Software Defect Prediction Based on Classification Rule Mining

BY

SHARADINDU CHAKRABORTY

GOVT. COLLEGE OF ENGINEERING & CERAMIC TECHNOLOGY

DEPARTMENT OF INFORMETION TECHNOLOGY

Author Note

[Include any grant/funding information and a complete correspondence address.]

Abstract

There has been rapid growth of software development. Due to various causes, the software comes with many defects. In Software development process, testing of software is the main phase which reduces the defects of the software. If a developer or a tester can predict the software defects properly then, it reduces the cost, time and effort.In this paper, we show a comparative analysis of software defect based on classification rule of mining

Various metrics in software like Cyclomatic complexity, Lines of Code have been calculated and effectively used for predicting faults. Techniques like statistical methods, data mining, machine learning, and mixed algorithms, which were based on software metrics associated with the software, have also been used to predict software defects. Here, different data mining techniques are discussed for identifying fault prone modules.

Keywords: Data-mining, cyclomatic complexity

Software Defect Prediction Based on Classification Rule Mining

Introduction

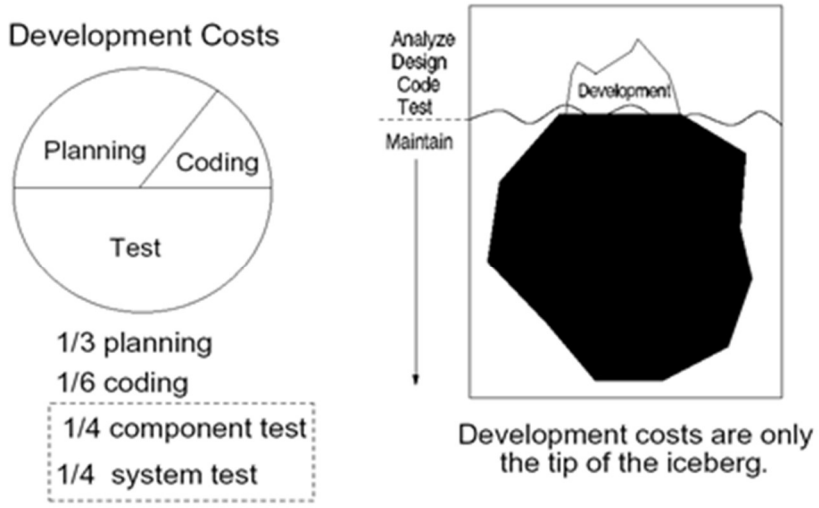
There has been a huge growth in the demand for software quality during recent ages. As a consequence, issues are related to testing, becoming increasingly critical. The ability to measure software defect can be extremely important for minimizing cost and improving the overall effectiveness of the testing process. The major amount of faults in a software system are found in a few of its components. Now days individuals and society increasingly rely on advanced software systems. Because software is intertwined with all aspects of our lives, it is essential to produce reliable and trustworthy systems economically and quickly

Software defect prediction is one of the SQA(Software quality assurance)activities that aims to automatically predict fault-prone software modules using historical software information from an earlier deployment or identical objects, for example source code edit logs and bug reports , before the actual testing process begins. Effective defect prediction could help test managers locate bugs and facilitate the allocation of limited SQA resources optimally and economically; thus, it has become an extremely important research topic . Commonly, a prediction model is used to predict the defective software modules in one of the three categories: binary class classification of defects number of defects/defect density prediction, and severity of defect prediction .

1.Research background and significance [1]

Defects are basic properties of a system. They come from design or manufacture, or external environment. The systems which run well at the moment may also have defects not trigged now or not so important at the moment. Software defects are programming errors which cause the different behavior compared with expectation. Most of the defects are from source code or deign, some of them are from the wrong code generating from compilers.

For software developers and users, software defects are a headache problem. Software defects not only reduce software quality, increase costing but also suspend the development schedule. No matter in software engineering or in research area, to control the number of defects is an important aspect.Finding and fixing the bugs cost lots of money. The data of US department of defense shows that in 2006, American spent around 780 billion dollars for software bugs related problem. And it also shows that there is around 42% money spend on software bugs in IT products[1]. Until now there is no similar report in the world, but there is an estimation that the cost for software bugs account for 30% of the whole cost. So it is very worktable to research the software defects. Figure 1.1 shows the cost of each phase of software



developments.

Usually during the development process, software development team can only know about the software defects by software testing results. But it is expensive to testing the whole process completely, at the same time most of software testing happens at the later stage of software development. If during testing process we find the number of software defects/defects rates are higher that the demanding level, software development team falls into a dilemma, either to postpone software release to fix these defects or release the software products containing defects.

Software defects predicting is proposed to solve this kind of problem. The assumption is that the quantity of software is related with the complexity of software modules. More complex modules normally contain more bugs. We can use the historical data, the already discovered software defects and other metric data which can represent software product, software development process to predict the software defects quantity and decide whether the module is defects-prone. In this case, software development team can allocate resources to high risk software module to improve the reliability and quality. By adopting software defects prediction, software development team can forecast the costing in early stage of software development at a relatively lower cost. This will help software development team to optimize allocation of project resources, also help to improve the quality of software.

Research status.

The widely used software defects prediction techniques are regression and classification techniques. Regression technique is aimed to predict the quantity and density of software defects. Classification technique is aimed to determine whether software module (which can be a package, code file, or the like) has a higher defect risks or not. Classification usually learns the data in earlier versions of the same project or similar data of other projects to establish a classification model. The model will be used to forecast the software defects in the projects which need to be predicted. By doing this, we can make more reasonable allocation for resources and time, to improve the efficiency and quality of software development.

Akiyama is the first person who suggests the relationship between software defects and the line of code. Barry Boehum[15] brings up the COCOMO (1970) model to control the software quality. This model considers several parameters during software development to estimate the

defects rates when software is released. The model raised up to manifest prediction based on collection of previous versions of software, changing history, and previous defects of software

Shang Liu et al bring up the software defects prediction method based on machine learning. This method possesses a higher accuracy rate and is more stable.

Norman Fenton(Risk Information Management Research Group) take advantage of Bayesian networks in their model which is different software development phase.

Main work.

Most of the current software defects prediction techniques have something to do with software metrics and classification algorithm. By adopting the popular software development repository techniques.

WE get a scrap dataset from web, and then analyze the characters of these datasets, use different models based on different characters, choose different but suitable software metrics, to build different prediction models. And then I use these models in future version of software and predict software defects on different granularity, and then compare the performance of different models.

2.Background & Literature Survey

The purpose of this chapter is to establish a theoretical background for the project. The focus of this study will be on software defects and effort spent correcting software defects. However, it is necessary to explore research areas which influence or touches software defects. Poor software quality may be manifested through severe software defects, or software maintenance may be costly due to many defects requiring extensive effort to correct. Last, we explore relevant research methods for this study.

2.1 Data Mining for software Engineering

To improve the software productivity and quality, software engineers are applying data mining algorithms to various SE tasks. Many algorithms can help engineers

figure out how to invoke API methods provided by a complex library or framework with insufficient documentation. In terms of maintenance, such type of data mining algorithms can assist in determining what code locations must be changed when another code location is changed. Software engineers can also use data mining algorithms to hunt for potential bugs that can cause future in-field failures as well as identify buggy lines of code (LOC) responsible for already-known failures.

Table 2.1: Example software engineering data, Mining algorithm, SE tasks .

SE data	Mining Algo	SE Tasks
Sequences: execution/static traces, co-changes	Frequent item set or dataset /sequence/partial –order mining. Sequence matching/clustering/classification	Programming, maintenance, bug-detection, debugging
Graphs: dynamic/static graphs, program dependent graph	Frequent sub graph mining, graph matching/clustering/classification	Bug dectection,debugging
Text: bug reports, emails, code ,comments, doccumentation	Textmatching/clustering/classifcetion	Maintenance, debugging

2.2 Software defect predictor

A defect predictor is a tool or method that guides testing activities and software development lifecycle. According to Brooks, half the cost of software development is in unit and systems testing .research also conform that testing phase requires approximately 50% or more of the whole project schedule. Therefore, the main challenge is the testing phase and practitioners seek predictors that indicate where the defects might exist before they start testing. This allows them to efficiently allocate their resources.

2.3 Defect Prediction as a Classification Problem

Software defect prediction can be viewed as a supervised binary classification problem. Software modules are represented with software metrics, and are labelled as either defective or non-defective. To learn defect predictors, data tables of historical examples are formed where one column has a Boolean value for "defects detected" (i.e. dependent variable) and the other columns describe software Characteristics in terms of software metrics (i.e. independent variables).

2.4 Binary classification

In machine learning and statistics, classi_cation is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.

Binary or binomial classification is the task of classifying the members of a given set of objects into two groups on the basis of whether they have some property or

not.

Data Classification is two step process. In the 1st step, a classifier is built describing a predetermined set of data classes or concepts. This is the learning step(or training phase), where a classification algorithm is builds the classi_er by analyzing or "learning form" a training set made up of database tuples and there associated class labels.

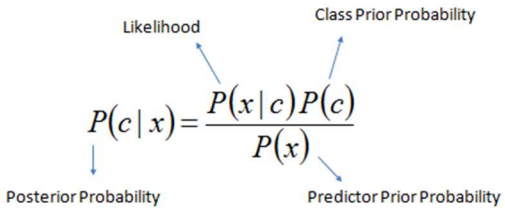
In the 2nd step the model is used for classification. Therefore, a test set is used, make up of test tupples and there associated class labels.

A classification rule takes the form X=> C, where X is a set of data items, and C is the class (label) and a predetermined target. With such a rule, a transaction or data record t in a given database could be classified into class C if t contains X.

2.5 Binary Classification Algorithms

2.5.1 Bayesian Classification

The Naive Bayesian classifier is based on Bayes theorem with independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classi_er often does surprisingly well and is widely used because it often outperforms more sophisticated classi_cation methods.



Algorithm:

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

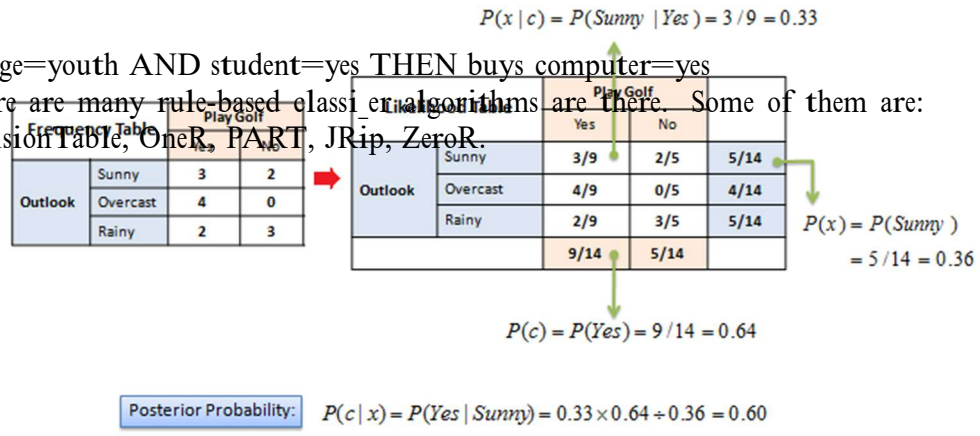
Bayes theorem provides a way of calculating the posterior probability, P(cjx), from P(c), P(x), and P(xjc). Naive Bayes classi_er assume that the eect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.

- P(cjx) is the posterior probability of class (target) given predictor (attribute).
- P(c) is the prior probability of class.
- P(xjc) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor.

2.5.2 Rule-Based Classification

Rules are a good way of representing information or bits of knowledge. A rule-based classi_er uses a set of IF-THEN rules for classification. An IF-THEN rule is an expression of the form-

expression of the form-



IF age=youth AND student=yes THEN buys computer=yes

There are many rule-based classi_er algorithms are there. Some of them are: DecisionTable, OneR, PART, JRip, ZeroR.

2.5.3 Logistic Regression

In statistics, logistic regression or logit regression is a type of regression analysis used for predicting the outcome of a categorical dependent variable (a dependent variable that can take on a limited number of values, whose magnitudes are not meaningful but whose ordering of magnitudes may or may not be meaningful) based on one or more predictor variables.

An explanation of logistic regression begins with an explanation of the logistic function, which always takes on values between zero and one:

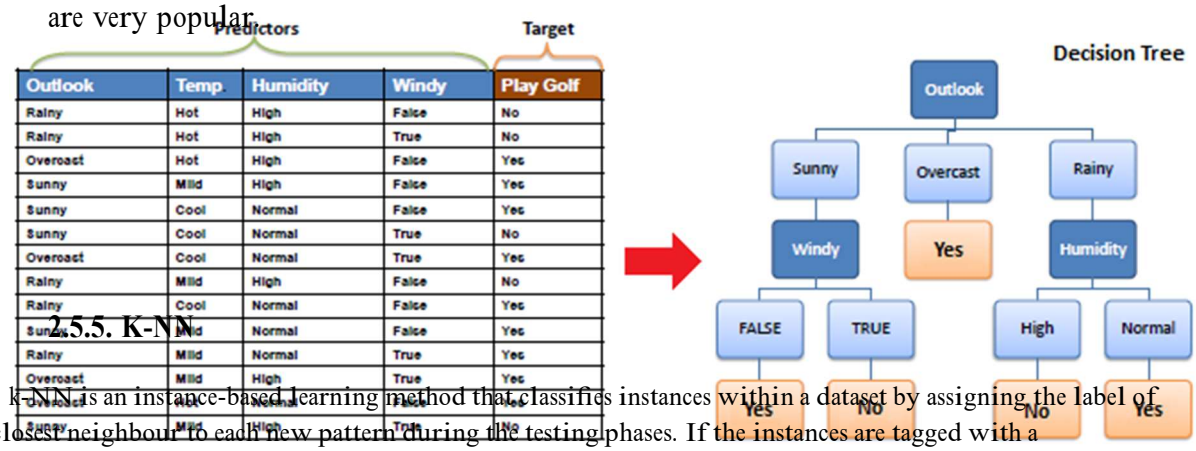
$$f(t)=\frac{}{1+e^{t}}$$

2.5.4 Decision Tree classification

Decision tree induction is the learning of decision trees from class-labled training

tuples. A desision tree is a flow chart like tree structure, where each internal nodes(non-leaf node) denotes a test on an attribute , each brunch represents an outcome of the test, each leaf node(or internal node) holds a class label. The topmostnode in a tree is the root node.

There are many algorithms developed using decision tree for classi_cation with some dierences. Some of them like BFTree, C4.8/J48, J48Graft,and SimpleCart are very popular.



2.5.5. K-NN

k-NN is an instance-based learning method that classifies instances within a dataset by assigning the label of the closest neighbour to each new pattern during the testing phases. If the instances are tagged with a classification label, then the majority class of the closest k neighbours is assigned to the unclassified instance. Although the power of k-NN has been proven in a number of real domains, they have large storage requirements and their performance is sensitive to the choice of the k.

2.6 Related Works

2.6.1 Regression via classification

In 2006, Bibi, Tsoumakas, Stamelos, Vlahavas, apply a machine learning approach to the problem of estimating the number of defects called Regression via Classi_cation (RvC).The whole process of Regression via Classi_cation (RvC) comprises two important stages:

a) The discretization of the numeric target variable in order to learn a classi_cation

b) the reverse process of transforming the class output of the model into a numeric
prediction.

2.6.2 Static Code Attribute

Menzies, Greenwald, and Frank (MGF) published a study in this journal in 2007 in which they compared the performance of two machine learning techniques (Rule Induction and Naive Bayes) to predict software components containing defects. To do this, they used the NASA MDP repository, which, at the time of their research, contained 10 separate data sets.

2.6.3 ANN

In 2007, Iker Gondra used a machine learning methods for defect prediction. He used Arti_cial neural network as a machine learner.

2.6.4 Embedded software defect prediction

In 2007, Oral and Bener used Multilayer Perception (MLP), NB, VFI(Voting Feature Intervals) for Embedded software defect prediction. there they used only 7 data sets for evaluation.

2.6.5 Association rule classification

In 2011 Baojun, Karel used classification based association rule named CBA2 for software defect prediction.In these research they used assocition rule for classification. and they compare with other classification rules such as C4.5 and Ripper.

2.6.6 Defect-proneness Prediction framework

In 2011, Song, Jia, Ying, and Liu propased a general framework for software defect-pronness prediction. in this research they use M*N cross validation with the dataset(NASA, Softlab Dataset) for learning process. and they used 3 classification algorithms(Naive baysed, OneR, J48). and they copared with MGF [5] framework. In 2010 a research has benn done by Chen, Sen, Du Ge, [8] on software defect prediction using datamining. In this reseach they used probabilistic Relational model and Baysean Network.

TABLE 1. A taxonomy of the related works along with the proposed method

Approach	PAPER	Method (Mining, Learning, Optimization)	Methodology	Preprocessing step	Class imbalance problem	Supervised Semi-supervised	Datasets
The Proposed Method	-	**Deep Learning**	**DBN, SSAE**	**Normalization**	**Not Considered**	-	**CM1, KC1, KC2, KC3, KC4, PC1, PC2, PC3, PC4, PC5, JM1, MW1, MC1, MC2**
WPDP	[17]	Minining & Learning	OneR, J48, and naïve Bayes	removing the module identifier attribute	Not Considered	Supervised	KC3,CM1,KC4,MW1,PC1, PC2,PC3,PC4
	[18]	Minining & Learning	Extended transfer component analysis +logistic regression	min-max and z-score normalization methods	Not Considered	Supervised	ReLink,AEEEM
	[19]	Minining & Learning	WC and CC-data models	NN-filtering	Not Considered	Supervised	PC1,KC1,KC2,CM1,KC3, AR3,AR4,AR5 ,MW1,MC2
CPDP	[20]	Learning	Transfer Naive Bayes	NN-filtering	Not Considered	Supervised	kc3,Pc1,kc1,kc2, cm1, ar3,ar4,ar5, mw1,mc2
	[21]	Mining & Learning	context-aware rank transformations	Clean Data (Understand)	Not Considered	Semi-supervised	Generate a dataset
	[22]	Learning	ensemble approaches	Minimizing collinearity	Considered	Supervised	Bugzilla ,Columba ,Gimp ,Eclipse JDT ,Maven-2 ,Mozilla
HDP	[23]	Learning	canonical correlation analysis (CCA) nearest neighbor (NN)	z-score normalization	Considered	Supervised	NASA,SOFTLAB,ReLink AEEEM
	[24]	Learning	Logistic regression	Feature selection (gain ratio, chi-square, relief-F)	Considered	Supervised	AEEEM,ReLink,MORPH NASA,SOFTLAB

Proposed Scheme

3.1 Overview Of the Framework

In General, before building defect prediction model and using them for prediction purposes, we first need to decide which learning scheme or learning algorithm should be used to construct the model. Thus, the predictive performance of the learning scheme should be determined, especially for future data. However, this step is often neglected and so the resultant prediction model may not be Reliable. As a consequence, we use a software defect prediction framework that provides guidance to address these potential shortcomings.

The framework consists of two components:

1) scheme evaluation and

2) defect prediction.

3.2CHALLENGES OF THE WORK

Some of the drawbacks of the existing classifiers are:

 It is very difficult to fully understand the connection between input and output sometimes. It is hard to accept the complex nonlinear algorithms with poor credibility.

 Rather than generating entire rule set, rule-based/tree-based classifiers can generate only partially biased ruleset .

 According to the characteristics of the software, defect prediction model from other project data cannot be adjusted. Once the prediction model is established, the result cannot be modified or if there are variance in characteristics of the training data, the test efficiency will be deteriorated .

 If the use of unnecessary code inspection increases, there will be increase in risk over budget and time.

3.3 Scheme Evaluation

The scheme evaluation is a fundamental part of the software defect prediction framework. At this stage, different learning schemes are evaluated by building and evaluating learners with them. The first problem of scheme evaluation is how to divide historical data into training and test data. As mentioned above, the test data should be independent of the learner construction. This is a necessary precondition to evaluate the performance of a learner for new data. Cross-validationis usually used to estimate how accurately a predictive model will perform in practice. One round of cross-validation involves partitioning a data set into complementary subsets, performing the analysis on one subset, and validating the analysis on the other

subset. To reduce variability, multiple rounds of cross-validation are performed using dierent partitions, and the validation results are averaged over the rounds.
In our framework, an percentage split used for estimating the performance of each predictive model, that is, each data set is first divided into 2 parts, and after that a predictor is learned on 70% intances, and then tested on the remaining 30%. To overcome any ordering eect and to achieve reliable statistics, each holdout experiment is also repeated M times and in each repetition the data sets are randomized. So overall, M*N(N=Data sets) models are built in all during the period of evaluation; thus M*N results are obtained on each data set about the performance of the each learning scheme.
After the training-test splitting is done each round, both the training data and learning scheme(s) are used to build a learner. A learning scheme consists of a data preprocessing method, an attribute selection method, and a learning algorithm.

Evaluation of the proposed framework is comprised of:
1. A data preprocessor
· The training data are preprocessed, such as removing outliers, handling missing values, and discretizing or transforming numeric attributes.
· Here Preprocessor used-
scipy preprocessor

2. An attribute selector

Here we have considered all the attributes pvovided by the kaggel (Creators: NASA, then the NASA Metrics Data Program)

3. Learning Algorithms
   3.1 Naïve-Bayse Simple from bayse classification
   3.2 Logistic classification
   3.3 Tree based classification
   3.4. Nearest neighbor classificetion

3.4 Scheme Evaluation Algorithm

Data: Historical Data Set
Result: The mean performance values
1 M=n :No of Data Set
2 i=1;
3 while i<=M do
4 Read Historical Data Set D[i];
5 Split Data set Intances using % split;
6 Train[i]=70% of D; % Training Data;
7 Learning(Train[i],scheme);
8 Test Data=D[i]-Train[i];% Test Data;
9Result=TestClassi_er(Test[i],Learner);
10 end

Algorithm 1: Scheme Evaluation

3.5 Defect prediction

The defect prediction part of our framework is straightforward; it consists of predictor construction and defect prediction. During the period of the predictor construction

1. A learning scheme is chosen according to the Performance Report.
2. A predictor is built with the selected learning scheme and the whole historical data. While evaluating a learning scheme, a learner is built with the training data and tested on the test data. Its final performance is the mean over all rounds. This reveals that the evaluation indeed covers all the data. Therefore, as we use all of the historical data to build the predictor, it is expected that the constructed predictor has stronger generalization ability.
3. After the predictor is built, new data are preprocessed in same way as historical data, then the constructed predictor can be used to predict software defect with

3.5 Deference between Our Framework and others

So, to summarize, the main difference between our framework and that of others in

1) We choose the entire learning scheme, not just one out of the learning algorithm,
attribute selector, or data preprocessor;

2) we use the appropriate data to evaluate the performance of a scheme.

3)We choose percentage split for training data set(70%) and test dataset(30%).

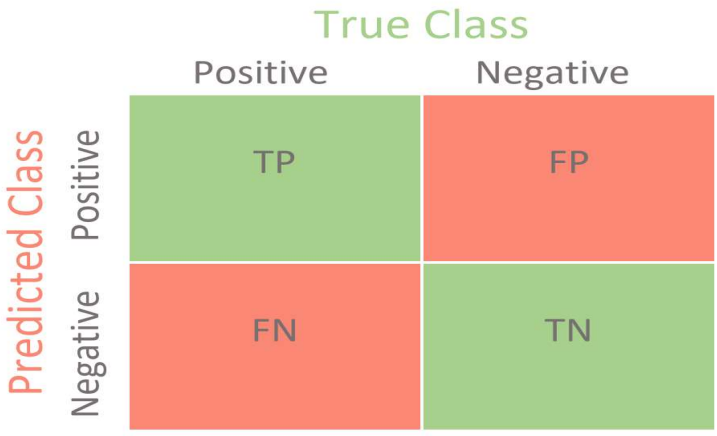
3.7 Data Set

We used a scrap data from kaggle. There are 22 columns & 10885 rows of data in total.

```
In [36]: df1.head()
Out[36]:
     loc  v(g)  ev(g)  iv(g)  ...  total_Op  total_Opnd  branchCount  defects
0    1.1   1.4    1.4    1.4  ...       1.2         1.2          1.4    False
1    1.0   1.0    1.0    1.0  ...         1           1            1     True
2   72.0   7.0    1.0    6.0  ...       112          86           13     True
3  190.0   3.0    1.0    3.0  ...       329         271            5     True
4   37.0   4.0    1.0    4.0  ...        76          50            7     True

[5 rows x 22 columns]

In [37]:
```

3.8 Performance Measurement

The Performance measured according to the Confusion matix given in table:3.3, which is used by many researchers e. Table 3.3 illustartes a confusion matrix for a two class problem having positive and negative class values.



Software defect predictor performance of the proposed scheme based on Accurcy, Sensitivity, Specificity, Balance, ROC Area defined as

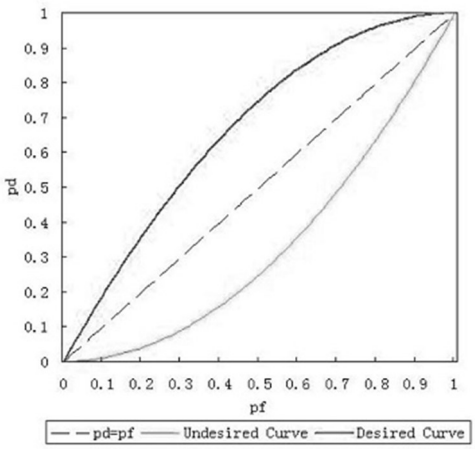
Accuracy=TP+TN/TP+FP+TN+FN
=The percentage of prediction that are correct.

pd=True Positive Rate(tpr)=Sensitivity = TP/TP+FN
=The percentage of positive labled instances that predicted as positive

Specif icity =TN/FP+TN
=The percentage of positive labled instances that predicted as negative.

pf=False Positive Rate(fpr)=1-specificity
=The percentage of Negative labled instances that predicted as negative

The receiver operating characteristic(ROC) [15] [28], curve is often used to evaluate the performance of binary predictors. A typical ROC curve is shown in Fig.The y-axis shows probability of detection (pd) and the x-axis shows probability of false alarms (pf).
Formal definitions for pd and pf are given above. Obviously, higher pds and lower pfs are desired. The point (pf=0, pd=1) is the ideal position where we recognize all defective modules and never make mistakes.
The Area Under ROC Curve (AUC) is often calculated to compare dierent ROC curves. Higher AUC values indicate the classi_er is, on average, more to the upper left region of the graph. AUC represents the most informative and commonly used, thus it is used as another performance measure in this paper.



Result Discussion

This section provides simulation results of some of the Classification algorithm

According to best accuracy value we choose 4 classification algorithm among many classification algorithms. All the evaluted values are collected and compare with dierent performance measurement parameter.

Accurecy scores:-

Logistic regression:-86.23578413%

Naïve bayes:- 82.36130867709 %

Decision tree:-85.02356946 %

K-NN:-84.88625644%

Conclusion

In our project work we have attempted to solve the Software defect prediction problem through different Data mining (Classification) algorithms.
In our research NB,K-NN ,Decision tree and Logistic algorithm gives the overall better performance for defect prediction.
From these results, we see that a data preprocessor/attribute selector can play different roles with different learning algorithms for different data sets and that no learning scheme dominates, i.e., always outperforms the others for all data sets. This means we should choose different learning schemes for different data sets, and consequently, the evaluation and decision process is important.

In order to improve the e_ciency and quality of software development, we can

make use of the advantage of data mining to analysis and predict large number of defect data collected in the software development. This small project reviewed the current state of software defect management, software defect prediction models and datamining technology briefly. Then proposed an ideal software defect management and prediction system, can be used and analyzed on several software defect prediction dataset using methods based on data mining techniques and speci_c models(NB, Logistic, PART, Decision tree,K-NN,)

5.2 Scope for Further Research

· Clustering based classification can be used.
· Future studies could focus on comparing more classification methods and

improving association rule based classifiation methods

· Furthermore, the reduction of rules for association rule based classification methods can be considered.

Codes:

```
import numpy as np
import pandas as pd
import matplotlib as mlt

import matplotlib.pyplot as plt # ploting , visualization
```

```
import seaborn as sns # ploting
from sklearn import model_selection #scikit learn
from sklearn import linear_model
from sklearn import metrics
from sklearn import preprocessing
from sklearn import utils
import warnings
warnings.filterwarnings("ignore")
from sklearn import ensemble
from sklearn import naive_bayes
from sklearn import feature_selection
from sklearn.feature_extraction import DictVectorizer
import os
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_squared_error,roc_auc_score
from sklearn.linear_model import LinearRegression
from sklearn import neighbors
from sklearn import tree
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

import chart_studio.plotly  as py
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
import plotly.graph_objs as go
import os



df1=pd.read_csv("C:/Users/sonu/Desktop/Project2/jm1.csv")

df1.info()

df1.head()

df1.columns
df1.dtypes
df1.shape

cleancolumn = []
for i in range(len(df1.columns)):
    cleancolumn.append(df1.columns[i].replace(' ','').lower())
df1.columns = cleancolumn

df1.isnull().sum()

df1.describe()

##Distribution graphs (histogram/bar graph) of column data
def plotPerColumnDistribution(df1, nGraphShown, nGraphPerRow):
```

```
        nunique = df1.nunique()
        df1 = df1[[col for col in df1 if nunique[col] > 1 and nunique[col] < 50]] # For
displaying purposes, pick columns that have between 1 and 50 unique values
        nRow, nCol = df1.shape
        columnNames = list(df1)
        nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
        plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80,
facecolor = 'w', edgecolor = 'k')
        for i in range(min(nCol, nGraphShown)):
            plt.subplot(nGraphRow, nGraphPerRow, i + 1)
            columnDf = df1.iloc[:, i]
            if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
                valueCounts = columnDf.value_counts()
                valueCounts.plot.bar()
            else:
                columnDf.hist()
            plt.ylabel('counts')
            plt.xticks(rotation = 90)
            plt.title(f'{columnNames[i]} (column {i})')
        plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
        plt.show()



    #Correlation matrix
    def plotCorrelationMatrix(df1, graphWidth):
        df1 = df1.dropna('columns') # drop columns with NaN
        df1 = df1[[col for col in df1 if df1[col].nunique() > 1]] # keep columns where there
are more than 1 unique values
        if df1.shape[1] < 2:
            print(f'No correlation plots shown: The number of non-NaN or constant columns
({df1.shape[1]}) is less than 2')
            return
        corr = df1.corr()
        plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w',
edgecolor='k')
        corrMat = plt.matshow(corr, fignum = 1)
        plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
        plt.yticks(range(len(corr.columns)), corr.columns)
        plt.gca().xaxis.tick_bottom()
        plt.colorbar(corrMat)
        plt.title(f'Correlation Matrix for dataset', fontsize=15)
        plt.show()



    # Scatter and density plots
    def plotScatterMatrix(df1, plotSize, textSize):
        df1 = df1.select_dtypes(include =[np.number]) # keep only numerical columns
        # Remove rows and columns that would lead to df1 being singular
```

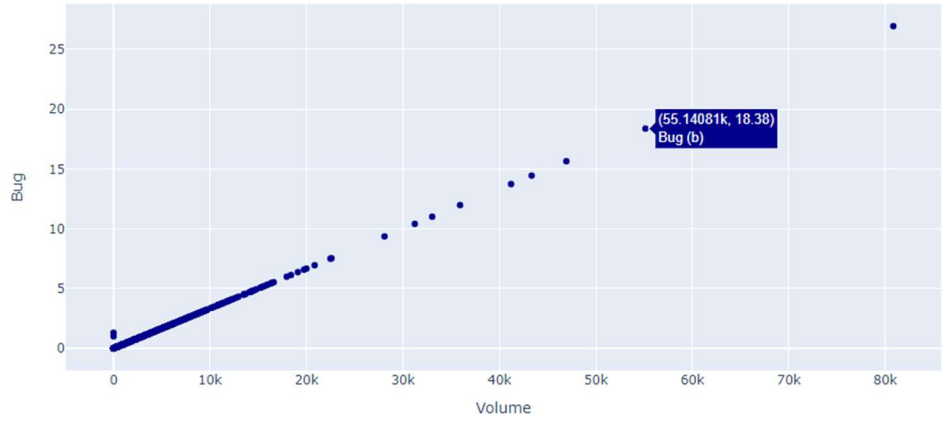
```
            df1 = df1.dropna('columns')
            df1 = df1[[col for col in df1 if df1[col].nunique() > 1]] # keep columns where there
are more than 1 unique values
            columnNames = list(df1)
            if len(columnNames) > 10: # reduce the number of columns for matrix inversion of
kernel density plots
                columnNames = columnNames[:10]
            df1 = df1[columnNames]
            ax = pd.plotting.scatter_matrix(df1, alpha=0.75, figsize=[plotSize, plotSize],
diagonal='kde')
            corrs = df1.corr().values
            for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
                ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes
fraction', ha='center', va='center', size=textSize)
            plt.suptitle('Scatter and Density Plot')
            plt.show()

        from mpl_toolkits.mplot3d import Axes3D

        plotPerColumnDistribution(df1, 10, 5)

        plotCorrelationMatrix(df1, 7)

        plotScatterMatrix(df1, 20, 10)

        defects_true_false = df1.groupby('defects')['b'].apply(lambda x: x.count()) #defect
rates (true/false)
        print('False : ' , defects_true_false[0])
        print('True : ' , defects_true_false[1])

        trace = go.Histogram(
            x = df1.defects,
            opacity = 0.75,
            name = "Defects",
            marker = dict(color = 'green'))

        hist_data = [trace]
        hist_layout = go.Layout(barmode='overlay',
                        title = 'Defects',
                        xaxis = dict(title = 'True - False'),
                        yaxis = dict(title = 'Frequency'),
        )
        fig = go.Figure(data = hist_data, layout = hist_layout)
        iplot(fig)


        f,ax = plt.subplots(figsize = (15, 15))
        sns.heatmap(df1.corr(), annot = True, linewidths = .5, fmt = '.2f')
        plt.show()

        trace = go.Scatter(
            x = df1.v,
```

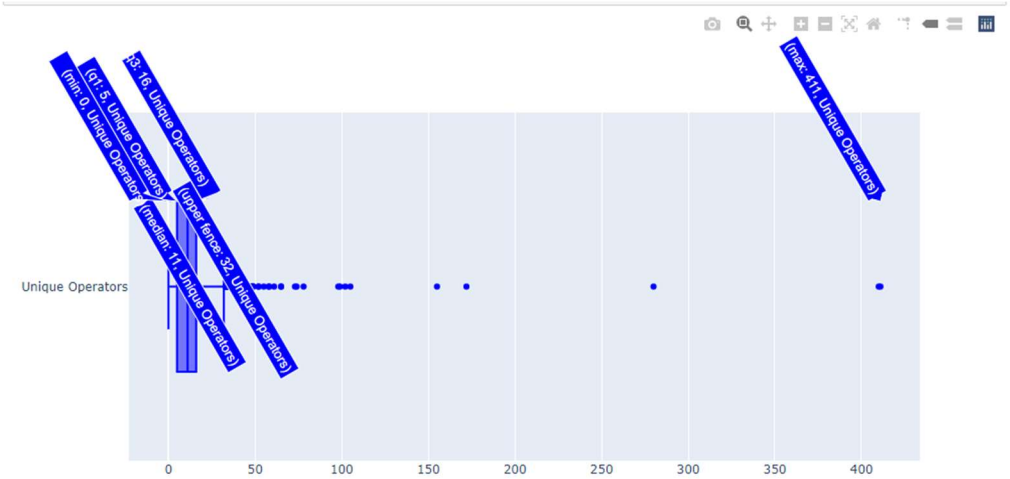
```
            y = df1.b,
            mode = "markers",
            name = "Volume - Bug",
            marker = dict(color = 'darkblue'),
            text = "Bug (b)")

        scatter_data = [trace]
        scatter_layout = dict(title = 'Volume - Bug',
                    xaxis = dict(title = 'Volume', ticklen = 5),
                    yaxis = dict(title = 'Bug' , ticklen = 5),
                   )
        fig = dict(data = scatter_data, layout = scatter_layout)
        iplot(fig)

        trace1 = go.Box(
            x = df1.uniq_op,
            name = 'Unique Operators',
            marker = dict(color = 'blue')
            )
        box_data = [trace1]
        iplot(box_data)

        def evaluation_control(df1):
            evaluation = (df1.n < 300) & (df1.v < 1000 ) & (df1.d < 50) & (df1.e < 500000) &
(df1.t < 5000)
            df1['defects'] = pd.DataFrame(evaluation)
            df1['defects'] = ['true' if evaluation == True else 'false' for evaluation in df1.defects]

        evaluation_control(df1)
        df1

        X =df1.drop(["defects"],axis=1)

        X.head()

        y = df1[["defects"]]
        y.head()

        scaler = MinMaxScaler()
        scl_X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

        x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)
        x_train.shape  , y_train.shape

        from sklearn.svm import SVC
        from sklearn.metrics import accuracy_score

        svc_model.fit(x_train,y_train)
        svc_pred = svc_model.predict(x_test)
        svc_score = accuracy_score(svc_pred,y_test)*100
        svc_score
```

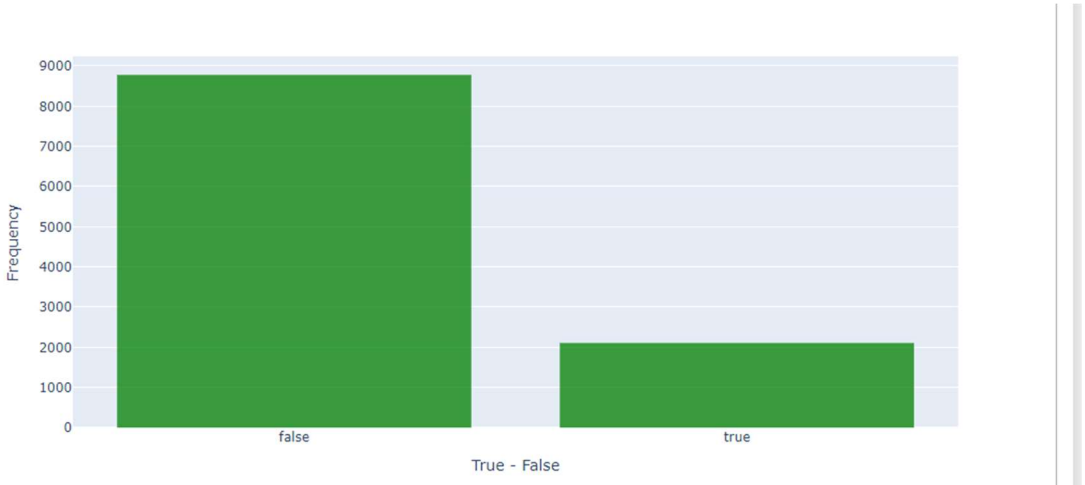
```
from sklearn.naive_bayes import GaussianNB
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(x_train,y_train)
naive_bayes_pred = naive_bayes_model.predict(x_test)
naive_bayes_score = accuracy_score(naive_bayes_pred,y_test)*100
naive_bayes_score


from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import KFold

k_fold = KFold(len(df), n_folds=10, shuffle=True, random_state=0)

svc_cv_model = SVC()
svc_cv_score = cross_val_score(svc_cv_model,X,y,cv=k_fold,scoring =
'accuracy')*100
svc_cv_score
svc_cv_score.mean()


naive_bayes_cv_model = GaussianNB()
naive_bayes_cv_score =

cross_val_score(naive_bayes_cv_model,X,y,cv=k_fold,scoring = 'accuracy')*100

naive_bayes_cv_score

naive_bayes_cv_score

naive_bayes_cv_model.fit(X,y)

naive_bayes_cv_pred = naive_bayes_cv_model.predict(X)

naive_bayes_cv_score = accuracy_score(naive_bayes_cv_pred,y)*100

naive_bayes_cv_score



from sklearn.tree import DecisionTreeClassifier

tree_cv_score = cross_val_score(tree_model,X,y,cv=k_fold,scoring = 'accuracy')*100

tree_cv_score

tree_cv_score.mean()

from sklearn.linear_model import LogisticRegression

logistic_model = LogisticRegression()
```

```
logistic_cv_score = cross_val_score(logistic_model,X,y,cv=k_fold,scoring =
'accuracy')*100


logistic_cv_score
logistic_cv_score.mean()


from sklearn.neighbors import KNeighborsClassifier
k_range = range(1,26)
scores = []
for k in k_range :
   KNN = KNeighborsClassifier(n_neighbors=k)
   KNN.fit(x_train,y_train)
   pred = KNN.predict(x_test)
   scores.append(accuracy_score(pred,y_test)*100)


print(pd.DataFrame(scores))


plt.plot(k_range,scores)
plt.xlabel("K for KNN")
plt.ylabel("Testing scores")
plt.show()


k_range = range(1,26)
scores = []
for k in k_range :
   KNN = KNeighborsClassifier(n_neighbors=k)
```

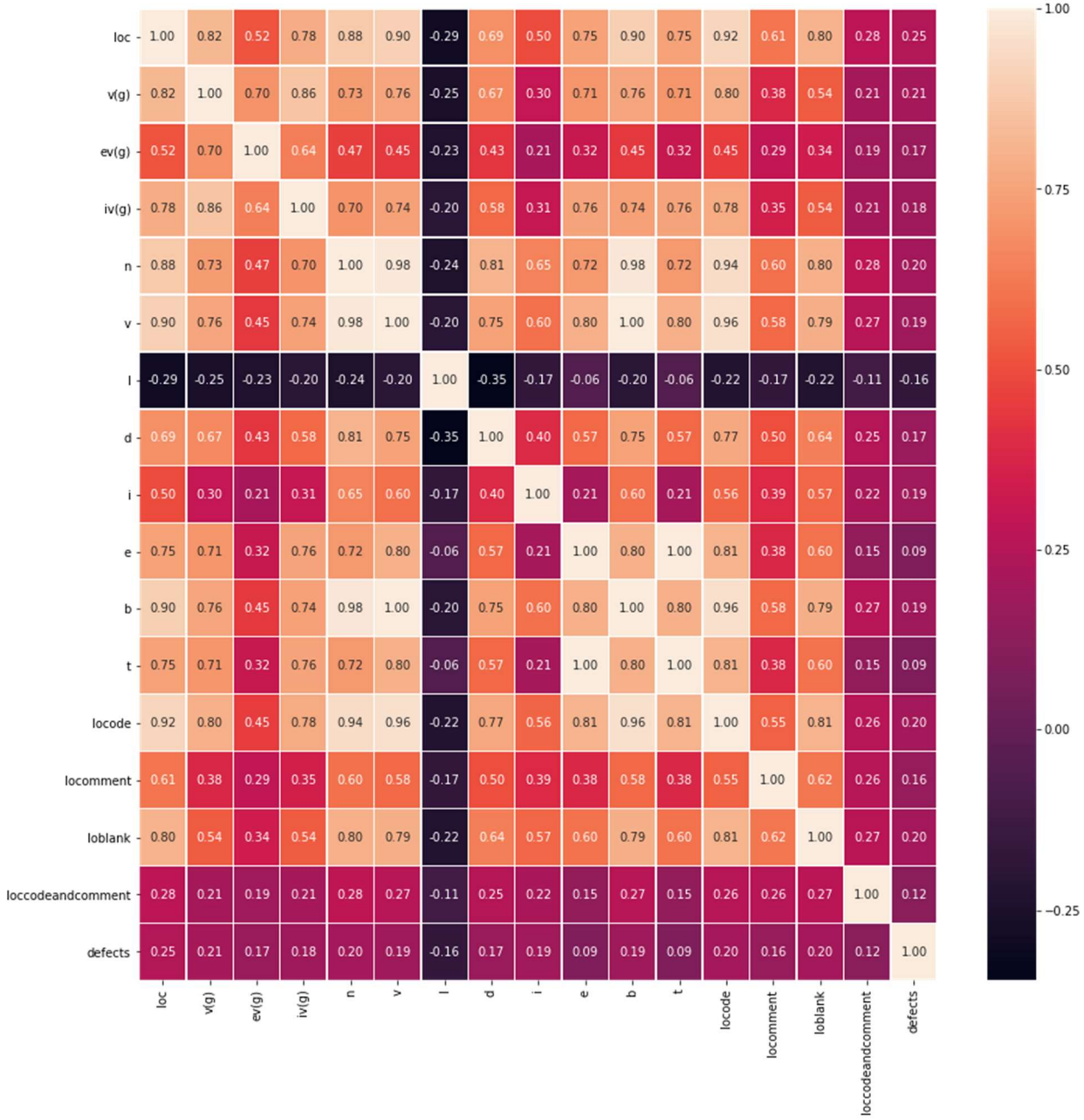
```
    KNN_cv_score = cross_val_score(KNN,X,y,cv=k_fold,scoring = 'accuracy')*100
    cv_score = scores.append(KNN_cv_score)

print(pd.DataFrame(scores))

KNN_cv_score.mean()

plt.scatter(X_train, y_train, color = 'red')
modelin_tahmin_ettigi_y = model.predict(X_train)
plt.plot(X_train, modelin_tahmin_ettigi_y, color = 'black')
plt.title('Line of Code - Bug', size = 15)
plt.xlabel('Line of Code')
plt.ylabel('Bug')
plt.show()

from sklearn import metrics
print('Mean Squared Error (MSE):', metrics.mean_squared_error(y_test, y_pred))
print('RootMeanSquaredError(RMSE):',np.sqrt(metrics.mean_squared_error(y_test,
y_pred)))
```

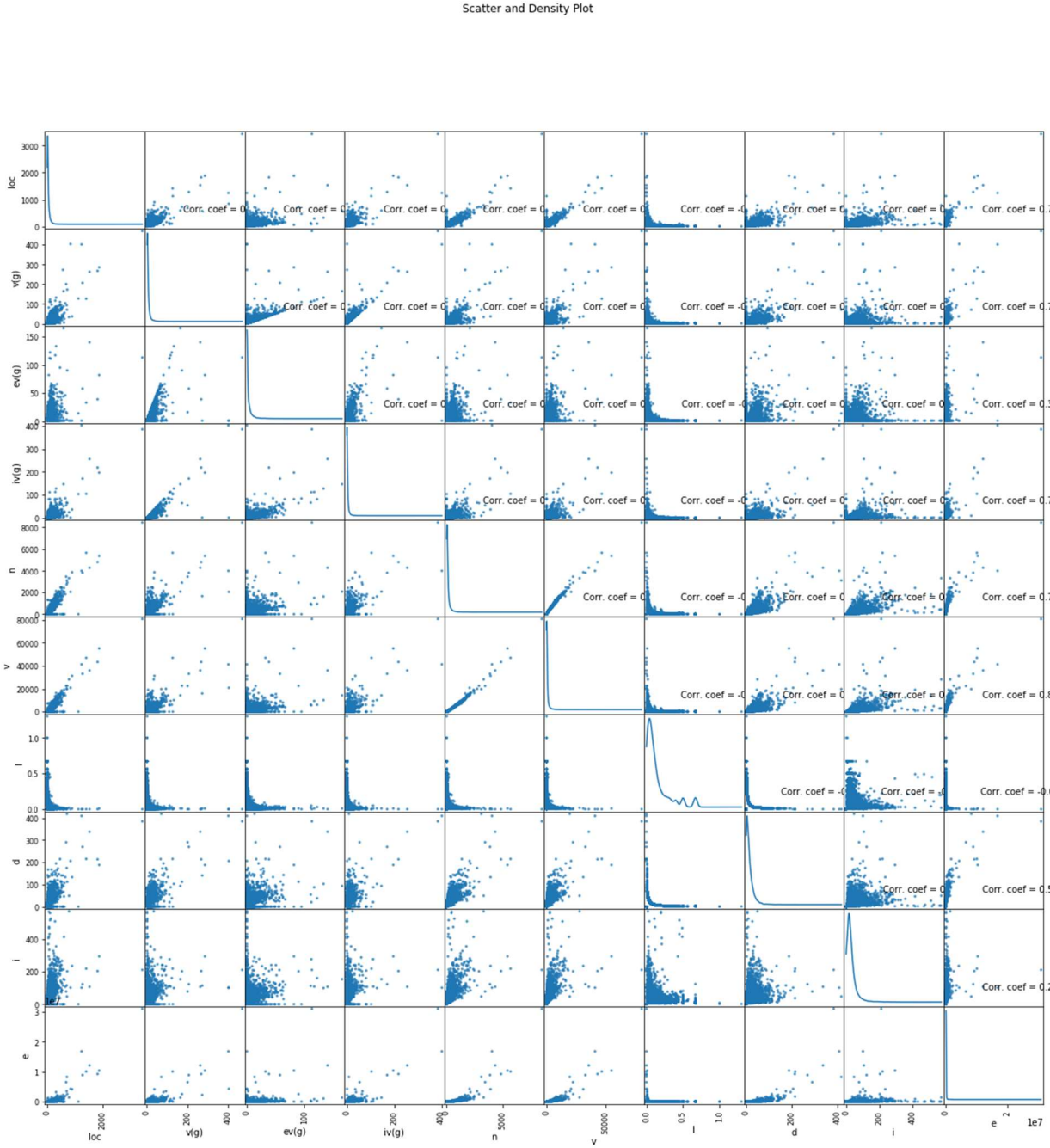







```
In [116]: df1.corr()
```


loc 1.00 0.82 0.52 0.78 0.88 0.90 -0.29 0.69 0.50 0.75 0.90 0.75 0.92 0.61 0.80 0.28 0.25
v(g) 0.82 1.00 0.70 0.86 0.73 0.76 -0.25 0.67 0.30 0.71 0.76 0.71 0.80 0.38 0.54 0.21 0.21
ev(g) 0.52 0.70 1.00 0.64 0.47 0.45 -0.23 0.43 0.21 0.32 0.45 0.32 0.45 0.29 0.34 0.19 0.17
iv(g) 0.78 0.86 0.64 1.00 0.70 0.74 -0.20 0.58 0.31 0.76 0.74 0.76 0.78 0.35 0.54 0.21 0.18
n 0.88 0.73 0.47 0.70 1.00 0.98 -0.24 0.81 0.65 0.72 0.98 0.72 0.94 0.60 0.80 0.28 0.20
v 0.90 0.76 0.45 0.74 0.98 1.00 -0.20 0.75 0.60 0.80 1.00 0.80 0.96 0.58 0.79 0.27 0.19
l -0.29 -0.25 -0.23 -0.20 -0.24 -0.20 1.00 -0.35 -0.17 -0.06 -0.20 -0.06 -0.22 -0.17 -0.22 -0.11 -0.16
d 0.69 0.67 0.43 0.58 0.81 0.75 -0.35 1.00 0.40 0.57 0.75 0.57 0.77 0.50 0.64 0.25 0.17
i 0.50 0.30 0.21 0.31 0.65 0.60 -0.17 0.40 1.00 0.21 0.60 0.21 0.56 0.39 0.57 0.22 0.19
e 0.75 0.71 0.32 0.76 0.72 0.80 -0.06 0.57 0.21 1.00 0.80 1.00 0.81 0.38 0.60 0.15 0.09
b 0.90 0.76 0.45 0.74 0.98 1.00 -0.20 0.75 0.60 0.80 1.00 0.80 0.96 0.58 0.79 0.27 0.19
t 0.75 0.71 0.32 0.76 0.72 0.80 -0.06 0.57 0.21 1.00 0.80 1.00 0.81 0.38 0.60 0.15 0.09
locode 0.92 0.80 0.45 0.78 0.94 0.96 -0.22 0.77 0.56 0.81 0.96 0.81 1.00 0.55 0.81 0.26 0.20
locomment 0.61 0.38 0.29 0.35 0.60 0.58 -0.17 0.50 0.39 0.38 0.58 0.38 0.55 1.00 0.62 0.26 0.16
loblank 0.80 0.54 0.34 0.54 0.80 0.79 -0.22 0.64 0.57 0.60 0.79 0.60 0.81 0.62 1.00 0.27 0.20
loccodeandcomment 0.28 0.21 0.19 0.21 0.28 0.27 -0.11 0.25 0.22 0.15 0.27 0.15 0.26 0.26 0.27 1.00 0.12
defects 0.25 0.21 0.17 0.18 0.20 0.19 -0.16 0.17 0.19 0.09 0.19 0.09 0.20 0.16 0.20 0.12 1.00
loc v(g) ev(g) iv(g) n v l d i e b t locode locomment loblank loccodeandcomment defects
1.00 0.75 0.50 0.25 0.00 −0.25


Scatter and Density Plot
loc
v(g)
ev(g)
iv(g)
n
v
l
d
i
e
Corr. coef = 0.751
Corr. coef = 0.710
Corr. coef = 0.316
Corr. coef = 0.758
Corr. coef = 0.717
Corr. coef = 0.800
Corr. coef = -0.062
Corr. coef = 0.574
Corr. coef = 0.209
1e7