logit link function.

B. Clustering Techniques

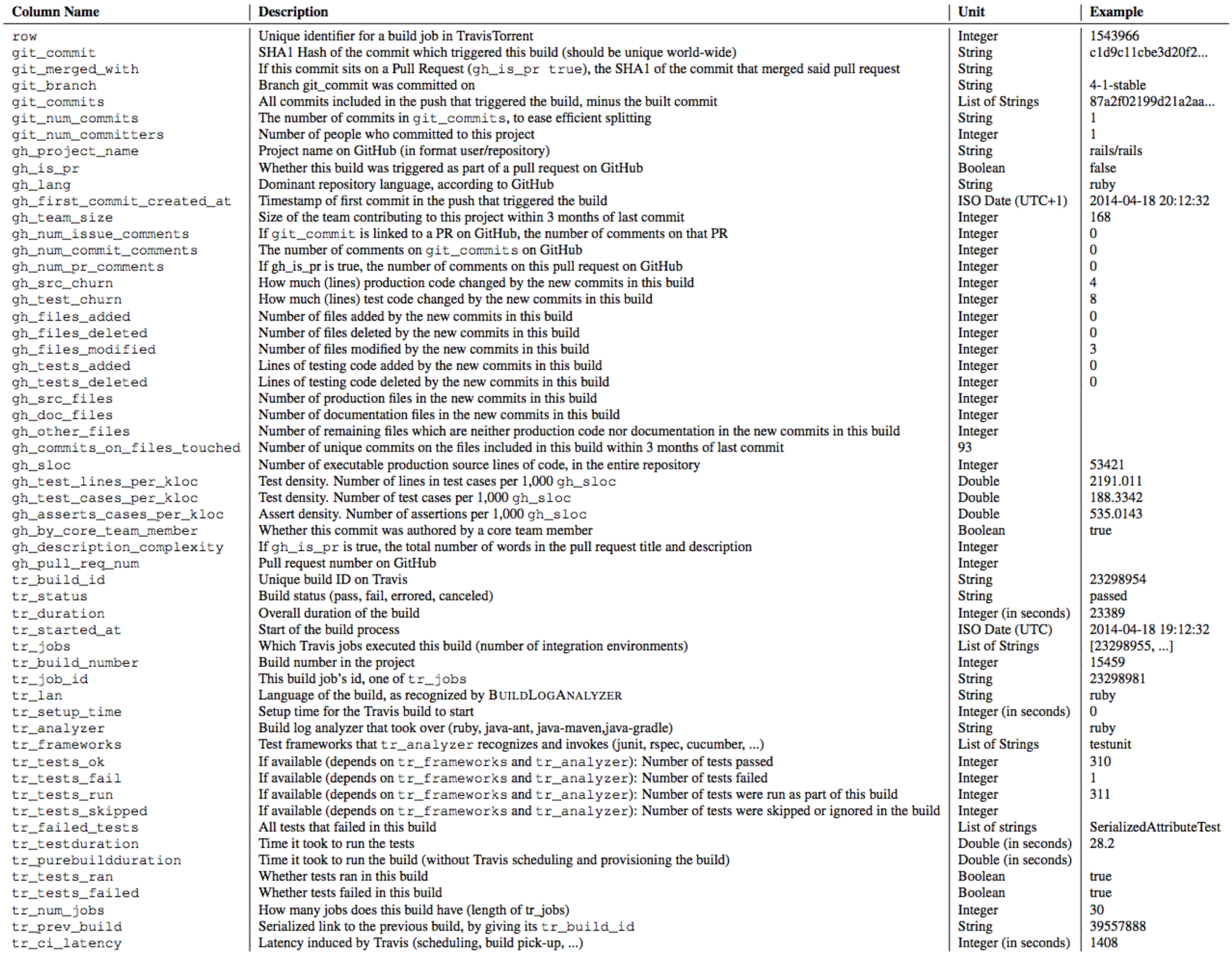
Clustering techniques divide the training data into small groups such that the similarity within groups is more than across the groups [14]. Clustering techniques use distance and similarity measures to find the similarity between two objects to group them together.

In this paper, we study the K-means technique. K-means divides the data into k clusters and centroids are chosen randomly in an iterative manner [28]. The value of k impacts the performance of the technique [19]. We used k as the same number of classes in the dataset, k = 2.

C. Neural Networks

Neural networks are systems that change their structure according to the flow of information through the network during training [25]. Neural network techniques are repeatedly run on training instances to find a classification vector that is correct for each training set [18].

In this paper, we study the Radial Basis Functions neural network technique. Radial Basis Functions [5] consists



Column Name	Description	Unit	Example
row	Unique identifier for a build job in TravisTorrent	Integer	1543966
git_commit	SHA1 Hash of the commit which triggered this build (should be unique world-wide)	String	c1d9c11cbe3d20f2...
git_merged_with	If this commit sits on a Pull Request (gh_is_pr true), the SHA1 of the commit that merged said pull request	String	
git_branch	Branch git_commit was committed on	String	4-1-stable
git_commits	All commits included in the push that triggered the build, minus the built commit	List of Strings	87a2f02199d21a2aa...
git_num_commits	The number of commits in git_commits, to ease efficient splitting	String	1
git_num_committers	Number of people who committed to this project	Integer	1
gh_project_name	Project name on GitHub (in format user/repository)	String	rails/rails
gh_is_pr	Whether this build was triggered as part of a pull request on GitHub	Boolean	false
gh_lang	Dominant repository language, according to GitHub	String	ruby
gh_first_commit_created_at	Timestamp of first commit in the push that triggered the build	ISO Date (UTC+1)	2014-04-18 20:12:32
gh_team_size	Size of the team contributing to this project within 3 months of last commit	Integer	168
gh_num_issue_comments	If git_commit is linked to a PR on GitHub, the number of comments on that PR	Integer	0
gh_num_commit_comments	The number of comments on git_commits on GitHub	Integer	0
gh_num_pr_comments	If gh_is_pr is true, the number of comments on this pull request on GitHub	Integer	0
gh_src_churn	How much (lines) production code changed by the new commits in this build	Integer	4
gh_test_churn	How much (lines) test code changed by the new commits in this build	Integer	8
gh_files_added	Number of files added by the new commits in this build	Integer	0
gh_files_deleted	Number of files deleted by the new commits in this build	Integer	0
gh_files_modified	Number of files modified by the new commits in this build	Integer	3
gh_tests_added	Lines of testing code added by the new commits in this build	Integer	0
gh_tests_deleted	Lines of testing code deleted by the new commits in this build	Integer	0
gh_src_files	Number of production files in the new commits in this build	Integer	
gh_doc_files	Number of documentation files in the new commits in this build	Integer	
gh_other_files	Number of remaining files which are neither production code nor documentation in the new commits in this build	Integer	
gh_commits_on_files_touched	Number of unique commits on the files included in this build within 3 months of last commit	93	
gh_sloc	Number of executable production source lines of code, in the entire repository	Integer	53421
gh_test_lines_per_kloc	Test density. Number of lines in test cases per 1,000 gh_sloc	Double	2191.011
gh_test_cases_per_kloc	Test density. Number of test cases per 1,000 gh_sloc	Double	188.3342
gh_asserts_cases_per_kloc	Assert density. Number of assertions per 1,000 gh_sloc	Double	535.0143
gh_by_core_team_member	Whether this commit was authored by a core team member	Boolean	true
gh_description_complexity	If gh_is_pr is true, the total number of words in the pull request title and description	Integer	
gh_pull_req_num	Pull request number on GitHub	Integer	
tr_build_id	Unique build ID on Travis	String	23298954
tr_status	Build status (pass, fail, errored, canceled)	String	passed
tr_duration	Overall duration of the build	Integer (in seconds)	23389
tr_started_at	Start of the build process	ISO Date (UTC)	2014-04-18 19:12:32
tr_jobs	Which Travis jobs executed this build (number of integration environments)	List of Strings	[23298955, ...]
tr_build_number	Build number in the project	Integer	15459
tr_job_id	This build job's id, one of tr_jobs	String	23298981
tr_lan	Language of the build, as recognized by BUILDLOGANALYZER	String	ruby
tr_setup_time	Setup time for the Travis build to start	Integer (in seconds)	0
tr_analyzer	Build log analyzer that took over (ruby, java-ant, java-maven,java-gradle)	String	ruby
tr_frameworks	Test frameworks that tr_analyzer recognizes and invokes (junit, rspec, cucumber, ...)	List of Strings	testunit
tr_tests_ok	If available (depends on tr_frameworks and tr_analyzer): Number of tests passed	Integer	310
tr_tests_fail	If available (depends on tr_frameworks and tr_analyzer): Number of tests failed	Integer	1
tr_tests_run	If available (depends on tr_frameworks and tr_analyzer): Number of tests were run as part of this build	Integer	311
tr_tests_skipped	If available (depends on tr_frameworks and tr_analyzer): Number of tests were skipped or ignored in the build	Integer	
tr_failed_tests	All tests that failed in this build	List of strings	SerializedAttributeTest
tr_testduration	Time it took to run the tests	Double (in seconds)	28.2
tr_purebuildduration	Time it took to run the build (without Travis scheduling and provisioning the build)	Double (in seconds)	
tr_tests_ran	Whether tests ran in this build	Boolean	true
tr_tests_failed	Whether tests failed in this build	Boolean	true
tr_num_jobs	How many jobs does this build have (length of tr_jobs)	Integer	30
tr_prev_build	Serialized link to the previous build, by giving its tr_build_id	String	39557888
tr_ci_latency	Latency induced by Travis (scheduling, build pick-up, ...)	Integer (in seconds)	1408

Figure 1: Description of TravisTorrent's data fields and one sample data point from RAILS/RAILS

of three different layers: an input layer (which consists of independent variables), output layer (which consists of the dependent variable) and the layer which connects the input and output layer to build a model [21].

D. Nearest Neighbour

Nearest neighbour (a.k.a., lazy-learning) techniques are another category of statistical techniques. Nearest neighbour learners take more time in the testing phase, while taking less time than techniques like decision trees, neural networks, and Bayesian networks during the training phase [18].

In this paper, we study the KNN nearest neighbour technique. KNN [6] considers the K most similar training examples to classify an instance. KNN computes the Euclidean distance to measure the distance between instances [20]. We used the default value of k = 8 for the purpose of this experiment.

E. Decision Trees

Decision trees use feature values for the classification of instances. A feature in an instance that has to be classified is represented by each node of the decision tree, while the assumption values taken by each node is represented by each branch. The classification of instances is performed by following a path through the tree from root to leaf nodes by checking feature values against rules at each node. The root node is the node that best divides the training data [18].

In this paper, we study the J48 decision tree techniques. J48 [23] is a C4.5-based technique that uses information entropy to build the decision tree. At each node of the decision tree, a rule is chosen by C4.5 such that it divides the set of training samples into subsets effectively [24].

F. Ensemble Methods

Ensemble methods combine different base learners together to solve one problem. Models trained using ensemble methods typically generalize better than those trained using the standalone techniques [7].

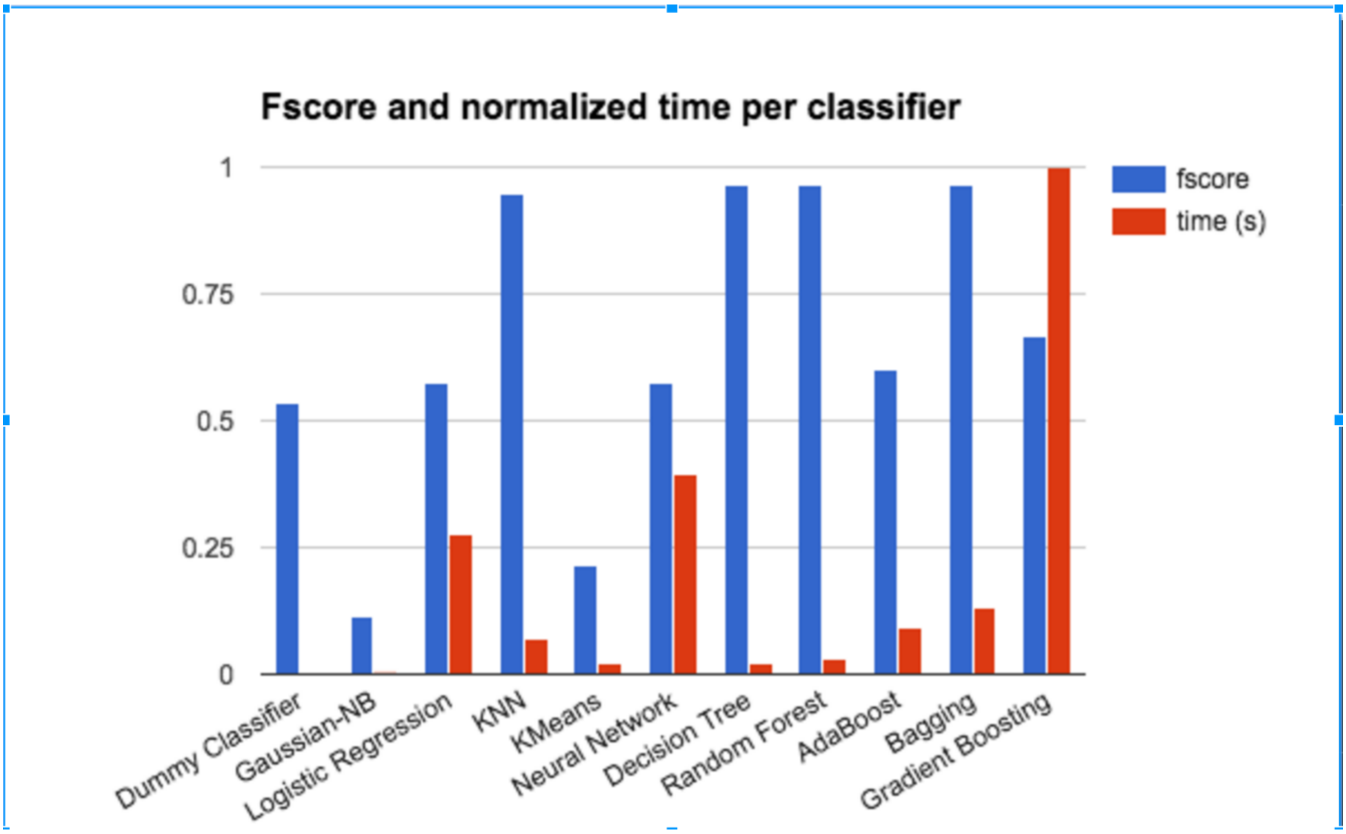


Figure 2: F1-score and normalized time per learner

In this paper, we study the Bagging, Adaboost, Gradient Boosting, and Random Forest ensemble methods. Bagging (Bootstrap Aggregating) [4] is designed to improve the stability and accuracy of machine learning algorithms. Bagging predicts an outcome multiple times from different training sets that are combined together either by uniform averaging or with voting [30]. Adaboost [10] performs multiple iterations each time with different example weights, and gives a final prediction through combined voting of techniques [30]. Gradient Boosting [11] builds an additive model in a forward stage-wise fashion, allowing for the optimization of arbitrary differentiable loss functions. In each stage multiple regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Random Forest [15] creates a random forest of multiple decision trees using a random selection attribute approach. A subset of instances is chosen randomly from the selected attributes and assigned to the learning technique [30].

G. Baseline Classifier

For comparison purposes only, we used a DummyClassifier that simply predicts for the majority class.

5. RESULTS

Figure 2 shows the F1-score [13] of applying each classification technique and the relative time it took them to build and run the model. The time results were normalized on a scale [0,1] for easy comparison between techniques.

As we can see from analyzing the results, four different techniques (*KNN, Decision Tree, Random Forest and Bagging*) showed F1-scores of over 0.9. Out of those four, three of them (*KNN, Decision Tree and Random Forest*) also presented very low runtime, making them desirable candidates for just in time prediction.

Figure 3 shows the results of using the data from just one project for training and testing (no transfer learning), for four different results. As we can see, bigger projects do have better results (F1-score close to 1), although smaller projects still show desireble numbers (F1-score over 0.8).

6. THREATS TO VALIDITY

Even though the dataset represents a diverse group of different software projects, they were all open-source projects obtained from public repositories in GitHub. Further analysis needs to be made to see if the results also hold for industry projects.

Although being one of the most used continuous integration tool, the experiments only considered projects that were built on Travis-CI. More experiments need to be executed to see if the overall results about continuous integration hold when the software projects used different CI tools, like Jenkins.

7. CONCLUSIONS AND FUTURE WORK

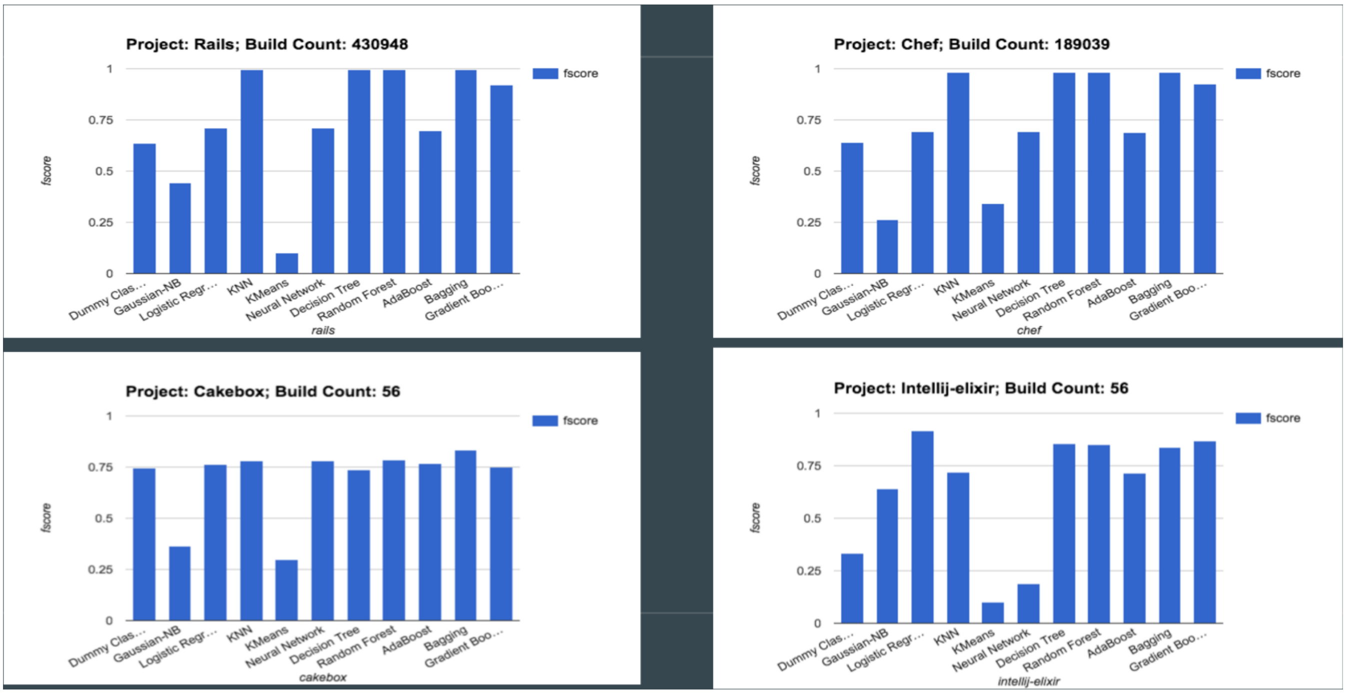


Figure 3: F1-score for four different projects

From our experiments we can clearly see that it is indeed possible to accurately predict the build results using just data from Github, and in a relatively small amount of time. Moreover, we also showed that not only we can obtain successful predictions even when the number of builds for a project is small, but we can apply transfer learning and use data from other projects in order to increase the accuracy of the prediction for projects that have a small number of builds.

As future work we plan on running this suite of experiments on industry, non-open source data to see if the results we achieved remain the same. We also intend on running the experiments on other popular CI tools like Jenkins.

For the purposes of this paper, we used the default values of the parameters of the classification techniques. As future work we plan on investigating whether parameter tuning can also affect build result prediction.

Another line of future direction is to use the build prediction result as a feature to aid in defect prediction and discovery.

8. REFERENCES

[1] K. Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[2] M. Beller, G. Gousios, and A. Zaidman. Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In *Proceedings of the 14th working conference on mining software repositories*, 2017.

[3] A. Berson, S. Smith, and K. Thearling. An overview of data mining techniques. *Building Data Mining Application for CRM*, 2004.

[4] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[5] M. D. Buhmann. Radial basis functions. *Acta Numerica 2000*, 9:1–38, 2000.

[6] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[7] T. G. Dietterich. Ensemble learning. *The handbook of brain theory and neural networks*, 2:110–125, 2002.

[8] P. M. Duvall. *Continuous Integration*. Pearson Education India, 2007.

[9] M. Fowler and M. Foemmel. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, page 122, 2006.

[10] Y. Freund, R. E. Schapire, et al. Experiments with a new boosting algorithm. In *Icml*, volume 96, pages 148–156, 1996.

[11] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[12] G. Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[13] C. Goutte and E. Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005.

[14] K. Hammouda and F. Karray. A comparative study of data clustering techniques. *Fakhreddine Karray University of Waterloo, Ontario, Canada*, 2000.

[15] T. K. Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*,

20(8):832–844, 1998.

[16] J. Holck and N. Jørgensen. Continuous integration and quality assurance: A case study of two open source projects. *Australasian Journal of Information Systems*, 11(1), 2003.

[17] M. Karlesky, G. Williams, W. Bereza, and M. Fletcher. Mocking the embedded world: Test-driven development, continuous integration, and design patterns. In *Proc. Emb. Systems Conf, CA, USA*, pages 1518–1532, 2007.

[18] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas. Supervised machine learning: A review of classification techniques, 2007.

[19] B. Lemon. *The effect of locality based learning on software defect prediction*. PhD thesis, Citeseer, 2010.

[20] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485–496, 2008.

[21] A. Panichella, R. Oliveto, and A. De Lucia. Cross-project defect prediction models: L'union fait la force. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 164–173. IEEE, 2014.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[23] J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[24] L. Sehgal, N. Mohan, and P. S. Sandhu. Quality prediction of function based software using decision tree approach. In *International conference on computer engineering and multimedia technologies (ICCEMT)*, pages 43–47, 2012.

[25] Y. Singh and A. S. Chauhan. Neural networks in data mining. *Journal of Theoretical and Applied Information Technology*, 5(6):36–42, 2009.

[26] D. Ståhl and J. Bosch. Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, 87:48–59, 2014.

[27] S. Stolberg. Enabling agile testing through continuous integration. In *Agile Conference, 2009. AGILE'09.*, pages 369–374. IEEE, 2009.

[28] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, et al. Constrained k-means clustering with background knowledge. In *ICML*, volume 1, pages 577–584, 2001.

[29] H. M. Wallach. Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*, pages 977–984. ACM, 2006.

[30] T. Wang, W. Li, H. Shi, and Z. Liu. Software defect prediction based on classifiers ensemble. *Journal of Information & Computational Science*, 8(16):4241–4254, 2011.