PROTEIN LYSINE METHYLATION CLASSIFICATION USING DECISION FOREST MODELS

Joshua Tanner - 100997120   Nhat Hieu Le - 101124254   Zein Hajj-Ali - 101020677

I. Abstract

Lysine methylation plays a crucial role in gene regulation for many human diseases such as cancer, heart diseases, and neurodegenerative disorders, thus understanding the mechanism of methylation for drug design and research works is essential for disease treatment, especially, an initial but important step is to detect methylation sites. In this project report, we apply a machine learning classifier model called Random Forest, an ensemble learning algorithm that is trained and validated on a provided dataset and tested on a blind test set. Additionally, an iterative process of an artificial intelligence model cycle starts from data preparation, model tuning, ensemble learning, and testing stages have been comprehensively investigated in this report. Our method archives the second-highest place, surprisingly close to the first position, based on the metric used in the course project.

II. Introduction

Protein methylation was discovered more than 50 years ago [1] and, together with other post-translational modifications (PTM), which are chemical modifications that play a crucial role in functional proteomic since they manipulate activity, localization, and interaction with other cellular molecules such as proteins, nucleic acids, lipids, and cofactors. Lysine methylation is a PTM that has been well established on histones and it can have one of three distinguished methyl forms (mono-, di- and trimethylation). Traditional biological experimental methods for identifying methylation sites on proteins are expensive in terms of time-consuming and labor-intensive. Therefore, silico computational methods are widely used to obtain potential methylation sites for further analysis. Furthermore, thanks to the rapid emergence of artificial intelligence, machine learning techniques have been widely used to tackle the complexity of identifying methylation sites. Artificial intelligence prediction methods overcome challenges and limitations by uncovering methylation events merely based on the general underlying biochemical characteristics of known modification sites. In particular, this research report applies Random Forest (RF) algorithms, trained on the provided dataset then tested on the blind test set for protein methylation classification.

The remainder of the project paper will be organized as follows. Section III will introduce algorithms and open-source libraries used in this project. Subsequently, Section IV and Section V describe the iterative model optimization starting from data preparation to model tuning, and at last, ensemble learning. Finally, results and conclusions will be analyzed in Section VI and Section VII respectively.

III. Background

Machine learning is a set of methods that are able to discover patterns in existing data and then use the learned patterns to predict future data, predict the class of a piece of unseen data, or conduct other types of decision-making under uncertainty [2]. Generally, it can be categorized into unsupervised learning, which looks for patterns within the given samples, supervised learning, which predicts the mapping between input features and outputs, and reinforcement learning, which learns based on reward and punishment feedback system [2]. In this project, the supervised learning approach was used to solve the prediction problem of detecting methylation sites. Moreover, both unsupervised and supervised approaches were applied during our iterative model optimization process as illustrated in Fig.1.

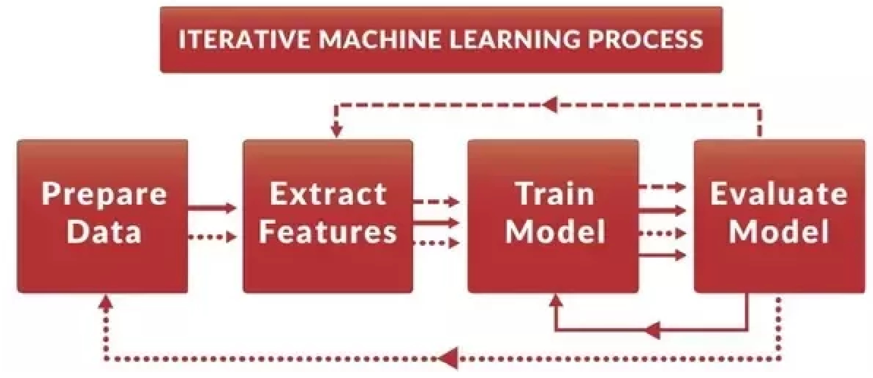

Fig. 1. Primary Steps in our Project [3]

Below are some of the open-source libraries, and machine learning algorithms used for the project:

Tensorflow and Keras: Tensorflow is an open-source software library written in Python, C++, and CUDA that was developed by Google. Although it can be used for various machine learning tasks, TensorFlow focuses on the training and inference of Deep Neural Networks. Keras is the high-level API that runs on top of Tensorflow. Tensorflow and Keras were used together to implement the deep learning algorithm used as one of the Level 0 Classifiers in the Meta-Learning/Ensemble model.

Scikit-Learn: is an open-source software machine learning library for the Python programming language. In addition to myriad support of supervised and unsupervised learning algorithms, Scikit-Learn integrates well with popular numerical and scientific libraries like Pandas and Numpy whilst providing efficient data transformer libraries. Sciket-Learn's implementation of the Random Forest algorithm was used to create the models for this project.

Further, it's PCA, Stratified K-Fold Cross-Validation, and Standard Scaler implementations are used at the various stages of the project for model optimization and model testing.

Numpy and Pandas: are some of the most popular Python libraries used for data manipulation when interacting with a multi-dimensional dataset. In this project, both Numpy and Pandas were used for data preparation and implementing deep learning algorithms.

Seaborn: is a statistical visualization library for Python, built on Matplotlib. It was used to make the graphs for data visualization and interpretation.

Imbalanced Learn: is a Python library that implements many different re-sampling algorithms in a way that is compatible with SciKit-Learn. SMOTE, RandomOverSampler, and Neighbourhood Cleaning Rule were a few of some of the sampling techniques implemented. Pipeline from Imbalanced Learn was also used to chain various sampling methods together.

XGBoost: is an especially efficient implementation of gradient boosted decision trees, where in each training iteration new models are trained to predict the residuals of prior models and then added together to make the final prediction. XGBoost employs more precise approximations than Gradient Boosting by using second-order gradients and advanced regularization. The XGBClassifier from XBoost was used to choose the relevant features from the dataset, within a relevancy threshold.

Random Forest: is an ensemble tree-based learning algorithm which consists of a large number of individual decision trees that randomly bootstrap a subset of the training set. It then aggregates the votes from different decision trees using bagging, to make the final prediction.

Principal Component Analysis (PCA): is an unsupervised, non-parametric statistical technique used for dimensionality-reduction. PCA transforms a larger set of features into smaller sets that still sufficiently contain most of the original information by the means of eigenvectors and eigenvalues from the covariance matrix.

A variety of resampling techniques were implemented in order to help improve the random forest model's performance due to the highly class imbalanced methylation classification problem. Oversampling is a method of training a classifier while over-selecting data points from the minority class to better match the proportion of the majority class. Undersampling is a method of training a classifier while under-selecting data points from the majority class to better match the minority class. Oversampling with undersampling was also implemented.

The oversampling techniques implemented were random oversampling, SMOTE (Synthetic Minority Oversampling TEchnique), Borderline SMOTE and ADASYN. Random oversampling consists of randomly oversampling with replacement points from the minority class. SMOTE is a technique that generates lines between closest points in the minority class, and oversampling points from these synthetically generated lines. Borderline SMOTE is similar to SMOTE, however emphasizes its oversampling from generated lines that are close to the class decision boundary. ADASYN is a technique that highly samples from the lower density regions and samples less from the higher density regions within the minority class.

The undersampling techniques implemented were NearMiss (1, 2 and 3), Tomek Links, Edited Nearest Neighbour, One Sided Selection and Neighborhood Cleaning Rule. NearMiss 1 selects the samples with the smallest average distance to 3 of its closest minority samples. NearMiss 2 selects examples with minimum average distance to the 3 furthest minority examples. NearMiss 3 selects examples with minimum distance to the rest of the minority class. Tomek Links is a modified method of Condensed Nearest Neighbor, which is a method that keeps only the data points needed to perform the nearest neighbour algorithm. Edited Nearest

Neighbour edits pre-classified samples based upon data points' 3 nearest neighbours. One Sided Selection combines Condensed Nearest Neighbour and Tomek Links. Lastly, Neighborhood Cleaning Rule is a combination of Condensed Nearest Neighbour with Edited Nearest Neighbors.

SMOTE with random undersampling was the oversampling with undersampling technique that was implemented. SMOTE with random undersampling is a method that uses SMOTE to oversample from the minority class, and randomly undersamples from the majority class.

IV. Data Preparation

In this section, the dataset that the classification methods was trained and validated on is introduced. We begin by describing how the dataset is constructed, after which, data cleaning and data manipulation steps are discussed. Finally, the section is concluded by discussing the details of transforming and preparing features.

The full dataset combining both provided training and validation CSV files contains more than 24900 observations. Each observation consists of twenty-nine features or descriptors including its class label, methylated or non-methylated site, of the local sequence window centered on lysine. The provided data set is highly skewed as shown in Fig. 2 below with methylated and non-methylated lysine denoted as "1" and "0" respectively.

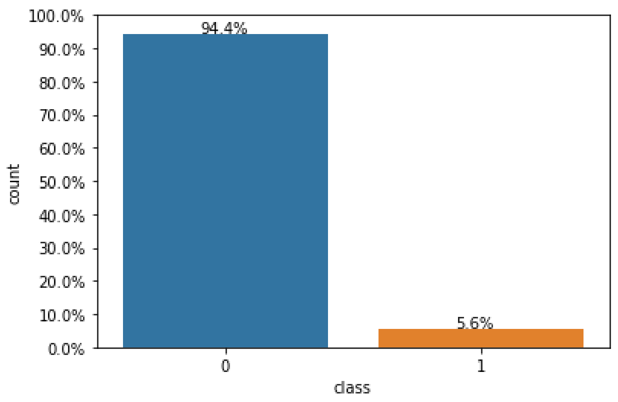

Fig. 2. Class label distribution.

The combined dataset was checked for duplicated observation and missing data. Subsequently, in order to avoid overfitting highly correlated features are dropped based on the correlation matrix of all descriptors as presented in Fig. 3. It can be seen that "Pb_NO_PCR_V" and "ECI_NO_PCR_CV" columns have perfect linear correlations, as a result, the former column was removed.

The processed dataset then underwent feature selection for faster model training, to avoid overfitting, reduce model complexity and improve model performance. Four experiments with different combinations of preprocessing algorithms were conducted to obtain the best performing method of preprocessing to optimize model performance. The baseline model with a single RF classifier was built to evaluate which feature selection technique, with a performance metric of maximum achievable precision at a recall of at least 50% (Pr@Re50). In our first experiment, the original feature data is compared with the second one in which features are standardized.

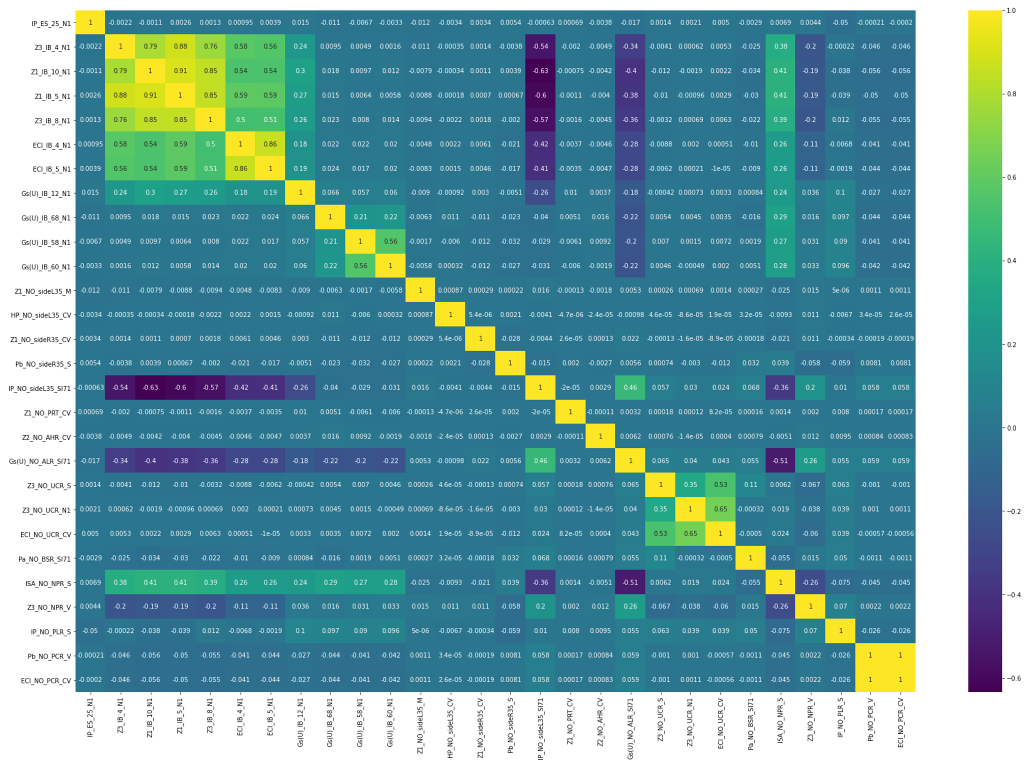

Fig. 3. Correlation Matrix of all features

The third experiment used PCA with twenty principal components to capture at least 95% of the entire dataset's variance as displayed on the cumulative variance plot in Fig. 4.

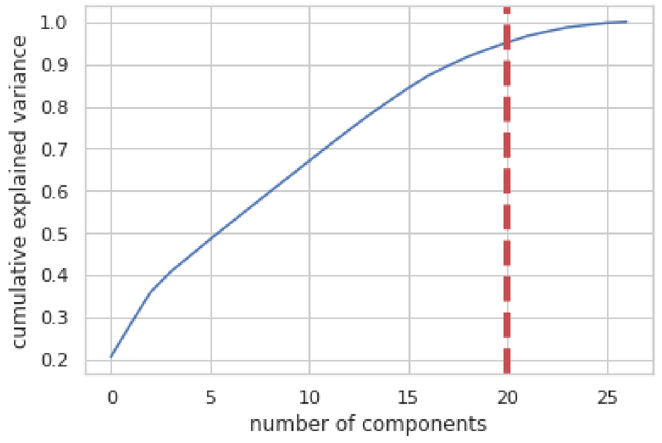

Fig. 4. Cumulative Variance Plot

The last experiment involves XGBoost which was used to select only features of importance at least 0.03. Table 1 below summarizes the result for each preprocessing method. Note that, five fold cross validation was used to compare preprocessing steps, the normalization and XGBoost algorithms were chosen as the best feature selections to use for the next step.

Table 1. Summary of Feature Selection Algorithms' Result

Original Data	**Normalized Data**	**PCA**	**XGBoost**
0.082	0.084	0.078	0.083

V. Methodology

All of the sampling techniques discussed in III. Background for oversampling (random oversampling, SMOTE, Borderline SMOTE and ADASYN), undersampling (NearMiss, Tomek Links, Edited Nearest Neighbour, One Sided Selection and Neighborhood Cleaning Rule) and oversampling with understanding (SMOTE with random undersampling) were used to train and test random forest models with five fold cross validation to compare its precision at a recall of 50% to a baseline random forest model trained without any sampling method. All of the different sampling techniques were iterated over two different preprocessing steps, the random forests' criterion (Gini Index vs Entropy) as well as the number of trees pre forest, which ranged from 10 to 500 trees in steps of 10. The preprocessing steps that were iterated over were normalizing the data, as well as normalizing the data followed by using XGBoost feature reduction, as these were the best performing prepossessing steps as determined in IV. Data Preparation. The percisions at recalls of 50% for the five fold cross validation results for the grid search optimization for the random forest models is shown in Table 2 below. Table 2 only shows the number of trees per forest of the best performing model for conciseness.

Table 2. Grid Search Optimization for Random Forest Parameter and Sampling Technique Optimization

Resampling Method	Oversampling, Undersmaplin or Both	Normalization (Gini) [n trees]	**Normalization (Entropy) [n trees]**	XGBoost Feature Reduction (Gini) [n trees]	XGBoost Feature Reduction (Entropy) [n trees]
None	Neither	0.0810 [160]	0.0867 [190]	0.0802 [80]	0.0864 [160]
Random Over Sampling	Oversampling	0.0883 [110]	0.0870 [200]	0.0855 [150]	0.0889 [340]
SMOTE	Oversampling	0.0856 [120]	0.0834 [160]	0.0823 [160]	0.0817 [100]
Borderline SMOTE	Oversampling	0.0873 [150]	0.0876 [160]	0.0868 [130]	0.087 [170]
ADASYN	Oversampling	0.0847 [200]	0.0838 [120]	0.0824 [190]	0.0823 [140]
NearMiss 1	Undersampling	0.0651 [150]	0.0647 [70]	0.0659 [70]	0.0660 [60]
NearMiss 2	Undersampling	0.0532 [5]	0.0523 [5]	0.0579 [150]	0.0565 [140]
NearMiss 3	Undersampling	0.0582 [10]	0.0577 [20]	0.0572 [20]	0.0563 [5]
Tomek Links	Undersampling	0.0820 [130]	0.0866 [170]	0.0819 [130]	0.0877 [180]
Edited Nearest Neighbour	Undersampling	0.0833 [140]	0.0900 [420]	0.0835 [80]	0.0897 [340]
One Sided Selection	Undersampling	0.0792 [140]	0.0848 [200]	0.0795 [160]	0.0853 [150]
Neighborhood Cleaning Rule	Undersampling	0.0837 [190]	0.0908 [300]	0.0834 [190]	0.0914 [200]
SMOTE for Oversampling with Random Undersampling	Both	0.0942 [440]	**0.0963 [500]**	0.0932 [150]	0.0962 [180]

As shown by the highlighted text in Table 2, the best performing model consisted of SMOTE for oversampling with Random Undersampling, with simple normalization for preprocessing, using the Entropy criterion and had 500 trees per forest. The best performing model had a precision at a recall of 50% of 0.0963.

After tuning the different Random Forest models by trying combinations from a wide array of pre-processing methods, as well as optimizing the hyperparameters of the model; the top ten performing models were chosen to be combined into a stacking ensemble classifier. A stacking ensemble was chosen because it can in many cases perform better than any individual model, although it may take more time to train and tune. The stacking ensemble classifier consists of two levels of models, the base level (Level 0) and the meta-model level (Level 1) also called the final classifier. The Level 0 classifiers were chosen by taking the top ten performing Random Forest model configurations and attaching their pre-processing pipeline to the model building functions. A simple Neural Network model was also built to diversify the prediction scores. Level 1 consisted of a final Random Forest classifier which took the scores predicted by the eleven Level 0 classifiers as an input, and outputs a final prediction/probability for the class. The reasoning behind using a Random Forest model for the final classifier instead of something else like a logistic regression model was to conform with the specifications of our Random Forest based project.

VI. Results

100 iterations of different (varying random states) 5 fold cross validation tests were used to obtain an estimate for the best performing random forest model and the meta learning model. The histogram result for the 100 iterations of 5 fold cross validation for the best performing model is shown in Figure 5 below. The mean (+/- standard deviation) precision at a recall of 50% from the 100 iteration test, and the models expected performance result was 0.0914 +/- 0.0019. An example of the precision recall curve for the best performing model is illustrated in Figure 6. The histogram result for the 100 iterations of 5 fold cross validation for the meta learning model is shown in Figure 7 below. The mean (+/- standard deviation) precision at a recall of 50% from the 100 iteration test, and the meta learning models expected performance result was 0.0744 +/- 0.0059. An example of the precision recall curve for the meta learning model is illustrated in Figure 8. Since the best random forest model outperformed the meta learning model, the best

performing random forest model was used as the model of choice and final prediction (0.0914 +/- 0.0019).

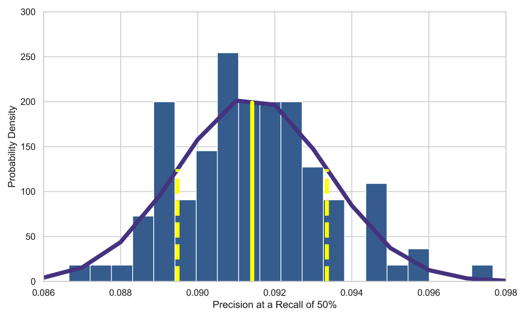

Fig. 5. Histogram of Best Performing Random Forest Model's Test Results

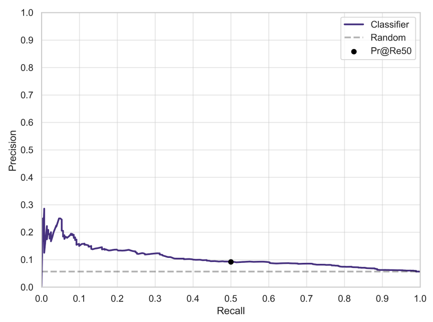

Fig. 6. Precision Recall Curve for Best Performing Random Forest Model

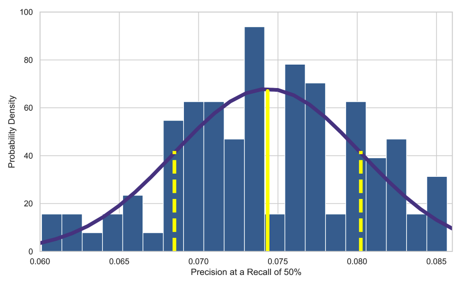

Fig. 7. Histogram of Meta Learning Model's Test Results

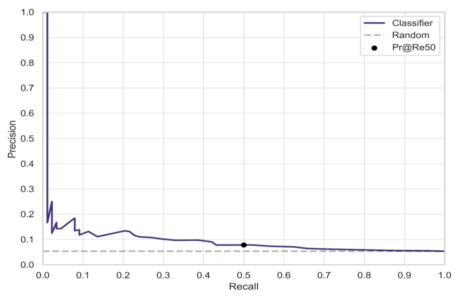

Fig. 8. Precision Recall Curve for Ensemble Model

The best performing Random Forest model was used to predict the probabilities of each class on the blind dataset when the dataset was released. The models prediction scores were sent to be evaluated against the precision at a recall of 50% metric. The blind dataset consisted of 5150 records with a class imbalance corresponding to a 3.75% positivity rate (193 Positive : 4947 Negative). The blind dataset had the same features as the datasets used to train and test the models. Our final model performed worse than expected on the blind dataset, with a precision at a recall of 50% of **0.067880**. It is worth noting that this performance ranked second compared to other classifiers, only behind a CNN model that outperformed our Random Forest model by only 0.000006. When looking at how close our prediction was to the actual score, we can see that although our group's estimated prediction was high, the final result was nowhere near the actual score. As a result the 'Score2' was found to be essentially **0.0000**.

VII. Conclusion

It is interesting to note that the best performing model was found to use a combination of over and under-sampling, resulting in a final positivity rate of 33.3%. This might not be as intuitive as choosing only under-sampling for a dataset like the one used here, due to the very low positivity rate, but this combination of SMOTE with random undersampling consistently outperformed the other models tested. The ensemble seemed like a promising solution at first, but it failed to outperform the Random Forest models, perhaps due to the low number of positive samples and the overall size of the dataset. The use of a stacking ensemble meant the need for further splitting the dataset so that the Level 0 and Level 1 classifiers would not train and test on the same samples. The ensemble model could have also benefited from a longer project timeframe, since more could have been done to tune it's hyperparameters with more time.

After watching the presentations of the other groups, a number of different ideas came up that we hadn't implemented. For example, some groups chose to detect and remove outliers from their training sets. This seemed counter-intuitive, due to the already low positivity rate of the training set. Removing outliers from a set with so few positive samples might mean removing important information.

The predictions made for the precision at a recall of 50% did not ultimately match the actual performance on the blind dataset. One explanation might be that the records in the blind dataset have some explicit difference to the samples in the released portion. Another might be due to the random nature of the Random Forest algorithm. Due to the random nature of the Random Forest model, re-training the model under the same conditions with the same training data might produce a result that performs better than the one used for the final predictions that were submitted. The ‘Score2’ of our predictions was found to be very low, likely due to our standard deviation having been the second smallest out of all the groups.

References:

[1] W. Paik and S. Kim, "Enzymatic methylation of protein fractions from calf thymus nuclei", Biochemical and Biophysical Research Communications, vol. 29, no. 1, pp. 14-20, 1967. Available: 10.1016/0006-291x(67)90533-5 [Accessed: 09- Apr- 2021].

[2] C. Robert, "Machine Learning, a Probabilistic Perspective", CHANCE, vol. 27, no. 2, pp. 62-63, 2014. Available: 10.1080/09332480.2014.914768 [Accessed: 09- Apr- 2021].

[3] C. Granville, "Machine Learning Process Summarized in Two Pictures", Datasciencecentral.com, 2021. [Online]. Available: https://www.datasciencecentral.com/profiles/blogs/deep-learning-pictures. [Accessed: 09- Apr- 2021].

[4] G. Antonio, and S. Pal. Deep learning with Keras. Packt Publishing Ltd, 2017. [Accessed: 09- Apr- 2021].

[5] H. Charles. "Array programming with NumPy." Nature 585.7825 (2020): 357-362. [Accessed: 09- Apr- 2021]

[6] M. Wes. "Pandas, python data analysis library." see http://pandas.pydata.org (2015). [Accessed: 09- Apr- 2021]

[7] B. Ekaba. "Matplotlib and Seaborn." Building machine learning and deep learning models on google cloud platform. Apress, Berkeley, CA, 2019. 151-165. [Accessed: 09- Apr- 2021]

[8] L. Guillaume, F. Nogueira, and C. Aridas. "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning." The Journal of Machine Learning Research 18.1 (2017): 559-563. [Accessed: 09- Apr- 2021]

[9] B. Jason. "Xgboost with python." Machine Learning Mastery (2019). [Accessed: 09- Apr- 2021]

[10] P. Fabian, et al. "Scikit-learn: Machine learning in Python." The Journal of machine Learning research 12 (2011): 2825-2830. [Accessed: 09- Apr- 2021]

[11] V. Guido. "Python." (1991). [Accessed: 09- Apr- 2021].

[12] J. Brownlee, "SMOTE for Imbalanced Classification with Python", Machine Learning Mastery, 2021. [Online]. Available: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/. [Accessed: 09- Apr- 2021].

[13] J. Brownlee, "Random Oversampling and Undersampling for Imbalanced Classification", Machine Learning Mastery, 2021. [Online]. Available: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/. [Accessed: 09- Apr- 2021].