Comparative Analysis of Undersampling Techniques for Credit Card Fraud Detection in Big Data

1st Chunfeng Xia
dept. of Computer Science
University of Nottingham
Nottingham, United Kingdom
psycx1@nottingham.ac.uk

2nd Yixuan Xu
dept. of Computer Science
University of Nottingham
Nottingham, United Kingdom
psyyx6@nottingham.ac.uk

3rd Rong Jian Yee
dept. of Computer Science
University of Nottingham
Nottingham, United Kingdom
psyrjy@nottingham.ac.uk

4th Guanda Zhao
dept. of Computer Science
University of Nottingham
Nottingham, United Kingdom
psygz@nottingham.ac.uk

***Abstract*—Credit card fraud detection (CCFD) is the process of identifying and nullifying purchases that are not performed by the actual owner of the card, and it plays a vital role in mitigating the loses incurred by this criminal activity. Though these systems are in place, billions of dollars are still lost every year, and this number is only projected to keep on rising as credit cards become even more ubiquitous.**

Due to the enormous amount of transactions that are made on a daily basis, there is a need for systems that are capable of quickly and efficiently analysing transactions in real-time to prevent any that are believed to be fraudulent. Thus, we present a Big Data solution to CCFD, and explore 3 undersampling techniques to handle the severe imbalance found in this problem domain. The techniques explored include random undersampling and k-nearest neighbours. We find that the local approach to k-nearest neighbours undersampling offers the best performance, though using it in combination with random undersampling also offers relatively good performance while being significantly faster and thus, more scalable to Big Data.

***Index Terms*—undersampling, credit card fraud detection, imbalanced binary classification**

I. Introduction

Credit cards have become an everyday necessity to the general population, and they play a vital role in supporting society's increased online consumption habits by providing a way to bridge the gap between paychecks, and offering merchants a guaranteed method of payment. The convenience of credit cards however, is also one of its downfalls. Because of their ease of use, it is very easy for criminals to steal credit card details, and use them to make fraudulent purchases—the cost of credit card fraud has reached nearly $28.65 billion in 2019 alone, and it is projected to keep on rising [1].

To mitigate credit card fraud, systems are in place to analyze transactions, and detect any that appear to be fraudulent so that they can be blocked. This however, requires a Big Data solution as the volume of transactions is very large; Visa alone processes 150 million transactions per day.[1] Furthermore, another challenge in credit card fraud detection (CCFD) is the severe class imbalance caused by the minuscule number of fraudulent transactions compared to the legitimate ones. This is problematic because most machine learning algorithms underperform when working with imbalanced datasets [2].

[1]https://usa.visa.com/run-your-business/small-business-tools/retail.html

Thus, this paper explores which undersampling methods are the most effective for CCFD and because it is being performed in the context of Big Data, one of the objectives is also to investigate which algorithms are the most scalable while providing the necessary performance. Using the *PySpark* library, we implement three undersampling algorithms: random undersampling (RUS), and two based on k-nearest neighbours (KNN). To assess the performance of each technique, we implement a gradient boosted decision tree (GBDT) classifier, and train it separately on each dataset undersampled by the previously mentioned techniques. To robustly evaluate and validate the performance of the classifier, we define a set of metrics and perform grid search cross validation. The main contribution of this work is the empirical evidence on which undersampling algorithms work best for CCFD while providing an assessment of which methods are more feasible to implement in the context of Big Data.

II. Background & Literature

Classification of highly imbalanced datasets is an active area of research in predictive modelling, and numerous techniques can be found in the literature in regards to how to handle such data-sets. Thus, the goal of this work is not to review the entire body of literature on imbalanced classification but to implement and analyse multiple Big Data undersampling solutions to CCFD.

A. Credit Card Fraud Detection

There is a substantial amount of research in the literature surrounding CCFD, however, there is a scarcity of it when applied in the context of Big Data. Most studies proposed frameworks to perform CCFD but do not consider the large volume of data associated with it.

Olowookere and Adewale [3] presented a framework using a cost-sensitive meta-learning ensemble. Their framework assigns a higher cost to misclassifying fraudulent transactions to handle the data imbalance instead of using a sampling technique. Their study however, doesn't take into consideration the Big Data nature of CCFD and thus, implement a framework using distributed computing technologies. Furthermore, this approach may not be the best because undersampling not only displays good performance, but the resulting smaller

datasets will lead to lower computational costs when training the models.

Shen et al. [4] empirically compared different types of models for CCFD, namely artificial neural networks (ANN), logistic regression, and decision trees. Their results show that ANN displayed the best performance. ANN however, may not be the most suitable for Big Data because graphical processing units (GPU) are required to considerably reduce the computational time for training the ANN, but scaling these out can be very costly, especially due to the current inability for the supply to meet the market's demand for GPUs.[2]

Awoyemi et al. [5] used a hybrid sampling approach that involved oversampling the minority and undersampling the majority class. They found that this technique significantly improved the performance of all classifiers.

B. Undersampling Techniques

According to a study conducted by Drummond and Holte, undersampling is better than oversampling even though it introduces non-determinism into the learning process [6]. Another reason to consider undersampling techniques over oversampling techniques is because the latter have been shown to introduce a greater risk to overestimating the predictive accuracy of models [7]. Another benefit to using undersampling techniques is that it significantly reduces the amount of data the model has to train on, thus, reducing the computational cost. This is especially important, when dealing with Big Data.

1) Random Undersampling: RUS involves randomly selecting a subset of data points belonging to the majority class to use for training. In the literature, it is often used in combination with bagging by fitting each classifier on its own randomly undersampled dataset [8], [9]. A study conducted by Joseph Prusa [10] showed that RUS significantly improved classification performance of *tweets*.

2) K-Nearest-Neighbours: KNN is a non-parametric algorithm that is commonly used for classification, but it may also be leveraged to perform undersampling. A study conducted by Beckmann et al.[11] showed that the classification results conducted with KNN undersampling on 33 datasets outperformed the results o six other methods. Throughout the paper, KNN will refer to KNN undersampling.

C. Global and Local Approach

Local approach refers to the divide-and-conquer algorithm where the data is split among the workers and the algorithm is applied locally to each partition. Models are trained locally with the data in each partition and then collected as a list of models. When making predictions, every individual model makes a prediction on the testing instance, and the final result is the majority vote of these predictions.

While the global approach means the model is trained on the entire dataset without partitioning. Global approach are expected to yield a better result as the model use a larger dataset during training.

[2]https://www.bloomberg.com/opinion/articles/2021-04-14/nvidia-risks-a-reckoning-as-crypto-miners-deprive-gamers-of-graphics-cards

III. Methodology

A. Random Undersampling

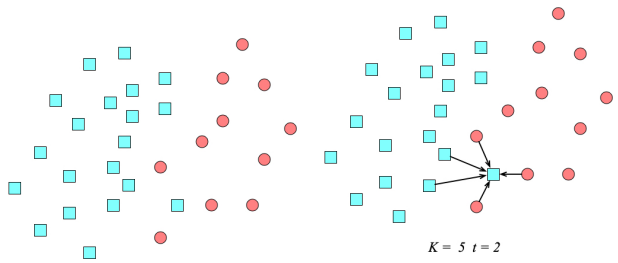
One of the main advantages to RUS is its simplicity and low computational cost compared to KNN. The computational intensity of an algorithm is an extremely important factor to consider in Big Data because of enormous scale of data. Thus, its performance will be used as a baseline to compare KNN to.

RUS may be used to balance the data in any ratio. Split the dataset into majority class and minority class. Randomly delete rows from the majority class to the desired ratio between majority and minority class.

B. K-Nearest-Neighbours

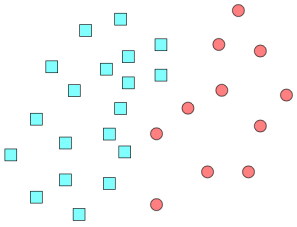
The KNN method used in this paper is modelled after the one proposed by Beckmann et al.[11]. For a certain dataset, the k nearest neighbours of each data point in the majority class are calculated, and then deleting points if t or more of its neighbours are part of the minority class. The parameter t essentially determines the maximum number of minority instances a majority instance can have out of its k-nearest neighbours where $1 \leq t \leq k$. The value of t and k can be changed accordingly. A large k makes it very computational expensive and defeats the basic philosophy of KNN where instances that are close are similar in classes or density, while a low k will be more sensitive to noise [11]. Additionally, the lower t is, the more aggressive it is at removing instances.

Fig. 1 shows an illustration of how KNN is used to undersample a majority class while its pseudocode can be found in the appendix.



(a) Original Dataset

(b) Identification of Nearest Neighbours



(c) Removal of Instance

Fig. 1. Visualisation of KNN undersampling

not only addresses the problem of cross class instances that are similar, it also overcomes problem that other K-Nearest

Neighbour based method has such as Edited Nearest Neighbours proposed by Wilson in 1972[12] which only removes small amount of instances. This method mainly acts in the class overlapping areas which will not remove instances in the majority class with high density that prevents information loss.

1) Global Undersampling Approach: The global undersampling approach for KNN was combined with RUS because of the vast amount of memory required to run KNN. RUS was first performed to obtain a dataset with a ratio of 1:3 minority to majority instances, which was then used to perform global KNN undersampling.

In this approach, each instance in the training dataset is assigned a unique index number, and the majority class instances are extracted from the training dataset. The distance of each majority instance to all other instances is calculated by our user defined function which utilises the `squared distance` function from Spark. The index number of majority instances will then be recorded if t or more neighbours are from the minority class as mentioned in the previous section. After iterating all majority instances, this will leave us with a dataframe containing the unique index number of the instances that will be removed. This dataframe will be used to select the instances from the training set which will return the undersampled dataset.

Algorithm 1: RUS with global KNN undersampling

Input: $RUS\_Dataset$, t, K
Output: $UndersampledDataset$

Extract $MajorityClass$ from $RUS\_Dataset$
for $x \in MajorityClass$ **do**
  $Distance \leftarrow \{x.squaredDistance(y) \mid \vee\ y \in RUS\_Dataset\}$
  $K\_nearest \leftarrow K$ nearest points calculated from $Distance$
  $num\_Minority \leftarrow Minority.count()$ in $K\_nearest$
  if $num\_Minority \geq t$ **then**
    Append index number to $indexDataframe$
  end
end
$UndersampledDataset \leftarrow RUS\_Dataset \notin indexDataframe$
return $UndersampledDataset$

2) Local Undersampling Approach: Initially, our team attempted to implement the local approach for CCFD. That is, by using `mapPartition`, we undersample the majority class training instances of each local partition, and the training of each GBDT model is performed on the undersampled partition in the worker node, thus we collect a list of GBDT models, and the prediction result is decided based on the majority vote of predictions made by the models. In this context, `mapPartition` requires PySpark dataframe to be converted to RDD, and the minority training instances are `broadcast` to each partition as ordinary Python list. Therefore, majority training instances in RDD is required to be converted to Pandas dataframe to be used together with minority training instance. However, it was found during the development phase that Spark cannot be used in the worker node, which means pandas dataframe cannot be converted to PySpark dataframe so as to train the GBDT models from PySpark MLlib. In the light of the above considerations, our team has decided to only perform local undersampling.

As depicted in Fig. 2, in the local undersampling approach, each instance of the dataset is first assigned a unique index, and then the dataset is split into two: one containing only samples from the majority class, and the other containing only samples belonging to the minority class. The majority dataset is then randomly split across all workers using `repartition`, while the entire minority dataset is broadcasted to each of the workers without any modifications, since the minority class is the key for performing KNN undersampling. Following the divide-and-conquer principle, KNN undersampling is then performed on each partition of data. Each partition will return a list containing the unique indexes assigned before partitioning of the undersampled dataset in the partition, then the `mapParitions` function will aggregate them into one single list. The indexes of the undersampled majority instances are then used to select them from the entire training set, these are then joined with the minority-only dataset to form the final undersampled dataset.

Algorithm 2: Local KNN undersampling

Input: $MajorityClass$, t, K, $MinorityClass$
Output: $Index$

Union $MajorityClass$ and $MinorityClass$ into $AllClass$
for $x \in MajorityClass$ **do**
  $Distance \leftarrow \{x.euclideanDistance(y) \mid \vee\ y \in AllClass\}$
  $K\_nearest \leftarrow K$ nearest points calculated from $Distance$
  $num\_Minority \leftarrow Minority.count()$ in $K\_nearest$
  if $num\_Minority \leq t$ **then**
    Append index number to $Index$
  end
end
return $Index$

C. Distance metrics

As described in the previous subsection, KNN sorts the distances of instance pairs to select the nearest point(s) to the minority class instances, it is therefore crucial to have accurate distance metrics.

Due to the fact that PySpark's built-in models and functions are highly optimised, we naturally turn to use existing models to calculate the distance of instance pairs. Unfortunately,

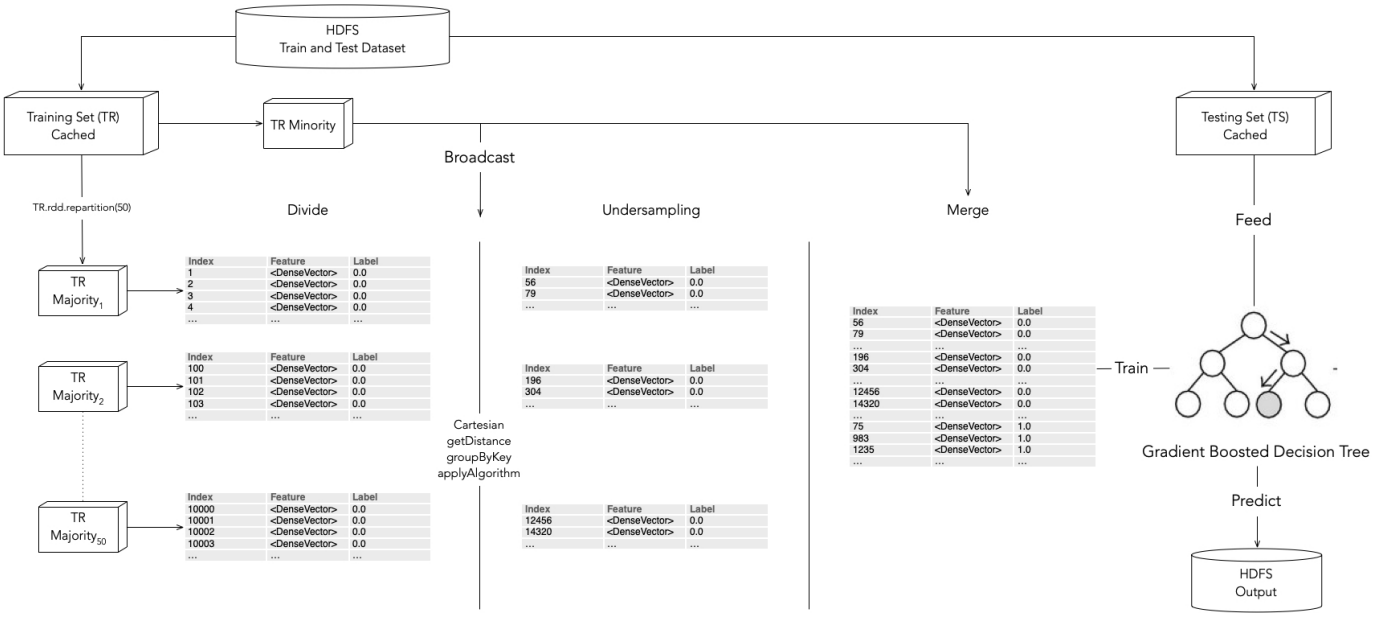


Fig. 2. Local Undersampling Approach

PySpark does not have a built-in KNN model for calculating the actual distance, though there is an alternative called *BucketedRandomProjectionLSH*, where 'LSH' stands for *locality-sensitive hashing*.

LSH works by building up a hash table and mapping all instances from the training data into the table. It follows the principle that if two points in the same feature space are close to each other, the probability that their hashes will have the same value are high[13]. Thus, unlike other use cases of hashes, LSH uses a hashing function that aims to maximise collisions.

One potential problem detected when we examine the LSH model is that, the points with hash collisions will be given the same euclidean distance. For example, as depicted in Fig. 3, instance 235911 from the majority class has 13 nearest points from training data with exactly the same distance. It is therefore confusing for the KNN undersampling algorithm to select the K nearest neighbour(s) if K is less than 13.

	majority_index	all_index	EuclideanDistance
1	235911	208375	0.00006405569493342691
2	235911	203055	0.00006405569493342691
3	235911	207206	0.00006405569493342691
4	235911	210590	0.00006405569493342691
5	235911	205185	0.00006405569493342691
6	235911	210291	0.00006405569493342691
7	235911	209184	0.00006405569493342691
8	235911	204800	0.00006405569493342691
9	235911	211319	0.00006405569493342691
10	235911	212093	0.00006405569493342691
11	235911	210004	0.00006405569493342691
12	235911	209827	0.00006405569493342691
13	235911	206522	0.00006405569493342691

Fig. 3. LSH Output Example

In the light of the above considerations, we have decided to implement our own user defined function for KNN.

D. Gradient Boosted Decision Tree

Gradient boosted decision trees (GBDT) essentially work by sequentially training an ensemble of decision trees, where each tree improves upon its previous trees mistakes [14]. They have proven to be extremely versatile and reliable, showing considerable success in a wide spectrum of tasks [15]. Thus, they were chosen as the classifier to model this problem, and to properly asses the generalization performance of the models trained on the datasets undersampled by the different algorithms, grid search (GS) cross validation (CV) will be performed as explained in the next section.

IV. Experimental Setup

A. Dataset

The dataset was obtained from Kaggle[3] and it contains credit cards transactions made in two days of September 2013 by European cardholders. It contains 492 fraudulent cases out of the 248,807 transactions such that only 0.172% of the transactions are fraudulent, which is evidently severely imbalanced. It only contains numerical features, most of which are the results from a dimensionality reduction technique, except for 'Time' and 'Amount'. 'Time' is the number of seconds elapsed between a transaction and the first transaction, thus it was dropped as we considered it to be unnecessary. 'Amount' is the value of a transaction and it was normalised with a p-norm of 3 alongside the other numeric features. 'Class' is the label where 1 corresponds to fraudulent and 0 otherwise.

[3]https://www.kaggle.com/mlg-ulb/creditcardfraud

The dataset was split into two, with one set being used for training and the other being used as a hold out set for evaluation. The undersampling techniques were only applied on the training dataset to prevent information leakage.

B. Performance Metrics

A meaningful and robust set of metrics is crucial in evaluating the performance of machine learning models. In the case of imbalanced datasets, it is especially important that the metrics chosen are accurately represent the models performance. Thus, the following metrics were measured to compare the performance of the models:

Area under PR Curve (AUPRC) indicates a general measure of performance irrespective of any particular threshold or operating point. The models were optimised on this metric because in the problem of CCFD, the main goal is to correctly identify fraudulent cases (true positives) while minimizing the amount of false negatives (FN). It is important to note that though false positives (FP) may pose inconveniences to customers, preventing fraudulent transactions is still the priority.

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

Recall is the ratio of correctly predicted positive observations to the all observations in actual class.

F1-score is the weighted average of *precision* and *recall*, it is a better measure than accuracy in the case of unbalanced positive and negative class.

C. Validation procedure

Another key component of evaluating the performance of our models was validation to analyse whether they would generalise well to new data and not be overfitted to the training data set.

K-fold cross validation (CV) is a widely used approach for validation because it can measure how well a model generalises, thus, it was used alongside a holdout dataset to measure the performance of models. Because of the severe class imbalance, CV was performed with 5 folds because a higher number would lead to too little fraudulent cases in the testing phase of each fold and to have higher odds of preserving the class distribution across each fold.

D. Hyper-Parameter Tuning

5-fold CV was performed for all possible combinations of hyper-parameters within the ranges specified by us. Because of the scope of this work and due to time constraints, it was not feasible to manually or exhaustively tune the hyper-parameters over a wider ranger of values. The maximum number of iterations was set to 100, and it determines the number number of trees that are grown

Maximum depth of the tree: 3, 4, 5, 6, 7. Used to control over-fitting: lower depths do not allow the model to learn relations very specific to a particular sample while higher depth may lead to over-fitting.

Learning rate: 0.05, 0.1, 0.15. Used to control the performance of tree model. Lower learning rate is computationally expensive as it takes longer to convergence, and higher learning rate will cause overshooting (exceed the minimum of the loss function).

Minimum number of instances each child must have after split: 5, 10, 15. Used to control over-fitting: lower value will cause over-fitting and higher value will cause under-fitting.

E. Baseline

Random undersampling is the baseline method for reducing instances from the majority class. RUS is considered the simplest and naive method for undersampling due to the fact that this approach does not take into account of any information about the data and no heuristics are used when removing instances.

V. Results and Discussion

It is important to note that due to time constraints, the experiments were run on half of the whole dataset. As shown by the results in table I, local KNN performed the best across all metrics.

TABLE I
SCORES FOR EACH UNDERSAMPLING TECHNIQUE

	Metric			
Algorithm	Precision	Recall	F1	AUPRC
RUS	0.073	0.898	0.135	0.616
RUS-KNN	0.115	0.843	0.203	0.683
KNN Local	0.271	0.863	0.412	0.736

When taking account the purpose of CCFD, the aim is to have a high precision and recall. All methods have low precision because only approximately 400 training data for positive cases is too few when considering the complexity of the task and the model.

Despite the low precision, all methods have high recall. Apart from RUS, the results suggest that lower the ratio of majority class over minority class, higher the precision, which because RUS does not remove noisy and similar data between majority and minority class.

The local approach to KNN most likely had the highest performance because it was able to exploit the properties of the whole dataset. RUS-KNN on the other hand only performed training on a limited dataset as KNN was performed on a randomly undersampled dataset.

The experiments were run on a Google Cloud computing instance with a 16-core CPU, 128 gigabytes of random access memory, and 50 gigabytes of storage memory. In terms of the scalability of the algorithms, RUS is obviously the most scalable because it only involves randomly selecting a subset of the majority class—its run time was merely 3 seconds. RUS-KNN had a run time of roughly 1 minute, whereas the local KNN undersampling algorithm had a run time of roughly 2 hours. From the performance results obtained by the metrics and the differences in run-time, it is clear that local KNN should be the method of choice if enough time and resources

are available. RUS-KNN however, may be appropriate if the amount of time and resources is limited.

VI. Conclusion

From the experimental results, it can be concluded that KNN provides the best performance out of the other techniques. RUS-KNN however, also performed relatively well, and considering how much more scalable this method is relative to KNN, it should definitely be considered and trialed in other Big Data contexts. Furthermore, RUS also performed relatively well even though it is very simple to implement and how fast it is compared to the other algorithms. Thus, the practical results obtained in this study may be followed when dealing with similar Big Data issues of severe class imbalance.

Future research could involve applying our experimental framework on other datasets to further assess the performance of the techniques explored in this paper and determine whether they generalise to other tasks. Furthermore, the combination of the local KNN undersampling approach could be combined with a local approach to obtaining the classification models.

References

[1] "Card fraud losses reach $28.65 billion." Accessed on Apr. 24, 2021. (Dec. 2020), [Online]. Available: https://nilsonreport.com/mention/1313/1link/.

[2] A. Dal Pozzolo, O. Caelen, Y.-A. Le Borgne, S. Waterschoot, and G. Bontempi, "Learned lessons in credit card fraud detection from a practitioner perspective," *Expert Systems with Applications*, vol. 41, no. 10, pp. 4915–4928, 2014, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.02.026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741741400089X.

[3] T. A. Olowookere and O. S. Adewale, "A framework for detecting credit card fraud with cost-sensitive meta-learning ensemble approach," *Scientific African*, vol. 8, e00464, 2020, ISSN: 2468-2276. DOI: https://doi.org/10.1016/j.sciaf.2020.e00464. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2468227620302027.

[4] A. Shen, R. Tong, and Y. Deng, "Application of classification models on credit card fraud detection," in *2007 International Conference on Service Systems and Service Management*, 2007, pp. 1–4. DOI: 10.1109/ICSSSM.2007.4280163.

[5] J. O. Awoyemi, A. O. Adetunmbi, and S. A. Oluwadare, "Credit card fraud detection using machine learning techniques: A comparative analysis," in *2017 International Conference on Computing Networking and Informatics (ICCNI)*, 2017, pp. 1–9. DOI: 10.1109/ICCNI.2017.8123782.

[6] C. Drummond and R. Holte, "C4.5, class imbalance, and cost sensitivity: Why under-sampling beats over-sampling," *Proceedings of the ICML'03 Workshop on Learning from Imbalanced Datasets*, Jan. 2003.

[7] R. Blagus and L. Lusa, "Joint use of over- and under-sampling techniques and cross-validation for the development and assessment of prediction models," *BMC Bioinformatics*, vol. 16, Nov. 2015. DOI: 10.1186/s12859-015-0784-9.

[8] X. Shi, G. Xu, F. Shen, and J. Zhao, "Solving the data imbalance problem of p300 detection via random under-sampling bagging svms," in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–5. DOI: 10.1109/IJCNN.2015.7280834.

[9] B. W. Yap, K. A. Rani, H. A. A. Rahman, S. Fong, Z. Khairudin, and N. N. Abdullah, "An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets," in *Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013)*, T. Herawan, M. M. Deris, and J. Abawajy, Eds., Singapore: Springer Singapore, 2014, pp. 13–22, ISBN: 978-981-4585-18-7.

[10] J. Prusa, T. M. Khoshgoftaar, D. J. Dittman, and A. Napolitano, "Using random undersampling to alleviate class imbalance on tweet sentiment data," in *2015 IEEE International Conference on Information Reuse and Integration*, 2015, pp. 197–202. DOI: 10.1109/IRI.2015.39.

[11] M. Beckmann, N. Ebecken, and B. Lima, "A knn undersampling approach for data balancing," *Journal of Intelligent Learning Systems and Applications*, vol. 7, pp. 104–116, Nov. 2015. DOI: 10.4236/jilsa.2015.74010.

[12] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-2, no. 3, pp. 408–421, 1972. DOI: 10.1109/TSMC.1972.4309137.

[13] D. Cai, "A revisit of hashing algorithms for approximate nearest neighbor search," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2019. DOI: 10.1109/TKDE.2019.2953897.

[14] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001, ISSN: 00905364. [Online]. Available: http://www.jstor.org/stable/2699986.

[15] C. Zhang, C. Liu, X. Zhang, and G. Almpanidis, "An up-to-date comparison of state-of-the-art classification algorithms," *Expert Systems with Applications*, vol. 82, pp. 128–150, 2017, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2017.04.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417417302397.