Automating Software Development using Artificial Intelligence

Venkatasaideep Nagulapati [1], Sai Rohit Rapelli [2], and Jinan Fiaidhi [2]

[1]Lakehead University
[2]Affiliation not available

October 30, 2023

Abstract

A paper exploring one of the applications of Artificial Intelligence in the field of Software Development

Automating Software Development using Artificial Intelligence

Nagulapati, Venkata Saideep
Lakehead University
vnagulap@lakeheadu.ca

Dr. Jinan, Fiaidhi
Professor
Lakehead University
jfiaidhi@lakeheadu.ca

Rapelli, Sai Rohit
Lakehead University
srapelli@lakeheadu.ca

***Abstract*— Developing software and maintaining software requires a large amount of changes to the source code to be made to a software repository. Any modification to a repository can introduce new resource needs that will cost repository owners more time and money. To help determine and allocate resources, it is therefore useful to predict future code changes. Hence, we proposed a technique to predict whether elements within a repository will change in the near future, given the repository's development history and these are developed using the machine learning approaches Support Vector Machine and Random Forest algorithms and using previous commit data. Current software development practices have been improved in recent years by using Artificial Intelligence (AI) techniques that include genetic algorithms, machine learning, and deep learning. The use cases for AI in software development ranged from developer feedback to full automation of development tasks for software developers. Software development and software maintenance require a large amount of source code changes to be made to a software repository. Any alteration to a repository will add new resource needs that will cost repository owners more time and money. Hence, forecasting possible code changes is useful in an attempt to help assess and distribute resources. To demonstrate the breadth of application, we will present several recent examples of how AI can be leveraged to automate software development[1].**

I. INTRODUCTION

Creating and managing a software application can be time consuming and resource intensive. Software application development commonly integrates the use of the Version Control Systems to manage the application by storing both the current and previous versions in a repository. Development of a repository is constrained by the resources available to the application development team. Every allocation of these limited resources can be the decisive factor in whether or not the repository is going to be successful. Predictions are made by identifying elements that are associated through frequent co-changes inside the repository. This helps to build a preceding research to provide predictions of change to help improve resource allocation for both repository developers and Version control system managers. Software has become pervasive and integrated with numerous platforms and applications such as mobile devices, web sites, embedded systems, safety critical systems. Creating and managing a software application can be time consuming and resource intensive.

The software intensive systems we develop these days are becoming much more complex in terms of the number of functional and non-functional requirements they need to support. In many vital applications, the impact of poor quality may also have a devastating effect on the purpose of those systems. In addition, software development costs exceed the total cost of such program. Work in the application of artificial intelligence techniques to software engineering has evolved immensely over the past two decades, generating several projects and publications. A number of conferences and journals are dedicated to publishing the research in this field. The AI strategies are being introduced to reduce market time and increase the efficiency of software systems. Yet many of these AI techniques remain primarily used by the testing community and have no influence on the software engineering practicing processes and devices. The recent survey papers published in this area are aimed primarily at the research community. They are guided by the different AI techniques used instead of the supporting software engineering activities. They often concentrate on a common process of software engineering, such as software design. If the current collection of AI technologies is adequate to achieve intelligence at the human level or not, it is clear that software systems will progressively integrate them as components and subsystems. The shape of the solutions created from these AI / ML technologies often looks radically different from the software which is normally developed and used. Thus, not only does the AI technology itself change quickly and at an increasing pace, the solutions it provides typically look very different from what soft- ware organizations and engineers are used to. This poses a new and unique set of risks and opportunities for software organizations and they need to understand and analyze these risks to select appropriate strategies[2].

The latest AI-based software and applications are developed using state-of - the-art machine learning models and techniques through comprehensive data training to incorporate various artificial intelligence functions and capabilities. Current AI-based system functions and features can be classified into the following categories:

- Natural language processing capability with language understanding and translation.
- Detection and recognition function, for example, human face detection, voice recognition and object detection.
- Recommendation features in e-commerce and advertising.
- Unmanned-controlled vehicles, robots, and UAVs

- Question and answer functions to assist users in messaging, phone calling, search, and smart home appliance control.
- Object identification and classification
- prediction and business decision-making.

II. Literature Review

A. Understanding and predicting change

- Developing and sustaining large-scale projects can take years or even decades and require a significant investment in time and money. Project managers will make improvements to the project in the production process. When a developer makes a modification to add to a project, a new feature can be added, or bugs can be addressed. A change can also involve unintended bugs and allow them to be implemented.

- Tracing and tracking the improvements made by the developers will ensure that the project follows the correct course and aligns closely with the intended path of the project after the modifications have been made. Change forecasts are an attempt to theoretically know what changes will occur before they are made. The changes will be more conscious with the awareness of what changes will occur. For example, if a section of the source code is classified as likely to alter the resources required, a developer working on that section will be more likely to be tasked with updating it. Likewise, understanding which changes are likely to occur will help developers plan their resources based on the value of what change.

- Prediction for software development includes various fields of research that typically aim to enhance individual projects by concentrating on their progress and providing developers feedback and recommendations.

B. Change Analysis

- Changes happen to accomplish a particular purpose or mission within a repository. The function may be high-level, such as adding new system functionality, or lower level, such as fixing a syntactic error. Investigations into how improvements are implemented or used will help to provide a deeper understanding to make a better change or use the improvements better.

- Bieman, Andrews, and Yang are researching the change-proneness of different organizations in a software project [7]. Visualizations were also used to provide a better understanding.

C. Software Development Prediction

- Predicting faults and changes in a software repository will allow developers or managers to build strategies to mitigate both faults and changes negative effects.

- Intuitively, detecting a flaw inside a library earlier will help minimize maintenance costs and the number of bugs found in the shipped version of the program. With a software program less susceptible to error, end users are less likely to find an error and can use the program as intended. The advantages of anticipating change are more closely linked to the development cycle helping developers to formulate plans and successfully introduce functional improvements with less faults than anticipated.

- Moser, Pedrycz, and Succi analyse fault predictions using static and shift metrics [9]. The transition metrics used outperformed, and remembered, the static metrics in precision. Instead, Sisman and Kak look directly at shift indicators to make the forecasts using the Information Retrieval (IR) system.

D. Machine Learning

Machine learning is a dynamic method of attempting to avoid trends within the data for software algorithms. One such example of a problem would be an algorithm for detecting certain people within an image. For a human such a task may seem trivial, but it is far more difficult for a software system to detect it. When these patterns are extremely complex, algorithms which can evaluate patterns and replicate them from abstract data set are useful. There are various algorithms that apply the techniques to machine learning.
Each solution has both advantages and inconveniences. Support Vector Machine, Random Forest, Artificial neural networks are few examples of machine learning algorithms. Bhattacharyya, Jha, Tharakunnel and Westland provide a detailed description of Random Forest (RF) and Support Vector Machine (SVM) [6] below:

- **Support Vector Machines**
  - Based on a collection of features given, an SVM is used to predict what form of change will occur. A function is a data derived from a floating-point number defined by the project. In order to be useful a function must define the group to which it is assigned in some way. Nor must the function depend on the group it belongs to for calculation
  - For example, given a change of method category within the next 5 commit or not, then the features must not depend on awareness of potential project changes. Unless the features fail to describe effectively the group to which they are assigned then the SVM may have weak predictions. It is also important for the features not to negatively affect the categorization, to be independent of one another.

- SVM includes encoding all of the function data as floating-point numbers. Conversion to floating-point is trivial for any numerical results. The conversion is a little more complicated for more complex data, though. Categorical data can be translated by category into one single vector entry.

- **Random Forests**

  - RF is a popular machine learning algorithm and is used in many fields including software fault predictions, software development effort, credit card fraud, database indexing, malware detection [8].

  - Malhotra offers a detailed overview of the machine learning studies to predict program faults [7]. RF appeared to perform better than other tested machine learning algorithms, the findings showed. Moeyersoms, Fortuny, Dejaeger, Baesens and Martens used RF and SVM, as well as a few other approaches to data mining to predict program faults and estimate effort [5].

  - One problem with decision trees and more generally machine learning techniques is imbalanced data sets for model training [10]. Therefore, the data set used rarely provided even sample sizes of each set without taking necessary precautions the result will be biased by the algorithm. In the worst case the model classifies any input data as the broader classification of the data [11].

III. PROPOSED MODEL

- The goal is to predict whether a method within a repository will change or not.
- The commit data is collected from open source repository to make predictions.
- By predicting short term changes, we can focus on impending changes rather than changes happening in future.

Why Change prediction?

- Change predictions are an attempt to potentially know what changes will occur prior to the changes taking place.
- Developer who worked on this section can be assigned to change code.
- Similarly, knowing which changes are likely to happen, helps the developer to prioritize the resources accordingly.

Data Distribution:

- Most of the time our dataset contains, samples in one category.
- For example, a sample may contain 80 percent methods with no change and 20 percent with change in next 5 commits.
- Hence, training the number of methods with changes and without changes ideally, would be around 50 percent in next 5 commits.
- This is more apparent when dataset distribution category changes between training and testing's.

Techniques of Data distribution

- **Over sampling:**

  - Reducing the size difference between larger classification and smaller classification.
  - Over sampling increases number of samples by calculating each category and expanding the smaller category by re-sampling values from the dataset until both categories are equal.
  - If the smaller category is still smaller, the larger category will be reduced to size of smaller category.
  - This is used to ensure the distribution of data is preserved.

- **Under sampling:**

  - It removes samples from larger classification to reduce the difference in size with smaller classification.
  - This is applied to the dataset by measuring number of samples in each category.
  - The larger dataset is reduced by discarding samples until this dataset becomes same size as the small dataset.
  - This may reduce performance, hence under sampling may not be ideal.

IV. COMMIT DATA

The commit data is obtained from target Open Source Software repositories to make predictions. From this obtained open source commit data, the goal is to predict whether the code changes in the next commits or not. The different types of changes that are possible are additions, deletions and modifications. Below fig. 1 is generic structure of any GIT repository.

- **Data Training Range:**

  - The machine learning model which is used to predict the changes requires the data to be trained using train set. The training samples need to be categorized before providing them to machine learning algorithms. This training allows to predict when a new observation show up.

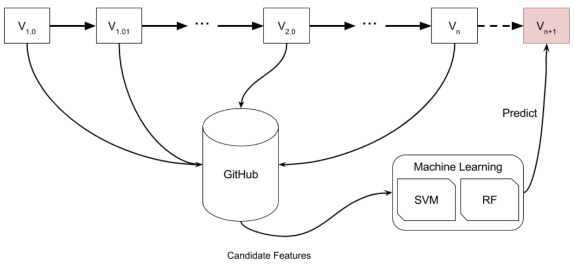

Fig. 1. GIT Repository Structure

– As the goal is to predict the changes occur in next commits, samples are made in the commit data and a sample of commit data is stored from the current commit.
– The sample set is restricted by a variable value sample window range (SWR) which deals with the number of commits considered to form a sample.

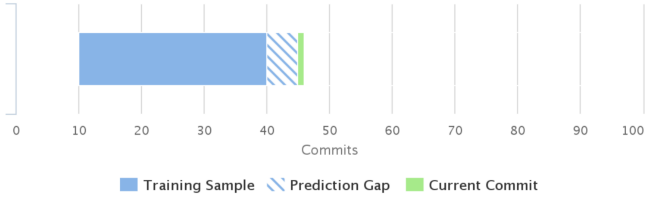

Fig. 2. Sample Window Range

The data will be sample with in only the limit of the range of dataset as show in the Fig. 2. Only 30 commits are considered and a prediction gap of 5 is taken. The prediction gap of 5 is used for the following reasons:

- So that the data sampling will be balanced and also improves the prediction capability in training model.
- Small prediction range will be more practical to obtain best results compared to large range.

Details in Commit Data

- Developer related.
- Source files.
- Changes made in the commit.
- Project release information in the form of tags.

V. Dataset Used

The dataset used is the commit data which is obtained from an opencv public repository from github. To obtain the commit data, as a first step git is installed in our system. All the commit logs from the opencv repository are pulled using local git.

Process of colleting log data

- As intial step, the repository url obtained from github. In our case it is **https://github.com/opencv/opencv**.

- This particular repository is cloned to our local system using git commands. In our case we used the command, **git clone https://github.com/opencv/opencv**

- After cloning the repository, we used to command to retrive all the log information and made a comma separated file (CSV) out of it. For that, we used the command,
git log –after="2016-09-06" –before="2017-04-11" –oneline> data.xls;
from the command, before and after are the date ranges in between which we need the log information.

- So, the obtained log information is shown in the below Fig. 3.

Fig. 3. Sample Data Set

- Commit data obtained contains the following details.

  – **Commit:**It contains the information about the commit id.
  – **Merge:** It contains the hashcode or merge id if the particular commit is a merge commit
  – **Author:** This field contains the information about the particular person which did the commit
  – **Date:** This field contains the information about the data on which particular commit occurred.
  – **Message:** It is the detailed message which is written which an author commits the data.

VI. Implementation

The log data obtained is in the format of string. This log data undergoes different preprocessing steps to feed it into any machine learning algorithm. Firstly, the data is split into different rows and then these data is parsed to extract features. The respective flow chart is given below in Fig. 4. The respective main step which are splitting, and parsing are explained below.

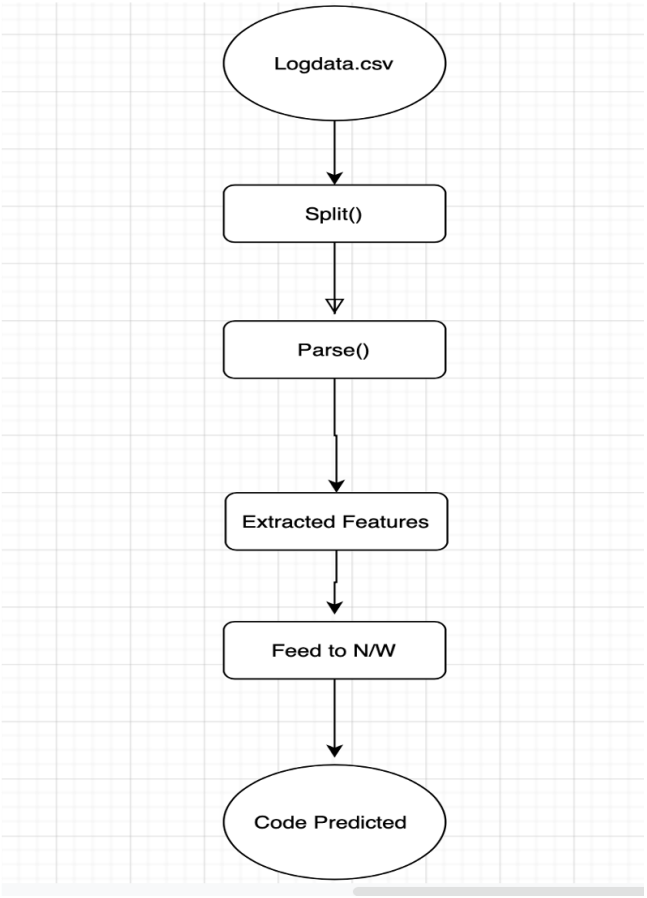

Fig. 4. Proposed Model Architecture

- **Splitting of data:** The commit data is divided between multiple lines as shown in Fig. 3. So, these multiple files are first split with respect to commit id. Following are the steps involved in split method.
  - Split the output of git log into separate entries per commit.
  - Parameters - whole_log: str A string containing the entire git log.
  - Returns - list(str) A list of log entries, with each commit as its own string.
  - Find the indices which separate each commit's entry.
  - Split the lines from the whole log into subsets for each log entry.
- **Parsing the data:** The lines split between the commit contains features which are to be extracted to pass them to the machine learning algorithms. The steps involved in parsing the data are given below. The respective code is shown in Fig. 6
  - Extract features from the text of a commit log entry
  - Parse the commit line, dateline, change lines, additions and deletions.
  - If this is a merge commit fill some fields with nan.

Following are the features to be extracted:

- Hash

```
def split_commits(whole_log):
    """Split the output of git log into separate entries per commit.

    Parameters
    ----------
    whole_log: str
        A string containing the entire git log.

    Returns
    -------
    list(str)
        A list of log entries, with each commit as its own string.
    """
    #print(whole_log)
    lines = whole_log.splitlines()
    #lines = whole_log
    #print(lines[1:100])
    # find the indices which separate each commit's entry
    commit_line_idxs = [i for i, line in enumerate(lines)
                        if re.match(r'^commit \w{40}$', line)]

    # split the lines from the whole log into subsets for each log entry
    commit_lines = np.array_split(lines, commit_line_idxs)

    return ["\n".join(arr) for arr in commit_lines[1:]]
```

Fig. 5. Code for Splitting commit data

- Day of week
- Hour
- Message Length
- Changed files
- Additions
- Deletions

The extracted features are fed into machine learning model random forest in our case.

The steps performed after extracting. features are:

- The extracted features as shown in Fig. 7. are sent to random forest tree algorithm. This uses ensemble learning methods from classification.
- Train and test the extracted features from previous phase
- Visualizing and analyzing the results.

Its respective code is shown in the below Fig.8.

VII. Conclusion

In conclusion, We suggested a way to use the commit history to predict potential changes in the repository. The data used for predictions was obtained from the GitHub repository of Open Source Software (OSS). Then, several methods were used to simulate the data to help define key features for use in the prediction model. The features have been selected and a model has been developed to predict whether the code changes in the future.

From our experiment we could predict code changes correctly with 50 % accuracy

References

[1] https://www.sqrlab.ca/blog/2018/03/23/automating-software-development-using-artificial-intelligence/
[2] https://ir.library.dc-uoit.ca/xmlui/handle/10155/730
[3] Alam, M. S., and Vuong, S. T. Random Forest Classification for Detecting Android Malware. In Proceedings of the Green Computing and Communications, IEEE Internet of Things, IEEE Cyber, Physical and Social Computing (2013), pp. 663–669.

```
def parse_commit(commit_str):

    feats = defaultdict(lambda: None)
    lines = commit_str.splitlines()

    # parse the commit line
    commit_line = [line for line in lines if line.startswith('commit')][0]
    feats['hash'] = \
        trim_hash(re.match(r'commit (\w{40})', commit_line).group(1))


    # parse the date line
    time_line = [line for line in lines if line.startswith('Date:')][0]
    timestamp = re.match(r'Date: (.*)', time_line).group(1)
    # TODO: fix the hardcoded timezone
    created_at = pd.to_datetime(timestamp, utc=True).tz_convert('US/Central')
    feats['dayofweek'] = created_at.dayofweek
    feats['hour'] = created_at.hour

    # parse the body lines
    body_lines = [line.lstrip() for line in lines if line.startswith('    ')]
    feats['len_message'] = len('\n'.join(body_lines))


    if any([line.startswith('Merge:') for line in lines]):
        # feats['tag'] = 'MERGE'
        feats['changed_files'] = np.NaN
        feats['additions'] = np.NaN
        feats['deletions'] = np.NaN

        return feats

    # parse the changes line
    changes_line = lines[-1]

    changed_regex = r' ([0-9]+) file[s]{0,1} changed'
    insert_regex = r'.* ([0-9]+) insertion[s]{0,1}'
    delete_regex = r'.* ([0-9]+) deletion[s]{0,1}'
```

Fig. 6. Code for Parsing Commit Data

hash	dayofweek	hour	len_message	changed_files	additions	deletions
9b32771	2.00000	11.00000	21.00000	2.00000	8.00000	11.00000
a1460ef3	2.00000	9.00000	22.00000	9.00000	80.00000	56.00000
64204eac	2.00000	3.00000	74.00000	nan	nan	nan
d6e99d0b	1.00000	4.00000	65.00000	nan	nan	nan
ee61cba9	3.00000	11.00000	145.00000	11.00000	23.00000	26.00000
8ebce16e	1.00000	7.00000	28.00000	1.00000	3.00000	nan
9818f481	4.00000	6.00000	64.00000	nan	nan	nan
bbe38930	3.00000	3.00000	19.00000	1.00000	2.00000	2.00000
c1bfb00d	3.00000	3.00000	51.00000	1.00000	289.00000	232.00000
f0121eba	3.00000	2.00000	54.00000	2.00000	2.00000	27.00000
0c691ff0	2.00000	8.00000	24.00000	1.00000	248.00000	248.00000
12b530c6	1.00000	18.00000	34.00000	3.00000	46.00000	53.00000
e496e9bc	1.00000	2.00000	74.00000	nan	nan	nan
8fdf85e0	1.00000	2.00000	26.00000	1.00000	16.00000	11.00000
dc30a205	0.00000	22.00000	20.00000	1.00000	nan	2.00000

Fig. 7. Sample Features Data

```
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size = 0.25, random_state = 42)
rf = RandomForestRegressor(n_estimators = 10, random_state = 42)
# Train the model on training data
rf.fit(x_train, y_train)
predictions = rf.predict(x_test)

from sklearn.metrics import accuracy_score


from sklearn.metrics import roc_auc_score
roc_value = roc_auc_score(y_test, predictions)
print(roc_value)

from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(x_train, y_train)
pred = neigh.predict(x_test)
print(accuracy_score(y_test, pred))
```

Fig. 8. Code for Feeding into Network

[4] Anto n, J. C. A ., Nieto, P. J. G., Viejo, C. B., and Vila n, J. A. V. Support Vector Machines Used to Estimate the Battery State of Charge. IEEE Transactions on Power Electronics 28, 12 (2013), 5919 – 5926.

[5] Bantelay, F., Zanjani, M. B., and Kagdi, H. Comparing and combining evolutionary couplings from interactions and commits. In Proceedings of the Working Conference on Reverse Engineering, WCRE (2013), pp. 311–320.

[6] Bhattacharyya, S., Jha, S., Tharakunnel, K., and Westland, J. C. Data mining for credit card fraud: A comparative study. Decision Support Systems 50, 3 (2011), 601–613.

[7] Bieman, J., Andrews, A., and Yang, H. Understanding change-proneness in OO software through visualization. In Proceedings of the 11th IEEE Interna- tional Workshop on Program Comprehension, 2003. (2003), pp. 44 – 53. 133

[8] Burbidge, R., Trotter, M., Buxton, B., and Holden, S. Drug design by machine learning : support vector machines for pharmaceutical data analysis. Computers and Chemistry 26, 1 (2001), 5 – 14.

[9] Canfora, G., Cerulo, L., and Di Penta, M. Identifying changed source code lines from version repositories. In Proceedings of the 4th International Work- shop on Mining Software Repositories, MSR 2007 (2007), pp. 14 – 22.

[10] Chaturvedi, K. K., Kapur, P. K., Anand, S., and Singh, V. B. Predicting the complexity of code changes using entropy based measures. International Journal of System Assurance Engineering and Management 5, 2 (2014), 155–164.

[11] Collberg, C., Kobourov, S., Nagra, J., Pitts, J., and Wampler, K. A system for graph-based visualization of the evolution of software. In Proceedings of the 2003 ACM symposium on Software visualization - SoftVis '03 (2003), pp. 77 – 86.

[12] De Souza, C. R., Quirk, S., Trainer, E., and Redmiles, D. F. Supporting collaborative software development through the visualization of socio- technical dependencies. 2007 International ACM Conference on Supporting Group Work, GROUP'07, November 4, 2007 - November 7, 2007 (2007), 147– 156.

[13] Dit, B., Holtzhauer, A., Poshyvanyk, D., and Kagdi, H. A dataset from change history to support evaluation of software maintenance tasks. In Proceedings of the 10th Working Conference on Mining Software Repositories (2013), pp. 131–134. 134

[14] Erturk, E., and Sezer, E. A. A comparison of some soft computing methods for software fault prediction. Expert Systems with Applications 42, 4 (2015), 1872–1879.

[15] Gall, H. C., and Lanza, M. Software Evolution : Analysis and Visualiza- tion. In Proceedings of the 28th international conference on Software engineering (2006), pp. 1055–1056.

[16] Giger, E., Pinzger, M., and Gall, H. C. Can we predict types of code changes? An empirical analysis. In Proceedings of the 9th IEEE Working Con- ference on Mining Software Repositories (MSR), 2012 (2012), pp. 217–226.

[17] Gilbert, E., and Karahalios, K. CodeSaw: A social visualization of distributed software development. In Proceedings of the 11th IFIP TC 13 interna- tional conference on Humancomputer interactionVolume Part II (2007), pp. 303– 316.

[18] Gondra, I. Applying machine learning to software fault-proneness prediction. Journal of Systems and Software 81, 2 (2008), 186–195.