Synthetic Embedding-based Data Generation Methods for Student Performance

Dom Huh
George Mason University
dhuh4@gmu.edu

Huzefa Rangwala
George Mason University
rangwala@gmu.edu

ABSTRACT

In this work, we introduce a framework for synthetic data generation for academic performance prediction formulations. A common problem in these academic performance prediction dataset is that outcomes/grades are not distributed evenly, leading to class imbalance. This poses a challenge for predictive machine learning algorithms to learn important characteristics at the edges of the target class distribution. We present a general framework for synthetic embedding-based data generation (SEDG), a search-based approach to generate new synthetic samples using embeddings to correct the detriment effects of class imbalances. We compare the SEDG framework to traditional synthetic data generation methods and relate the framework in relation to deep generative models. In our results, we find SEDG to outperform the traditional re-sampling methods for deep neural networks and perform competitively for common machine learning classifiers at the student performance task in several standard performance metrics.

1. INTRODUCTION

In the educational domain, academic performance prediction approaches are at the crux of degree planning and early warning systems. Classifiers that seek to predict the performance of a student given input features can be biased because these datasets are often skewed centrally to the mean as a relative grading scale is often employed. The limited number of examples belonging to outlier students, specifically those demonstrating success and failure (needing interventions/help), are not sufficiently represented. As these predictive models are useful for identifying at-risk students or helping students plan their degree and career pathways, this imbalance directly conflicts with this purpose. We refer to these examples belonging to the minority classes. In contrast, there exists an overwhelming number of students with average grade scores. We refer to these examples belonging to the majority classes. This evident disproportion results in classifiers failing to understand and learn how to classify within the minority classes, which was supported in [13]. This issue we are describing is known in the machine learning community as class imbalances. Class imbalances are recognized when working with a dataset that exhibits a notable disparity in the number of examples amongst different classes. This abnormality often disproportionately harm the classifier's performance on the minority classes. As [17] discusses, these datasets describe a true nature of the problem, which correlates heavily with rare events, small sample size, low class separability, and/or existence of within-class subconcepts. For student performance datasets, we find all to be the case, and even so, we argue gaining insight into the outlier students is significant for the educators as it may point to indicators of failure and success. In this paper, we want to alleviate the setbacks caused by class imbalance.

As described in [17], there exists three common approaches to alleviate the effects of class imbalances: data modification, algorithm modifications and learning modifications. Each approach modifies different aspects of the learning system to counter the issue of class imbalances. In this paper, we will focus on a class of data modification approach called sampling methods. We consider two classes of sampling methods: oversampling and under-sampling. Oversampling methods append samples to the pre-existing training set, whereas under-sampling methods remove samples from the training set. With all sampling approaches, they aim to balance the number of samples, given some criteria, in the training set. We suggest taking a more systematic process. We argue sampling methods must consider two important components. First, the criteria that is targeted must be defined. For the student performance dataset, the criteria may be the number of samples in the class. Second, the method of creating the balance should be well-defined, meaning the use of this method should improve upon the criteria. Examples of naive approaches are randomly re-sampling or randomly removing samples. In [18], a promising oversampling approach has been generating new synthetic samples from pre-existing samples. This approach is known as Synthetic Data Generation (SDG). The most popular SDG algorithm is Synthetic Minority Over-sampling Technique (SMOTE) [2], which generates new samples by modifying a pre-existing sample by adding the scaled (0-1) difference between feature vectors of its nearest neighbors, with the addition of noise. There have been notable improvements to this method, such as DataBoost-IM [7]and ADASYN [8] changing the sampling criteria to be based on predictive error. However, these methods have yet to demonstrate sufficient performance improvement of minority classes, but rather on the

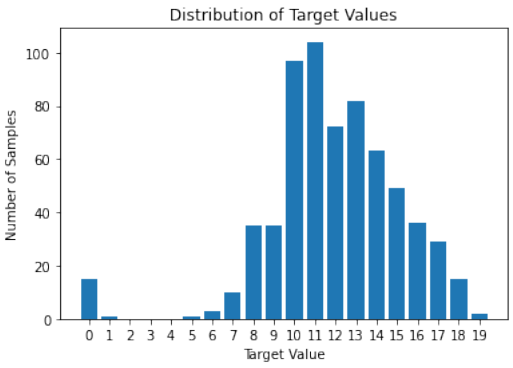

Figure 1: Target Value Distribution: From the student performance dataset, the class imbalance in the distribution skews centrally, indicating a disproportionate number of examples between the central classes and the outer classes.

entire dataset. Thus, in our approach, we systematically break down the process of SDG, to target the improvement of minority classes.

Deep learning methods have also exhibited advances in robustness, efficacy, and allowing end-to-end training for SDG methods as seen in [10, 15, 19]. In specific, deep generative models (DGMs), such as generative adversarial models, have been introduced as methods of SDG. [10, 15]. In our approach, we leverage the advances in deep learning, using embeddings learned by deep models.

In this paper, we introduce an novel framework for synthetic embedding-based data generation methods, called SEDG, to target the effects of class imbalances on academic performance prediction tasks. In Section 2, the academic performance dataset, the performance metrics, and the classifiers used will be described. Further details of SEDG framework and deep generative models will be discussed in Section 3. In Section 4, the results are presented and evaluated.

2. PRELIMINARIES

In this section, we will discuss the details of the Student Performance Dataset from [5], the performance metric, and the classifiers used to test the SEDG framework introduced in this paper. The training methods and implementation decisions will also be described.

2.1 Student Performance Dataset

The dataset provided by [5], can be used to predict student performance, or the final grade, on a scale from 0 to 20. The data was collected from two Portuguese secondary level schools, and has 649 samples of 32 features. In Figure 1, we show the class imbalance in the dataset, skewed centrally as mentioned in Section 1. In Figure 2, we show the distribution of features in the dataset, where some features, such as parent's cohabitation status, extra paid classes, and extra educational school support, also demonstrated imbalance. Further information on the meaning of these features to can be found in [5]. We will not opt to use the binary or five-level classification found in other works on this dataset, but will adhere to the the 20 classes preset. While regression

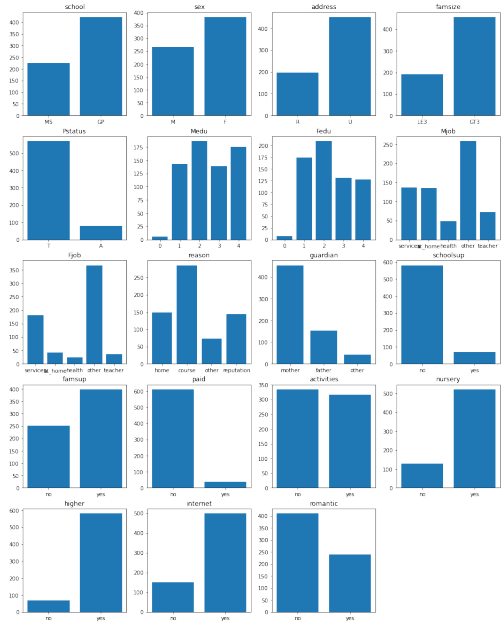

Figure 2: Feature Distribution: From the student performance dataset, we can see an imbalance in some of the distribution of feature values.

methods can be used, we stuck to classification methods to better distinguish improvements in individual target values.

2.2 Performance Metrics

It is important to consider an appropriate performance metric for datasets with class imbalance, as [12] found there to exists an imbalance in the performance itself: a relatively high predictive accuracy for the majority classes and significantly lower in the minority classes. Thus, if we consider traditional classification accuracy to evaluate the model's efficiency, we would obtain an overly optimistic belief in the classifier. Thus, we consider using Area under Curve (AUC) on the Receiver Operating Characteristic (ROC) curve. The ROC graph is used to show the trade-off between true positive and false positive error, and is represented in the ROC space. A mapping to the ROC space requires the true positive rate (TPR), or more commonly known as the sensitivity, and the false positive rate (FPR), or more commonly known as fall-out. For multi-class settings [11], we can use an one-vs-rest paradigm, and either obtain the AUC scores at the class level, or aggregate with or without class weighing to obtain a micro-AUC or macro-AUC score respectively. When we view at a class level, we call this AUC-class score.

2.3 Classifiers

We will use three traditional machine learning classifiers (support vector machines, random forest, gradient-boosted decision trees/XGBoost) and an neural network architecture which we will denote as NNModel.

Support vector machines (SVM) [9] are optimal-margin classifiers that seek to find the hyper-plane that maximize the geometric margins between binary classes. In a multi-class setting, we can proceed by using an one-vs-rest approach, where the maximum score from n SVMs, where each SVM

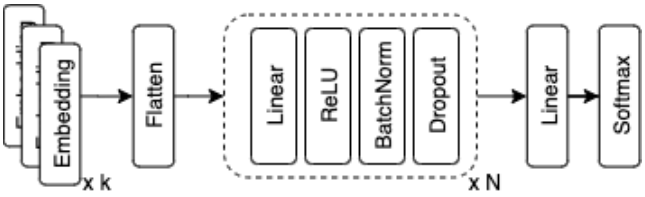

Figure 3: NNModel Block Diagram: In the diagram, the input is composed of k features. The input data is first mapped into embeddings, then passed into the processing component, which has N LRND blocks, denoted as the dotted block. To obtain the final prediction for a multi-class classification, the output is transformed using a softmax activation.

is trained on masking all but one class to formulate a binary setting, is selected, or an one-vs-one approach, where similar to one-vs rest, we formulate a binary setting, but with pairs of classes instead. Decision trees [14] are tree-based classifier that create binary splits based on a condition on the features that maximizes information gain, a metric of weighted entropy. Random forest [3] is a bootstrap aggregation, or bagging, algorithm using decision trees, where n learners learn on bootstrapped subsets of the dataset and are also limited to a subset of features it can split on. Thus, inference of random forest is run on majority vote of the learners. XGBoost [4] is a gradient boosting algorithm using decision trees, where learners are introduced to additively correct errors of trained learners. Gradient boosting refers to correcting the errors through gradient learning when adding new learners [6]. For our experiments, the neural network architecture we call NNModel is a multi-layer perceptron model that is sequentially made up of N LRND blocks, seen in Figure 3. The LRND block is composed of a linear layer, rectified linear activation, batch normalization, and followed by a dropout layer. This model is used for classification, and its parameters are optimized using gradient learning on cross entropy loss, seen in Equation 1, where p is the true, target function and q is the hypothesis function, and an Adam optimizer scheduled to reduce and anneal the learning rate on plateaus by an order magnitude.

$$L(p, q) = -\frac{1}{N} \sum_{\forall x} p(x) \log(q(x)), |x| = N \tag{1}$$

3. SYNTHETIC EMBEDDING-BASED DATA GENERATION

We will describe the SEDG approach step by step, dividing the discussion into 4 chronological parts: sample selection, feature selection, feature modification and synthetic sample usage. In each step, we offer various design considerations that we have tested. For future works, we encourage new variations and designs of this general framework. We will then write on deep generative models in the context of the SEDG framework.

3.1 Sample Selection

We must first consider how to select the samples from the original dataset to base the synthetic samples on. We can set how many samples k we want to select to be fixed constant or stochastically chosen within a defined range.

3.1.1 Random Sample Selection

Naively, we can randomly select k samples from the entire dataset D, all with equal probability. Once the samples are selected, we place them into a sample pool, S, that will be used to generate the new synthetic samples. This approach will be referred to as random sample selection.

3.1.2 Partition-based Sample Selection

We can define a sample selection approach that partitions the dataset D and select a member M_i of the partition $P = \{M_0, M_1...\}$ to sample from. The member selection can occur at a probability p or be deterministic. The member will be stochastically sampled from, with or without replacement. Consider we want k samples from the datasets, we can implement a partition P, where the equivalence relation is defined by the samples' associated class. We set each member of the partition $M_i \in P$ to be associated to a selection probability p_i proportional to their cardinality $p_i \propto |M_i|$. The selection probability represent how likely the member will be selected. We choose to define the surjective mapping from the partition $P = \{M_0, M_1...M_n\}$ to the probability distribution $\bar{p} = \{p_0, p_1...p_m\}$ with a hashmap for simplicity. A special case would be to deterministically select the member M_i based on the cardinality, where $|M_i| \geq |M_j|$ for all $j \in [0, n]$. Once a member M_i is selected, we sample m examples from the set, where $m \leq k$. The value of m, similar to the value of k, is a hyper-parameter. The m samples are stochastically selected from M by performing random sample selection, and we add this to our sample pool S. We repeat this process until $|S| = k$. This approach will be referred to as partition-based sample selection. We note that this approach is dependent solely on the dataset, and remains entirely independent on the classifier.

3.1.3 Performance-based Sample Selection

We formulate a sample selection approach to select samples based on the classifier's performance, similar to concept of boosting. Specifically, we can associate higher selection likelihoods for samples the learner has the largest margin for error or uncertainty. In this context, we can consider classification accuracy to quantify error and uncertainty. Thus, we can place the highest probability of selection to the samples that were misclassified, and rank the correctly classified samples based on the classifier's certainty of the prediction. So, given a classifier C and an loss function $L(C, s)$, where $s \in S$ is a sample belonging to the training set S, we let the selection probability p_i for x_i to be greater than the selection probability p_j for x_j if and only if $L(C, x_i) > L(C, x_j)$. The loss function returns a scalar value that represents the correlation between the target and prediction values. This approach will be referred to as performance-based sample selection. We note that this approach, in contrast to partition-based approach, is dependent on the classifier, thus also requires trained model for its operation. Consequently, the performance-based sample selection method is more computationally costly.

3.1.4 Partition-Performance Sample Selection

We now can formulate a selection method that incorporates from both partition and performance sample selection approaches. In this paper, we use the class-AUC score to do so. We first partition as described above and use the class-AUC score as our uncertainty metric to develop a selection probability distribution to select the member to sample from. Thus, we implement a class partition to first select a member to sample from, and we incorporate the AUC score from the classifier to develop the probability associated to member selection. We call this the partition-performance sample selection approach.

3.2 Feature Selection

At a more granular level, we can think of every sample $S_i \in \bar{S}$ from the dataset D, where $\bar{S} \subseteq D$ to be made up of features. For example, a sample $S_i = \{f_1^i, f_2^i, f_3^i ..., f_n^i\}$ can be defined to be a set of features where $|S_i| = n$. We define equality as $S_i = S_j$ if and only if $f_k^i = f_k^j$, $\forall k \in \mathbb{Z}$. After sample selection, we can modify the selected samples to generate new samples. Thus, given some mapping ϕ from S_i to S_i' where $\phi : \{f_1^i, f_2^i, f_3^i ...\} \rightarrow \{f_1^{i'}, f_2^{i'}, f_3^{i'} ...\}$ would provide us new samples S_i', where $S_i' \neq S_i$. But in order to modify these features, we must consider how to select which features to modify efficiently as to appropriately increase variance in the dataset without creating class overlaps. But, to modify the samples, we must first select the features of the sample that we wish to adjust. We will discuss three approaches to select features, which we will call random, imbalance-based, and importance-based feature selection approach. We can label the imbalance-based and importance-based feature selection methods together and classify them as weighted feature selection.

3.2.1 *Random Feature Selection*

The random selection approach, similar to random sample selection, stochastically sample a subset of feature $\bar{f}^i \subseteq S_i = \{f_0^i, f_1^i, f_2^i ... f_k^i\}$, where $|\bar{f}^i| \leq |S_i|$, with all features with equal probability. The number of feature selected $k = |\bar{f}|$ can be fixed or randomly chosen per sample.

3.2.2 *Imbalance-based Feature Selection*

We can weigh the feature selection based on a metric based in feature value imbalances, as seen in Figure 2. We introduce a metric that quantifies this imbalance by using the following method, which differs from past works [17]. We first create an nested set $c = \{c_0, c_1 ...\}$ where each c_i is a set of counts that represent the number of examples that contains each unique feature value for feature f_i. For clarity, for sample S_i, $|c| = |S_i|$. Then, for each $c_i \in c$, we cluster on a 1-dimensional space using a clustering method. For example, if we use the k-means algorithm, we can obtain two centroids to obtain the ratio r_i to be set to the difference between the coordinate of the two centroids and the maximum count in c_i. In the end, we obtain a vector of ratios $R = \{r_0, r_1 ...\}$, which we normalize to treat as a probability distribution to select subsets of features. This approach is largely motivated to place higher emphasis on more imbalanced features, and thus attempting to balance all features' value representation.

3.2.3 *Importance-based Feature Selection*

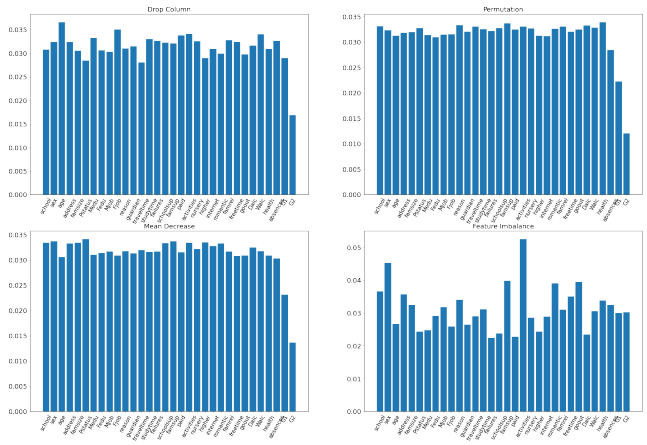

Figure 4: Selection distribution from weighted feature selection methods for classifiers.

The feature importance approach creates a probability distribution $\bar{p}$ based on feature importance, which will be used to select the features. Feature importance provides insight into the relationship between the predictive system and the data, and how much significance each feature in the data may have on the overall system's performance at the given task at hand. We will consider three approaches to calculate and obtain the feature importance: mean decrease impurity importance, permutation importance, drop-column importance.

Mean decrease, or gini impurity, importance methods rely on the use of tree-based classifiers, and is calculated by aggregating the gini decreases of each feature. This is criteria used to determine the splits at every tree. However, [16] has shown gini impurity importance approach is biased given the scale of measurement or number of categories of the features. Additionally, this approach can only be used for tree-based classifiers, thus will only be consider such models. Permutation importance methods, on the other hand, calculate the feature importance by evaluating the decrease in performance of a trained model on a test set that is shuffled on at the single feature value of interest. The shuffling process can be repeated to test multiple permutations of the feature values. Thus, permutation importance only requires the model to be trained once, but also needs to use the testing dataset. Also, this approach can be used for any models. Similarly, drop-column importance methods obtain the feature importance using the difference between the baseline performance of a model, which would trained on the entire dataset, and the performance of the model that has been trained on a limited dataset with a single feature value dropped. Thus, drop-column importance can also be used for any model. However, we see that drop-column importance is very computationally costly, proportionally to the number of features in the dataset.

3.3 Feature Modification

We now consider how to modify the selected features optimally. We define optimality as maximizing the variance between the synthetic samples from original samples while minimizing the class overlap in the new dataset. Our objec-

tive is to maximize the improvements in the minority classes, since that is the deficient area of performance. To modify the features, we can naively inject noise to the continuous features and replace discrete features with sampling, but we wish to formulate a more targeted approach. In this paper, we will focus on embedding-based modifications, a search-based method where we will leverage learned embeddings to offer insights in how to optimally modify the features. For future works, we encourage different methodologies to feature modification.

Embeddings are mappings from the raw feature domain to a domain that can be more useful and understandable for some classifier to perform the task at hand. In other words, we can think of embeddings as optimized pre-processing transformation. We consider two methods of embedding generation in Section 3.3.1 and Section 3.3.2: weights from transfer learning (WTL) and latent representation from auto-encoder (LRA). There are many other embedding generation approaches [1] that have been successfully demonstrated in field of representation learning.

We now formulate a method for feature modification using embeddings. Recall a sample S can be thought of as a set of features $\bar{f}_s$. These features can either be discrete and continuous. We will consider both cases, and discuss how we can handle each case separately. Assume a set of features $\bar{f}_d \subseteq S$ that was selected for modification is homogeneously discrete, for each feature $f_{d,i}$ in $\bar{f}_d$, then there exists a set of possible feature values $f_{d,i}$ can take on $y_i = \{y_{i,0}, y_{i,1}, ...y_{i,k}\}$. We can sample a subset c_i of y_i, where $|c_i| \leq |y_i|$ and if $f_{d,i} = y_{i,m}$ then $y_{i,m} \notin c_i$. If the search space for unique feature values y_i is small enough, we can try all possible values for $f_{d,i}$, however if it is large, then we could sample a subset as mentioned. Given the embedding mapping $\phi_i : y_i \rightarrow E_i$, we calculate the similarity score between $\phi_i(f_{d,i})$ and $\phi_i(y_{i,j})$ for all $y_{i,j} \in c_i$, and replace the feature value with the one with the highest similarity score. We repeat this process for every features in $\bar{f}_d$. In other form, given a embedding mapping $\phi : S \rightarrow E$, where the input is the entire sample, we will again follow the same procedure, however we must compare $\phi(S)$ and $\phi(S')$ where $S = \{f_0, f_1...f_i, ...\}$ and $S' = \{f_0, f_1...f'_i, ...\}$ where $f'_i \in c_i$. When we repeat this process for each feature in $\bar{f}_d$, we can choose to replace S to the updated S' for the proceeding feature modifications, or choose not to replace S. To place higher emphasis on lower class overlaps, we can place a soft constraint on the similarity score using the mean similarity score from most similar, or randomly selected, samples to S from different classes. To place higher emphasis on class variance, we can set a threshold value for the similarity score, or set a soft constraint on the similarity score using the mean similarity score from different samples from the same class as S. Now we assume a set of features $\bar{f}_c \subseteq S$ that was selected for modification is homogeneously continuous. For each element $f_{c,i}$ in $\bar{f}_c$, we define a range with a fixed step $y_i = \{y_{i,0}, y_{i,1}, ...y_{i,k}\}$ where $f_{d,i} \in (y_{i,0}, y_{i,k})$ and $y_{i,0} < y_{i,k}$. Then, we proceed similarly to the discrete feature case for feature modification.

3.3.1 Weights from Transfer Learning

The WTL approach consists of learning the embedding mapping ϕ through training an model F on some relevant task. For example, we choose the task to be classification of student performance. We note that both ϕ and F are parameterized functions by their own independent weights θ_ϕ and θ_F. Thus, inference on the model can be seen in Equation 2, where $\hat{y}$ is the prediction, and x is the input data.

$$\hat{E} = \phi_\theta(x)\hat{y} = F_{\theta_F}(\phi_\theta(x)) \tag{2}$$

Following the specification from the example above, a classifier F and the embedding mapping ϕ can be optimized using the same loss function L, defined in Equation 1. With gradient learning, the weight updates can be seen in Equation 3.

$$\begin{aligned} \theta_\phi &= \theta_\phi - \nabla_{\theta_\phi} L(y, \hat{y}) \\ \theta_F &= \theta_F - \nabla_{\theta_F} L_F(y, \hat{y}) \end{aligned} \tag{3}$$

3.3.2 Latent Representation from Auto-encoder

Auto-encoders are parameterized functions learn to compress and decompress the input data using unsupervised learning. The architecture thus has a bottleneck structure. Let the set of layers $l = \{l_0, l_1...l_n\}$ be the layer that make up the autoencoder, and the cardinality $|l_i|$ for $i \in [0, n]$ be the number of parameters in layer l_i. Given $0 \leq a < c \leq n$ and $b \in (a, c)$, there exists an unique l_b such that $|l_b| < |l_a|$ and $|l_b| < |l_c|$, where we will refer to l_b as the bottleneck layer. We define $\phi = \{l_0, l_1...l_b\}$ and $D = \{l_{b+1}, l_{b+2}...l_n\}$. The aim of the autoencoder is to accurately build a reconstruction of the input given this decompression. Inference on auto-encoders can be seen in Equation 4, where $\hat{x}$ is the prediction, and x is the input data.

$$\begin{aligned} \hat{E} &= \phi(x) \\ \hat{x} &= D(\hat{E}) \end{aligned} \tag{4}$$

Auto-encoders are optimized given the reconstruction loss L, and the target is the input data x. Typically, the loss function L will compare the input x to the predicted value $\hat{x}$ directly, using a function like mean-square error or cross entropy. If gradient learning is used, the weight updates can be seen in Equation 5.

$$\begin{aligned} \theta_\phi &= \theta_\phi - \nabla_{\theta_\phi} L(x, \hat{x}) \\ \theta_D &= \theta_D - \nabla_{\theta_D} L(x, \hat{x}) \end{aligned} \tag{5}$$

Once the auto-encoder is optimized, we can export the compressing subset of layers ϕ as the embedding mapping. We call this procedure LRA.

We also consider variational auto-encoders (VAE), which follows the same inference and optimization methods, however, we represent the bottleneck layer l_b as a probability distribution, often a Gaussian $N(\mu, \sigma)$. Thus, the input of layer l_{b+1} will be samples from the distribution from l_b. The loss function used is the empirical lower bound (ELBO) loss, which combines the reconstruction loss normally used in auto-encoders and the Kullback–Leibler divergence loss

term $KL(P_\phi(z|x), P_D(z))$, where ϕ is a mapping that uses the layers $\{l_0, l_1, ...l_b\}$, D is a mapping that uses the layers $\{l_{b+1}, l_{b+2}, ...l_n\}$ and z is the latent space representation, or the output of the ϕ. Thus, z will be considered to be the embedding of the sample.

3.4 Synthetic Sample Usage

Once we obtain the synthetic samples, we now must decide how the samples will be utilized. We evaluate two considerations: cold or warm start and iterative or non-iterative.

Initially, the model learns on the training set D_{train}. Then, the set of synthetic samples $D_{s,t}$ is appended to the training set D_{train}. We can choose to either re-initialize the weights of the model, which we will call cold-start, or keep the weights from the previous training cycle, which we will call warm-start. By doing cold starts, we remove the bias from the previous dataset, but lose what was learned from that dataset. We consider whether to repeat the process of creating a new $D_{s,t+1}$ from the newly trained model, making the process iterative, or end the training cycle entirely, making the process non-iterative. If the process is iterative, we can choose to dropout and replace the previous $D_{s,t}$ partially or wholly with $D_{s,t+1}$, or append $D_{s,t+1}$ to the current $D_{s,t}$ at each step using a finite queue system. Another consideration is whether to re-sample the distributions used for sample and feature selection.

3.5 Deep Generative Models

In this section, we will discuss deep generative models (DGM) in context with the SEDG framework.

Generative models can model the conditional probability $P(x|y)$, the joint probability $P(x, y)$ or the prior $P(x)$. A DGM is expressed using a deep neural network parameterized by learnable weights θ. Equivalently, DGM are mappings $f_\theta : y \to x$, $f_\theta : x \to x$, or $f_\theta : x, y \to x$. In context of SEDG, DGM can be viewed as an end to end approach to generating synthetic samples, bypassing the need for Section 3.2, and 3.3. For our purposes, the objective is to learn how to generate new samples following the definition of optimality discussed in Section 3.3. We can set the input of the DGM to either be random noise, or samples selected using methods from Section 3.1. A potential issue is the possibility of the generative model learning to map the input to itself without any variance, especially for traditional auto-encoders and given the nature of the optimization. This paper does not propose a proven solution to this issue. In fact, we address this issue by simply preventing over-fitting with early stopping with heuristics and holding out a subset of the training dataset, discussed in further detail in Section 4. Now, we will discuss two main approaches to training DGMs: unsupervised and adversarial training.

3.5.1 Unsupervised Training

Unsupervised training refers to learning the prior $P(x)$, thus in more practical terms, optimizing the model using only the input x. An example of unsupervised training is training auto-encoders, where we treat the input as the output as well. Instead of exporting the embedding function, as seen in Section 3.3.2, we treat the auto-encoder as the generator model $f_\theta : x \to \hat{x}$, where $\hat{x}$ is the synthetic sample. In this paper, we will solely evaluate DGMs using auto-encoders and VAEs.

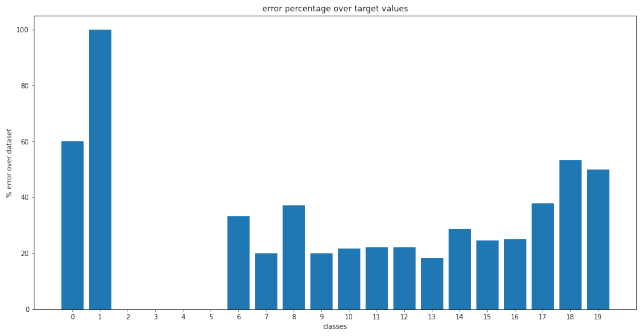

Figure 5: Percent error distribution on the testing set over all target classes averaged over classifiers from Section 2.3

3.5.2 Adversarial Training

Adversarial training adds onto unsupervised learning with the discriminator model, which learns to discern real or fake data. At a high level, the generator model attempts to trick the discriminator model by learning how to generate samples that are realistic. We will consider the traditional approach and the conditional approach.

The traditional approach is to train the generator and discriminator models separately using unsupervised and supervised learning respectively. The training for the generator model will follow the procedure discussed in Section 3.3.2, however we aggregate the discriminator's inaccuracy and the reconstruction loss, scaled by α and $1-\alpha$ respectively, where $|\alpha| \leq 1$, to obtain our generative model's loss. The training of the discriminator will optimize the classification loss between real and fake data using the dataset and the generator model. The conditional approach follows most of what is stated in the traditional approach, however we aim to have both the generative and discriminative models conditioned on the class label, y. Similar to practices discussed in [19], we can append y to the input paired with an embedding layer, and is trained accordingly.

4. RESULTS

In this section, we will evaluate the methods discussed in Section 3 using the student performance dataset and classifiers described in Section 2. We will compare our results with other balancing methods from past works, such as random oversampling, random under-sampling, SMOTE, Tomek Links, extended nearest neighbors, and combinations methods of these approaches. We see that if the classifiers are trained on the dataset normally, optimizing the cross entropy loss, we obtain performance seen in Figure 5, which follows our hypothesis of disproportion performance between minority and majority classes. We seek to mitigate these effects of class imbalances with the SEDG framework we have proposed.

We now will discuss our results from our experiments using the SEDG and DGM methods mentioned in Section 3 and using the NNModel and the traditional models mentioned in Section 2.3 as the classifier. For all of the exper-

Model	% improvement
OvR SVM	8.4615
OvO SVM	3.0769
Random Forest	13.846
XGBoost	4.2307

Table 1: Performance improvement when using trained NN-Model's embedding as data pre-processing to following classifiers. In the table above, OvR SVM represents one-vs-rest SVM and OvO SVM represents one-vs-one SVM.

iments below, we set the number of synthetic samples that will be generated to 100 samples. All recorded improvements are on the testing set, which encompasses 40% of the dataset. We define % improvement PI in Equation 6, where $M(y_{syn}, \hat{y}_{syn})$ is the score from some performance metric M from the dataset with the synthetic samples, and $M(y, \hat{y})$ is the score from the performance metric M from the original dataset.

$$PI = M(y_{syn}, \hat{y}_{syn}) - M(y, \hat{y}) \tag{6}$$

We consider using the NNModel defined in Section 2.3 as our classifier, and we will test our embedding-based SDG methods and DGMs on the student performance dataset. In Figure 6, we find that the SEDG method outperforms all traditional sampling methods for all performance metrics when using NNModel as the classifier. Even given 50 independent trials to run, many of the classic sampling method perform poorly, some even harming the performance. Most notably, the class-AUC score demonstrate a significant positive difference in targeting the minority classes when using SEDG methods.

Now we consider the traditional classifiers seen in Section 2.3, and how SEDG methods affect their performances. First, we show that when we transfer the embeddings from the deep models, to act as data pre-processing mappings to these learners. We find their performance increases noticeably, as seen in Table 1. This supports and allows us to proceed to use embedding-based SDG methods without worries of the embeddings being incompatible to these classifiers.

In Figure 7, we find that the SEDG method, while it does not excel in accuracy and macro-AUC score % improvements for traditional machine learning classifiers, it is able to target the minority classes more effectively in class-AUC score than all other methods. Additionally, compared to the other sampling methods, the results show when considering all the performance metric, SEDG and DGMs performs the best as other approaches that may excel in one metric often fail to replicate the same success in other metrics.

Here, we discuss design considerations made in Section 3 from what was demonstrated in these trials. We found partition-performance based sample selection demonstrates highest improvements in classification accuracy over other sample selection methods. However, based on the macro-AUC and class AUC scores, there doesn't seem to be a clear sample selection method that outperforms the others, all performing comparatively targeting the minority classes than the traditional sampling methods.

We also remark that DGMs demonstrates very minimal improvements in predictive accuracy and macro-AUC score when using the NNModel classifier. We assume this to be related to the issue of over-fitting, which we accommodated for by employing early stopping with basic heuristics: if the majority of training data ($\geq 50\%$) share $\geq 80\%$ of the features to the target sample, then we end training. We suggest for future works to handle this issue of over-fitting in the context of SDG more efficiently.

We further confirm that weighted feature selection is more effective than random feature selection in all performance metrics. The feature imbalance approach performs better in all metrics, with permutation importance and drop-column importance, the two feature importance methods, performing similarly in predictive accuracy. We find that for both accuracy and macro-AUC score, using the embedding matrix from the NNModel outperform the other methods, with VAE being a close second in terms of macro-AUC score. For class AUC scores, we find that VAEs and embedding matrix from NNModel improve the scores of the minority classes the most.

4.1 Understanding Student Performance

In this section, we investigate the synthetic samples more in-depth for each class to improve our understanding for student performance by showing how likely certain features are allowed to change without much significant changes to the data distribution, similar to feature importance. However, with synthetic samples, we can also see how we can change these features, thus seeing possible feature value candidates.

In Figure 8, we use a VAE as DGM to create 100 synthetic samples for each class to show how likely each feature in each class is subject to change, and what feature values is reasonable. For example, we can see for low performing students, the likelihood of changing absences is low, with greater number of absences being a possible candidates. We also see that some features, such as travel time, school supplies, and famrel, are highly susceptible to change, inferring to their lack of distinction between possible candidates. Thus, we can leverage this method for feature understanding at a class level. Its usage can inform educators further on different performing students.

5. CONCLUSION

In this paper, we proposed a general framework for embedding-based SDG and investigated DGMs for academic performance tasks. We tested the SEDG approach and DGMs against standard re-sampling methods, and found greater improvements in our proposed approaches in all performance metrics when we use our NNModel as the classifier and a more comprehensive improvement when we use traditional machine learning classifiers. We also introduced a technique for greater interpretibilty and insight into the dataset using a DGM by looking at the synthetic samples generated and seeing how likely each feature is modified and to what values it can take on for each class.

6. ACKNOWLEDGEMENT

We acknowledge NSF #1757064 Grant for support this work.

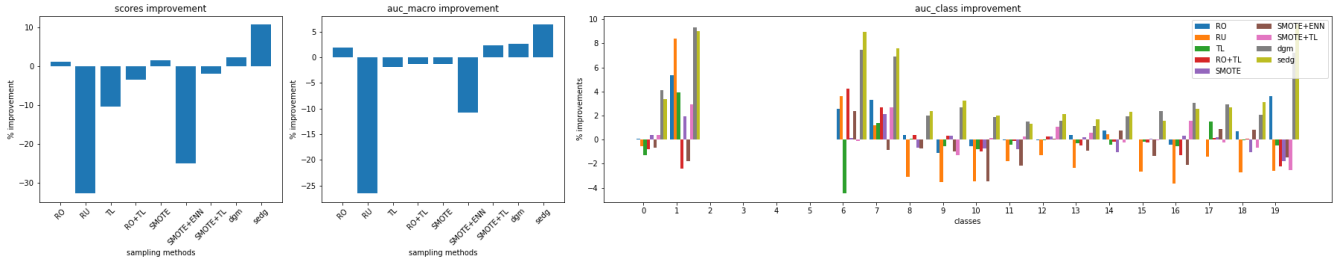

Figure 6: Performance improvement comparing traditional sampling methods against DGMs and SEDG using NNModel as classifier. The graphs above use the following notation: score refers to classification accuracy, auc_macro refers to macro-AUC score, and auc_class refers to AUC score per class.

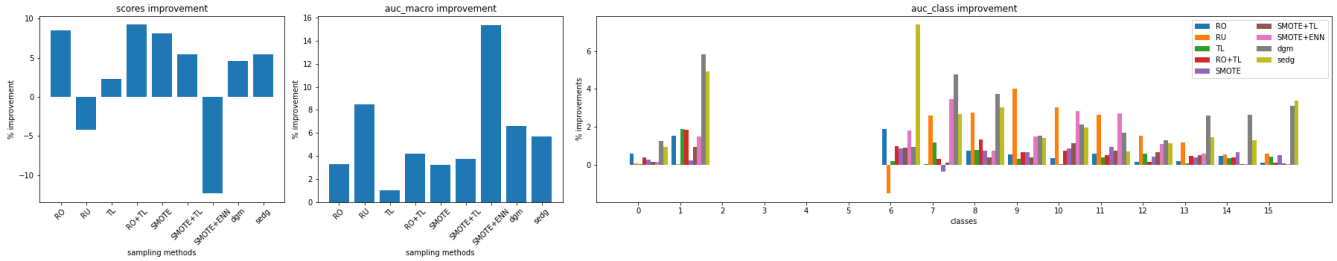

Figure 7: Performance improvement comparing traditional sampling methods against DGMs and SEDG with traditional machine learning classifiers. The graphs above use the same notation as Figure 6.

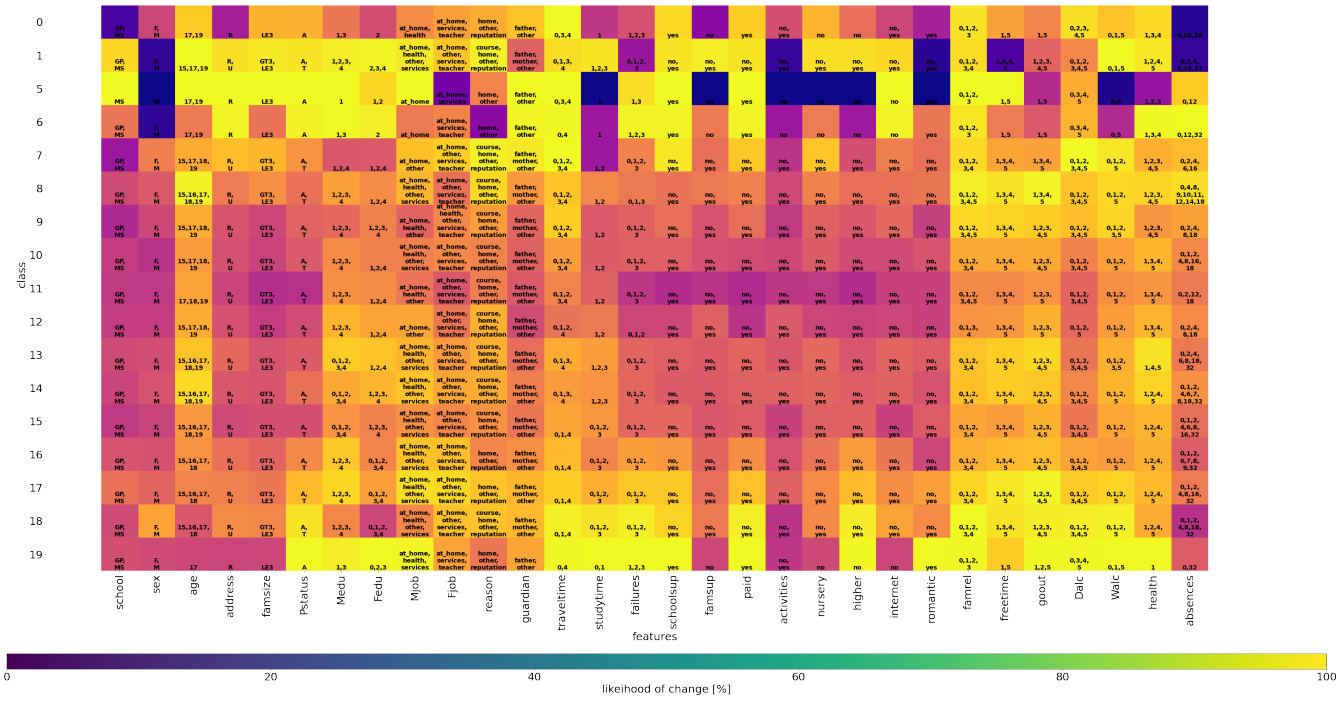

Figure 8: Likelihood of feature modification and feature value candidates using generative VAE from 100 synthetic samples per class.

7. REFERENCES

[1] Y. Bengio, A. C. Courville, and P. Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[2] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011.

[3] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, Oct. 2001.

[4] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[5] P. Cortez and A. Silva. Using data mining to predict secondary school student performance. *EUROSIS*, 01 2008.

[6] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[7] H. Guo and H. L. Viktor. Learning from imbalanced data sets with boosting and data generation: The databoost-im approach. *SIGKDD Explor. Newsl.*, 6(1):30–39, June 2004.

[8] Haibo He, Yang Bai, E. A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328, 2008.

[9] M. A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, July 1998.

[10] S. S. Mullick, S. Datta, and S. Das. Generative adversarial minority oversampling. *CoRR*, abs/1903.09730, 2019.

[11] H. Narasimhan, W. Pan, P. Kar, P. Protopapas, and H. G. Ramaswamy. Optimizing the multiclass f-measure via biconcave programming. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1101–1106, 2016.

[12] F. Provost and T. Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, KDD'97, page 43–48. AAAI Press, 1997.

[13] F. J. Provost and G. M. Weiss. Learning when training data are costly: The effect of class distribution on tree induction. *CoRR*, abs/1106.4557, 2011.

[14] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, Mar. 1986.

[15] P. Shamsolmoali, M. Zareapoor, L. Shen, A. H. Sadka, and J. Yang. Imbalanced data learning by minority class augmentation using capsule adversarial networks, 2020.

[16] C. Strobl, A. Boulesteix, A. Zeileis, and T. Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8:25 – 25, 2006.

[17] Y. Sun, A. Wong, and M. S. Kamel. Classification of imbalanced data: a review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23, 11 2011.

[18] S. Susan and A. Kumar. The balancing trick: Optimized sampling of imbalanced datasets—a brief survey of the recent state of the art. *Engineering Reports*, n/a(n/a):e12298.

[19] C. G. Turhan and H. S. Bilge. Recent trends in deep generative models: a review. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, pages 574–579, 2018.