Part I: Naive Base Classifier

Project	**F1**
jackrabbit	0.2686414708886619
xorg	0.3061667183142237
jdt	0.2731565564838358
lucene	0.35405405405405405

Part II

Project	**F1/Logistic Regression**	**F1/Decission Tree**	**F1/SVM**	**F1/Random Forest**
jackrabbit	0.38014311270125	0.3705035971223	0.3539094650205762	0.2759336099585062
xorg	0.076923076923076	0.268141592920354	0.048780487804878044	0.09037037037037038
jdt	0.1016949152542373	0.2713513513513513	0.08993576017130621	0.1
lucene	0.0910031023784901	0.26575931232091693	0.0563230605738576	0.10885167464114832

Parameters Used while tunning the models:
1) Support Vector Machine: penalty='l1', dual=False, max_iter=10000000, loss='squared_hinge', tol=1e-3
setting the penalty to l1 and dual to false improved the svm model and gave the best value of F1.

2) **Decession Tree:** random_state=None, criterion='gini', max_depth=10, max_leaf_nodes=50, min_samples_leaf=1, splitter='best'
with the above parameters, the classifer gave the best value of F1

3) **Logistic Regression:** random_state=0, max_iter=12000, solver='lbfgs'

4) **Random Forest Regressor:** n_estimators=100, criterion='gini', max_depth=10, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', min_impurity_decrease=0.0, random_state=None

Part III

Project	Logistic Regression	Decission Tree	SVM	Random Forest	Naive base model	metrics
jackrabbit	0.43630166169578183	0.4233637116818558	0.4455100261551874	0.23644251626898047	0.4337599446558284	**F1**
xorg	0.15796519410977242	0.1611675126903553	0.1608725289706885	0.026438569206842927	0.2725749394898529	**F1**
jdt	0.2702104097452935	0.22365339578454335	0.21916092673763307	0.05309734513274337	0.29917032698877505	**F1**
lucene	0.1722113502935421	0.14317180616740088	0.1366649668536461	0.028259473346178548	0.31889236266190263	**F1**

Feature Collection:
To create new set of features for each project, first of all a bag of words was created for each project. The bag of word was created by tokenizing each single patch file and counted the word frequency of all patch files. Then, a dictionary of words of frequency of more than 3 was created of all the tokens collected for the patch files. The dictionary was cleaned of all non words symbols and of tokens of more than 20 chars length.
Then of the tokens of the created dictionary, a loop iterated over all the change_ id of the train and test data and calculated the frequency of each key of the dictionary in each change and save it to a numpy array.
The numpy array then converted to a datafram and saved as a new csv file with all the features added to a folder named data.

Comparing results from PartII and PartIII:
By comparing the results of partII and partII, we can see that the F1 vlaues have imporeved for some models while it decreased for others when adding new features and increasing the dimentionality of the input data. The table below shows F1 values for differnt projects using different classifiers that have been trained using partII and partIII data:

Project	F1/Logistic Reg PartIII	F1/Logistic Reg PartII
jackrabbit	0.43630166169578183	0.38014311270125
xorg	0.15796519410977242	0.076923076923076
jdt	0.2702104097452935	0.1016949152542373
lucene	0.1722113502935421	0.0910031023784901

Project	F1/Decission Tree PartIII	F1/Decission Tree PartII
jackrabbit	0.4233637116818558	0.3705035971223
xorg	0.1611675126903553	0.268141592920354
jdt	0.22365339578454335	0.2713513513513513
lucene	0.14317180616740088	0.26575931232091693

Project	F1/SVM PartIII	F1/SVM PartII
jackrabbit	0.4455100261551874	0.3539094650205762
xorg	0.1608725289706885	0.048780487804878044
jdt	0.21916092673763307	0.08993576017130621
lucene	0.1366649668536461	0.0563230605738576

Project	F1/Niave Bays PartIII	F1/Niave Bays PartI
jackrabbit	0.4337599446558284	0.2686414708886619
xorg	0.2725749394898529	0.3061667183142237
jdt	0.29917032698877505	0.2731565564838358
lucene	0.31889236266190263	0.35405405405405405

Part IV- Improving the training set by resampling
Resampling Methods used:
1) SMOTE
first selects instances from the minority class and finds k nearest neighbors for each instance, where k is a given number. It then creates new instances using the selected instances and their neighbors.
2) Resampling with Replacement
The most common way to address this issue of imbalanced learning is based on a resampling procedure. This approach is simple but it has its own drawbacks. We could decide to upsample the class 1, so as to match the number of samples belonging to class 0. However, we can only use the existing data and, after every sampling step, we restart from the original dataset (replacement).
3) Spread Subsample
This method eliminates instances in the majority class until the ratio of majority instances over minority instances is equal to a given ratio. Weight is a property of the instance that reflects the level of impact this instance has on its class. Spread subsample also updates the weight of each instance in order to maintain the overall weight of the two classes.

Project	F1/SVM	F1/Decission Tree	Data Sampling method
jackrabbit	0.5227963525835866	0.48393854748603354	SMOTE
xorg	0.2431736218444101	0.3466780238500852	SMOTE
jdt	0.2778884462151394	0.26867119301648884	SMOTE
lucene	0.2651814588573482	0.269539501478665	SMOTE
jackrabbit	0.5418559377027904	0.5036080516521078	Resampling with Replacement
xorg	0.24489795918367346	0.34895833333333337	Resampling with Replacement
jdt	0.28169014084507044	0.32724107919930373	Resampling with Replacement
lucene	0.22735042735042732	0.27761065749892566	Resampling with Replacement
jackrabbit	0.5835847917923959	0.3978269954032595	Spread Subsample
xorg	0.40176470588235297	0.27598566308243727	Spread Subsample
jdt	0.39955522609340255	0.3106976744186047	Spread Subsample
lucene	0.39270687237026647	0.3288186606471031	Spread Subsample

For resampling, SVM and Decission tree models were used for the resampled data. By comparing the results that were obtained in partIII and partIV, we notice that with resampling the values of F1 have imporved as shown in the table bellow which show the comparison of resutls obtained in partIII and partIV.

Project	**F1/ SVM with SMOTE resamping**	**F1/SVM PartIII**
jackrabbit	0.5227963525835866	0.43630166169578183
xorg	0.2431736218444101	0.15796519410977242
jdt	0.2778884462151394	0.2702104097452935
lucene	0.2651814588573482	0.1722113502935421

Project	**F1/Decission Tree with SMOTE resamping**	**F1/Decission Tree PartIII**
jackrabbit	0.48393854748603354	0.4233637116818558
xorg	0.3466780238500852	0.1611675126903553
jdt	0.26867119301648884	0.22365339578454335
lucene	0.269539501478665	0.14317180616740088

Part V: Further Improvement(bonus)
In this part I used Neural Network deep learning technique. The model that was used to make the pridictoin is Keras model.
The Keras models were defined as a sequence of layers.

The model was run on the original features given in the data to avoid time and space complexity

A sequential model was created and layers were added one at a time until network architecture is complete to the ouput layer.

In this model, I used a fully-connected network structure with three layers.

Fully connected layers are defined using the Dense class. The number of neurons or nodes were specified in the layer as the first argument, and the activation function si specified using the **activation** argument.

I used the rectified linear unit activate function know as ReLU on the first two layers and the Sigmoid function in the output layer.

Compiling the model uses the efficient numerical libraries of TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions.

When compiling, we must specify some additional properties required when training the network. To train the network, it means finding the best set of weights to map inputs to outputs in our dataset.

We must specify the loss function to use to evaluate a set of weights, the optimizer is used to search through different weights for the network and any optional metrics we would like to collect and report during training.

In our case, we will use cross entropy as the **loss** argument. This loss is for a binary classification problems.

Training occurs over epochs and each epoch is split into batches.

- **Epoch**: One pass through all of the rows in the training dataset.
- **Batch**: One or more samples considered by the model within an epoch before weights are updated.

Project	**F1/NN- Keras Model**	**F1/Decission Tree PartIV(SMOTE)**
jackrabbit	0.23838162930563148	0.48393854748603354
xorg	0.1816920943134535	0.3466780238500852
jdt	0.073706591070163	0.26867119301648884
lucene	0.16161616161616163	0.269539501478665

References:
https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/