Explore the impact of techniques for handling data imbalance on perceptron prediction

Yi Han
a1876251
University of Adelaide
yi.han@adelaide.edu.au

Abstract

Diabetes is a serious disease worldwide [11].This study explores three resampling methods for handling imbalanced datasets: SMOTE, random oversampling, and random undersampling to predict patients with diabetes. Although there are more positive samples than negative samples in the original data set, I found that the data processed using these three methods did not significantly improve model performance. This paper describes the implementation process of each method in detail and compares their effects, aiming to provide empirical research for dealing with imbalanced data.

1. Introduction

Handling imbalanced data sets is a common challenge in machine learning and deep learning [1], especially in the healthcare field. When we try to predict rare events or diseases, such as diabetes, the imbalance of the data may cause the model's predictions to be biased towards the majority class, thereby reducing the prediction performance for the minority class. In order to solve this problem, researchers have proposed various resampling techniques, such as SMOTE [6], random oversampling [2] and random undersampling [10].

In this paper, I selected Multi Layer Perceptron (MLP) as the prediction model [5]. MLP is a feedforward neural network composed of multiple layers, including input layer, hidden layer and output layer. It is capable of learning and representing non-linear relationships and therefore has been widely used in many machine learning tasks.

The purpose of this paper is to evaluate the effectiveness of resampling methods in improving the performance of MLP models and provide a valuable reference for handling imbalanced medical data.

The main contribution of this work can be summarised as:

- I not only conducted a detailed experimental evaluation of three commonly used resampling methods: SMOTE, random oversampling, and random undersampling, but also conducted an analysis of their results to explore the effects of the resampling method as well as the factors that may affect its effects.
- By conducting experiments on a real diabetes datasets, I found that resampling methods may not always be effective in some cases. This provides valuable practical experience in dealing with imbalanced data.
- I compared model performance with and without resampling methods in detail, including the F1 score, recall, precision from both class of samples respectively together with accuracy, providing an intuitive reference for researchers and practitioners.

2. Methodology

2.1. SMOTE

The main idea of SMOTE is to create synthetic samples in the feature space, thereby increase the presence of the minority class which is the class with the rarest number of samples and ensuring a more balanced dataset for model training.

For each data point in the minority class, the algorithm of SMOTE computes its k nearest neighbors using Euclidean distance due to its universal acceptance. A subset is selected from these k neighbors depends on the degree of over-sampling. This subset plays a key role in the synthesis of new samples. The algorithm then generate a random scalar in the range [0, 1], which we call δ. This scalar introduces a degree of randomness to the synthesis process, ensuring that the generated samples, not only aligned with the feature space, but also introduce variability. Then it computes the difference between the feature vector of the minority class instance and its selected neighbors. This difference is then multiplied by δ. The difference obtained in the previous step is then added to the feature vector of the original minority class instance. The result of this addition is a new synthetic data point that lies in the feature space between the original data point and its selected neighbors. This procedure is described in Algorithm 1.

$$x_{\text{new}} = x_i + \delta \cdot (x_{zi} - x_i) \tag{1}$$

As shown in Eq. 1, x_i represents the original minority instance, while x_{zi} denotes one of its k nearest neighbors. The factor δ is the aforementioned random scalar, introducing variability to the synthetic sample creation process.

Algorithm 1 SMOTE

```
1:  procedure SMOTE(data, N, k)
2:      Input: data    ▷ samples from training datasets of minority class
3:      $N$    ▷ number of synthetic samples per minority sample
4:      $k$    ▷ number of neighbors
5:      Output: new_data
6:      synthetic samples ← []
7:      for each sample in data do
8:          neighbors ← k-nearest-neighbors(sample, $k$, data)
9:          for i ← 1 to $N$ do
10:             $n$ ← randomly choose a neighbor from neighbors
11:             diff ← $n$ - sample
12:             $\delta$ ← random value between 0 and 1
13:             synthetic sample ← sample + $\delta\times$ diff
14:             synthetic samples.append(synthetic sample)
15:     new_data ← data + synthetic samples
16:     return new_data
```

2.2. Random Oversampling

Random Oversampling is the algorithm that firstly identify the minority class in the datasets which is the class with the rarest number of samples. Once this category is determined, the algorithm randomly selects and replicates samples from these minority classes of training datasets, ensuring that these newly generated replicated samples are completely consistent in features with the original samples. Subsequently, these replicated samples are integrated into the training datasets to enhance the representation of the minority class. This process continues until the number of samples in the minority class approaches that of the majority class, or at least reaches to a predetermined balance standard. This strategy aims to provide a more balanced data environment for model training by increasing the number of minority class samples in the training datasets. This procedure is described in Algorithm 2.

Algorithm 2 Random Oversampling

```
1: procedure RANDOM OVERSAMPLE(data, desired_number)
2:     Input: data    ▷ samples from training datasets of minority class
3:     desired_number    ▷ desired number of minority samples after random over-sampling
4:     Output: new_data
5:     new_data ← copy(data)
6:     while new_data < desired_number do
7:         sample ← randomly select a sample from data
8:         new_data ← sample + new_data
9:     return new_data
```

2.3. Random Undersampling

Random undersampling is a straightforward algorithm to address the class imbalance problem. The core idea is to achieve a balance between classes by randomly removing samples from the majority class. Specifically, this algorithm first determines the majority class in the training datasets, which is the class with the largest number of samples. Then, the system randomly selects and deletes samples from the majority class until the number of samples from the majority class is reduced to be similar to that of the minority class or reaches a predetermined target number. This procedure is described in Algorithm 3.

Algorithm 3 Random Undersampling

```
1: procedure RANDOM UNDERSAMPLE(data, desired_number)
2:     Input: data    ▷ samples from training datasets of majority class
3:     desired_number    ▷ desired number of majority samples after under-sampling
4:     Output: new_data
5:     new_data ← copy(data)
6:     while new_data > desired_number do
7:         sample ← randomly select a sample from data
8:         new_data ← new_data - sample
9:     return new_data
```

3. Experiments and Analysis

3.1. Experiments setup

I implemented the three methods above on training datasets of Pima Indians Diabetes datasets [4], then employed MLP for all our experiments with the same random seed. Then I utilised the F1 score, recall, precision from both class of samples respectively together with accuracy as metrics to compare the effectiveness of these resampling

methods.
[1]

I took MLP without any resampling methods as control group. To concentrate on comparison on different resampling methods, I chose the same MLP model to conduct experiments. Considering the superiority of Adam optimizer[3] and ReLU activation function [8] on classification tasks, I kept these through all experiments on the three layers perceptron model [9] for better performance. There is also certain degree of $\mathcal{L}^2$ regularization in Adam optimizer in every experiment to prevent overfitting [12]. The loss function is binary cross-entropy (BCE) because it performs well on binary classification problems [7].

I used a MacBook Air with Apple M1 chip and 16 GB memory to conduct all of the experiments. Before the experiments, I deleted all of the samples with empty feature values. Then I separated features and labels and convert -1 class to 0 class for PyTorch's BCE. After that, the datasets is randomly splitted into training, validation, and test sets which took 60%, 20% and 20% of the datasets respectively.

During the stage of processing data, the three resampling methods are conducted respectively and the datasets with no resampling method is also performed as control group. For the convenience of comparison, the algorithm stopped resampling until it got a balanced training datasets by the three resampling methods.

The experiments are conducted for 1000 epochs of training. After every 100 epochs of training, it evaluated the trained model on the validation datasets computing the average F1 score of the positive and negative samples. After training, there were 10 results and I loaded the weights with the largest average F1 score into the model which was the process of getting the best model.

Next, the best model was evaluated by the test datasets with the F1 score, recall, precision from both class of samples respectively together with accuracy.

3.2. Analysis

As shown in Fig. 1, the training losses all start at about 0.7 and decrease sharply at the beginning of the training. After trained at around 100 epoch where the losses are around 0.46, the downward trend has flattened clearly and showd an obvious convergence trend. The losses for validation datasets stayed basically constant through all experiments, suggesting that the model might not be generalizing well to unseen data or there might be overfitting since the training loss is decreasing but the validation loss is not. It indicates that the model might not be generalizing well to unseen data or there might be overfitting since the training loss is decreasing but the validation loss is not. An overfitted model may perform poorly on new, unseen data. However, overfitting occurred in all experiments and the differences between training loss and validation loss are not significant. Thus, overfitting is probably caused by inherent characteristics of the dataset or model structure which is not affected by resampling methods.

Fig. 2 describes how the average F1 score changing with epoch after processing the training datasets without

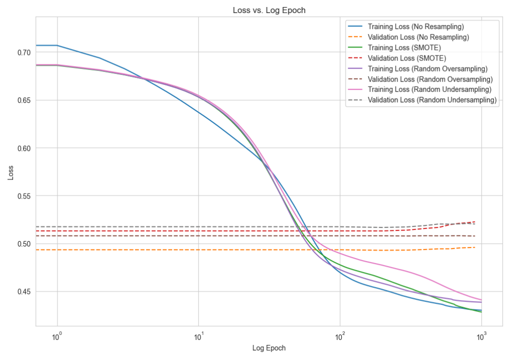
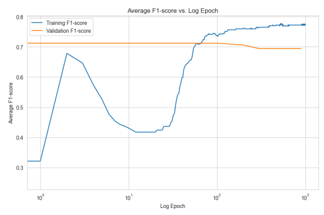
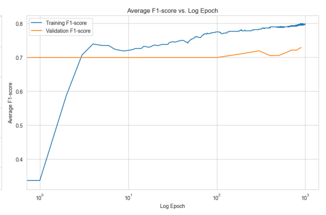


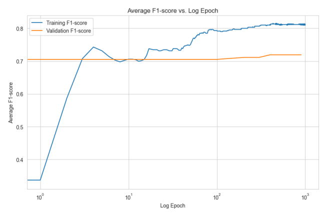
Figure 1: This graph represents the "Loss vs. Log Epoch" for both training and validation data. The x-axis denotes logarithm of the number of epochs, and the y-axis represents the loss. It shows the losses changing with epoch after processing the training set without any method, using the smote method, using the random oversampling method and using the random undersampling method respectively.



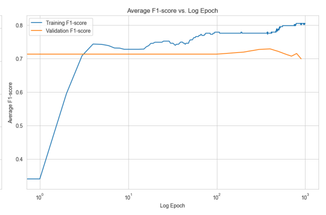
(a)



(b)



(c)



(d)

Figure 2: These four graphs represent the "Average F1 score vs. Log Epoch" for both training and validation data. The x-axis denotes logarithm of the number of epochs, and the y-axis represents the average F1 score. It shows the average F1 score changing with epoch after processing the training datasets without any method, using the smote method, using the random oversampling method and using the random undersampling method from (a) to (d).

[1]GitHub Repository: https://github.com/Yi126/dlf.git

	no resampling method	SMOTE	random oversample	random undersample
F1 (Positive)	0.8276	0.7956	0.8152	0.8111
F1 (Negative)	0.6535	0.6992	0.7167	0.7258
Avg F1	0.7405	0.7474	0.7659	0.7685
Recall (Positive)	0.8660	0.7423	0.7732	0.7526
Recall (Negative)	0.6000	0.7818	0.7818	0.8182
Avg Recall	0.7330	0.7620	0.7775	0.7854
Precision (Positive)	0.7925	0.8571	0.8621	0.8795
Precision (Negative)	0.7174	0.6324	0.6615	0.6522
Avg Precision	0.7549	0.7447	0.7618	0.7658
Accuracy	0.7697	0.7566	0.7763	0.7763

Table 1: This is the results from the test datasets from four methods respectively. The results are the F1 score, recall, precision from both class of samples and the average respectively together with accuracy.

any method, using the smote method, using the random oversampling method and using the random undersampling method from (a) to (d). Overall, the average F1 score all start from low and increase sharply then decrease followed by increase until convergent to a certain degree. However, the decrease in Fig. 2(a) is much more significant than others which means that in the early stages of training, the model may focus mainly on the dominant class, thereby neglecting the minority class. When the model starts trying to classify minority classes, it can cause drastic fluctuations in performance metrics due to their small number. The average F1 score of training datasets in all graphs are all convergent to around 0.8 and in validation datasets it remains almost constant. This is also a sign of overfitting which as discussed above, is probably caused by inherent characteristics of the dataset or model structure which is not affected by resampling methods.

By comparison between the results from table1, it can be concluded that generally there is no significant difference between each resampling method except recall from positive and negtive samples. This is reasonable since when the number of one class is larger than the other, the model may focus mainly on the dominant class, thereby neglecting the minority class. When the model starts trying to classify minority classes, it can lead to mistakes in predictions due to their small number.

Considering the nature of diabetes predictions, the recall of positive samples is more important as a high recall rate means that the model is able to capture more real diabetic patients, thereby reducing the risk of missed diagnoses. Therefore, not implementing any resampling method is better. However, focusing only on recall may result in many false positives which is misclassifying healthy people as diabetics. Therefore, we also need to consider precision, which represents the proportion of samples predicted to be positive by the model that are actually positive.

In order to consider both recall and precision, we can use the F1 score as the evaluation metric, which is the harmonic mean of recall and precision. But in a scenario like diabetes prediction, we may be more inclined to give a higher weight to the recall rate, because it is more important to ensure that no diagnosis is missed.

The three resampling methods do not affect the results largely, this is mainly because the imbalance of the original datasets may not be serious, or the model has been able to capture the characteristics of positive and negative samples very well, so resampling does not bring significant improvement. Resampling methods may not always work, and their effectiveness may depend on the characteristics of the data, the complexity of the model, and other factors.

4. Conclusion

Overall, the resampling methods did not bring the expected performance improvements which is mainly because the imbalance of the original datasets may not be serious, or the model has been able to capture the characteristics of positive and negative samples very well, thus resampling does not bring significant improvement. In the future, other methods of handling imbalanced data, such as using different loss functions, model structures, or ensemble methods may be considered. Furthermore, we can also explore the characteristics of the data to see if there are other factors that cause the model performance to not be significantly improved. It is also recommended to verify the effects of these resampling methods on other data sets or practical application scenarios.

References

[1] Salvador García Alberto Fernández. Learning from imbalanced data sets, 2018. Available at `https://link.springer.com/book/10.1007/978-3-319-98074-4`.

[2] Andrea Esuli Alejandro Moreo. Distributional random oversampling for imbalanced text classification, 2016. Available at DOI: https://doi.org/10.1145/2911451.2914722.

[3] Sebastian Bock and Martin Weiß. A proof of local convergence for the adam optimizer, 2019. In: 2019 International Joint Conference on Neural Networks (IJCNN), pages 1-8, doi: https://doi.org/10.1109/IJCNN.2019.8852239.

[4] Chih-Jen Lin. Libsvm data: Diabetes dataset, 2023. Available online at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/diabetes.

[5] Fionn Murtagh. Multilayer perceptrons for classification and regression, 1991. Available at DOI: https://doi.org/10.1016/0925-2312(91)90023-5.

[6] K. W. Bowyer N. V. Chawla. Smote: Synthetic minority over-sampling technique, 2002. Available at DOI: https://doi.org/10.1613/jair.953.

[7] Michel Daoud Yacoub Nidhi Simmons, David E Simmons. Outage performance and novel loss function for an ml-assisted resource allocation: An exact analytical framework, 2023. Available at arXiv: https://arxiv.org/abs/2305.09739.

[8] Dabal Pedamonti. Comparison of non-linear activation functions for deep neural networks on mnist classification task, 2018. Available at arXiv: https://arxiv.org/abs/1804.02763.

[9] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks, 2009. In: WSEAS Transactions on Circuits and Systems, volume 8, number 7, pages 579-588, publisher: World Scientific and Engineering Academy and Society (WSEAS) Stevens Point.

[10] Joseph Prusa, Taghi M. Khoshgoftaar, and Dittman. Using random undersampling to alleviate class imbalance on tweet sentiment data, 2015. Available at DOI: https://doi.org/10.1109/IRI.2015.39.

[11] Gojka Roglic. Who global report on diabetes: A summary, 2016. Available on International Journal of Noncommunicable Diseases.

[12] Xue Ying. An overview of overfitting and its solutions, 2019. Available at DOI: https://doi.org/10.1088/1742-6596/1168/2/022022.