Number of lines in a repository
https://stackoverflow.com/questions/4822471/count-number-of-lines-in-a-git-repository

This is a project to analyse the logging statements using BIgquery and which is a similar approach has to investigate what is it exactly

https://blog.overops.com/779236-java-logging-statements-1313-github-repositories-error-warn-or-fatal/

I run some similarity test on the data

Paper: Automated suggestions for log changes

- Importance of log statements - debugging, software system understanding and development
- Logging should be consistent [not too little and not too large
- Proper logging will help to maintain the logging statements
- What to log is always an afterthought of after all the development
- They used the SVM blame statement to get the commitment in the code
- Check the type of code in the try-catch was the example
- And then they checked whether there is a logging statement within or outside the if statement
- They analysed that when they committed the statement and what all were advantageous if they logged beforehand

There are many logging statement enhancement tools like LogEnhancer and ErrLog to diagnose and suggest the logging statements
The main challenge of these existing project is that they do not incorporate the domain knowledge of the experts
All of the previous study suggested post coding guidelines for inserting loggin statements into the source code

The current paper that I am reading is basically focusing on just in time suggestion for logging statements
Where the log change is needed when a code change occurs
There are studies that advocate the importance of suggestion at commit time

This paper is all about the importance of change in the logging statement whenever there is a code change

1. What are the reasons for changing logging statements?
    a. Block change
    b. Log improvement

        c. Dependence-driven change and logging issues
2. How well we can provide a just-in-time log change suggestions?
    a. Provide a RF : use the line of codes before each commit as a training data to suggest is there any need for log change in the commit - it achieved a balanced accuracy of 0.76-0.82
3. What are the influential factors that explain log changes?
    a. Change measures
        i. Number of changed control flows
    b. Current analysis of the code
        i. Product measures
        ii. Number of existing Logging statements

Balanced Accuracy

It is calculated as the average of the proportion corrects of each class individually.

It is used when evaluating how good is a binary classifier is.
It is especially useful when the classes are imbalanced. Balanced accuracy is based on two more commonly used metrics: sensitivity(Tp rate/recall) and specificity (FP rate)
Recall = tp / tp+ fn whereas specificity = tn/tn+fp
Sensitivity answers the question: How many of the positive cases did I detect? Or How many (truly) defective product did I manage to recall
Specificity answers the same question but for the negative class
Balanced Accuracy is the arithmetic mean of the two (sensitivity + specificity)/ 2

Case study setup

They studied the reasons for log changes and explore the feasibility of providing accurate just-in-time suggestions for log changes

Svn log to retrieve the development history of the open-source java projects - Hadoop is one of them
Here they unroll all the merge commits

Data extraction -

Get the commit history from the version control repositories
Identify the log changes in the codebase

Classify them to log changes and labelled commits
Analyse the log changes manually and rationale for log changes
For the labelled commits + the source code at the time of commit create a random forest classifier
And observe the model evaluation and model performance
And also infer what all are the factors for log change

Log addition, deletion, and modification -> considered as the log change
They considered the basic standard libraries has the common format of inputting the logging statements - thus it is easy to identify the logging statements

They used reglar expressions to find out the added and deleted logging statements across commits
The textual similarity of the logging statements are considered by levenshtein distance ratio

Informally, the Levenshtein distance between two words is the minimum number of single character edits (insertion, deletion, or substitution) required to change one word into the other

It is also known as the edit distance

Two logging statements are considered similar if teh levenshteindistance ratio between them is larger than a specified threshold which we choose .5

Also analysing the impact of this threshold on the sensitivity of the model

There were lots of logging statement for the manual evaluation of understanding the reasons for logging change they took some random sample and evaluated the same
They concluded that the four categories of log change - block change log improvement and dependence driven change, and logging issue. The log change reasons give us valuable insights for defining measures to model the drivers for log changes

How well can we provide just-in-time log change suggestion?

- The approach is that they used a RF model a binary response variable which measure the likelihood of a log change recurring in a pariticular code commit
- Set of measures from 3 dimensions - change , historical , product

Soon after they finished finding the attribute they conducted a pair wise correlation analysis where they checked the pairwise correlation between our proposed measures using Spearman rank correlation test (ρ) - the reason for using this method was, robust for non-normally distributed data
- Threshold considered - 0.8
- They started the prediction of the model from 51 commit to evaluate the evaluation of the performance of the current model\
- They put a sliding window where it will predict for for each 101th commit

- Point to ve noted that they are using balanced accuracy for training / evaluating the model
- The main aim of habing the sliding window approach is that they want to ensure that this model will work for very low number of data points which is the case when a project is in it's initial stage of development
- They used AUC to evaluate the performance as well
- Basically they proved that their just in time classifier can provide a better performance

What all are the influential factors
They used a bootstrap approach to analyse what all are the influential factors of the project
Bootstrap is a general approach to infer the relationship between a sample data and overall population, resampling the sample data with replacement and analyzing the relationship between the resample data and the sample data

1. From the original dataset with n instances we choode a random bootstrap sample of n instances with replacement
2. Build the RF with bootstrap data
3. We collect the influence of each factor in the classifier
4. Repeat for 1000 times

They used the importance function in teh R package to evaluate of the factors - the data OOB data
This followed by Scott-Knott clustering - it is an homogenous algorithm which will partition the results into distinct homogenous groups by comparing their means
Sk heirarchically divides the factors into groups and uses the likelihood ratio test to judge the significance of difference among the group. The SK method generates statically distinct groups of factors ie, each two groups of factors have a p-value < 0.05 in a likelihood ratio test of their influence value

The final result proved that the change measures and the product measures are the most influential factors for log changes
For Hadoop

- Log density
- If statements
- - catch statements
- Log number
- Method declaration
- Average log length
- Average log varaialbes
- SLOC
- Else clause
- Average log level

Result - Code changes in a commit and the current snapshot of the source code are the most influential indicators for a log change in a commit.

Unfortunately this model will fail when the developers change the log statements without changing the code in such cases this model will not help