IS- Commit Classification

hsmalapa

September 2018

1 Introduction

Due to the requirement of version control in today's life, there are many tools available to handle it. There is so much that we can know from it by understanding it in a deeper sense. We might just think, we are having a copy of all the version of the same task (say a program) saved. But we can say there's something more than it if we dig deeper.

Every time a version is made and saved, we called it a commit in terms of version control context. When ever a commit is issued by user/developer,some text is given along with it. There's so much knowledge in this text if we can analyze. If we can gather all the commits and start to read them, no wonder, we might realize that there is something behind them, and interested to know them.

To start classifying these commit messages, we can start off by knowing what kinds of works are already existing in this area and knowing the advantages and disadvantages of each of the method that was used.

2 Literature Study (Before starting the project)

I saved all the papers which I read in the drive and here's the link for it :
https://drive.google.com/open?id=1uxKWLCyxVs8Y4g882FzEHWPavBgC7ORl

This is the link for the excel sheet where I classified the commits. There were 100 commit messages int he file which Niki shared.
https://drive.google.com/open?id=12IqFeACsS4CjgbHnmUPAA4K8hBI9IMDR

How to get further information?

- Can we get the description on the commit along with the 1 line subject message?
- Can we even access the code?
- Can we know the timeline of commits?
- Can we see which branch has most of the commits?
- Can we see the popularity of a repo and then categorize the commits?

Generally, the code commits should be in present tense and not in past tense so that we can revert them conveniently because we understand them. If they are in past tense, after reverting, the tense doesn't look apt.

But If we observe the data we have, almost all commit messages are in the form of "verb + object" and verbs are in past tense. However there might be around 5-6 categories to classify them. The main categories are something like : Corrective, Adaptive, Perfective. The secondary categories are something like: Testing, Merging, Progressing.

Going even further deep, if we know the language they are using from the keywords (if mentioned in commit messages) would be even better.

3 Summary of Papers Read

Here's a brief summary of all the research papers I've read so far :

1. **Sentiment Analysis of Commit Comments in GitHub: An Empirical Study**

   This work concentrates on determining whether a commit can be classified as positive or negative. This completely depends on context and semantics of the commit message. Sentiment Analysis using SentiStrength was done. It's a lexical sentiment extraction tool. The work focussed on projects where there are over 200 commit messages. It collected all the commits, started giving scores to each posotove/negative word. Words with a negative emotion are given a value between [-5, -1] and words with a positive emotion are given a value in the [1, 5] range. The 1 and -1 values are used to give neutral scores to words, whereas 5 and -5 are used for words with a very positive and very negative emotion respectively. After this, they took mean and standard deviation grouped by weekday over all projects.

   Their study had 4 potential questions :

   R1) Are emotions in commit comments related to the programming language in which a project is developed?

   R2) Are emotions in commit comments related to the day of the week or time in which the commits were written?

   R3) Are emotions in commit comments related to the team geographical distribution?

   R4) Are emotions in commit comments related to project approval?

2. **How Do Centralized and Distributed Version Control Systems Impact Software Changes?**

   This work focuses on grouping commits according to the following categories :

   1. Distribution
   2. Maintainance
   3. Enhancement

   820 software developers were selected as participants for this work.To get further insights into how DVCS affects code changes, also we analyzed 409M lines of code changes from 358300 commits, made by 5890 developers, in 132 repositories containing a total of 73M LOC. SVN and Git were chosen as representatives for repositories. This work discovered that many commits do not represent actual programming changes carried out by a developer (e.g., adding features, bug fixing, refactoring, etc.), but are the result of applying tools such as code formatters.

   It concentrated on commit metrics like :

   1. Commit Id
   2. Commit date
   3. commit author

   4.Number of LOC changed by the commit

   5.Number of files impacted by the commit

   6.Number of issues referenced in the commit message

The research focuses on how the team size affects VCS usage. Much of this paper concentrates on differentiating between CVCS and DVCS.

3. **What Do Large Commits Tell Us? A taxonomical study of large commits**

   The main goal of this paper is anlayze large commits and determine patterns from them. To address this goal we performed a case study that included the manual classification of large commits of nine open source projects. Large commits are more perfective while small commits are more corrective. A common purpose of large commit is massive copyright change, or reformatting its source code. This work emphasizes that learning commit patterns helps development and maintainance teams. The resarch point if view is behind the questions :

   1. What are the different types of large commits that occur in the development of a software product?
   2. What do large commits tell us about the evolution of a software product?

   Focusing on part 1 : different types of commits are : Corrective, Adaptive, Perfective, Implementation, Non Functional.

   The way they classified them were : Only 1 percent commits, per each project, were chosen that contained the largest number of files.

   They came up with 26 types of categories. It now became hard to place these 26 into their initial 5 categories. So a new taxonomy was created for Large Commits :

   Implementation, Maintenance, Module Management, Legal, Non Functional Source Changes, SCS Management, Meta Program

   They manually read every commit - and identified the files changed. The diff of the commit was studied and compared to the commit log. This 2-sided information would improve the classification quality according to the authors. Then each commit was classified into one or more types (7 types stated above) Then quantitative analysis of each project regarding it's commits was done and even on average of all projects.

4. **Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes**

   This work uses 3 main classification categories for maintenance activities:

   Corrective: fault fixing;

   Perfective: system improvements;

   Adaptive: new feature introduction.

   The commits are automatically classified based on source code changes. The data used is 11 popular open source projects from various professional domains. For the testing set, an accuracy of 76% and kappa of 63%. They even tested against cross projects and it's accuracy came out to be

5. **GitcProc: A Tool for Processing and Classifying GitHub Commits** This paper presents

   GitcProc, a lightweight tool based on regular expressions and source code blocks, which downloads projects and extracts their project history, including ne-grained source code information and development time bug xes. GitcProc can track changes to both single-line and block source code structures and associate these changes to the surrounding function context with minimal set up required from users.Managers, de- velopers, and researchers are often interested in analyzing such

   evolutionary data to understand the characteristics of software development and maintenance.

6. **"May the Fork Be with You": Novel Metrics to Analyze Collaboration on GitHub**

   A single repository (is enough to analyze)to measure the characteristics of a single software product is no longer a safe assumption. This project makes use the essence of of Decentralized Version Control Systems (DVCS). This work concentrates on aggregation and analyzing commit histories of GitHub forks related to the same project. A classification of commits, explicitly conceived for the analysis of DVCSs, which characterizes their distributed development process.

unique: Commits existing in one fork only. – vip: Commits existing in several (but not all) forks and in the mainline. – u–vip: Commits existing in the mainline and in one other fork only. – scattered: Commits existing in several forks, but not in the mainline. – pervasive: Commits existing in all repositories.

This way of classification is not an apt idea since we are dealing with educational data (where commits are done by students and not industry developers). And generally students do not use multiple branches and forks for a semester long project.

7. **Investigation and Detection of Split Commit** commits. In this research, we firstly investigate how many and what kinds of split commits are included in repositories. It's focused on merging commits into which a task is split as a single appropriate commit. In this research, we call such a pair of commits containing a split task split commit.

   The contributions of this research are as follows:

   • investigating how many and what kinds of split commits are included in repositories, and

   • proposing a new technique to find split commits automatically.

   Two repositories, Apache Commons Collection1 and Retrofit2 are considered in this work. Manually checking were done to see whether every pair in these commits was a split commit or not.

   It was found that 81 split commits were included in both repositories by checking 1,714 commit pairs. By investigating characteristics of the 81 split commits in more detail, they categorized them with the following three-level classification.

   1) Level-1 (snippet level changes) : c2 need not be a single independent commit at all. For example, adding some changes for overlooked code fragments or reverting previous changes are Level-1. In Level-1, c1 and c2 change the same or very closely located code.

   2) Level-2 (method level changes) : the changes in c2 depend on the changes in c1. In most cases of Level-2, two methods having a calling or an inheritance relationship are changed in c1 and c2. For example, a commit adding a new method and a commit adding calling the method to another method are Level-2 split commit.

   3) Level-3 (function level changes) : changes in c1 and c2 are adding, deleting or modifying functions of the same class or module. For example, a commit adding a getter method for a member of a class and a commit adding a setter method for the same member form a Level-3 split commit.

8. **Dataset of Developer-Labeled Commit Messages** This work is a collaboration on author and his 6 close aquiantances. They all are frequent commiters for development projects. This work can be split into 4 parts:

   1. The 7 developers had to fill a survey where they were given their codes and commit messages. They had to classify the commit into the 3 categories :

   Maintanance related - these are again subdivided into : Corrective, Adaptive, Perfective

   NFR (Non Functional Requirements) - there are again subdivided into Functionality, Reliability, Usability, Efficiency, Maintainability and Portability

   Software Evolution Tasks - these are further classified into : Forward Engineering, Re-engineering, Corrective-Engineering and Management

   2. Now all the results were aggregated as per the developer name. The top 10 rows were removed from the dataset in every developer's classification as it would be good enough as a training set.

9. **Mining GitHub: Why Commit Stops** This work is about developer commit patterns in GitHub, and try to mine the relationship between these patterns and code that got effected due to that commit.Four metrics to measure commit activity and code evolution:

   the changes in each commit,

   the time between two commits,

   the author of each changes,

and the source code dependency.

After analyzing these, 2 patterns were developed and using these 2 patterns and developer's commit model, we can predict when his next commit comes and which file may be changed in that commit.New version of freeware usually is released in an indefinite frequency; and developers often do not commit anything for a long time. This is a common phenomenon in GitHub.To find more this phenomenon, this paper aims at exploring the relation between developer commit patterns and file version evolution.

They developed a model by name CAT (Commit Analysis Tool) which has modules to generate metric which are more explained in next paragraph. Finally, through the case study of eight projects from GitHub, we reveal the following patterns:

1)the later changes can be mostly associated with the previous changes according to the edges in dependency direct graphs

2) the developers' commits can be grouped into several series and the average gap between huge commits is much longer than other commits.

10. **Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes**

    In this work, the commits were classified as maintainance activities taking place in repository. The 3 classes are : Corrective, Perfective Adaptive. Previous works are done in the same lines, but this work aims at data from different projects (Previous works were based only on 1 project) The procedure they followed was : First they collected commit data, commit messgae and the code changes of them Then one expert would manually label them into the above mentioned 3 categories. Later, second expert would do the same independently to check the agreement level. Then created a model which takes 85 percent and was tested on the rest 15 percent. The accuracy of the model was 76 percent and kappa was 63 percent.

4 Citations and Related Work

The papers presented in before section are repeated in same order. If they are cited in any paper, that work might be continuation/future work for this particular paper. Hence they have been studied too. In case they have anything else which aren't covered in the above section are summarized here.

For few papers, although it's cited, no new/related work which is relevant to this study is there.

- 1.Sentiment Analysis of Commit Comments in GitHub: An Empirical Study

  Work was done based on classifying commits on polarity based - positive and negative. Paper : Sentiment Polarity Detection for Software Development

  Among the top predicting keyword-based features, we find positive and negative emoticons (Positive emotions and Negative emotions, respectively). Expressions of gratitude (i.e., 'thanks') and appreciation (i.e., 'great', 'excellent') are also among the top uni-gram predictors, thus confirming evidence from previous research that paying gratitude for the help received as well as enthusiasm for the solution provided are the main causes for positive sentiment in the social programmer ecosystem

- 2. How Do Centralized and Distributed Version Control Systems Impact Software Changes?

  Paper : Accurate and Efficient Refactoring Detection in Commit History

  Mainly used data regarding refactor methods and used RMiner - Repository Miner.

  These 4 methods were used :

  Package

  Method

  Field

  Type

- 3.What Do Large Commits Tell Us? A taxonomical study of large commits

  Paper : Judging a commit by its cover: Correlating commit message entropy with build status on travis-ci

  Developers summarize their changes to code in commit messages.When a message seems "unusual", however, this puts doubt into the quality of the code contained in the commit. We trained n-gram language models and used cross-entropy as an indicator of commit message "unusualness" of over 120,000 commits from open source projects.Build statuses collected from Travis-CI were used as a proxy for code quality. We then compared the distributions of failed and successful commits with regards to the "unusualness" of their commit message.

- 4.Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes

  Paper : Judging a Commit by Its Cover

  The cross-entropy of a commit message with respect to a language model in order to quantify its unusualness. To determine the quality of a commit, we use its build status as provided by Travis-CI,1 a continuous integration service popular among open source projects.The build status of any commit can then be evaluated with respect to the unusualness of its message as measured by cross-entropy

- 5. GitcProc: A Tool for Processing and Classifying GitHub Commits

  Nothing found so far - still searching

- 6. May the Fork Be with You": Novel Metrics to Analyze Collaboration on GitHub

  Paper : On Analyzing the Topology of Commit Histories in Decentralized Version Control Systems

  Thus there is information hidden in the very structure of the commit history graph, which is relevant to understand key characteristics of the development process typical of DVCS.In this paper we show that the topology of commit graphs presents recognizable patterns, whose recurrence is due to the nature of the underlying development process itself. Such patterns are not purposely designed by developers, who rather pursuit goals related to the state of their codebase. They thus emerge from the topologically rich structure of DVCS's and characterize the way the codebase evolves over time.

- 7. Investigation and Detection of Split Commit

  paper : Clustering Commits for Understanding the Intents of Implementation

  This approach detects identifier names related to changes in every commit, and classifies every commit via the bag-of-words model with the identifier names. Our approach does not use commit messages, which allows it to cover the weak point of the commit message based approach. Furthermore, it can consider semantics of commits because identifier names should provide us with an insight into semantics of changes occurring in each commit

- 8. Dataset of Developer-Labeled Commit Messages

  paper : Tracing Your Maintenance Work – A Cross-Project Validation of an Automated Classification Dictionary for Commit Messages

  The research area of the identification and classification of maintenance tasks in the software development process has evolved for decades. A maintenance task as an activity that can be assigned to one of the following three categories:

  Corrective Software Maintenance : Activities that are necessary to fix processing failures, performance failures or implementation failures

  Adaptive Software Maintenance : Activities that focus on changes in the data environment or changes in the processing environment

  Perfective Software Maintenance : Activities that strive to decrease processing inefficiency, enhance the performance or increase the maintainability

- 9. Mining GitHub: Why Commit Stops

  Paper :

- 10. Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes

  Not yet found - still searching

5 Tagging

So far the commit messages of project 3 in the given dataset is used. Manual tagging of random 240 commits are done so far. They will be more tagged as research proceeds.

The way they were tagged into the categories : Bug-fix(B), No-codeChange(N), Tuning/Adaptive/Perfective(T), Unknown(U), Implementation/Enhancements (E)

- If the commit message says "Fixed a bug" or a similar style sentence - it would be tagged as B-Bug-Fix
- If the commit message says "generated Javadoc" or "Finished testing", we can infer that there's no code changes. Thus they would be tagged as N - No-CodeChange
- If the commit message says something like "Updated something" or "Working on something", implies the code is being updated and enhanced, thus we tag them as E - Enhancement
- If the commit message says something like "removed unused packages" or "Test cases added", they are similar to improving the performance of the task. This would be slightly different than Enhancements. It's more like improving task rather than code. Thus it would be tagged as T - Tuning
- The last category is something like which doesn't belong to any of these. If we cannot infer anything from the commit message, it would be tagged as U - Unknown

6 How to implement?

After observing many works on commit classification, the taxonomy given below was found apt keeping in mind about Educational data (not industry data) where all the datapoints correspond to students. Considering our dataset, all the commits are generated by students. A student might be either beginner in using version control system and doesn't know how long and informative a commit should be. Or he might be lazy to give a descriptive commit. Thus in many cases, the commit message which we see is not too descriptive.

4 features were built based on the commit message. They are :

1. Tags : The values will be one among the following :

- Unknown(U)
- No code Change (N)
- Bug Fix/ Corrective (B)
- Tuning -Adaptive/Perfective (T)
- Implementation/Enhancing/Testing (E)

2. Size :

small(S)

Large(L)

3.Polarity of sentiment

- Positive
- Negative

4. Yet to decide

7 Result set 1

NOTE : results keep on changing. So the following might not be updated one. I'll keep updating as I progress.

I've implemented Decision Tree, Naive Bayes and SVM classifier. Since the category of commits isn't evenly distributed, I used stratified sampling and not random.

Here are my results :
The length of bag or words before processing 232
The length of bag of words after processing 153
The bag of words after processing are :

['check', 'illegal', 'remove', 'ui', 'finish', 'something', 'implementation', 'previous', 'testplan', 'bbtp', 'bound', 'complete', 'problem', 'arary', 'coverage', 'list', 'nullpointer', 'set', 'plan', 'push', 'listner', 'unused', 'fail', 'reoving', 'statements', 'figure', 'linkedlist', 'categories', 'pattern', 'implement', 'editable', 'git', 'drop', 'setchanged', 'generate', 'throw', 'taskid', 'exception', 'nfields', 'nfor', 'merge', 'object', 'lnkedlist', 'modify', 'visual', 'current', 'observers', 'fillfields', 'bug', 'properly', 'still', 'flag', 'grow', 'https', 'hopefully', 'checkstyle', 'warn', 'combobox', 'test', 'array', 'behavior', 'categorylisttest', 'run', 'length', 'repeatedly', 'pmd', 'contain', 'taskeditpane', 'increase', 'problems', 'task', 'ncontains', 'code', 'date', 'regain', 'failure', 'comment', 'edit', 'patern', 'errors', 'create', 'return', 'case', 'functionality', 'indexoutofboundsexception', 'nan', 'act', 'change', 'method', 'size', 'small', 'populate', 'show', 'branch', 'piece', 'element', 'believe', 'longer', 'categorylist', 'master', 'tasklist', 'green', 'throwwing', 'last', 'nnow', 'fix', 'format', 'udated', 'print', 'start', 'get', 'revert', 'like', 'todolist', 'npoiter', 'node', 'add', 'tasklistpane', 'gui', 'update', 'try', 'due', 'new', 'null', 'work', 'constructor', 'taks', 'observable', 'atempt', 'field', 'p3', 'improve', 'exceptions', 'order', 'debug', 'category', 'teacher', 'testfiles', 'argument', 'nullpointerexception', 'java', 'arraylist', 'mode', 'nexceptions', 'name', 'observer', 'growarray', 'package', 'javadoc', 'index', 'clean', 'containsin', 'box']

The shape of Dataframe is (100, 154)
The counts for N, E, I, B, T, U are : 23 5 15 43 10 2

DECISION TREE
Accuracy : 0.57
Classification Report
precision recall f1-score support
B 0.86 0.67 0.75 18
D 0.00 0.00 0.00 1
E 0.00 0.00 0.00 1
I 0.40 1.00 0.57 2
N 0.43 0.38 0.40 8
T 0.67 0.67 0.67 3
U 0.00 0.00 0.00 0
avg / total 0.66 0.58 0.60 33
Confusion Matrix :
[[12 0 1 1 2 1 1]
[0 0 0 0 1 0 0]
[1 0 0 0 0 0 0]
[0 0 0 2 0 0 0]
[1 1 1 2 3 0 0]

[0 0 0 0 0 1 2 0]
[0 0 0 0 0 0 0 0]]

Naive Bayes
Accuracy : 0.36
Classification report
precision recall f1-score support
B 0.14 1.00 0.25 2
D 0.00 0.00 0.00 1
E 0.50 0.12 0.20 8
I 0.60 0.50 0.55 6
N 0.57 0.31 0.40 13
T 0.67 0.67 0.67 3
U 0.00 0.00 0.00 0
avg / total 0.52 0.36 0.38 33
Confusion Matrix :
[[2 0 0 0 0 0 0]
[0 0 0 0 1 0 0]
[4 0 1 1 1 1 0]
[2 0 0 3 0 0 1]
[6 1 1 1 4 0 0]
[0 0 0 0 1 2 0]
[0 0 0 0 0 0 0]]

SVM
Accuracy :0.57
Classification Report
precision recall f1-score support
B 0.86 0.55 0.67 22
D 0.00 0.00 0.00 0
E 0.00 0.00 0.00 0
I 0.40 1.00 0.57 2
N 0.57 0.50 0.53 8
T 0.00 0.00 0.00 1
U 0.00 0.00 0.00 0
avg / total 0.73 0.55 0.61 33
Confusion Matrix:
[[12 0 1 3 2 2 1]
[0 0 0 0 0 0 0]
[0 0 0 0 0 0 0]
[0 0 0 2 0 0 0]
[2 1 1 0 4 0 0]
[0 0 0 0 1 1 0]
[0 0 0 0 0 0 0]]

8 Results Set 2

The output from program is
(8883, 7)
(8780, 7)
8780
The length of bag or words before processing 2216
After processing 2107

The length of bag of words after processing 1744
The shape of Dataframe is (240, 1744)
The counts for N, E, B, T, U are : 44 76 67 44 9
(240, 1500) (240, 1)
(240, 1500) (240, 1) 240 (240, 1)
DECISION TREE
Accuracy : 0.6458333333333334
Classification report
precision recall f1-score support
1 0.67 1.00 0.80 10
2 0.87 0.54 0.67 24
3 0.50 0.60 0.55 5
4 0.71 0.56 0.63 9
5 0.00 0.00 0.00 0
avg / total 0.76 0.65 0.67 48
Confusion Matrix
[[10 0 0 0 0]
[5 13 2 0 4]
[0 0 3 2 0]
[0 2 1 5 1]
[0 0 0 0 0]]
Naive Bayes
Accuracy : 0.4583333333333333
Classification report
precision recall f1-score support
1 0.53 0.80 0.64 10
2 0.47 0.78 0.58 9
3 0.67 0.29 0.40 14
4 0.29 0.22 0.25 9
5 0.20 0.17 0.18 6
avg / total 0.47 0.46 0.43 48
Confusion Matrix
[[8 1 0 1 0]
[1 7 0 0 1]
[1 4 4 4 1]
[2 2 1 2 2]
[3 1 1 0 1]]
SVM
Accuracy : 0.6041666666666666
Classification Report
precision recall f1-score support
1 0.53 1.00 0.70 8
2 0.93 0.47 0.62 30
3 0.50 1.00 0.67 3
4 0.57 0.57 0.57 7
5 0.00 0.00 0.00 0
avg / total 0.79 0.60 0.63 48
Confusion Matrix
[[8 0 0 0 0]
[5 14 3 3 5]
[0 0 3 0 0]
[2 1 0 4 0]
[0 0 0 0 0]]

9 Result set 3

DECISION TREE

Accuracy : 0.5694444444444444

Classification report

precision recall f1-score support

1 0.48 1.00 0.65 20

2 0.73 0.48 0.58 23

3 0.71 0.38 0.50 13

4 0.67 0.31 0.42 13

5 0.50 0.33 0.40 3

avg / total 0.64 0.57 0.55 72

Confusion Matrix

[[20 0 0 0 0]

[11 11 1 0 0]

[7 0 5 1 0]

[3 4 1 4 1]

[1 0 0 1 1]]

Naive Bayes

Accuracy : 0.2777777777777778

Classification report

precision recall f1-score support

1 0.56 0.25 0.34 20

2 0.50 0.22 0.30 23

3 0.25 0.38 0.30 13

4 0.27 0.31 0.29 13

5 0.06 0.33 0.10 3

avg / total 0.41 0.28 0.30 72

Confusion Matrix

[[5 1 3 7 4]

[3 5 7 2 6]

[1 0 5 2 5]

[0 2 5 4 2]

[0 2 0 0 1]]

SVM

Accuracy : 0.6111111111111112

Classification Report

precision recall f1-score support

1 0.92 0.60 0.73 20

2 0.46 0.91 0.61 23

3 0.83 0.38 0.53 13

4 0.86 0.46 0.60 13

5 0.00 0.00 0.00 3

avg / total 0.71 0.61 0.60 72

Confusion Matrix

[[12 7 0 1 0]

[1 21 1 0 0]

[0 8 5 0 0]

[0 7 0 6 0]

[0 3 0 0 0]]

10 Results set 4

The following changes were made, with which the accuracies increased as well.

1. the commit messages were increased from 240 to 300
2. Added Number of files changed as a new feature

The accuracies for all the three algorithms are increased.

DECISION TREE

Accuracy : 0.6666666666666666

Classification report

precision recall f1-score support

1 0.76 0.64 0.70 25

2 0.55 0.84 0.67 25

3 1.00 0.73 0.85 15

4 0.65 0.52 0.58 21

5 0.33 0.25 0.29 4

avg / total 0.70 0.67 0.67 90

Confusion Matrix

[[16 7 0 2 0]

[1 21 0 2 1]

[0 3 11 1 0]

[4 5 0 11 1]

[0 2 0 1 1]]

Naive Bayes

Accuracy : 0.4

Classification report

precision recall f1-score support

1 0.54 0.28 0.37 25

2 0.45 0.36 0.40 25

3 0.43 0.80 0.56 15

4 0.60 0.29 0.39 21

5 0.11 0.50 0.17 4

avg / total 0.49 0.40 0.40 90

Confusion Matrix

[[7 3 3 2 10]

[2 9 11 1 2]

[0 1 12 0 2]

[4 6 2 6 3]

[0 1 0 1 2]]

SVM

Accuracy : 0.6555555555555556

Classification Report

precision recall f1-score support

1 0.67 0.56 0.61 25

2 0.54 0.84 0.66 25

3 0.92 0.73 0.81 15

4 0.72 0.62 0.67 21

5 0.00 0.00 0.00 4

avg / total 0.66 0.66 0.64 90

Confusion Matrix

[[14 8 1 2 0]

[2 21 0 2 0]

[0 3 11 1 0]

[4 4 0 13 0]

[1 3 0 0 0]]

11 Result set 5

Now, the words in commit messages whose frequency is 1 are removed. Accuracy decreased.

DECISION TREE
Accuracy : 0.5111111111111111
Classification report
precision recall f1-score support
1 0.42 0.80 0.55 25
2 0.60 0.36 0.45 25
3 0.80 0.53 0.64 15
4 0.53 0.43 0.47 21
5 0.00 0.00 0.00 4
avg / total 0.54 0.51 0.49 90
Confusion Matrix
[[20 1 2 2 0]
[11 9 0 5 0]
[4 3 8 0 0]
[10 2 0 9 0]
[3 0 0 1 0]]
Naive Bayes
Accuracy : 0.3111111111111111
Classification report
precision recall f1-score support
1 0.70 0.28 0.40 25
2 0.43 0.24 0.31 25
3 0.41 0.47 0.44 15
4 0.36 0.38 0.37 21
5 0.00 0.00 0.00 4
avg / total 0.47 0.31 0.36 90
Confusion Matrix
[[7 3 3 3 9]
[2 6 2 6 9]
[0 2 7 3 3]
[1 2 4 8 6]
[0 1 1 2 0]]
SVM
Accuracy : 0.5
Classification Report
precision recall f1-score support
1 0.47 0.64 0.54 25
2 0.40 0.48 0.44 25
3 1.00 0.40 0.57 15
4 0.55 0.52 0.54 21
5 0.00 0.00 0.00 4
avg / total 0.54 0.50 0.49 90
Confusion Matrix
[[16 6 0 3 0]
[9 12 0 4 0]
[2 5 6 2 0]
[7 3 0 11 0]
[0 4 0 0 0]]

12 Result set 6

Here, unsupervised Learning - KNN was used

— Still in Progress

13 Result Set 7

300 with removing proj specific words

(8883, 7) ¡class 'collections.Counter'¿ The length of bag or words before processing 2216 1741 The length of bag of words after processing 1741 (300, 9) The shape of Dataframe is (300, 1938) The counts for N, E, B, T, U are : 49 83 83 71 13 (300, 1501) (300, 1) (300, 1) ¡class 'numpy.ndarray'¿ 210 (210, 1501) ¡class 'numpy.ndarray'¿ 90 (90, 1501) ¡class 'pandas.core.frame.DataFrame'¿ 210 (210, 1) ¡class 'pandas.core.frame.DataFrame'¿ 90 (90, 1) The test set after splitting using stratify [[3] [2] [3] [4] [3] [2] [2] [1] [2] [2] [2] [1] [1] [1] [4] [3] [2] [2] [2] [2] [2] [3] [4] [1] [3] [2] [3] [4] [4] [1] [2] [1] [1] [5] [3] [2] [2] [4] [1] [4] [4] [1] [5] [2] [1] [1] [3] [4] [2] [2] [2] [1] [1] [1] [4] [4] [4] [5] [3] [3] [1] [1] [5] [1] [2] [1] [4] [4] [3] [2] [2] [4] [4] [4] [3] [2] [1] [1] [4] [2] [1] [1] [4] [2] [4] [1] [1] [3] [3] [4]] DECISION TREE Accuracy : 0.6333333333333333 Classification report precision recall f1-score support

1 0.76 0.64 0.70 25 2 0.59 0.88 0.71 25 3 0.59 0.67 0.62 15 4 0.69 0.43 0.53 21 5 0.00 0.00 0.00 4

avg / total 0.64 0.63 0.62 90

Confusion Matrix [[16 7 0 1 1] [0 22 2 1 0] [0 4 10 0 1] [4 4 4 9 0] [1 0 1 2 0]]

Naive Bayes Accuracy : 0.4777777777777778 Classification report precision recall f1-score support

1 0.65 0.52 0.58 25 2 0.64 0.36 0.46 25 3 0.64 0.60 0.62 15 4 0.55 0.52 0.54 21 5 0.05 0.25 0.08 4

avg / total 0.60 0.48 0.52 90

Confusion Matrix [[13 0 1 3 8] [4 9 1 4 7] [0 1 9 1 4] [2 4 2 11 2] [1 0 1 1 1]] SVM Accuracy : 0.6555555555555556 Classification Report

precision recall f1-score support

1 0.80 0.64 0.71 25 2 0.50 0.92 0.65 25 3 0.90 0.60 0.72 15 4 0.79 0.52 0.63 21 5 0.00 0.00 0.00 4

avg / total 0.69 0.66 0.64 90

Confusion Matrix [[16 9 0 0 0] [1 23 0 1 0] [0 6 9 0 0] [2 7 1 11 0] [1 1 0 2 0]]

[0 0 0 1 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 4 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 4 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 3 4 0 0 0 3 0 0 0 0 1 0 4 0 4 4 4 4 0 4 2 4 4 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] [[0. 0. 0. ... 0. 0. 1.5202952] [0. 0. 0. ... 0. 0. 91.] [0. 0. 0. ... 0. 0. 24.] [0. 0. 0. ... 0. 0. 76.71428571] [0. 0. 0. ... 0. 0. 7.64705882]]