VELAGAPUDI RAMAKRISHNA SIDDHARTHA ENGINEERING COLLEGE

(AUTONOMOUS **)**

AI TOOLS AND TECHNIQUES

HOME – ASSIGNMENT

Submitted To :

DR. Sangeetha Mam

Submitted By :

208W1A1298

208W1A1299

208W1A12A0

Problem : Write a solution for a software defects by Using the learning algorithms

Objective :

In this paper, we propose Transfer Learning Code Vectorizer, a novel method that derives features from the text of the software source code itself and uses those features for defect prediction. Here, we mainly focus on the software code and to convert it into vectors using a pre-trained learning language models (SVM, Decision ….) .

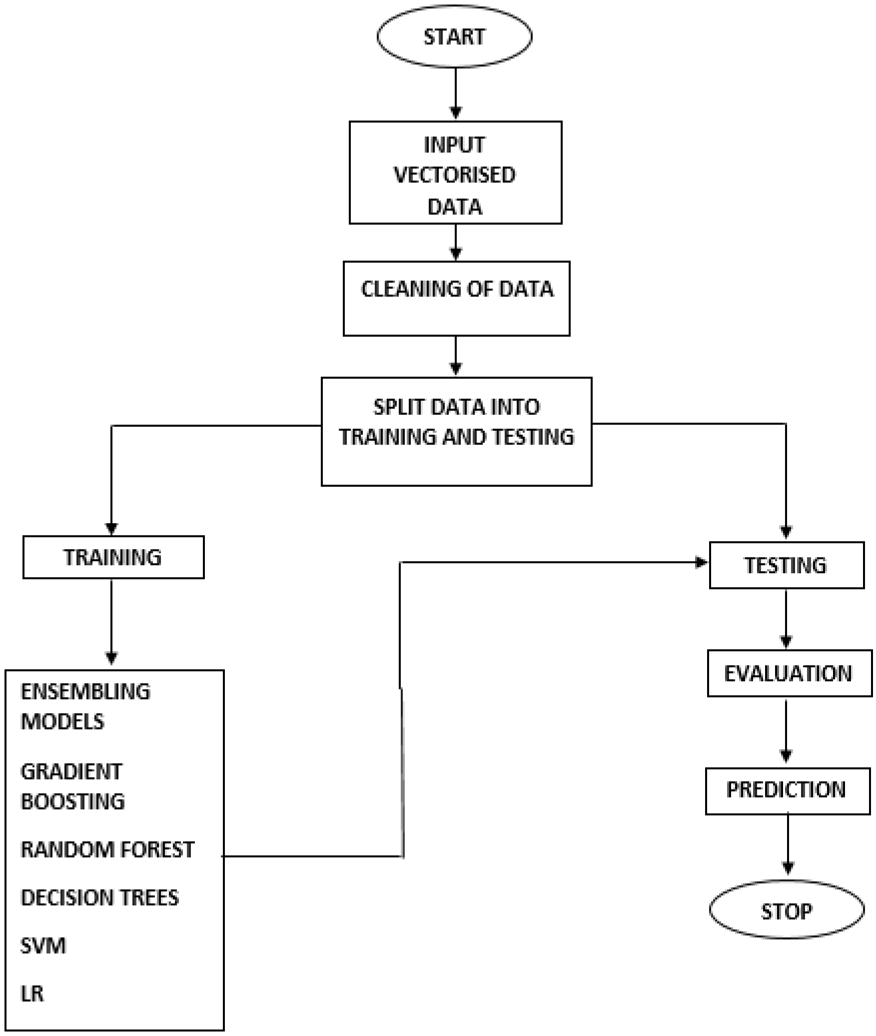
Abstract :

Despite of great planning, well documentation and proper process during software development, occurrences of certain defects are inevitable. These software defects may lead to degradation of the quality which might be the underlying cause of failure. Researchers have devised various methods that can be used for effective software defect prediction.

The prediction of the presence of defects or bugs in a software module can facilitate the testing process as it would enable developers and testers to allocate their time and resources on modules that are prone to defects. In this paper, we propose Transfer Learning Code Vectorizer, a novel method that derives

features from the text of the software source code itself and uses those features for defect prediction.

Block Diagram :



COMMON CODE FOR ALL USED ALGORITHMS :

We can Predict the defects of the Software by using the 6 Algorithms and they are :

- Decision Tree Classifier Algorithm
- KNN Algorithm
- Logistic Regression
- Naive Bayes Algorithm
- Random Forest Algorithm
- Support Machine Vector (SVM) Algorithm

The Above Algorithms are part of the Learning Models in the Ai

Installing the Packages

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# import plotly.plotly as py

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


data = pd.read_csv('pc1.csv')

print(data)

data.head( )


defect_true_false = data.groupby('defects')['b'].apply(lambda x : x.count())

print(" false values : ", defect_true_false[0])

print(" True values : ", defect_true_false[1])


defect_true_false
```

```
trace = go.Histogram(

    x = data.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)

data.corr( )

type(data.corr( ))


f,ax = plt.subplots(figsize = (15, 15))

sns.heatmap(data.corr(), annot = True, linewidths = .5, fmt = '.2f')

plt.show()


trace = go.Scatter(

    x = data.v,

    y = data.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)

data.isnull( ).sum( )



trace1 = go.Box(

    x = data.uniq_Op,

    name = "unique Operators",

    marker = dict(color = "blue"))

box_data = [trace1]

iplot(box_data)



def evaluation_control(data):

    evaluation = (data.n < 300) & (data.v < 1000) & (data.d < 50) & (data.e < 500000) &
(data.t < 5000)

    data['complexityEvaluation'] = pd.DataFrame(evaluation)
```

```
    data['complexityEvaluation'] = ['Succesful' if evaluation == True else 'Redesign' for evaluation in data.complexityEvaluation]


evaluation_control(data)

data

data.info( )


data.groupby("complexityEvaluation").size()

# Histogram Graph

trace = go.Histogram(

    x = data.complexityEvaluation,

    opacity = 0.75,

    name = 'Complexity Evaluation',

    marker = dict(color = "darkorange"))

hist_data = [trace]

hist_layout = go.Layout(barmode = 'overlay',

                        title = 'Complexity Evaluation',

                        xaxis = dict(title = "Succesful - Re design"),

                        yaxis = dict(title = " Frequency "))

fig = go.Figure(data = hist_data, layout = hist_layout)


iplot(fig)
```

```
from sklearn import preprocessing
scale_v = data[['v']]
scale_b = data[['b']]
minmax_scaler = preprocessing.MinMaxScaler()
v_scaled = minmax_scaler.fit_transform(scale_v)
b_scaled = minmax_scaler.fit_transform(scale_b)
data['v_ScaledUp'] = pd.DataFrame(v_scaled)
data['b_ScaledUp'] = pd.DataFrame(b_scaled)


data
scaled_data = pd.concat([data.v, data.b, data.v_ScaledUp,
data.b_ScaledUp], axis = 1)
scaled_data


data.info( )
from sklearn.metrics import confusion_matrix,
classification_report
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn import model_selection
X = data.iloc[:, :-10].values
Y = data.complexityEvaluation.values
Y
```

```
# parsing and verification on data
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size = validation_size, random_state = seed)
print(X_train)
print(X_validation)
print(Y_train)
print(Y_validation)
```

Decision Tree :

```
 from sklearn import tree
model = tree.DecisionTreeClassifier()
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)


model.fit(X_train, y_train)


y_pred = model.predict(X_test)
```

```
# Summary of the Pedictions
print("Decision Tree Algorithm \n \n \n")
print(classification_report(y_test, y_pred))
print("\n\n\n")
print(confusion_matrix(y_test, y_pred))
print("\n\n\n")

# Accuracy

from sklearn.metrics import accuracy_score

print(" Accuracy Model Score :  ",
accuracy_score(y_pred,y_test))
```

KNN :

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=5)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size
= 0.2, random_state = 0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
#Summary of the predictions made by the classifier
print("K-Nearest Neighbors Algorithm")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
from sklearn.metrics import accuracy_score
print("ACCURACY : ",accuracy_score(y_pred,y_test))
```

Logistic Regression :

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size
= 0.2, random_state = 0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

#Summary of the predictions made by the classifier
print("SVM Algorithm")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
```

```
from sklearn.metrics import accuracy_score
print("ACC: ",accuracy_score(y_pred,y_test))
```

Naive’s Bayes :

```
#Creation of Naive Bayes model
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
#Calculation of ACC value by K-fold cross validation of NB model
scoring = 'accuracy'
kfold = model_selection.KFold(n_splits = 10, random_state = seed, shuffle = True)
cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv = kfold, scoring = scoring)
msg = "Mean : %f - Std : (%f)" % (cv_results.mean(), cv_results.std())
msg
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
#Summary of the predictions made by the classifier
print("Naive Bayes Algorithm")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
from sklearn.metrics import accuracy_score
print("ACC: ",accuracy_score(y_pred,y_test))
```

Random Forest :

```
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier(n_estimators=100)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size
= 0.2, random_state = 0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

#Summary of the predictions made by the classifier
print("Random Forests Algorithm")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
```

```
from sklearn.metrics import accuracy_score
print("ACC: ",accuracy_score(y_pred,y_test))
```

SVM :

```
from sklearn import svm
model = svm.SVC(kernel='linear', C=0.01)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size
= 0.2, random_state = 0)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

#Summary of the predictions made by the classifier
print("SVM Algorithm")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
#Accuracy score
from sklearn.metrics import accuracy_score
print("ACCURACY : ",accuracy_score(y_pred,y_test))
```

```
error_rate = 1 - accuracy_score(y_pred,y_test)

error_rate
```

Result : Successfully Completed