The Intersection of Classification, Imbalance, and Interpretability

Andrea Boskovic *
Department of Mathematics and Statistics, Amherst College

December 12, 2020

Abstract

Many real world classification problems involve imbalanced datasets, where the number of instances in one class vastly outweighs the number of instances in the other classes. Machine learning classifiers, however, struggle to classify imbalanced data correctly. In order to improve upon imbalanced data classification methods, we first need to improve the interpretability of classifiers by understandig why they make certain choices. Model interpretability has often been described as a tradeoff between performance and understanding, but this tradeoff is even steeper with classification of imbalanced datasets. This report examines the interpretability of imbalanced data classification by first providing background for classification, class imbalance, and interpretability and then studying the intersection of the three areas.

Keywords: imbalanced data, classification, machine learning, interpretability

*I gratefully acknowledge Professor Nicholas Horton for his feedback on this report.

1 Introduction

A primary goal in statistics involves predicting a response variable based on data. One example of this goal is in binary classification. In this setting, the response variable, often referred to as the target variable or class, takes on two possible values, and classification techniques aim to classify this response based on data in a way that maximizes accuracy. Classification techniques involve subsetting data into a training dataset, which is used to train the model with data instances for which the response is known, and a testing dataset, which contains instances where the response is unknown and is used to test the trained model.

We are particularly interested in classifying imbalanced data, a common data type in many real world datasets, where the number of instances in each class varies significantly. If we are examining a dataset containing information about incidence of a disease, we may be interested in predicting whether a patient is sick or healthy. This dataset will likely contain more instances of healthy patients than sick patients, making the dataset highly imbalanced. Canonical classification methods often perform poorly on imbalanced datasets, however. In the case of the health dataset, a canonical classification method would likely predict almost exclusively healthy patients in order to optimize for accuracy.

Issues of model interpretability arise in many machine learning models, even those tailored to balanced datasets. As in most machine learning methods, the reasons that many classifiers and neural networks make any given decision is difficult for humans to understand. These are known as black box models because they cannot be understood based on their parameters. This report discusses how classification of imbalanced datasets relates to interpretable machine learning.

Section 2 gives an overview of interpretable classification techniques, and Section 3 gives an example of class imbalance and reviews the meaning of the class imbalance problem. We follow with a synopsis of the general principles of model interpretability in Section 4. A discussion of interpretability in the context of imbalanced data in Section 5 ties the report together, and it concludes with a brief summary of conclusions in Section 6.

2 Classification Techniques

Classifiers are tools used for classification tasks, where the goal is to predict a target, or class, based on data. Many different classifiers exist, and each is appropriate in different types of problems. Classification tasks can be binary or multi-class. In binary tasks, the class can only take on the values 0 or 1, as when classifying whether a credit card transaction is fraudulent (1) or not (0), for instance. In multi-class problems, there are more than two values the class can take on. One example of such a problem is classifying news articles from five sources to the correct source.

A variety of techniques are used to evaluate the performance of classifiers. One common technique is the confusion matrix, which compares the amount of instances in the actual dataset to the predicted dataset for each class, as shown in Figure 1.

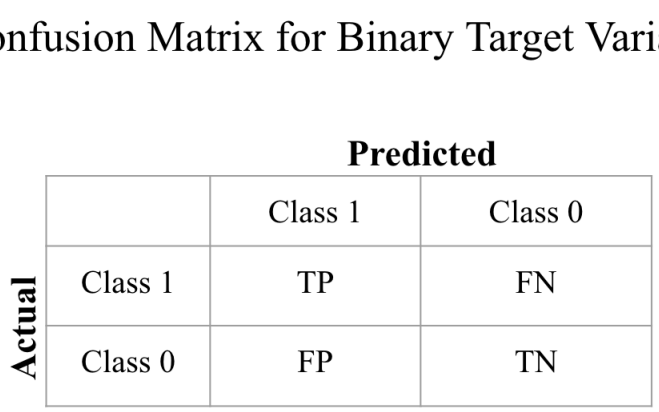

Figure 1: An example of a confusion matrix for a binary classification problem. Here, TP represents True Positive, FP represents False Positive, FN represents False Negative, and TN represents True Negative.

One of the most common metrics for evaluating classifier performance is accuracy, which measures how often the classifier correctly labels instances (Johnson & Khoshgoftaar 2019). It is mathematically defined as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}}.$$

Despite the popularity of this metric, it can be misleading when used to evaluate model performance on an imbalanced dataset. This problem is discussed in detail in Section 3.

In R, the `caret` and `yardstick` packages are particularly useful with classification tasks. The `caret` package, or Classification and REgression Training package, streamlines the model training process, making code easier to read. The `yardstick` package, on the other hand, provides useful metrics to evaluate models used in classification with tidy principles. R users can install `caret` and `yardstick` from CRAN with `install.packages('caret')` and `install.packages('yardstick')`, respectively.

We now outline several classifiers: logistic regression (Section 2.1), Gaussian Naive Bayes (Section 2.2), and decision trees (Section 2.3). Although this section highlights the theoretical underpinnings and implementations of these methods, Section 4 discusses the interpretability of these methods.

2.1 Logistic Regression

Logistic regression, one of the simplest binary classifiers, provides an alternative to linear regression for binary response variables. This method involves modeling the probability of the response with a function that outputs 0 and 1 for all instances in the dataset (Greenwell 2020). The logistic function provides us with this functionality:

$$p\left(X\right) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}. \tag{1}$$

Here, $p(X)$ refers to the probability that the positive class occurs. If we apply the logit transformation to $p(X)$ in Equation (1), we get the logit form:

$$g\left(X\right) = \log\left[\frac{p\left(X\right)}{1-p\left(X\right)}\right] = \beta_0 + \beta_1 X, \tag{2}$$

where β_0 and β_1 represent model coefficients.

Both Equation (1) and Equation (2), which are in simple logistic regression form, can be generalized to multiple logistic regression, where there are n coefficients: β_0, β_1, ..., β_n. In this case, $p(X)$ becomes

$$p\left(X\right) = \frac{e^{\beta_0+\beta_1 X+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X+\cdots+\beta_p X_p}}.$$

The logistic regression model, however, solely outputs predicted class probabilities, which cannot be directly used for classification. By setting a cutoff value such that values above the cutoff are classified as one class and values below the cutoff are classified as the other class, however, logistic regression becomes a viable binary classification technique. This cutoff is almost always set to 0.5, but it may be appropriate to change the value in some cases.

Suppose we have a dataset called `dataset` with a target variable called `class`. If we want to create a 70-30 train-test split, fit a logistic regression using all the predictors in the dataset, and predict the class labels, we can use `caret` as follows:

```
train <- createDataPartition(dataset$class, p = 0.7)
training <- dataset[train, ]
testing <- dataset[-train, ]

mod_fit <- train(class ~ .,
  data = training,
  method = "glm",
  family = "binomial"
)

predict(mod_fit, newdata = testing)
```

2.2 Gaussian Naive Bayes

The Naive Bayes classifier is a probabilistic classifier that applies Bayes Theorem to make predictions, assuming independence and equal importance of predictors (Gayathri & Sumathi 2016). For some class y and a predictor $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ of length n, Bayes Theorem states that

$$P(y|\mathbf{x}) = \frac{P(y)P(\mathbf{x}|y)}{P(\mathbf{x})}.$$

Since the classifier assumes independence among predictors, we have that

$$P(y|\mathbf{x}) = \frac{P(y)\Pi_{i=1}^{n} P(x_i|y)}{P(\mathbf{x})}. \tag{3}$$

Note that the denominator remains constant, so we can define a classifier that predicts y as

$$\hat{y} = \arg\max_{y} P(y)\Pi_{i=1}^{n} P(x_i|y).$$

The Gaussian Naive Bayes classifier follows the same framework as general Naive Bayes classifiers, but they take on the additional assumption that $\mathbf{x}$, a continuous variable, is distributed according to a Gaussian distribution (Gayathri & Sumathi 2016). The classifier segments $\mathbf{x}$ by class and calculates the mean μ_j and variance σ_j^2 of $\mathbf{x}$ for each class level y_j. Using these calculations of mean and variance for each class, the classifier can then estimate probabilities with the equation for a normal distribution for some observed value v (Gayathri & Sumathi 2016). This yields the following mathematical formulation:

$$P(\mathbf{x} = v|y) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(v-\mu_j)^2}{2\sigma_j^2}}.$$

In general, Naive Bayes classifiers require small amounts of training data, making them advantageous for small datasets (Gayathri & Sumathi 2016). The equal importance and independence assumptions do not hold for all datasets, making this technique not applicable to some scenarios.

Again, suppose we are again dealing with a dataset called `dataset` with a target variable called `class`. If we want to create a 70-30 train-test split, fit a Gaussian Naive Bayes classifier using all the predictors in the dataset, and predict the class labels. We can use `caret` package again. Notice that the only lines that changed from our implementation for the logistic regression were in the call to `train()`, where we changed our classification method to Naive Bayes, which is Gaussian by default.

```
train <- createDataPartition(dataset$class, p = 0.7)
training <- dataset[train, ]
```

```
testing <- dataset[-train, ]


mod_fit <- train(class ~ .,
  data = training,
  method = "nb"
)


predict(mod_fit, newdata = testing)
```

2.3 Decision Tree

The foundation of machine learning classifiers ranging from random forests to gradient boosting machines, decision trees act as a fundamental structure used to create more complex techniques. Although many methods of constructing decision trees exist, the most common is the classification and regression tree (CART) (Breiman et al. 2001). The first step in this process is to partition the data into homogenous subgroups, or nodes, which are then formed recursively with binary partitions until a stopping criterion is satisfied (Greenwell 2020). The process of iterating recursively to partition the data is known as binary recursive partitioning (Chakure 2020). Following the partitioning of the data, the model predicts the output based on the class with majority representation (Greenwell 2020). The resulting tree has a structure similar to that of Figure 2

There are a variety of parameters that can control the performance of decision trees. It is particularly important to pay close attention to the depth of the decision tree, which controls its complexity (Greenwell 2020). Making the tree too deep can lead to a consequence called overfitting, while making it too shallow can lead to poor model performance.

Overfitting essentially makes models ungeneralizable. Suppose some classifier separates two classes by simply drawing a line between the classes. Although this would classify the training data perfectly, it would not generalize to other data, such as the testing data. This problem, where we capture accidental properties in the data, is known as overfitting. To avoid this, we can use early stopping, a method that restricts the growth of a tree (Greenwell 2020). We can restrict growth by restricting tree depth or the number of instances in a

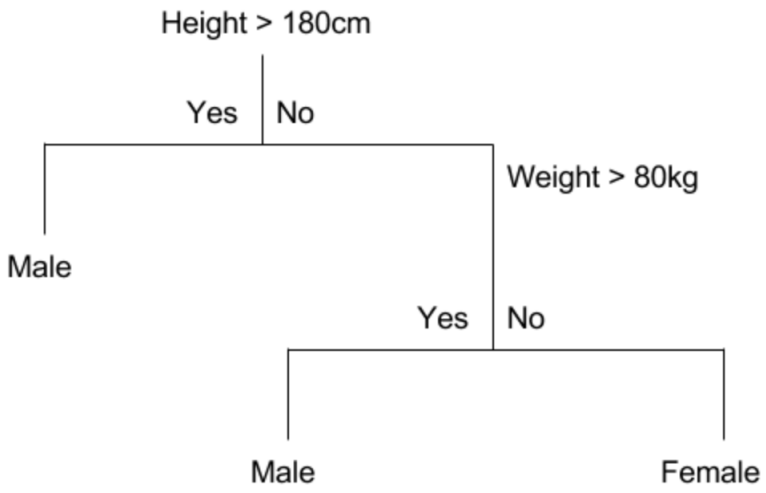

Figure 2: An example of a decision tree to classify males and females based on height and weight (Chakure, 2020).

terminal node, decreasing the variance in predictions (Greenwell 2020).

Using the same experiment described in Section 2.1 and Section 2.2, we can implement a decision tree in R using the `caret` package. Here, we specify that our method is `rpart`, indicating we are using a decision tree classifier.

```
train <- createDataPartition(dataset$class, p = 0.7)
training <- dataset[train, ]
testing <- dataset[-train, ]

mod_fit <- train(class ~ .,
  data = training,
  method = "rpart"
)

predict(mod_fit, newdata = testing)
```

3 Class Imbalance

Binary target variables often have unequal representation of two classes. In imbalanced data, common in health data and fraud data, it is common to have ten times as many instances of the negative class. Consider a dataset containing information about cancer incidence: we expect many more negative instances than positive instances, creating an inherently imbalanced dataset where the minority class, or positive class, contains instances where a patient has cancer and the majority class, or negative class, contains instances where a patient does not have cancer. In such datasets, we are often more interested in predicting the positive class correctly than predicting the negative class correctly. With the cancer dataset, predicting the positive class accurately means that we are detecting cancer effectively so that patients can get appropriate care quickly.

Despite the class imbalance problem, many machine learning techniques are not attuned to this issue. In order to mitigate class imbalance and improve model performance, we must change metrics for evaluating models to get an accurate sense of model performance, implement sampling methods to balance class distributions, and modify existing algorithm structures to better capture minority class behavior. By doing so, we more accurately predict the minority class and maximize model performance.

3.1 Motivating Example

The Kaggle Credit Cards dataset displays extreme class imbalance and will be used throughout this report. In addition to twenty-eight unknown features obtained through Principal Component Analysis (PCA), a common dimensionality reduction technique, the dataset also contains information about time of a credit card transaction, the amount of the transaction, and whether or not the transaction was fraudulent, which we refer to as the class.

Out of all of the credit card transactions in the dataset, only 0.17% were fraudulent, and the remaining 99.83% were not fraudulent. Figure 3 shows the imbalance in the dataset. Traditional machine learning techniques, however, often incorrectly classify positively labeled data because they are built to optimize for accuracy. In this example, standard techniques will tend to classify instances of fraudulent credit card transactions as non-fraudulent because there are many more instances of non-fraudulent transactions.

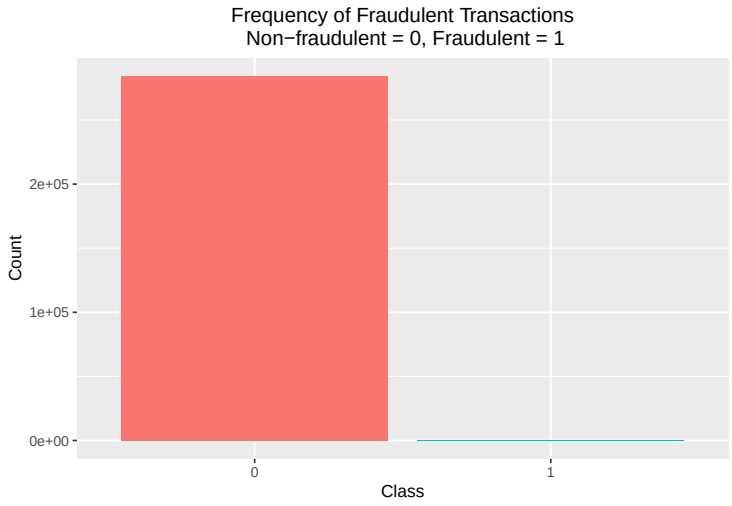

Figure 3: The Kaggle Credit Card Dataset is highly imblanaced, as nearly all of the transactions are not fraudulent and only a small proportion are fraudulent.

Assuming the same class distribution in the training and testing datasets, by predicting all non-fraudulent transactions, classifiers can achieve 99.83% accuracy, i.e., the proportion of non-fraudulent credit card transactions in the dataset.

3.2 Metrics for Model Evaluation

The high accuracy illustrated in the example would not be representative of true model performance. In that case, the model would incorrectly classify all of the majority class, which, in cases like credit card fraud detection or health data, are the instances for which we want to maximize accuracy. To combat this issue, we outline several metrics for model evaluation of imbalanced data that have been developed and explain them in relation to the previously mentioned example. Table 1 illustrates each unbiased metric on the Kaggle Credit Cards dataset using logistic regression classification.

1. **Precision:** Proportion of all instances labeled positive that are truly positive, given by
$$\frac{\text{TP}}{\text{TP} + \text{FP}}.$$
2. **Recall:** True positive rate, or the proportion of of correctly labeled positive instances.

This can be calculated the following way:

$$\frac{\text{TP}}{\text{TP} + \text{FN}}.$$

3. **Balanced Accuracy:** This is an accuracy more sensitive to the minority class, as it considers True Positive Rate (TPR) and True Negative Rate (TNR) in its calculation:

$$\frac{1}{2} \cdot (\text{TPR} + \text{TNR}).$$

4. **G-Mean:** A performance metric that combines TPR and TNR as follows:

$$\sqrt{\text{TPR} \cdot \text{TNR}}.$$

5. **F-Measure:** Often referred to as the F_1 score, this metric is a combination of precision and recall with harmonic mean:

$$\frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}.$$

Metric	**Value**
Precision	89.7%
Recall	64.1%
Balanced Accuracy	82.1%
G-Mean	80.1%
F-Measure	74.8%

Table 1: Summary of performance of logistic regression on the Credit Cards dataset with respect to the metrics outlined above. The code for these results is available in Section 7.1.

Another popular metric for model performance with imbalanced data is the receiver operating characteristics (ROC) curve, which plots TPR over FPR to depict the tradeoff between classifying the minority class correctly and classifying the majority class incorrectly (Johnson & Khoshgoftaar 2019). A summary of the ROC curve is the area under the curve (AUC). In classification problems, we aim to maximize AUC because a higher AUC means the model better distinguishes between the minority and majority classes. By implementing

performance metrics that are less biased towards the negative class, we are better able to evaluate model performance.

Classifier	Accuracy	Balanced Accuracy
Logistic Regression	0.999	0.821
Gaussian Naive Bayes	0.978	0.534
Decision Tree	0.999	0.937

Table 2: A comparison of accuracy and balanced accuracy metrics of three classifiers of interest: Logistic Regression, Gaussian Naive Bayes, and Decision Tree. The code for this analysis is available in Section 7.2.

We can also evaluate the performance of the classifiers discussed in Section 2. We are interested in further comparing accuracy and balanced accuracy of logistic regression, Gaussian Naive Bayes, and decision trees. Based on Table 2, we see that all three classifiers perform extremely well in terms of accuracy, illustrating the bias of the accuracy metric toward the majority class. There is much more variety in the balanced accuracy metric, with Gaussian Naive Bayes performing worst with balanced accuracy only 53.4% and decision tree performing best with balanced accuracy 93.7%. This difference in performance with respect to balanced accuracy is likely a product of the differences in the classifiers' decision-making process.

4 Model Interpretability

Machine learning models are often referred to as black box models due to their complex internal workings. When machine learning models perform well, we often do not question the theoretical underpinnings of the model to understand why the model performed well and made those specific predictions. Model interpretability involves filling in the gap between understanding what prediction a model made and why it made that prediction. In some scenarios, the tradeoff between interpretability and accuracy is futile because we only care about finding a model that performs best. However, when dealing with problems that are not well-studied or problems that can impact people, businesses, or institutions, we must pay close attention to interpretability (Molnar 2020). Understanding interpretability can help us improve machine learning methods, such as those used with imbalanced datasets.

4.1 Background

Although we can learn the structure of algorithms, this does not reflect model interpretability. Rather, it reflects model transparency: how the algorithm learns a model based on the data (Molnar 2020). In other words, we can understand the process by which neural networks produce an output, but we will not know how they predict each individual outcome for specific models and datasets (Molnar 2020). Advancements in interpretability allow us to answer questions including:

- What are the most important customer attributes driving behavior?
- How are these attributes related to the behavior output?
- Do multiple attributes interact to drive different behavior among customers?
- Why do we expect a customer to make a particular decision?
- Are the decisions we are making based on predicted results fair and reliable? (Molnar 2020)

The approaches to these questions can be described as either global, where we attempt to understand the algorithm as a whole, or local, where we explore individual data points or small groups of data points to extract insights about the model's predictions (Hall 2019). Global and local interpretations are just one of several ways of categorizing techniques for model interpretability.

In addition to determining whether an interpretability technique is global or local, we can also determine if it is model-specific and model-agnostic. Whereas model-agnostic techniques can be applied to multiple machine learning algorithms, model-specific techniques are unique to one type or class of algorithm (Hall 2019).

4.2 Global and Local Interpretations

Global and local interpretability involve the scope of of the interpretation method. As the names of these types of interpretations illustrate, global interpretability has a wider scope, while local interpretability has a narrower scope. Still, there are nuances to both types of interpretability.

Global interpretability methods help us understand the overall relationship between the inputs and the response variable (Greenwell 2020). These methods center around understanding how the model makes predictions, both on a holistic and modular level (Greenwell 2020). On a holistic level, we examine how the trained model makes predictions based on a holistic view of its features and learned components (Molnar 2020). On a modular level, however, we are interested in how parts of the model make predictions (Molnar 2020). We often discuss global modular interpretability when the dataset of interest has hundreds of features. In these cases, it becomes too difficult to keep track of weights and use the information gathered to make new predictions. To mitigate this issue, we examine only one weight at a time, for instance, and gather information about part of the model.

Local interpretability, on the other hand, involves either single predictions or a group of predictions (Molnar 2020). Even though certain predictors may influence the overall predictive accuracy of the model, they are not necessarily most important for a given observation or group of observations (Greenwell 2020). Two of the primary approaches for local interpretability are local interpretable model-agnostic explanations (LIME) and Shapley Values (Greenwell 2020). They both aim to explain which variables are most influential in predicting the response variable for a given set of observations (Greenwell 2020). Evidently, we can use global and local interpretations to examine interpretability of data with different scopes.

4.3 Model-specific and Model-agnostic Interpretations

Although model-agnostic interpretations can be applied to any algorithm and are seemingly more versatile, they often are not as accurate as model-specific interpretations. Model-agnostic methods often rely on surrogate models or other approximations, which degrade their accuracy (Hall 2019). These methods often analyze feature input-output pairs to make measurements (Molnar 2020). Model-specific interpretation techniques, however, interpret models directly, likely leading to more accurate measurements (Hall 2019).

Table 3: The results of fitting a logistic regression model on the Kaggle Credit Cards dataset. The table shows the estimated weights, odd ratios, and standard errors of the weights for features used in the logistic model (Molnar, 2020).

	Weight	Odds ratio	Std. Error
Intercept	-7.53	0.00	0.08
Feature 1	0.27	1.31	0.03
Feature 2	0.30	1.34	0.05
Feature 3	-0.70	0.50	0.03
Amount of Transaction (dollars)	0.00	1.00	0.00

4.4 Interpretable Classifiers

The classifiers discussed in Section 2 are all interpretable by design. We will now explore what makes each of the classifiers interpretable. The code for any examples in this section of the report is available in Section 7.3.

4.4.1 Logistic Regression

Unlike in linear regression, the weights do not affect the output linearly in logistic regression. Recall that we need to apply a logit transformation by taking the log of the odds to obtain the linear form of the function, shown in Equation (2).

To derive the interpretation of logistic regression, we can observe the effect of increasing one of the feature values by one in terms of the ratio of the new odds to the old odds:

$$\frac{odds_{x_j+1}}{odds} = \frac{exp\left(\beta_0 + \beta_1 x_1 + \ldots + \beta_j(x_j+1) + \ldots + \beta_p x_p\right)}{exp\left(\beta_0 + \beta_1 x_1 + \ldots + \beta_j x_j + \ldots + \beta_p x_p\right)},$$

shown in Molnar (2020). Exponentiating the top and bottom, we can apply this exponentiation rule: $\frac{exp(a)}{exp(b)} = exp(a-b)$. We then have

$$\frac{odds_{x_j+1}}{odds} = exp\left(\beta_j(x_j+1) - \beta_j x_j\right) = exp\left(\beta_j\right).$$

In other words, increasing a feature value by one unit changes the increases the odds ratio by a factor of $\exp(\beta_j)$ (Molnar 2020). In other words, increasing x_j by one unit increases the log odds ratio by the value of its corresponding weight (Molnar 2020).

Table 3 shows weights, odds ratios, and standard errors of predictors used in the logistic regression model. In this example, we would interpret Feature 2 as follows: An increase in Feature 2 increases the odds credit card fraud by a factor of 1.34, given that all other features are held constant. Although we do not know what Feature 2 represents this case due to the privacy of the company that released the dataset, we can see how this technique would be useful if we had an intuitive understanding of the features' meaning.

4.4.2 Gaussian Naive Bayes

As discussed in Section 2.2, the Gaussian Naive Bayes classifier uses Bayes' theorem to make predictions. Recall that the equation used to predict the class, given by Equation (3), can by rewritten as

$$P(y|\mathbf{x}) = \frac{1}{Z} P(y) \Pi_{i=1}^{n} P(x_i|y) \tag{4}$$

because $P(y|\mathbf{x}) \propto P(y)\Pi_{i=1}^{n} P(x_i|y)$, where Z is a scaling parameter (Molnar 2020).

This classifier is interpretable due to its independence assumption (Molnar 2020). Using the formulation in Equation (4), we can find how much a feature contributes toward a class prediction by calculating $P(y|\mathbf{x})$ (Molnar 2020).

4.4.3 Decision Tree

Interpreting decision trees involves starting at the root node, or highest node on the tree, and traveling down the tree sequentially until reaching the leaf. Note that each edge connection, represented by a line connecting a parent node to its child node, is read as "and" (Molnar 2020). Often, however, decision tree interpretability is viewed through the lens of feature importance or tree decomposition.

We can determine feature importance of a variable in a decision tree as follows:

1. Identify all the splits in which the feature of interest is used.
2. Measure how much it reduced some performance metric, such as the variance or Gini index, using the parent node as a baseline.
3. Sum over all the importances and scale to 100 so that each importance can be interpreted as a part of the total model performance (Molnar 2020).

In tree decomposition, on the other hand, the goal is to decompose the decision path into one component per feature. Starting at the root node, we aim to track a decision through the tree and explain a given prediction by the contributions of each decision node (Molnar 2020). By following the path of the instance we want to predict, we can add or subtract the mean of the outcome of the training data at each split in the tree. Mathematically, this is formulated as follows:

$$\hat{f}(x) = \bar{y} + \sum_{d=1}^{D} \text{split.contrib(d,x)} = \bar{y} + \sum_{j=1}^{p} \text{feat.contrib(j,x)}.$$

Over D splits in the tree, we add the contributions of each of the p features to determine how much a given feature has contributed to the prediction.

4.5 Model-Agnostic Approaches

Although simply using interpretable models, like decision trees, provides guaranteed interpretability, limiting techniques to these methods can hinder predictive performance. Model-agnostic interpretation methods avoid this issue. These techniques can be applied to any classifier to provide interpretability. We discuss two popular model-agnostic methods: permutation feature importance and global surrogate. Despite their utility with non-interpretable models, these methods have advantages and disadvantages that make them appropriate in different scenarios.

4.5.1 Permutation Feature Importance

Permutation feature importance measures the increase in the prediction error after permuting a feature's values (Molnar 2020). In this setting, a feature is "important" if permuting its values increases the model's error because this implies that the model relies on that feature to make its predictions. Shuffling values of "unimportant" features, however, do not significantly change the model's error because the model likely ignored the feature in its prediction. The permutation feature importance algorithm is outlined in Algorithm 1.

Algorithm 1: Pseudo-code for the permutation feature importance algorithm.

Data: Dataset used in classification

Result: Ordered list of features by importance

Input: model f, feature matrix X, target variable y, loss function $L(y, f)$.

Estimate original model error: $e_{\text{orig}} = L(y, f(X))$;

for *feature* $j = 1, 2, \ldots, p$ **do**

  Define X_{perm} by permuting feature j in the feature matrix X;

  Estimate permutation error $e_{\text{perm}} = L(y, f(X_{\text{perm}}))$ based on predictions using X_{perm};

  Calculate permutation feature importance: $F_j := \frac{e_{\text{perm}}}{e_{\text{orig}}}$;

Sort features by descending values of F

One possible flaw in this method is the relation of permutation feature importance to the error of the model (Molnar 2020). In general, this may not be an issue, but it restricts our ability to assess the output variation of a feature without considering its implications on model performance. Despite these shortcomings, feature importance generally provides compressed, global insight into model behavior in a way that is easy to interpret. This method takes interactions between other features into account and does not require retraining the model (Molnar 2020).

4.5.2 Global Surrogate

Global surrogate is a direct method of understanding the inner workings of black box models. By approximating the predictions of a black box model, global surrogate acts as a lens with which we can interpret machine learning models.

This method uses a surrogate model to approximate the predictions of the underlying model as accurately as possible while retaining interpretability (Molnar 2020). Mathematically, we are approximating our machine learning prediction function f with some surrogate, interpretable function g in a way that maximizes accuracy between the predictions of the original model and the surrogate (Molnar 2020). These functions g are simply interpretable machine learning models, such logistic regression, Gaussian Naive Bayes, and decision trees. Note that training the surrogate is model-agnostic, as it does not require

information about the structure of the black box model. Rather, it requires access to data and the prediction function (Molnar 2020).

To implement a global surrogate model, we follow a series of simple steps (Molnar 2020). After completing these steps, we can interpret our surrogate model if it closely approximates the black box model.

1. Select the training dataset or a subset of the training dataset, X, depending on the scope of interpretability you wish to achieve.
2. Determine the black box model's predictions on X.
3. Choose an interpretable model as the surrogate function g.
4. Train this model on X, obtaining predictions of the surrogate.
5. Measure the accuracy of the surrogate as compared to the black box model.

A common metric for measuring the deviation between the surrogate and the black box model is R-squared (Molnar 2020). This metric is defined as

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_*^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{n}(\hat{y}^{(i)} - \bar{\hat{y}})^2}. \tag{5}$$

R-squared measures the percentage of variance explained by the surrogate model. In Equation (5), $\hat{y}_*^{(i)}$ represents the surrogate model's prediction, $\hat{y}^{(i)}$ the black box model's prediction, and $\bar{\hat{y}}$ the mean of black box model predictions (Molnar 2020). Note that R-squared values close to 0 indicate that the surrogate poorly approximates the black box model, while a value close to 1 indicates that the surrogate approximates the black box model well.

Like any technique, global surrogate has both advantages and disadvantages. Global surrogate is a flexible technique that can use any interpretable model as its base, and its accuracy relative to the black box model is easy to measure using R-squared. The easy to calculate R-squared value comes at the cost of the ambiguity of defining a cutoff of a decent value for this metric. There is an additional complication in this method's flexibility, as each surrogate model comes with its own advantages and disadvantages that should be considered in the interpretation process.

5 Imbalanced Classification Interpretability

We seek to contextualize interpretability methods with imbalanced data. First, we discuss two black box models that are not interpretable and explain the properties that decrease their interpretability. The chapter concludes with a state of the art, interpretable technique for imbalanced classification.

5.1 Uninterpretable Classification Methods Background

When dealing with imbalanced datasets, interpretable machine learning methods are often not ideal due to their simplicity. Decision trees, though interpretable, split the feature space along the feature axis greedily (Goh 2018). These characteristics make interpretable machine learning algorithms difficult to use in imbalanced settings. In this part of the report, we delve into powerful classification methods that lack interpretability. We give an overview of their algorithms and discuss why they lack the interpretability of decision trees, even though they both use trees in their structure.

5.1.1 Random Forest

Bootstrap aggregation, or bagging, reduces variance of estimated prediction functions and works particularly well for trees, which have high variance and low bias (Hastie et al. 2009). Bagging averages noisy, unbiased models to reduce their variance (Hastie et al. 2009). Decision trees often have low bias when grown sufficiently deep but are generally noisy, making these ideal candidates for bagging (Hastie et al. 2009). The Random Forest classifier uses bagging to improve upon the popular Decision Tree classifier (Hastie et al. 2009). Random forests essentially build a large collection of uncorrelated trees and average them. Whereas decision trees split each node using the best split among all variables in the dataset using a majority vote, random forests split each node using the best among a subset of predictors randomly chosen at that node, constructing multiple trees, each on a bootstrapped dataset (Liaw et al. 2002).

Although each tree in a random forest is interpretable, as discussed in Section 4.4.3, random forests contain many such trees. The forest then likely has conflicting decisions,

hence the use of majority vote to make predictions. In that sense, it becomes challenging to understand the classification of individual data points with random forests. Therefore, this technique is considered to be a black box method.

5.1.2 Gradient Boosting

In contrast to random forests, which build an ensemble of deep independent trees, gradient boosting classifiers build an ensemble of shallow trees in a way where each tree improves upon the previous one (Greenwell 2020). Gradient boosting combines boosting with gradient descent to achieve optimal results.

Boosting adds new models to the ensemble sequentially (Greenwell 2020). By starting with a weak learner, often a decision tree, the model boosts performance by continually adding trees, where each new tree aims to fix errors in prediction from the previous one, as shown in Figure 4 (Greenwell 2020). Although most boosting algorithms, including gradient boosting, use decision trees as their base learners, boosting can improve upon predictions from any weak learner, which have error rates only slightly better than random guessing (Greenwell 2020).

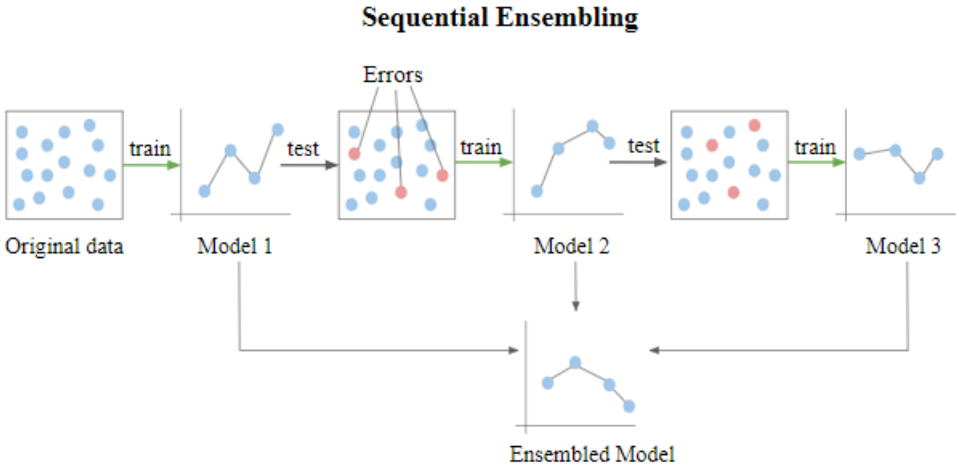

Figure 4: Boosting ensembles models sequentially (Greenwell, 2020).

Gradient descent is a function minimization process. In the case of machine learning problems, the function we want to minimize is a loss, or cost, function in order to ensure the algorithm gives optimal results. Often, this function is given by Mean Squared Error

(MSE), Mean Absolute Error (MAE), or Binary Cross Entropy (BCE), but gradient descent can be performed on any differentiable loss function (Greenwell 2020).

The general idea of gradient descent is to take steps in the direction negative to the gradient at a point until it reaches a local minimum, as shown in Figure 5. The size of these steps is determined by the learning rate parameter γ. Too small of a learning rate increases computational as the algorithm must run for more iterations, while too large of a learning rate means that the algorithm may miss the minimum (Greenwell 2020).

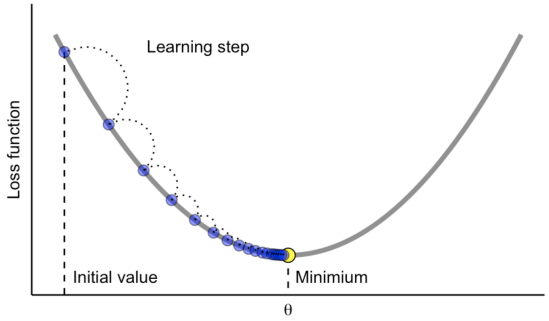

Figure 5: Gradient descent is used to minimize loss functions (Greenwell, 2020).

The algorithm for gradient boosting classification is shown in Algorithm 2 (Hastie et al. 2009).

Algorithm 2: Pseudo-code for the Gradient Boosting Algorithm.

Initialize loss function $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$;
for *m = 1 to M* **do**
  for *i = 1, 2, . . . , N* **do**
    Compute $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$;
  Fit a regression tree to class r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$;
  for $j = 1, 2, \ldots, J_m$ **do**
    Compute $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$;
  Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$;
Output $\hat{f(x)} = f_M(x)$;

Like random forests, the gradient boosting classifier is also referred to as a black box. Unlike decision trees, a fundamental piece of gradient boosting, ensembling many trees

makes it much more difficult to track specific data instances and thus understand why gradient boosting makes certain predictions.

5.2 State of the Art Technique

Although the topic of imbalanced data interpretability with classification is not-well studied, Siong Thye Goh proposes a novel classification technique tailored to this purpose in *Machine Learning Approaches to Challenging Problems: Interpretable Imbalanced Classification, Interpretable Density Estimation, and Causal Inference*. Goh proposes the Exact Boxes method, a classifier formed as a union of axis parallel rectangles around the positive instances in the dataset (Goh 2018). Such a structure makes it more interpretable to people due to its ease of visualization, as shown in Figure 6.

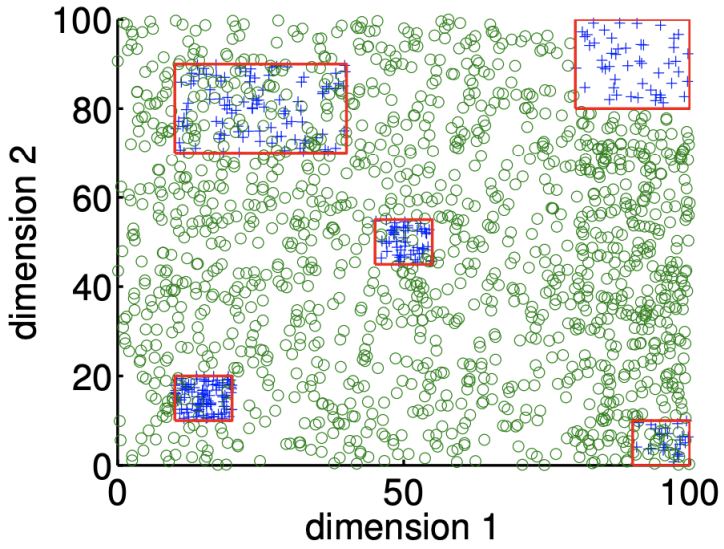

Figure 6: The Exact Boxes method draws boxes around positive instances in imbalanced dataset even when they are contained among many negative instances (Goh, 2018).

This method performs at the level of other state of the art classifiers despite the fact that it is restricted to producing interpretable results (Goh 2018). By using Bayesian priors, the priors control the shape of the tree in the method, improving both generalization and interpretability (Goh 2018).

Model performance, as shown by unbiased metrics, suffers in the classification of imbalanced datasets, steepening the tradeoff between interpretability and performance. Goh's

method balances this tradeoff, performing well, but not at the cost of interpretability.

6 Conclusion

Imbalanced data poses challenges to standard classification techniques. When trying to improve upon classification methods for imbalanced data, we often think about improving model performance, not model interpretability. As statisticians become more interested in learning about model interpretability and creating tools for this problem, they often build upon classification techniques that are well-studied and understood, leaving a gap in the literature involving interpretability of imbalanced data. In this report, we explored canonical classification methods and why they fail on the class imbalance problem. Adding interpretability into this problem, we discussed uninterpretable classifiers and why they are difficult for humans to understand, especially in the context of imbalanced data. Though some novel methods exist that balance interpretability and performance for imbalanced data, such as Goh's Exact Boxes, there is a dearth of such techniques. By better understanding the intersection of classification, class imbalance, and model interpretability, as investigated in this report, we can begin to create new techniques for these settings.

7 Appendix

7.1 Class Imbalance Initial Example

The first code chunk was used to create the bar graph demonstrating class imbalance in the Kaggle Credit Cards dataset, and the second code chunk compares standard performance metrics for imbalanced datasets.

```
# Numbers pulled from confusion_lr
tn <- 227436
fn <- 137
fp <- 28
tp <- 245

tpr <- tpr(tp = tp, fn = fn)
fpr <- fpr(fp = fp, tn = tn)
tnr <- tnr(fp = fp, tn = tn)
fnr <- fnr(tp = tp, fn = fn)

precision <- tp / (tp + fp)
recall <- tp / (tp + fn)
bal_accuracy <- 0.5 * (tpr + tnr)
g_mean <- sqrt(tpr * tnr)
f_measure <- (2 * recall * precision) / (recall + precision)
```

7.2 Classifier Comparison

This code shows the data wrangling, modeling, and testing of a logistic regression classifier, Gaussian Naive Bayes classifier, and decision tree classifier to compare accuracy and balanced accuracy.

```r
# Set seed for reproducibility
set.seed(495)

# Remove time variable
credit_small <- creditcard[, -1] %>%
  drop_na()

# Training and testing dataset
n <- nrow(credit_small)
credit_parts <- credit_small %>%
  initial_split(prop = 0.8)

train <- credit_parts %>%
  training()
nrow(train)

test <- credit_parts %>%
  testing()
nrow(test)
```

```r
# Set seed for reproducibility
set.seed(495)

# Create model
form <- as.formula(
  "Class ~ ."
)
mod <- glm(form, family = "binomial", data = train)
msummary(mod)
mod_glm <- logistic_reg(mode = "classification") %>%
```

```
  set_engine("glm") %>%
  fit(form, data = train)

# Evaluate Model
pred <- train %>%
  select(Class) %>%
  mutate(
    credit_dtree = mod_glm %>%
      predict(new_data = train, type = "class") %>%
      pull(.pred_class)
  )
confusion_lr <- pred %>%
  conf_mat(truth = Class, estimate = credit_dtree)
confusion_lr

# Accuracy vs Balanced Accuracy
yardstick::accuracy(pred, Class, credit_dtree)
bal_accuracy(pred, Class, credit_dtree)
```

```
# Create model
gnb_mod <- naiveBayes(Class ~ ., data = train)
gnb_pred <- predict(object = gnb_mod, newdata = test, type = "class")

# Confusion matrix and model evaluation
cm_gnb <- table(test$Class, gnb_pred)
confusionMatrix(cm_gnb)
```

```
# Create model
dtree_mod <- rpart(Class ~ ., data = train, method = "class")
dtree_pred <- predict(object = dtree_mod, newdata = test, type = "class")
```

```r
# Confusion matrix and model evaluation
cm_dtree <- table(test$Class, dtree_pred)
confusionMatrix(cm_dtree)
```

7.3 Classifier Interpretability Examples

We display an example of logistic regression to demonstrate its interpretability in the context of the Kaggle Credit Cards dataset.

```r
neat_names <- c(
  "Intercept", "Feature 1",
  "Feature 2", "Feature 3",
  "Amount of Transaction (dollars)"
)

mod <- glm(Class ~ V1 + V2 + V3 + Amount,
  data = creditcard, family = binomial()
)

# Print table of coef, exp(coef), std, p-value
coef.table <- summary(mod)$coefficients[, c("Estimate", "Std. Error")]
coef.table <- cbind(coef.table,
                    "Odds ratio" = as.vector(exp(coef.table[, c("Estimate")])))

# Interpret one numerical and one factor
rownames(coef.table) <- neat_names
colnames(coef.table)[1] <- "Weight"
kable(coef.table[, c("Weight", "Odds ratio", "Std. Error")],
      digits = 2,
      caption = "The results of fitting a logistic regression model on
      the Kaggle Credit Cards dataset. The table shows the estimated weights,
```

```
      odd ratios, and standard errors of the weights for features used in the
      logistic model (Molnar, 2020).")
```

References

Breiman, L. et al. (2001), 'Statistical modeling: The two cultures (with comments and a rejoinder by the author)', *Statistical science* **16**(3), 199–231.

Chakure, A. (2020), 'Decision tree classification'.
URL: *https://medium.com/swlh/decision-tree-classification-de64fc4d5aac*

Gayathri, B. & Sumathi, C. (2016), 'An automated technique using gaussian naïve bayes classifier to classify breast cancer', *International Journal of Computer Applications* **148**(6), 16–21.

Goh, S. T. (2018), Machine learning approaches to challenging problems: interpretable imbalanced classification, interpretable density estimation, and causal inference, PhD thesis, Massachusetts Institute of Technology.

Greenwell, B. B. a. B. (2020), 'Hands-on machine learning with r'.
URL: *https://bradleyboehmke.github.io/HOML/gbm.html*

Hall, P. (2019), *An introduction to machine learning interpretability*, O'Reilly Media, Incorporated.

Hastie, T., Tibshirani, R. & Friedman, J. (2009), *The elements of statistical learning: data mining, inference and prediction*, 2 edn, Springer.
URL: *http://www-stat.stanford.edu/ tibs/ElemStatLearn/*

Johnson, J. M. & Khoshgoftaar, T. M. (2019), 'Survey on deep learning with class imbalance', *Journal of Big Data* **6**(1), 27.

Liaw, A., Wiener, M. et al. (2002), 'Classification and regression by randomforest', *R news* **2**(3), 18–22.

Molnar, C. (2020), 'Interpretable machine learning'.

URL: *https://christophm.github.io/interpretable-ml-book/evaluation-of-interpretability.html*