Glitch Classification for Gravitational Wave Interferometry

Name:	**Rita Abani**
Registration No./Roll No.:	19244
Institute/University Name:	IISER Bhopal
Program/Stream:	e.g., EECS
Problem Release date:	February 02, 2022
Date of Submission:	April 24 2022

1 Introduction

Gravitational waves are disturbances in the curvature of spacetime, generated by accelerated masses, that propagate as waves outward from their source at the speed of light. The detection of gravitational waves demands a thorough understanding of instrumental responses in the ecosystem of environmental noise. Hence of pertinent interest is the study of anomalous non Gaussian noise transients called ‘Glitches’. The classification of glitches is essential owing to their high occurrence rates in LIGO data that often hazard and mimic true gravitational wave signals. The data used in this project has been extracted from LIGO’s Gravity Sky portal and contains metadata about these ‘Glitches’. The train data contains information about the characteristics of a glitch like bandwidth, signal to noise ratio etc. (there are a total of 7 such features). The test data contains the glitch labels or the 22 types of glitches along with unique identification labels. In this project, various machine learning models from the sklearn or the scikit learn library in python namely K-nearest neighbours, Support Vector Machines, Random Forest and Decision Trees were used to train the data and develop an accurate model to classify glitches. That model was then run on the test data. In another sub-instance of this project, One Hot Encoding was used deal with the categorical variable ifo or detector location and hence to target the research question ’Does the location of the interferometer have any impact on the classification of glitches’

The scatter plot below shows the correlation between various variables in the train data set.

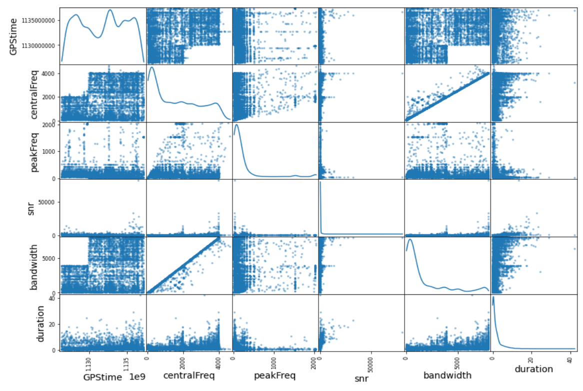
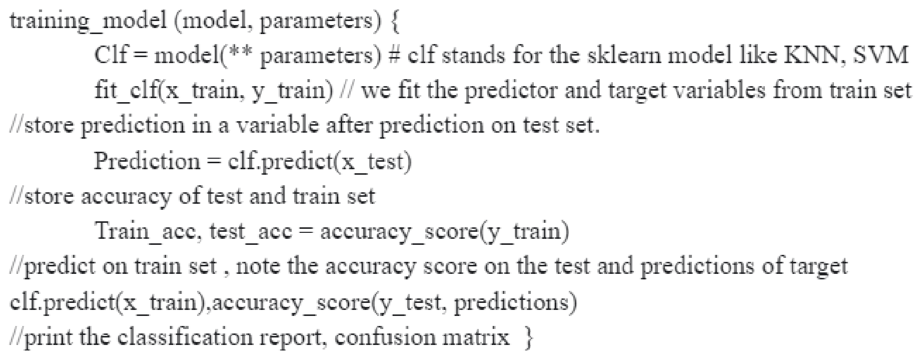


Figure 1: correlation between various glitch features in the train data set

2 Methods

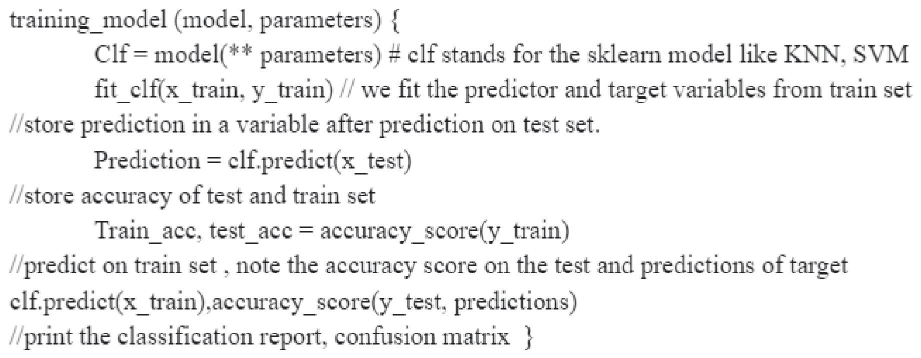
This project explored the use of sklearn ML models like KNNs, SVMs, Decision Trees and Random Forest via a pipeline based on the 'Divide and Conquer Algorithm' juxtaposed to the generally used pipeline which is defined to run everything from the pre-processing to hyperparameter tuning to the evaluation and prediction at one go. A training routine was first designed, the pseudo-code of which is shown below (feature selection was used to check the weighted contribution of the various columns in the data set): Before passing parameters to the training routing given above, we separately performed



```
training_model (model, parameters) {
        Clf = model(** parameters) # clf stands for the sklearn model like KNN, SVM
        fit_clf(x_train, y_train) // we fit the predictor and target variables from train set
//store prediction in a variable after prediction on test set.
        Prediction = clf.predict(x_test)
//store accuracy of test and train set
        Train_acc, test_acc = accuracy_score(y_train)
//predict on train set , note the accuracy score on the test and predictions of target
clf.predict(x_train),accuracy_score(y_test, predictions)
//print the classification report, confusion matrix  }
```

Figure 2: pseudo code for the training substructure

hyperparameter tuning using GridSearchCV.



```
training_model (model, parameters) {
        Clf = model(** parameters) # clf stands for the sklearn model like KNN, SVM
        fit_clf(x_train, y_train) // we fit the predictor and target variables from train set
//store prediction in a variable after prediction on test set.
        Prediction = clf.predict(x_test)
//store accuracy of test and train set
        Train_acc, test_acc = accuracy_score(y_train)
//predict on train set , note the accuracy score on the test and predictions of target
clf.predict(x_train),accuracy_score(y_test, predictions)
//print the classification report, confusion matrix  }
```

Figure 3: pseudo code for hyperparameter tuning

This divide and conquer approach of splitting the pipeline into a training routine and a parameter tuning routine reduced the time complexity which was evident owing to reduced time execution that was sufficed by a local intel i7 NVIDIA processor (without a GPU). For each of the ML models, i.e KNN, SVM, Decision Tree and Random Forest, parameter tuning was done followed by using the training routine with those optimal parameters. We considered 2 scenarios, the first one which didn't consider location of the detector, the second scenario where one hot encoding was used to convert the location (categorical variable into to numerical).

The detailed overview of the method, inferences and visulaizations can be viewed on my Github repostory for this project : GITHUB

3 Evaluation Criteria

Weighted F1 score , train and test accuracy were used to determine the best model for evaluation. The best model was run on the train samples and a scatter plot was made to visualize the results.

As can be seen from the table, the Decision Tree classifier had the highest F-measure value along with an accuracy of 1 on the train data and 0.87 on test data. In scenario 2, after one hot encoding of

Table 1: Performance Of Different Classifiers Without Considering Location of detctors

Classifier	Precision	Recall	F-measure
KNN	0.66	0.66	0.66
SVM	0.69	0.69	0.67
Decision Tree	0.80	0.79	0.79
Random Forest	0.11	0.12	0.11

ifo , i.e converting the two locations, Hanford and Livingston into separate columns in the table, we first checked for the best model on data in case a) exclusively from Hanford , where we obtained 0.86, 0.87,0.86 as the precision, recall and weighted f measure respectively. From the Livingston station, we obtained 0.13, 0.12 and 0.13 as the same values respectively. Scenario 2 was proposed to tacklle whether location has any impact on glitch classification or not.

4 Analysis of Results

Location	Precision	Recall	F-measure
Hanford	0.86	0.87	0.86
Livingston	0.13	0.12	0.13

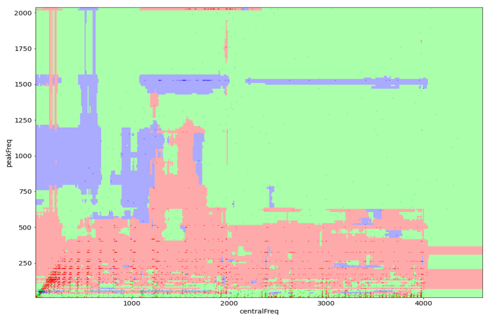


Figure 4: Listed Color Map for Hanford data

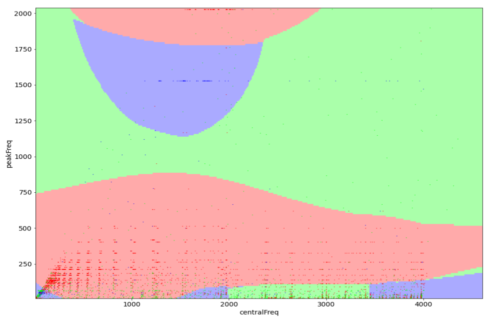


Figure 5: Listed Color Map for Livingston data

The research problem in Scenario 2, where we were trying to find out a correlation between the location of the interferometer and glitch classification hasn't been made conducively clear through the results obtained. The Listed Color Maps show the variations in visulaizing the data dsitribution after running the model with the highest f-score. Over-fitting of data, and most importantly imbalance

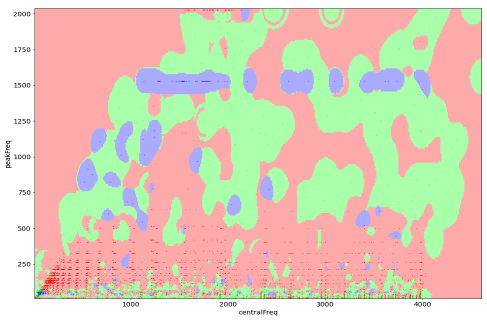


Figure 6: Listed Color Map for Scenario 1 without location

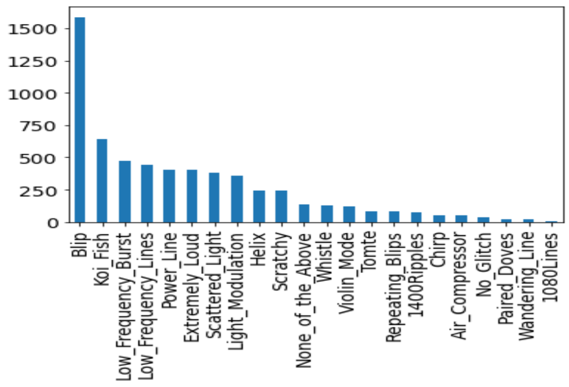


Figure 7: Distribution of Glitches, their types

might be a contributing factor. As can be seen from the Glitch distribution bar plot, the number of non glitch events is extremely low and the percentage of Blip glitches is very high.

5 Discussions and Conclusion

As can be seen from the Glitch distribution bar plot, the number of non glitch events is extremely low and the percentage of Blip glitches is very high. The very innate nature of imbalanced data obtained from the interferometers at LIGO prompt us to consider upsampling and other data balancing techniques which haven't been considered in this project. However, the very fact that we cannot seem to obtain a highly robust glitch classification model hints at how difficult it is in terms of the laws of physics and nature to identify black hole mergers which cause gravitational waves. The very minute proportion of 'No glitch' labels shows how rare it is to detect them, and hence re-sampling techniques may not help us infer much knowledge about the detection events. An optimum solution for the Gravitational Wave community would be to solve this problem using a combined approach of adopting resilient hardware as well as robust Ml networks which still have more scope with emerging techniques like Spiking Neural Networks, Quantum Computation, etc.