Diabetes Prediction Over Telephonic Health Survey

Name:	**Aditya Mishra**
Roll Number:	21013
University Name:	IISER Bhopal
Stream:	EECS
Problem Release date:	August 17, 2023
Date of Submission:	November 19, 2023

1 Introduction

In this project, the task is to predict whether a person is suffering from diabetes or not. The Behavioral Risk Factor Surveillance System (BRFSS)[1] dataset is used in the study. It comprises of three class labels: 0 (no diabetes or only during pregnancy), 1 (prediabetes), and 2 (diabetes). Figure 1 illustrates that the distribution of instances across these classes is imbalanced. No missing values in the training dataset eliminate the need for imputation. However, the high imbalance among different classes and the presence of categorical features present challenges that will be explored in greater detail in subsequent sections of the project report.

2 Methods

In the experimental analysis, various state-of-the-art ML models, including decision tree [1], random forest [2], logistic regression [3], naive Bayes classifier [4], k-nearest neighbors (KNN) [5], multi-layer perceptron (MLP) [6], artificial neural network (ANN) [7], extreme gradient boosting (XGB) [8], LightGBM [9] and Adaboost [10] with various base classifiers, were employed to assess their performance on the given task. XGBoost and LightGBM are gradient boosting algorithms which uses decision tree to build strong predictive models. XGBoost minimizes the cost function by adding new trees that correct the errors made by existing ones, using boosting technique. XGBoost is known for its speed, scalability, and regularization techniques, making it a powerful tool for various machine learning tasks. Contrary, LightGBM differs from traditional methods by optimizing the construction of decision trees using a leaf-wise strategy and histogram-based learning. It enhances training efficiency and is particularly effective for large datasets. The data has 21 features, out of which many will not even be relevant for the classification. For the same, we used feature selection techniques, using `SelectKBest`[2] module of sklearn along with correlation matrix. The `score_func` is chosen to be `chi2` because the features are categorical, with a categorical target variable [11]. On the basis of feature importance score 1 and correlation matrix1, total of 8 features are chosen for the model training. The features with magnitude of correlation greater than 0.35 are dropped because highly correlated features provide redundant information to the model and can lead to overfitting [12].

To deal with class imbalance, we employed various oversampling techniques: `RandomOverSampler`[3] and `SMOTE`[4]. `RandomOverSampler` address class imbalance by oversampling the minority classes (Class 1 and 2). It randomly duplicates instances of the minority classes until a more balanced distribution is achieved. `SMOTE` address class imbalance by generating synthetic examples of the minority classes to balance the class distribution. It creates synthetic instances by interpolating between existing minority

[1] www.cdc.gov/brfss/index.html

[2] www.scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

[3] www.imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html

[4] www.imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

class instances. Its working can be understood by Algorithm 2. The `RandomOverSampler` randomly duplicates instances, which causes the effect of randomization and bias towards particular group of instances. To tackle this, we oversampled by concatinating the entire minority class by a factor of `oversampling_ratio`. This oversampling technique best performed on all the models, and will be refered as `RatioOversampler` (say) in the later sections and can be uderstood by the Algorithm 1.

Algorithm 1 RatioOversampler

```
for each class c in minority classes
    minority_class = df_train[df_train['output' ] == c]
    oversampling_ratio_c = get_oversampling_ratio_for_class(c)
    oversampled_minority_class =
        pd.concat([minority_class] * oversampling_ratio_c, ignore_index=True)
    add oversampled_minority_class to df_train_oversampled
end for
```

In `SMOTE`, we iterate through each instance $X[i]$ in the minority classes dataset X. For every $X[i]$, we find its k nearest neighbors using `find_k_nearest_neighbors` with a predefined constant k. Nested loops are then employed, creating synthetic instances for each neighbor $X[j]$ through `generate_synthetic_instance`. The number of synthetic instances per neighbor is determined by `oversampling_ratio`. These synthetic instances are added to `synthetic_samples`, addressing class imbalance by oversampling the dataset.

Algorithm 2 Synthetic Minority Over-sampling Technique (SMOTE)

```
for i = 1 to n
    neighbors = find_k_nearest_neighbors(X[i], X, k)
    for j in neighbors
        for m = 1 to oversampling_ratio
            synthetic_instance = generate_synthetic_instance(X[i], X[j])
            add synthetic_instance to synthetic_samples
        end for
    end for
end for
```

The training dataset is further split into training and validation set usting `train_test_split`[5]. Oversampling is exclusively applied to the new training set, leaving the validation set unaffected. This approach is adopted to guarantee that during model training, the validation set comprises data not encountered by the model. This ensures a robust evaluation of the model's true performance and prevents the possibility of inflated results that might occur if oversampling is performed prior to the data split. `Stratify` is used during `train_test_split` to esnure that the split preserves the proportion of the target variable classes in both the training and validation sets, that means, it maintains the same class distribution in both the training and testing sets as in the original dataset.

All the mentioned ML models are trained on the created pipeline and cross-validated using `GridSearchCV`[6], which resulted in random forest as the best classifier, with and without oversampling. The `scoring` in the `GridSearchCV` is taken as `f1_macro`. Hyperparameter tuning in done on random forest using `GridSearchCV` to find optimum values of hyperparameters for the model's best performance. Adaboost (with different base classifers - logistic regression, decision tree, naive bayes and random forest - Table 2), XGB and LighGBM are also used with various combination of hyperparameters but they are unable to give better results as compared to random forest. It is evident from the Table 1. The github link is: www.github.com/adityamishraaaa/ECS308_21013.git.

[5] www.scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
[6] www.scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

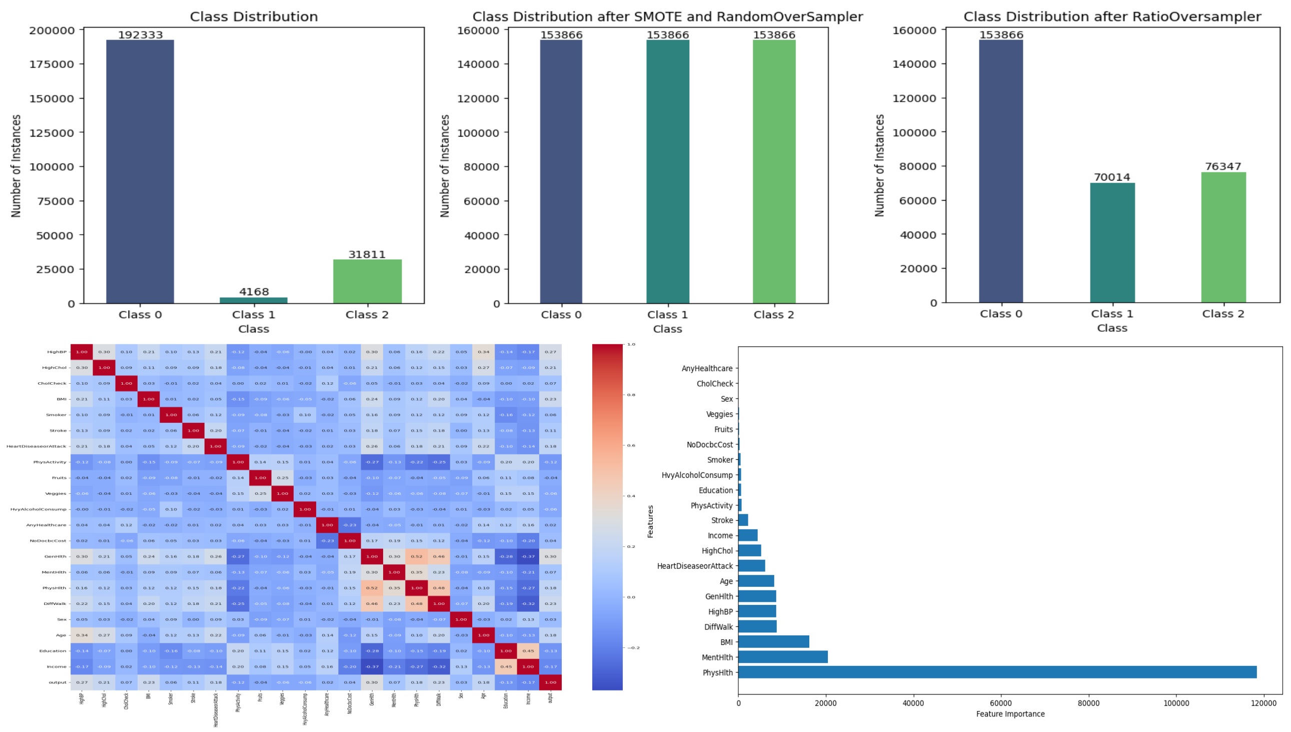

Figure 1: Plot 1 illustrates the original class distibution, plot 2 shows the distribution after RatioOversampler, plot 3 demonstrates class distributuion after RandomOversampler or SMOTE. Plot 4 gives the correlation heatmap and plot 5 depicts the feature importance scores using χ^2-statistics.

3 Experimental Setup

We evaluated the performance of the various models with three evaluation metrics – macro-averaged precision, macro-averaged recall, and the macro F1-score. It ensures that model is not biased towards a particular class, as macro-averaged scores give equal weight to every class, regardless of its frequency.

Macro-averaged Precision: The average precision across all classes, treating each class equally, regardless of class imbalance. **Macro-averaged Recall:** The average recall across all classes, providing a balanced assessment of the model's ability to capture positive instances across different classes.**F1-score:** The f-measure is the harmonic mean of precision and recall. It combines recall and precision with an equal weight. The mathematical formulae for each is given by:

$$\text{Macro-Precision} = \frac{1}{r}\sum_{i=1}^{r}\frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i} \qquad \text{Macro-Recall} = \frac{1}{r}\sum_{i=1}^{r}\frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i} \tag{1}$$

$$F_1 - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2}$$

Here, r is the total number of classes (3 for us), TP_i refers to the total number of true positive instance with respect to the i^{th} class, FP_i defines false positive instances in the i^{th} class, and FN_i is the false negative instances in the i^{th} class. Hyperparameter tuning refers to the process of systematically optimizing the hyperparameters of a machine learning model to achieve better performance. Hyperparameters are configuration settings for a model that are not learned from the data but are set prior to the training process. There were a combination of hyperparameters in various models, which required proper tuning for achieving the best performance. In this section we explain the opted methodology for hyperparameter tuning only the best performer model, i.e., random forest. In random forest, the number of `n_estimators`, splitting `criterion`, `max_depth` of the tree, and `minimum_samples_split` are the major hyperparameters which affected the model's performance most. After tuning using GridSearchCV with evaluation on the validation set and scoring as `f1_macro`, the optimum hyperparamters are determined as `1000` `n_estimators` `criterion` as `entropy`, `max_depth` of `8`, and `5` `minimum_samples_split`.

The number number of decision trees to be used in the random forest are controlled by the parameter `n_estimators`. The maximum of the decision tree and minimum number of instances required for a split are controlled by `max_depth` and `minimum_samples_split`, respectively. The parameter `criterion` decides splitting basis on which splitting of the nodes are done. An optimum balance between the value of each parameter is required for model to perform better and to avoid overfitting, which might be the result of very high value `max_depth` and `n_estimators`.

4 Results and Discussion

Table 1: Performance Of Different Classifiers Using Selected Features

Classifier	**Precision**	**Recall**	**F-measure**
Logistic Regression	0.44	0.52	0.43
AdaBoost	0.41	0.42	0.41
MLP	0.44	0.47	0.45
K-Nearest Neighbor	0.41	0.41	0.42
Naive Bayes	0.43	0.46	0.44
XGB	0.44	0.48	0.45
LightGBM	0.44	0.47	0.45
Random Forest	**0.44**	**0.47**	**0.46**

Table 2: Performance of Adaboost Using Selected Features

Base Classifier	**Precision**	**Recall**	**F-measure**
Logistic Regression	0.43	0.50	0.41
Decision Tree	0.41	0.42	0.41
Naive Bayes	0.49	0.42	0.27
Random Forest	0.44	0.47	0.46

Table 1 illustrates that the random forest model emerges as the best performer, having the maximum macro-F_1 score of 0.46. Tables 3, 4, 5, 6, 7, and 8 show the confusion matrix for various classifiers, in which, the rows are the actual labels and the columns represents the predicted labels. It can be seen that KNN classifies maximum instances of class 2 (diabetic) to class 0 (non-diabetic), which is concerning. Contrary, it classifies the least class 0 instances to class 2. The logistic regression works well for class 2, as it misclassifies the least number of instances from class 2 to class 0. Adaboost works best for classifing class 1 instances. Table 2 illustrates the performance of adaboost with various base classifiers. Additionally, we also tried bagging technique with LightGBM as the base classifier, but the performance is not good. Also, Artificial Neural Network (ANN) with 1-input layer, 2-hidden layer (each with ReLU activavtion function) and 1-output layer (SoftMax activation function) is trained, but it is inefficient with class 1, as the confusion matrix for the ANN has entire column of class 1 as 0. We also experimented K-Means clustering in hope that it can divide the data set in three clusters, however on evaluating with Normalised Mutual Information, we got a poor score of 0.03.

5 Conclusion

In the preceding analysis, it is clear that the Random Forest model outperforms other ML models. Nevertheless, its performance is deemed fair and not optimal. This limitation can be attributed to the substantial class imbalance within the dataset. Additionally, existing oversampling techniques fall short in adequately addressing this imbalance, hindering the models from achieving their peak performance. The future scope of this project is to overcome its limitation by using deep learning algorithms.

	0	1	2
0	24197	6911	7359
1	234	229	371
2	1169	1240	3953

Table 3: Logistic Regression

	0	1	2
0	25493	6224	6750
1	238	240	356
2	1196	1310	3856

Table 4: AdaBoost

	0	1	2
0	33998	917	3552
1	599	46	189
2	3942	400	2020

Table 5: KNN

	0	1	2
0	31812	1081	5574
1	498	55	281
2	2829	308	3225

Table 6: Naive Bayes

	0	1	2
0	31874	1415	5178
1	466	67	301
2	2510	520	3332

Table 7: LightGBM

	0	1	2
0	32948	211	5308
1	508	10	316
2	2714	75	3573

Table 8: Random Forest

References

[1] Leo Breiman. *Classification and regression trees.* Routledge, 2017.

[2] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[3] Raymond E Wright. Logistic regression. 1995.

[4] Thomas Bayes. Naive bayes classifier. *Article Sources and Contributors*, pages 1–9, 1968.

[5] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[6] Konstantinos Koutroumbas and Sergios Theodoridis. *Pattern recognition.* Academic Press, 2008.

[7] Simon Haykin. *Neural networks and learning machines, 3/E.* Pearson Education India, 2009.

[8] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, 2015.

[9] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[10] Robert E Schapire. Explaining adaboost. In *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 37–52. Springer, 2013.

[11] Huan Liu and Rudy Setiono. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of 7th IEEE international conference on tools with artificial intelligence*, pages 388–391. Ieee, 1995.

[12] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.