THE FUTURE OF FRAUD DETECTION

Detecting Fraudulent Insurance Claims Using Machine Learning Methods

R. V. Plaisant van der Wal

TU Delft

The Future of Fraud Detection: Detecting Fraudulent Insurance Claims Using Machine Learning Methods

by

R. V. Plaisant van der Wal

to obtain the degree of

Master of Science in Computer Engineering

at Delft University of Technology,
to be defended on Friday, August 17, 2018 at 10:00 AM

Student number:	4206169	
Date of submission:	August 12, 2018	
Responsible professor:	dr. ir. Zaid Al-Ars	TU Delft
Thesis committee:	dr. ir. Zaid Al-Ars	TU Delft, jury chairman
	dr. ir. Sicco Verwer	TU Delft, jury member
	Willem de Voogd	Voogd & Voogd, jury member

CE-MS-2018-18
Computer Engineering
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
Mekelweg 4,
2628 CD, Delft
The Netherlands

Abstract

Machine learning methods are explored in an attempt to achieve better predictive performance than the legacy rule-based fraud detection systems that are currently used to detect fraudulent car insurance claims. There are two key principles that lead the exploration of machine learning techniques and algorithms in this thesis, namely, the applicability to imbalanced data, and the interpretability of predictions. The dataset used for model training and evaluation contains only 0.3% fraudulent claims compared to 99.7% non-fraudulent claims, which can therefore be considered highly imbalanced. Furthermore, prediction interpretability is of great importance, since fraud experts are directly interfacing with the output of the machine learning models. With the key principles in mind, this thesis considers four algorithms, Logistic Regression, Random Forest, LightGBM and a Stacking classifier. The algorithms are trained on the imbalanced learning problem by using a combination of undersampling (random and Edited Nearest Neighbors), oversampling (SMOTE) and class weighting. Conclusively, each trained model meets the objective, with the Stacking classifier combining the best performance with the lowest variance. By benchmarking the baseline for two different parameters, the models can be evaluated for two boundary conditions, which leads to tunable performance between the two conditions. Ultimately, the performance of the Stacking classifier is tunable (by moving its classification threshold) to roughly a 70-80% increase in extra fraud caught or a 75% reduction in effort. Extra fraud will increase the amount of real fraudulent claims that fraud experts get to see, and effort reduction leads to an increase in capacity, which enables fraud experts to spend more time on other more relevant tasks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Insurance is an integral component of society and must be affordable and reliable in order to provide everyone with aid in time of need. However, insurance fraud brings disturbance to the insurance sector, increasing the cost for both insurers and insureds. With technology on the rise, insurance companies seek state-of-the-art solutions to improve fraud detection. This research is executed in collaboration with one such company, which provides an opportunity to improve fraud detection with machine learning methods on real-world data; the key focus of this thesis being to bring improvements to the detection of fraudulent car insurance claims.

Insurance fraud is any act of deception with the intent to obtain a benefit that does not rightfully belong to the deceiver, with financial profit being the main driver behind insurance fraud [1]. Studies have been carried out in an attempt to appraise the cost of insurance fraud in Europe. However, it is difficult to determine the exact losses due to insurance fraud, because insurance fraud is deliberately kept under the radar by fraudsters. Therefore, the portion of fraudulent claims that are detected is much smaller than the portion of fraudulent acts that are committed [1]. According to the European federation of insurance companies—"Insurance Europe"—detected and undetected fraud is estimated to represent up to 10% of all claims expenditure in Europe [2]. A study by the car insurance industry in Quebec has shown that 3 to 6.4% of all claim payments contained fraud [3].

Due to the sheer number of claims submitted each day, it would be far too expensive for insurance companies to have employees check each claim for symptoms of fraud [4]. Instead, many companies use automated systems to identify suspicious claims for further investigation [5]. This also applies to the collaborating insurance company. They have incorporated a two-step procedure to catch fraud. Firstly, the suspicious claims with the highest probability to be fraudulent are identified by a so-called rule-based detection system Thereafter, during the second step, the identified claims are referred to fraud experts that check the claims again for potentially fraudulent characteristics, to decide whether a more elaborate investigation is needed.

1.2 Objective

The advent of machine learning and other related technologies opens up new paths for exploration to bring about improvements to the predictive performance of the previously mentioned rule-based fraud detection system. This thesis makes an attempt at achieving this. The attempt is successful when the final results satisfy the following objective statement:

> **Achieve better predictive performance than the rule-based fraud detection system, which is currently installed at the collaborating insurance company, using machine learning methods**

"Better predictive performance" is defined by the following two constraints:

- Catching a higher amount of fraud than the rule-based detection system.
- Flagging fraudulent claims with a higher efficiency than the rule-based detection system. Thus, flagging a lower amount of claims as suspicious, while catching a similar amount of fraud.

Complying to the constraints above, ensures that the machine learning methods will catch a higher amount of fraudulent claims with a higher efficiency i.e. less misclassifications. Which in turn will increase fraud expert efficiency, reduce cost and leave more time to focus on relevant claims; increasing the chance to catch fraudsters.

1.3 Outline

Chapter 2 kicks things of with a discussion of the machine learning related topics that are used in this thesis. It provides information about a variety of algorithms as well as concepts like overfitting and data resampling. Chapters 3 and 4 build upon the theoretical foundation established in chapter 2, with a description of the approach used to turn data into a model that is able to predict fraud. Subsequently, the results obtained with the previously described approach are presented in chapter 5. The predictive performance of the models is compared to the baseline, in order to see if the yielded results match the requirements that are stated in the objective. Following the presentation of the results, chapter 6 concludes and evaluates the results. Moreover, the possible machine learning solution to satisfy the main objective is discussed, and some additional words are shared regarding the use of performance metrics on imbalanced data problems. Lastly, extra results are presented regarding the potential difference in predictive performance of the final model in a real world environment.

Chapter 2

Machine Learning Theory

2.1 Machine Learning

Machine learning can be defined as the practice of using algorithms on large amounts of data to learn and discover the underlying patterns and relationships in the data, and then make a determination or prediction about something in the world. The field of machine learning encompasses a wide variety of techniques and algorithms and a high level way of categorizing these techniques and algorithms is:

- **Supervised learning**
  Supervised machine learning algorithms are trained on datasets with labels. With supervised learning an algorithm learns a mapping function from the input variables to the output variable. The algorithm tries to approximate the mapping function to such a degree that it can predict the output variable for new input variables. Supervised learning derives its name from its learning process being supervised. For every wrongly predicted value the algorithm is corrected, and training is stopped when the algorithm achieves reasonable performance. Supervised learning can be split into two different problem types, regression and classification. A continuous output space indicates that one is dealing with a regression problem, whereas a target variable made up of classes indicated that one is dealing with a classification problem.
- **Unsupervised learning**
  With unsupervised machine learning, algorithms are trained on unlabeled data and try to model the underlying structure or distribution of the data. There are only input variables available and no output variables. Unlike supervised learning algorithms, unsupervised learning algorithms cannot be corrected with a target output. They are left to their own devises to find patterns and structures in the data. Clustering and dimension reduction are both examples of unsupervised learning problems.

A variation on supervised and unsupervised learning is semi-supervised learning. Semi-supervised learning algorithms train on data that is only

partially labeled and use a combination of supervised and unsupervised learning techniques. The above categorization will help to understand later on why a certain choice for techniques and algorithms is made. The rest of this chapter will elaborate more on common techniques that are used in the field of machine learning and also discuss the machine learning algorithms that will be used to achieve the objective of this thesis.

2.2 Data Preprocessing

The sections below discuss multiple data preprocessing techniques that are used in the field of machine learning to alter the input space for better predictive performance. Most techniques apply to imbalanced datasets, this is relevant, because that dataset that is used for this research is highly imbalanced; see section 3.3.1.

2.2.1 Sample Weighting

With sample weighting, weights get added to single instances or entire classes to bias an algorithm towards the weighted instances/classes. When dealing with imbalanced data, weighting is a useful tool to counter the naturally occurring bias an algorithm has towards the majority class. The following example illustrates how weighting samples can change the direction to which an algorithm tries to optimize.

> **Example A: An algorithm's natural bias towards majority class**
> Given an imbalanced dataset with a 1000/10 majority/minority instances ratio.
> Scenario A.1: A classification accuracy of 90% on the majority class and 10% on the minority class, gives a total score of:
> $(90\%) \cdot 1000_{maj} + (10\%) \cdot 10_{min} = 901$.
> Scenario A.2: A classification accuracy of 10% on the majority class and 90% on the minority class, gives a total score of:
> $(10\%) \cdot 1000_{maj} + (90\%) \cdot 10_{min} = 109$.

The first scenario in example A performs roughly 8 times better than the second scenario. Thus, for the algorithm it pays off to focus more on learning to predict the majority rather than the minority class. By adding weights to the minority class, it is possible to create a scenario where the algorithm gets rewarded more for classifying minority instances.

> **Example B: Sample weighting helps to shift class balance**
> Scenario B.1: Class distribution like scenario 1a plus a minority class weight of 1000, gives a total score of:
> $(90\%) \cdot 1000_{maj} + (10\%) \cdot 10_{min} \cdot 1000_{weight} = 1900$.
> Scenario B.2: Class distribution like scenario 2a plus a minority class weight of 1000, gives a total score of:
> $(10\%) \cdot 1000_{maj} + (90\%) \cdot 10_{min} \cdot 1000_{weight} = 9100$.

As one can see in example B, after weighting, it now pays off for the algorithm to shift its focus more towards learning the minority class. Algorithms have multiple ways of dealing with sample weights, some use the weights in their loss function, others duplicate minority instances to simulate the weight effect.

Weighting is similar to sampling (section 2.2.2), in the sense that it tries to restore data balance. Furthermore, an advantage weighting has over sampling, is that it does not alter the data and thus not remove any valuable entries from the dataset.

2.2.2 Data Resampling

Data resampling, just like sample weighting, is a technique to counter data imbalance. However, the main difference is that with data resampling one creates a different data distribution, rather than adding weights. By changing the distribution, the dataset will appear more balanced for the algorithm. Resampling by removing data points is called undersampling, and resampling by adding artificial data points is called oversampling.

Random Resampling

Random resampling is the most basic form of data resampling and is focused to restore data balance within a dataset to improve algorithm training. Data balance is restored by oversampling the minority class and/or undersampling the majority class.

Random undersampling randomly removes instances from the majority class, lowering the size of the majority class, and balancing the dataset to the required majority/minority ratio. A drawback of this technique is the possible loss of important information, some discarded instances can be essential for good performance.

Random oversampling does the exact opposite of random undersampling. Instead of decreasing the size of the majority class to restore balance, random oversampling increases the size of the minority class. It does so by adding replications of instances from the minority class, and thus balancing the dataset to the required majority/minority ratio. An advantage oversampling has over undersampling is that no valuable information is lost. However, random oversampling makes algorithms more prone to overfitting by replicating instances multiple times.

Synthetic Minority Oversampling TEchnique (SMOTE)

SMOTE (Synthetic Minority Oversampling TEchnique) [6] is an oversampling technique that uses synthetic data generation to overcome imbalance in the original dataset. An arbitrary number of synthetic minority instances are generated to shift the classifier learning bias towards the minority class.

Figure 2.1, illustrates the creation process of the artificial data points. SMOTE first picks a random instance from the minority class and computes the k-nearest minority neighbors of this instance. The artificial instances

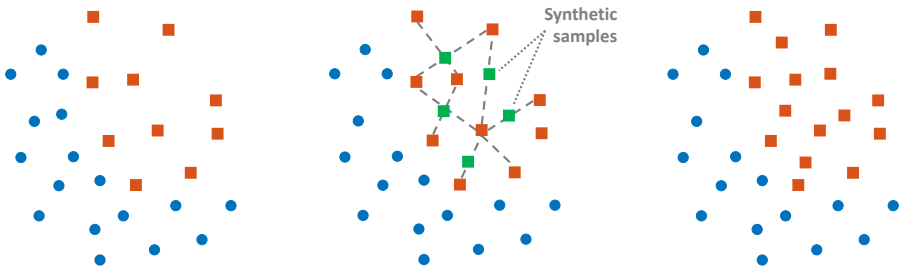


Figure 2.1: Artificial data generation with SMOTE. Data points get added on the line segment connecting two existing data points.

are then generated between one of the randomly chosen neighbors and the originally chosen instance. Below follows a more detailed approach of how new instance are being generated:

1. For every minority instance i, randomly select one of its k nearest neighbors x
2. Find the difference vector $\vec{v}$ between i and x
3. Randomly select a point p along $\vec{v}$, by $p = i + rand(0, 1) \cdot \vec{v}\,(i, x)$
4. Add point p to the minority class

This process gets repeated for every minority instance and is completed when the target imbalance ratio is reached.

By creating new *synthetic* samples instead of exact replications—like random oversampling—overfitting is less likely to happen. Since, replications tighten decision boundaries around one single observation, whereas synthetic samples soften the decision boundary for better generalization.

Tomek Links

The removal of Tomek links is a data cleaning method that attempts to remove overlap between classes, making it easier for a machine learning algorithm to distinguish between classes.

The definition of a Tomek link is given as: given an instance pair (x_i, x_j), where $x_i \in S_{min}, x_j \in S_{maj}$, and $d\,(x_i, x_j)$ is the distance between x_i and x_j, then the (x_i, x_j) pair is called a Tomek link if there is no instance x_k, such that $d\,(x_i, x_k) < d\,(x_i, x_j)$ or $d\,(x_j, x_k) < d\,(x_i, x_j)$ [7], [8].

Meaning that point pairs forming Tomek links are considered to be noise or class borders. By removing Tomek links, one can obtain well-defined class clusters in the training dataset, which can, in turn, improve classification performance.

Edited Nearest Neighbors (ENN)

Edited Nearest Neighbors (ENN) [9] attempts to edit out noisy instances and close border cases, a similar goal to that of Tomek links. Whenever the

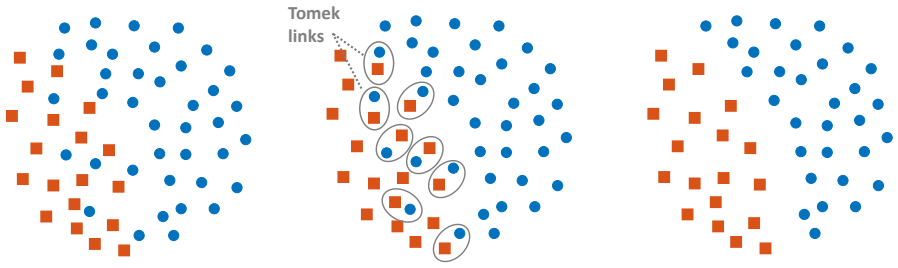


Figure 2.2: The identification and removal of Tomek links

majority of the k-nearest neighbors of an instance contradict with its own class, the instance is removed.

2.2.3 Feature Selection and Crossing

Feature selection and feature crossing are two techniques to alter the dimensions of the input space. Feature selection reduces the dimensionality by making a selection of features, to reduce the noise induced by input features that do not contain any information about the target. Feature crossing creates new features by combing existing features, to make certain relationships between features more explicit.

Feature Selection

Feature selection algorithms can be divided into three categories:

- **Filter methods**
  Filter methods score features by applying a statistical method. The features are then ranked by their score and generally the features with the lowest scores are discarded, and the features with the highest scores are kept for model training. A common technique of filtered feature selection is the selection of features based on their mutual information values. Mutual information [10] measures the dependency between two variables, in this case a feature and the target label. With this technique it is possible to identify whether the target label depends on the feature and thus if it contains information about the target. Higher mutual information means a higher dependency between feature and target label.
- **Wrapper methods**
  Wrapper methods prepare, compare and evaluate different feature sets to determine which selection of features has the best performance. An example of wrapped feature selection is the recursive feature elimination algorithm [11].
- **Embedded methods**
  Embedded methods find the features that contribute most to the

accuracy of a model during training. Regularization (see section 2.3.3) is the most common type of embedded feature selection.

As one can see, the field of feature selection encompasses many techniques and is widely covered in literature. However, various methods to select features were tried, but none yielded considerably improved performance. Therefore, the switch was made to select features by hand and not programmatically.

Feature Crossing

Feature crossing attempts to make the relationships between features more explicit. With feature crossing, features get transformed into a new one, through a mathematical operation. This makes sense when the meaning of each feature is known. For example, in the case that one has the features *length* and *width*, a new feature *area* could be created by multiplication. By including a crossed feature, one makes the relationship between two features more explicit to the model, which in return can improve training results. However, for this research, feature meanings and raw values are not disclosed, which complicates creating valuable crossed relationship and therefore is beyond the scope of this thesis.

2.3 Linear Models

This section will introduce both Linear and Logistic Regression. Since this thesis is about a classification problem, only Logistic Regression will actually be used, however, linear regression is easy to understand and provides good background knowledge for Logistic Regression.

2.3.1 Linear Regression

One of the most commonly used machine learning models to tackle regression problems is linear regression. A linear regression model attempts to model the linear relationship between the response variable y and input variables (features) $\mathbf{x}$, through an error variable ϵ. Combining the above, the model can be written as

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \epsilon$$

with $\mathbf{w}$ the model's weight vector, ϵ the residual error between the predictions and true response, and $\mathbf{x}$ a matrix made up of row vectors $\mathbf{x}_i$ for $i = 1, 2, ..., J$ where J the number of features. One often assumes ϵ to be normally distributed

$$\epsilon \sim \mathcal{N}\left(\mu, \sigma^2\right)$$

where μ is the mean and σ the variance. With this assumption, one can rewrite linear regression to the following form:

$$p(y|\mathbf{x}, \theta) = \mathcal{N}\left(y|\mu(\mathbf{x}), \sigma^2(\mathbf{x})\right)$$

In the simplest case, one can assume μ to be a linear function of $\mathbf{x}$, $\mu(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, and the noise to be fixed, $\sigma^2(\mathbf{x}) = \sigma^2$. Now, θ represents the parameters of the model $\theta = \left(\mathbf{w}, \sigma^2\right)$, and the model can be rewritten to:

$$p(y|\mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}\left(y|\mathbf{w}^T\mathbf{x}, \sigma^2\right)$$

Now with the final derivation of linear regression, all that is left is to estimate the parameters of linear regression. A common method for estimating the parameters of a linear regression model is: ordinary least squares (OLS). OLS tries to minimize the sum of square differences between the true response and predictions. Using OLS, the weight vector $\mathbf{w}$ is computed using:

$$\mathbf{w} = \left(\mathbf{x}^T\mathbf{x}\right)^{-1}\mathbf{x}^T y$$

2.3.2 Logistic Regression

Logistic regression is a generalization of linear regression to make it suitable for (binary) classification. In order to achieve this generalization two changes to standard linear regression need to be made.

Firstly, the normal distribution for y is replaced by a Bernoulli distribution (indicated by Ber) that better suits a binary response, $y \in \{0, 1\}$. The Bernoulli distribution is a discrete distribution that describes a boolean outcome, its probability density function is defined by

$$f(k; p) = \begin{cases} p & \text{if } n = 1, \\ 1 - p & \text{if } n = 0. \end{cases}$$

with p the probability and n the outcome. Now the model is described by

$$p(y|\mathbf{x}, \mathbf{w}) = \mathrm{Ber}\left(y|\mu(\mathbf{x})\right) \tag{2.1}$$

where $\mu(\mathbf{x}) = p(y = 1|\mathbf{x})$.

Secondly, to ensure that $\mu(\mathbf{x})$, the linear combination of the input variables $\mathbf{x}$ with weights $\mathbf{w}$, is bound between 0 and 1, it is passed through a *sigmoid* function

$$\mu(\mathbf{x}) = \mathrm{sigm}\left(\mathbf{w}^T\mathbf{x}\right)$$

where sigm is defined as:

$$\mathrm{sigm}(\eta) = \frac{1}{1 + \exp\left(-\eta\right)} \tag{2.2}$$

The term "sigmoid" means S-shaped. It is also known as a *squashing function*, since it squashes all values between 0 and 1.

Combining both steps together gives

$$p(y|\mathbf{x}, \mathbf{w}) = \mathrm{Ber}\left(y \mid \mathrm{sigm}\left(\mathbf{w}^T\mathbf{x}\right)\right)$$

the final representation of logistic regression[1].

[1]Note that, unlike its name implies, logistic regression is a form of classification not regression.

A big difference between Linear Regression and Logistic Regression is how the line is fit to the data, Linear Regression uses Ordinary Least Squares, whereas Logistic regression does not have the same concept as residuals, it uses maximum likelihood. The concept of maximum likelihood is elaborated on in section 2.4.2, where it is used to train gradient boosted decision trees. Furthermore, Logistic Regression uses *regularization* to improve its generalization to other data.

2.3.3 Regularization

Regularization can help to reduce overfitting (see section 2.4.1), and thus improve the generalization performance i.e. the performance on new unseen data. Regularization applies artificial constraints on an algorithm, to implicitly reduce the amount of free parameters of the algorithm, while maintaining its optimization efficiency.

The constraints are applied by adding an expression that penalizes the overfitting properties of the fit. The penalty of L2 regularization, is the squared ℓ_2-norm of $\mathbf{w}$:

$$R_{\ell_2}(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{i=0}^{n} w_i^2 \tag{2.3}$$

where $\mathbf{w}$ are the weights defined for both linear and logistic regression. The regularization strength can be tuned by a coefficient usually signified by λ_2. Taking Logistic Regression as an example, the larger its coefficients, the larger R will be, and because R is added to the loss function, the larger the loss function will be. Thus L2-regularization effectively tries to tune down the regression coefficients, which results in a more general model.

For L1 regularization the ℓ_1-norm is used:

$$R(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{i=0}^{n} |w_i| \tag{2.4}$$

which is the sum of the absolute values of the regression coefficients—also known as the Manhattan distance. instead of scaling everything by a uniform factor like L2 regularization, L1 regularization can make certain coefficients zero. By setting feature coefficients to zero, L1 regularization does a form of *feature selection*. Yuan et al. [12], gives a comprehensive review of ℓ_1 regularized Logistic Regression.

There is also a third form of regularization that combines both L1 and L2 regularization [13]. However, this form is not supported by the implementation of Logistic Regression that is used for this research, and therefore also not discussed in more detail.

2.4 Decision Tree Learning

A decision tree [14] performs classification by recursively considering a set of decision points—also called *branches*—that partition the input space $\mathbf{x}$

into multiple regions $\{R_1, R_2, ..., R_M\}$ with corresponding labels—also called *leaves*.

Starting from the root node, a tree is grown, by creating branches (based on decision criteria) that attempt to split data in a way to minimize a cost function. Every branch connects two nodes, but a node can have multiple branches sprouting from it. Branches are created until the tree reaches a certain size or exceeds its cost reduction threshold.

During prediction, data follows the appropriate branches, until it terminates in a so-called leaf node, where the data gets assigned the corresponding label of the region that the leaf node represents. All nodes in a tree—except leaf nodes—contain a decision criterion that gets evaluated to determine the next branch and corresponding node for the data to go to, see figure 2.3. A decision criterion can either be a threshold or equality constraint.

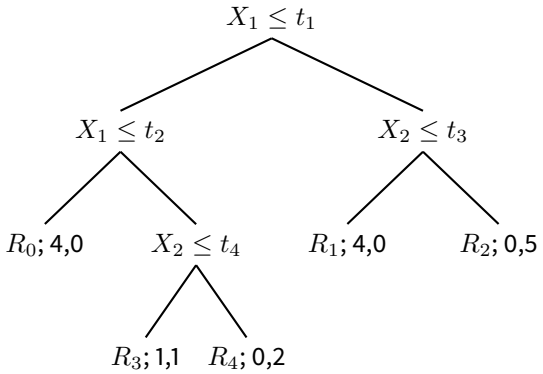


Figure 2.3: A simple decision tree for an input space with two features X_1 and X_2. The space is partitioned into 5 regions. Each leaf shows its class label distribution (n_0, n_1), with n_0 positive samples and n_1 negative samples.

The response of a classification tree is a probability value that is based on the class label distribution of the leaf node at the end of the path taken by the input data. This probability is the empirical fraction of positive samples that are grouped in the same region belonging to that leaf, defined by

$$p(y = 1|\mathbf{x}) = \frac{n_0}{n_0 + n_1} \tag{2.5}$$

where n_0 the positive class and n_1 the negative class. Using equation (2.5) and the class distributions of figure 2.3, a few examples of region probabilities can be computed. The probability of R_0 becomes $4/(4 + 0) = 1$, for R_1 the probability is $1/(1 + 1) = 0.5$, etc.

When growing a tree, finding the optimal data splits is an NP-complete problem. Hence, the reason why a heuristic based approach in the shape of algorithm 1 is used. Given a *node*, data $\mathcal{D}$ and maximum tree *depth*. The algorithm first computes the prediction using the class label distribution method described above. Secondly, it computes the best feature j^* and optimal threshold t^* using

$$(j^*, t^*) = \arg \min_{j \in 1,...,p} \min_{t \in \mathcal{T}_j} \left[\text{cost}\left(\{\mathbf{x}_i, y_i : x_{ij} \leq t\}\right) + \text{cost}\left(\{\mathbf{x}_i, y_i : x_{ij} > t\}\right) \right],$$

Algorithm 1 Fitting a decision tree

function FIT_TREE($node$, $\mathcal{D}$, $depth$)
  $node.prediction$ = CLASS_LABEL_PROBABILITY($y_i : i \in \mathcal{D}$)
  $(j^*, t^*, \mathcal{D}_L, \mathcal{D}_R)$ = SPLIT($\mathcal{D}$)

  if WORTH_SPLITTING($depth$, $cost$, $\mathcal{D}_L$, $\mathcal{D}_R$) **then**
    $node.test = \lambda \mathbf{x}.x_{j^*} > t^*$
    $node.left$ = FIT_TREE(new $node$, $\mathcal{D}_L$, $depth + 1$)
    $node.right$ = FIT_TREE(new $node$, $\mathcal{D}_R$, $depth + 1$)
  end if

  return $node$
end function

to split the dataset into $\mathcal{D}_L$ and $\mathcal{D}_R$. The algorithm is recursive and keeps repeating the described steps until it meets its stopping criteria.

When using decision trees for classification there are multiple ways to measure the quality/cost of a split. For either way, the class conditional probability is part of the computation, and is specified as:

$$\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbf{1}(y_i = c)$$

where $\mathcal{D}$ is the data in the leaf, and $\mathbf{1}$ the *indicator function* that is 1 when y_i belongs to class c and 0 when not. The class conditional probability $\hat{\pi}_c$ outputs a fraction between 0 and 1 that indicates the percentage of the data $\mathcal{D}$ belongs to class c. Below two ways of computing the classification cost are given:

- **Entropy**

$$H(\hat{\boldsymbol{\pi}}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$

- **Gini index**

$$\text{gini}(\hat{\boldsymbol{\pi}}) = 1 - \sum_c \hat{\pi}_c^2 = \sum_c \hat{\pi}_c - \sum_c \hat{\pi}_c^2 = 1 - \sum_c \hat{\pi}_c^2$$

Figure 2.4, shows the impurity scores of both Entropy and the Gini index for the entire probability space. Both measures are more sensitive for the lower and higher probability regions. Moreover, as one can see, entropy and Gini have a very similar response. Gini is generally preferred, because Gini is faster to compute and in only 2% of the cases Entropy gives different results.[15]

Once the split has been computed, the algorithm checks whether it is worth making the split, by using stopping heuristics like:

- is the reduction in cost too small?
- will the tree exceed the maximum specified depth?

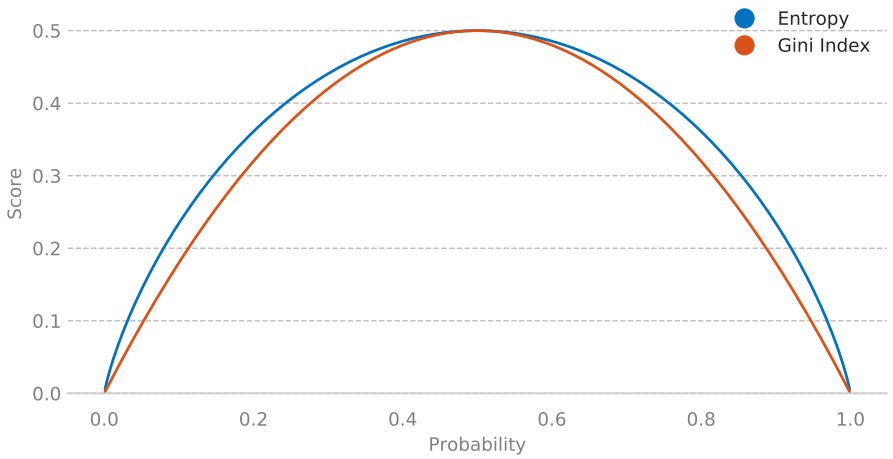


Figure 2.4: Node impurity measures for binary classification. Entropy has been rescaled to pass through the peak of the Gini index.

- is the distribution of the instances in the response sufficiently pure—e.g. either $\mathcal{D}_L$ or $\mathcal{D}_R$ contain labels that are mostly similar.
- are there enough samples present in both resulting data splits $\mathcal{D}_L$ and $\mathcal{D}_R$?

This process continues till the tree growth criteria are met, or the loss criterion is satisfied.

The advantages of using decision tree models are: interpretability, flexibility and also robustness to the input features. Decision trees are simple to understand and to interpret, they can be visualized. The conditions in the model are easily explained by boolean logic. For more complex trees it can be difficult to still comprehend the decision making, to solve this, section 4.4 describes an approach to extract the feature contributions of a decision tree for a specific prediction. Furthermore, decision trees are flexible and robust. They require little data preparation, and are able to handle both numerical and categorical data.

The main weakness of using a single decision tree is that it is prone to overfitting—unable to generalize to other data. Overfitting happens when no limit is posed on tree growth, and a tree is allowed to continue making splits to better split the data and minimize the cost function. A way to prevent overfitting is to limit tree growth, this will help to prevent making very specific splits that only work well on the train set. However, by limiting tree depth, underfitting is a serious risk, making the tree too general. Furthermore, decision trees can be unstable. Small variations in the data might result in a completely different tree being generated. Sections 2.4.2 and 2.4.3, discuss possible algorithmic solutions to mitigate the weaknesses of a single decision tree, by combining multiple decision trees to create a model that is specific, general and more stable.

2.4.1 Bias and Variance Trade-off

Before moving on to the algorithmic solutions to mitigate the weaknesses of a decision tree, bias and variance need to be discussed for a better understanding of the trade-offs that are involved. Bias and variance play a large role when improving decision trees or any other machine learning algorithm. This section attempts to clarify the terms bias and variance and why there is a trade-off between the two.

Supervised machine learning algorithms aim to estimate a mapping function for an output variable given some input data. This mapping function has a certain error, which is partially caused by bias and variance, defined by:

$$\text{error} = \text{bias}^2 + \text{variance} + \text{irreducible error}.$$

The irreducible error is irreducible in a sense that it cannot be reduced by the choice of algorithm or parameter tuning. It is deeply embedded in the problem itself and can only be tackled in the data engineering stage. Hence, the reason that, when choosing a model, one should focus on the bias and variance trade-off.

To introduce the concept of *bias*, let's introduce the simplest form of a decision tree (see section 2.4), called a *stump*, comprised of a single rule that divides data into two groups. This binary approach ignores the complexity and hidden relationships in the training data, making the model prone to errors due to *bias*. Bias indicates the level of mismatch between the predicted and true value. The more *bias* a model has, the more a model will ignore relevant details and produce wrong predictions i.e. the model is underfitting. In order to mitigate *bias*, additional splits can be added to the stump, creating a tree, and allowing the tree to base classifications on more complexity. One can keep adding splits until every instance has its own branch. However, by adding splits, the tree becomes more prone to modeling characteristics that are not part of the data, but rather of a second process. The tree starts to suffer from prediction errors due to *variance*. It is unable to generalize predictions to other relatively similar instances i.e. the model is overfitting. Overfitting makes a model sensitive to minor changes in the training data, increasing the *variance* at the output. Figure 2.5, illustrates the concepts of over and underfitting on a two-class dataset.

Summarizing, an overly-complex (error due to *variance*) model performs just as poor as an overly-simplistic (error due to *bias*) one. Therefore, a modeler has to make a trade-off between bias and variance, to minimize the overall error.

2.4.2 Gradient Boosting and Gradient Boosted Trees

Gradient boosting [16] is a method that fits a complex model by re-fitting simpler models (typically decision trees) to residuals, it works on both classification and regression problems. Gradient boosting is part of a family called ensemble methods. Ensemble methods are methods that perform classification by building a linear estimate over many other classifiers. There are two main types of ensemble methods, bagging (bootstrap aggregating,

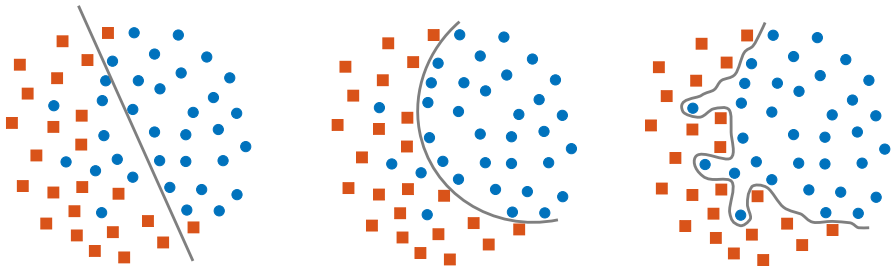


Figure 2.5: The squares and dots represent a dataset with two classes, and the line represents the fitted model. Left: underfitted model that suffers from high bias. Center: good fitted model that captures the underlying structure of the data. Right: overfitted model that suffers from high variance.

[17]) and boosting. Bagging inspired classifiers work by averaging sub-models, like Random Forest; see section 2.4.3. And boosting inspired classifiers fit complex models by iteratively fitting sub-models to residuals.

With gradient boosting, the model being *boosted* is called a *weak learner*. Many models can function as weak learners, but typically decision trees are used, because gradient boosting using decision trees is seen as one of the best off-the-shelf classifiers in the world [18]. As mentioned before, gradient boosting works by iteratively fitting new weak learners to the residual error of the previous iteration. Below follows a more detailed description of how gradient boosting achieves this.

The goal of gradient boosting is to find a function $f(\mathbf{x})$ that operates on a training set, such that the misclassification error at the testing set is as small as possible:

$$f(\mathbf{x}) = \min \sum_{(\mathbf{x},y)\in T} (f(\mathbf{x}) \neq y)$$

where T is a test set given by $T = (\mathbf{x}_i, y_i), ..., (\mathbf{x}_n, y_n)$. To find this function, a probabilistic model that models the probability of the loglink of an object is used:

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\sum_{m=1}^{M} h_m(\mathbf{x}))}$$

where $h_m(\mathbf{x})$ a decision tree. This probabilistic model is derived from the sigmoid function that maps all real values into the range between 0 and 1, see equation (2.2). Now, denote $f(\mathbf{x})$ as an ensemble of decision trees, by taking the sum of all decision trees:

$$f(\mathbf{x}) = \sum_{m=1}^{M} h_m(\mathbf{x}).$$

The probability of the loglink on the positive class can now be rewritten as:

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + exp(-f(\mathbf{x}))}.$$

Gradient boosting uses the principle of maximum likelihood. To understand this principle, let's first discuss likelihood. Likelihood is the probability of observing some data given a statistical model. For a dataset with n data entries, the likelihood function becomes:

$$\prod_{i=1}^{n} p(y_i|\mathbf{x}_i) = p(y_1|\mathbf{x}_1) \cdot ... \cdot p(y_n|\mathbf{x}_n).$$

The principle of maximum likelihood states that the algorithm should find a function $f(\mathbf{x})$ which maximizes the likelihood. Thus, the algorithm should find the underlying statistical model that maximizes the likelihood. This is equivalent to finding a function $f(\mathbf{x})$ that maximizes the logarithm of the likelihood.[2] The likelihood can now be denoted by:

$$Q(f) = \sum_{i=1}^{n} \log(p(y_i|\mathbf{x}_i))$$

which is the sum of all logarithms of probabilities. The maximum likelihood is computed by maximizing $Q(f)$. For simplicity reasons, one can rewrite $Q(f)$ as:

$$Q(f) = \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i))$$

with $L(y_i, f(\mathbf{x}_i))$ the loss function given by:

$$L(y_i, f(\mathbf{x}_i)) = \log(p(y_i|\mathbf{x}_i)).$$

Using the above, any weak learner can be trained using gradient boosting. Below follows an explanation of the training steps involved to train a gradient boosted model. In this case, decision trees are used as the weak learners. It all starts with the initial approximation, which is the optimal approximation of the sigmoid function, given by:

$$f_0(\mathbf{x}) = \log \frac{p_1}{1 - p_1}$$

Now, the steps below get repeated M (number of trees) times in an iterative manner. First, calculate the gradient of the loss function, using:

$$g_i = \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f=f_{m-1}}$$

In order to maximize $Q(f)$, the solutions should move in the direction of the gradient. Therefore, the gradient is computed for each data entry in the dataset. Now, a decision tree $h_m(\mathbf{x}_i)$ is fitted to an auxiliary training set $(\mathbf{x_i}, g_i), ..., (\mathbf{x}_n, g_n)$ that is created with the gradients from the previous step. The difference with the original set, is that the labels y_i are replaced by the

[2] From a computational point of view it is easier to deal with logarithms.

gradients g_i After fitting the decision tree, the optimal step size ρ_m can be computed using:

$$\rho_m = \arg\max_{\rho} Q(f_{m-1}(\mathbf{x}) + \rho h_m(\mathbf{x}))$$

which computes the step size that maximizes the likelihood for the addition of the newly fitted tree $h_m(\mathbf{x})$. Finally, the new decision tree $h_m(\mathbf{x})$ is added to the ensemble, with an extra *shrinkage* parameter ν to improve convergence:

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu\rho_m h_m(\mathbf{x}_i)$$

the value of ν typically ranges between 0.01 and 0.1.

For the full algorithm for fitting gradient boosted trees, see algorithm 2. Note that the method described above is not reserved solely for decision trees, it also applies to other weak learners.

Algorithm 2 Fitting gradient boosted trees

Initialize $f_0(\mathbf{x})$
for $m = 1 : M$ **do**
  Compute $g_i = \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f=f_{m-1}}$
  Fit a decision tree $h_m(\mathbf{x})$ to the target g_i on $(\mathbf{x_i}, g_i), ..., (\mathbf{x}_n, g_n)$
  Compute the step size $\rho_m = \arg\max_\rho Q(f_{m-1}(\mathbf{x}) + \rho h_m(\mathbf{x}))$
  Add $h_m(\mathbf{x})$ to the ensemble $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu\rho_m h_m(\mathbf{x}_i)$
end for
return $f(\mathbf{x}) = f_M(\mathbf{x})$

2.4.3 Random Forest

Random Forest is based on a technique called bootstrap aggregating (bagging) [17], which makes an attempt at mitigating the inability of a decision tree to generalize well to other data—so-called overfitting. Random forest is an ensemble of decision trees, that can perform both regression and classification tasks. Random Forest grows many specialized decision trees on different subsets of the training data and its features [19]. Separately, each specialized tree has a low bias and high variance. However, by creating a forest of trees that uses a combination of many trees for a prediction, a model is created with low bias and low variance.

As shown by equation (2.6), the final prediction value of a Random Forest is computed by summing and averaging the prediction values of the separate decision trees.

$$f(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} h_m(\mathbf{x}) \tag{2.6}$$

with M the number of trees and h_m the prediction function of a single decision tree.

2.5 Stacking Classifier

Stacking is a technique that can be applied to both regression and classification problems. A Stacking classifier is concerned with combining multiple classifiers into a single estimation [20]. The most basic form of stacking averages the predicted outputs of its input models into a final prediction. A more sophisticated approach, is to replace the averaging scheme for an estimator that gets trained on the predictions of the other models, to create a model that predicts with different weights for each separate sample. Ultimately, stacking is "a scheme for minimizing the generalization error rate of one or more generalizers" [20].

Chapter 3

Analysis, Preprocessing and Modeling

3.1 Overview

The main use case of machine learning is to make predictions on data. The process to go from data to trained models is described in this chapter. Chapter 4, follows this up with a description of the evaluation process of the predictive performance and how the predictions can be interpreted. Figure 3.1 illustrates how each of the different steps link to each other.

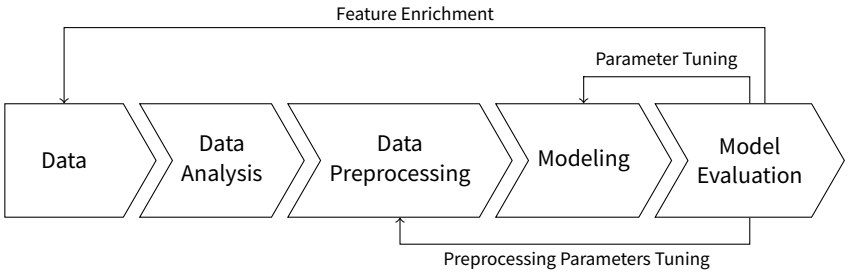


Figure 3.1: Illustration of machine learning workflow, where one starts with data and ends up in the evaluation stage with predictions. After the evaluation stage the models should be made ready for execution in a production environment.

The first step in the process is to acquire a dataset and analyze it. The dataset is provided and put together by the fraud experts that work for the collaborating insurance company. The dataset is analyzed to get an idea of the technologies that might perform well on the data and to get an idea of the challenges that lie ahead, see section 3.3.

With the challenges (section 3.4) in mind, the algorithms are chosen (section 3.6) and the data preprocessing steps are designed (section 3.5), and model training can begin.

After each training round, the predictive performance of the models is evaluated. The evaluation stage provides feedback on the chosen

preprocessing steps and algorithms. The effects of parameter tuning on the predictive performance is also evaluated. Furthermore, model evaluation also helps to identify new features that can be added to the dataset to improve model performance. Section 4.2, elaborates on the tasks carried out in the evaluation stage. The steps above are repeated until the models meet the requirements and are ready for release in production. Sections 4.3 and 4.4, conclude the approach with a look at the final workflow and the interpretation of model predictions.

3.2 Rule-Based Fraud Detection System

Before moving on to the rest of the machine learning work flow that is described above, it is helpful to understand the current system that is in place at the collaborating insurance company to detect fraud. Machine learning models can help to improve existing processes. However, in order to gauge the performance of a machine learning model, one needs a baseline. For this research, this happens to be the rule-based detection system that is currently installed at the collaborating insurance company.

The rule-based detection system operates by evaluating, for each claim, a set of rules. Whenever the conditions of a rule are satisfied, a weighted value is added to the output score of a claim. After evaluation of all the rules, the output score is converted, with the help of thresholding, to a 4-value *fraud score*—ranging from 0 (non-fraudulent) to 3 (fraudulent). This fraud score is then used to decide if a claim is worth investigating or not.

Currently, the rules that the system uses are rules that fraud experts perceive as important, based on their experience with previous fraudulent claims. This is not necessarily a bad thing, however, the system contains no feedback that checks the effectiveness of a certain rule or condition. A rule might increase the amount of fraud caught, but also double the workload, because a rule is too general and flags a lot of non-fraudulent claims. This is where machine learning can help, since machine learning algorithms are able to identify more complex relationships and more nuanced interaction between features; generating more complex condition structures that better describe the root cause of why a claim is fraudulent.

Since the rule-based system is just a collection of rules and conditions, the underlying decision making is easy to interpret. After the system awards a fraud score, the fraud experts get to see the rules that counted towards the final output. This way fraud experts can quickly decide if the claim is worth investigating, and it also helps targeting the investigation at the right variables. Fraud experts consider this model transparency as one of the key features of the rule-based system. Section 4.4, describes solutions to also add this transparency to machine learning models.

3.3 Data Acquisition, Analysis and Partitioning

The data-driven nature of machine learning makes data acquisition a logical first step in the process. With data acquisition, one makes a preselection of

features that seem valuable to solve the task at hand—detecting fraud for car insurance claims.

Data quantity and quality are both important in this step. Quality-wise, including too many features unrelated to the task can pose problems later on when training machine learning models. To ensure that the features are related to the detection of fraudulent claims, a fraud expert helps to make a selection of features that seem reasonable to include. Quantity-wise, more data instances is generally better, as the machine learning models have more data to train on. The sections below contain an elaborate description of the dataset and how it is partitioned to make it ready for machine learning.

3.3.1 The Dataset

The dataset assembled for this research contains 161,247 records that each represent a claim. Each claim is represented by 76 features, of which 54 are scalar and 22 are categorical. Examples of scalar features are: insurer age, insurance premium, car's value, etc. Examples of categorical features are: car brand, car color, etc. All features are provided anonymized with blind feature names (e.g. *x0*) and with encoded values. Scalar features are encoded by removing the mean and scaling to unit variance; also called *standard scaling*. Categorical features are encoded with numerical codes; also called *integer encoding*.

From all these records, roughly 0.3% (519 claims) is labeled as fraud, meaning that the dataset is highly imbalanced. A couple of reasons for this high imbalance are:

- Only a small percentage (3-6.4%) of claims is actually fraud [3].
- In the provided dataset only *proven fraud* has been labeled as fraud, (highly) suspicious uninvestigated claims have similar labels as legitimate claims. It is therefore not possible to distinguish *suspicious* claims from other *non-fraudulent* claims, by label, in the dataset.

Figure 3.2, illustrates the label distribution of the dataset, this visual impression makes the magnitude of label imbalance even more apparent. Data imbalance plays an important role throughout the rest of this thesis and influences the choice of algorithms, model evaluation etc.

3.3.2 Data Partitioning: Train, Validation and Test

To determine the predictive performance of a machine learning model, the dataset needs to be partitioned into at least a *train* and *test* set. The train set is used to fit a model, and the test set is used to determine its predictive performance. With this simple way of partitioning, one runs into problems with the tuning of model parameters. Whenever the test set is used for both tuning parameters and estimating predictive performance, one is basically polluting the model with prior knowledge of the test set. Giving the model an unfair performance boost that does not generalize well to other parts of the data. To mitigate this effect, a *validation* partition must be added, on which

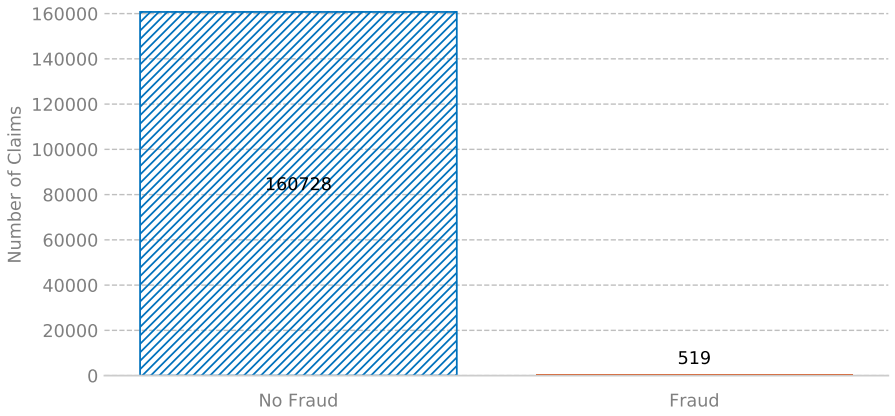


Figure 3.2: Label distribution of the entire dataset

the model parameters are tuned. The test set now becomes a hold out set that is only used to estimate the final predictive performance of the model. Below the use cases of each subset are described:

- **Train**: This subset is solely used to train a model.
- **Validation**: This subset is used to estimate model performance. Additionally, it is used to tune model parameters, in an attempt to increase a model's predictive performance.
- **Test (hold out)**: This subset is used to validate model performance on data that the trained model has never seen before and data that has also not been used to tune any of the model parameters. The performance on this set can be considered a good estimation for how well the model will perform in a production environment.[1]

The problem with using a single test set is that the predictive performance is based on a single partition and this does not guarantee that the performance will be similar on other partitions. By checking the difference in predictive performance between both the validation and test set, one can attempt to draw a conclusion on whether the model generalizes well to different parts of the data. However, as mentioned before, the predictive performance on the validation set is biased. Moreover, checking generalization on only two data partitions is not always enough, especially when dealing with models with high variance. In order to gain more confidence about the predictive performance of a model and whether it generalizes well to other parts of the data; cross validation can be used.

[1] To ensure that the predictive performance on the test set is not boosted by a "lucky" split, it is important to check whether a model's predictions generalizes well to other parts of the data.

3.3.3 Cross Validation

Cross validation is a way to estimate the confidence bounds of a model's predictive performance, these confidence bounds indicate how well the predictions of a model generalize to other parts of the data. Cross validation can be performed in many different ways, for this research, k-fold cross validation is chosen, because it allows for the highest amount of validations with the smallest loss of training data.

With k-fold cross validation, the train and validation sets are merged and randomly partitioned into k equal subsets. From these k subsets, 1 subset is used for performance validation and the other $k - 1$ subsets are used to train the model. Training is repeated k times using every subset once for validation. Choosing the value of k is a science on its own, since there is a bias-variance trade-off (section 2.4.1) associated with the choice of k in k-fold cross-validation. In this thesis the merely standard 5 or 10-fold cross validation is used, because: "as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance" [21]. Figure 3.3, illustrates the data partitioning for 10-fold cross validation.

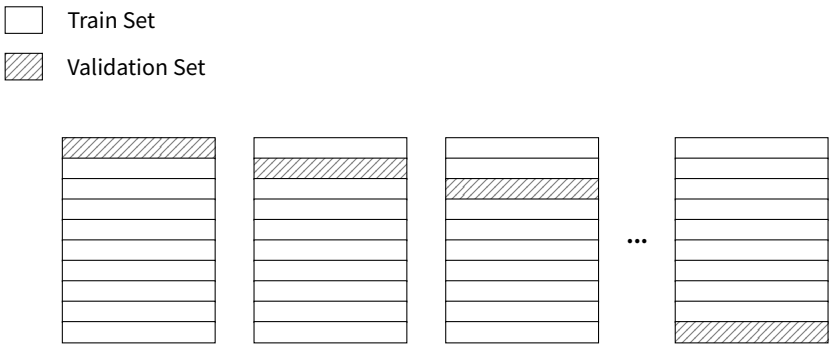


Figure 3.3: A visual representation of 10-fold cross validation. The dataset is split up in 10 folds, from which 10 distinctive datasets are created; consisting of 9 train folds and 1 validation fold.

After cross validation, one has k results that can be converted to a single performance estimation (mean) with the standard deviation. The final model is trained on all the data (train and validation combined) and is evaluated on the test set. One should expect the performance on the test set to be similar to the performance determined with cross validation.

3.3.4 Final Data Partitioning: TrainVal A, TrainVal B and Test

The final partitioning of the dataset consists of two train/validation pairs and one test set. This differs from the previously described partitioning that is discussed in the rest of this section. Instead of the regular train, validation and test partitioning, the dataset is partitioned into *train/validation A*, *train/validation B* and *test*, see figure 3.4. The reason for this is the use

of a Stacking classifier. Section 2.5 contains more information on what a Stacking classifier actually is, however, for now one just needs to know that a Stacking classifier uses the predicted output of other models as its input. Hence, the reason that an extra train/validation pair is needed to train the Stacking classifier on.

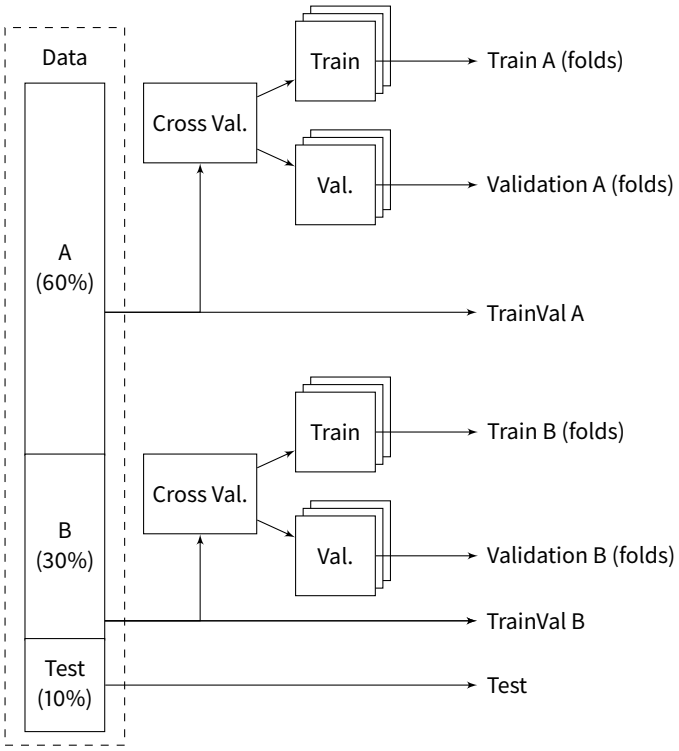


Figure 3.4: The final data partitioning of the dataset, and the names that belong to each partition. The dataset is split into three partitions, TrainVal A, TrainVal B and Test. Cross validation is also applied on TrainVal A and TrainVal B to measure the generalization of the predictions.

All models except the Stacking classifier are trained and validated on the *A* subset using 10-fold cross validation. In line with section 3.3.3, all final models are trained on the entire *A* subset. After which, their predictive performance is checked on the *test* subset.

The Stacking classifier is trained and validated on the *B* subset using 5-fold[2] cross validation. The models that are fit on subset *A* are now used to provide the input for the Stacking classifier, by performing predictions on the values of subset *B*. The *test* subset is again used to check the predictive performance of the Stacking classifier.

Table 3.1, shows the final label distribution and size for each of the subsets after the final partitioning round and figure 3.4 illustrates the final partitioning of the dataset.

[2]Due to the lower claim count in subset *B* as compared to subset *A*, 10-fold cross validation would leave too little fraud cases to train on.

Table 3.1: Label distribution across different splits

	# Fraud	# Non-Fraud	Fraud (%)	Size of Subset (%)
TrainVal A	312	96436	0.32	60.00
TrainVal B	161	48213	0.33	30.00
Test	46	16079	0.29	10.00

3.3.5 New Features through Claim Analysis

Lastly, data acquisition has a tight feedback loop with the prediction stage, certain claims are used to further enrich the dataset with new features based on their predicted fraud probability. A fraud expert inspects these claims and extracts data features that strongly profile these claims. Any features that are not present in the dataset yet are added for the next iteration. Section 4.2 contains more information on this process and explains how claims are filtered to look for new features.

3.4 Challenges

The three main challenges that this thesis faces are:

- **Imbalanced data**
  As mentioned in section 3.3.1 the dataset is highly imbalanced—only a small portion of the claims in the dataset is labeled as fraud. Imbalanced data complicates model training, pressing the need for data resampling and label weighting; see section 3.5.2. Moreover, imbalanced data also complicates the usage of metrics to evaluate the predictive performance of a model. Section 4.2.3 explores the most widely used performance metrics and tests them for their ability to give useful insights on an imbalanced data problem.

- **Unlabeled fraudulent/suspicious claims**
  Apart from data imbalance, section 3.3.1 also mentioned that suspicious claims are labeled as non-fraud in the dataset. Making it impossible to distinguish non-fraud from suspicious claims, even though the goal of the fraud detection system is to flag both fraudulent and suspicious claims as well. This complicates both model training and evaluation. Section 3.5.1, makes an attempt at reducing this problem.

- **Comparison to baseline**
  The research goal states that the machine learning models should achieve better predictive performance than the rule-based detection system, by finding more fraudulent claims with a higher efficiency. However, beyond these requirements, which metric should be favored more: finding extra fraudulent claims or an increase efficiency; section 4.2.5 tries to answer this question.

3.5 Data Preprocessing

Where data acquisition is meant for an initial selection of features and records, data preprocessing enhances the data and prepares it for use with machine learning algorithms.[3] Data preprocessing is an important step where much of the model's performance is determined. Machine learning is not a magical tool, and needs good data to function; hence, data preprocessing needs to distill the right features and records that contain information about claim fraudulence, guiding the algorithm in the right direction. The following sections elaborate on the types of preprocessing that are being used.

3.5.1 Label Enrichment

Data labels are an important part of the dataset, they indicate if a claim is fraudulent or not. Supervised learning algorithms use this information during training. Data labels are also used to check the predictive performance of a model, to see how many labels are predicted correctly. Below a few ways are proposed to increase label quality and potentially restore some balance within the dataset.

Section 3.3, briefly mentioned that only *proven fraud* has been labeled as fraud and that *suspicious* and *legitimate* claims are treated equally. Even though, suspicious claims should fall within the fraudulent category, in the dataset suspicious claims are labeled as legitimate instead of fraudulent; complicating supervised learning and the evaluation of the predictive performance:

- In the case of supervised learning, a model gets punished for misclassifying claims as fraudulent that are labeled as non-fraudulent, even though they are suspicious and should be classified as fraudulent.
- The same goes for the evaluation of predictive performance, faulty labels will induce errors in the final performance measurements.

To mitigate the problems stated above, an attempt is made to identify past suspicious claims and relabel them to being fraudulent. Two variables are found that potentially profile claims considered as suspicious in the past.

- **Fraud score history**
  Internally fraud experts use a so-called *fraud score* that integrates with the legacy rule-based system. When a claim comes in, the rule-based system awards a score between 0 and 3 (higher is more fraudulent), after which, fraud experts mutate this score based on their findings. Mutations can be both incremental and decremental, and in some cases a multitude of mutations are applied. On closing a claim, only proven fraudulent claims get to keep a fraud score of 3 and receive the fraud label. Suspicious and legitimate claims, get mutated to a score between 0 and 2 and receive a non-fraud label—even when their current fraud score is 3.

[3] Each algorithm has its own tailored preprocessing pipeline

One could argue that a score of 2 seems more fraudulent than a score of 0, however, based on the reports of fraud experts, the scoring between 0 and 2 is unreliable. Since, the fraud score levels are not explicitly defined and scores not always get mutated when new findings are added to the case. As a results of this, one cannot rely on the fact that higher means more fraudulent. The last option left to use as a measure to identify suspicious claims, is to check if claims received a fraud score of 3 somewhere during the period while the case was active; this method only partially works. On the one hand, many claims receive a score of 3 at the beginning of the process by the legacy rule-based system, and fraud experts might not always mutate fraud scores even when claims are not suspicious; which makes a score of 3 not very useful. On the other hand, mutations from a lower score to 3, are almost always intentional, and indicate that a fraud expert suspected fraud.

Therefore, every processed claim that had a mutation of its fraud score from 0-2 to 3 somewhere in the history of the case, will be considered as fraudulent.

- **Claim research cost**
  For each processed claim, the so called *research cost* are available. These costs indicate how much money was spend on both fraud inspection and claim handling for a specific claim. According to fraud experts, research costs above a certain threshold can be considered as suspicious, since commitments to further case inspections are only made when a claim is highly suspicious.

  Therefore, every processed claim with research costs higher than a certain threshold, will be considered as suspicious.

It is important to note that label enrichment can only be applied to processed claims, it solely aids training (for supervised models), and provides valuable insights for the predictive performance on the validation and test data. Label enrichment does not transform the actual features of new incoming future claims. The resulting label count is shown in table 3.2. The dataset is still highly imbalanced, however, it now contains more than twice as many 'fraudulent' cases.

Table 3.2: Label distribution for different subsets after label enrichment. "Extra Fraud" indicates the number of times more fraudulent claims that are now present in the dataset.

	# Fraud	# Non-Faud	Extra Fraud	Fraud (%)
TrainVal A	839	95909	2.69	0.87
TrainVal B	417	47957	2.59	0.86
Test	132	15993	2.87	0.82

3.5.2 Data Resampling and Label Weighting

Apart from data sanitization and label enrichment, more can be done to cope with the data imbalance mentioned in section 3.3. Namely, data resampling and sample weighting; see sections 2.2.1 and 2.2.2 for more background on these topics. Through experimentation is found that the way of dealing with data imbalance greatly depends on the type of algorithm that will be used. In general a mix of both undersampling and oversampling in combination with label weighting yields the best results. For each algorithm, except for the Stacking classifier that is presented later on, the following preprocessing pipeline is used. The resampling ratios and label weights that work best, have to be found through experimentation.

1. The majority class (non-fraud) is randomly undersampled.
2. The minority class is oversampled using SMOTE. This helps to further restore class balance and makes fraud stand out more among the rest of the data.
3. Both classes are resampled using Edited Nearest Neighbors[4] (ENN), to remove noisy data points and to make class borders more explicit.
4. Label weighting is used to further shift the importance to the minority class.

3.5.3 Category Encoding

Multiple ways exist to represent categorical features in a dataset. In this particular dataset integer encoding has already been applied to every categorical feature (section 3.3). Integer encoding is most useful in cases where the ordered relationship between categories is considered important, in such cases integer encoding is usually referred to as ordinal encoding. For example: importance levels that get encoded as integers, where 2 is more important than 1 and 1 is more important than 0.

Whenever no ordinal relationship exists, ordinal encoding may cause poor performance, because a model assumes an ordering between categories that does not exist. In this case, a one-hot encoding scheme can be used to transform the integer representation into a separate binary column for each category; creating n features for n categories in one feature x, see equation (3.1). By treating every category as a separate 'yes' 'no' entity the ordinal relationship is removed.

$$\vec{x}_n = \begin{cases} 0 & \text{if } x \neq n; \\ 1 & \text{if } x = n. \end{cases} \tag{3.1}$$

A disadvantage of one-hot encoding is that for a single categorical feature, many one-hot encoded features are added—e.g. for a categorical feature with 5 categories, 5 one-hot encoded features are created.

[4]Through experimentation was found that ENN produced more stable results with less variation than Tomek Links.

For each model discussed in this thesis one-hot encoding is used as one of the preprocessing steps. Especially when using SMOTE it makes more sense to use one-hot encoding. For example, let's take the categorical feature *car color*. Imagine that the colors red and black are good indicators that something is fraud. Now, when using SMOTE to create artificial fraudulent claims. For the integer encoded variant of car color it might generate a claim with a blue car, because the integer value for blue happens to be between the values of red and black. Whereas for the one-hot encoded variant of car color, SMOTE will generate a claim that is both half red and half black. Even though the latter option is maybe not physical possible. An algorithm will still be enabled to pick up the signal that a red or black car is an indicator for fraud. Whereas with the integer encoded case, the algorithm is trained on wrong data that does not represent reality.

3.6 Modeling

After data analysis, the preprocessing steps and algorithms can be chosen. This section discusses the algorithms that are chosen to reach the objective, the reasoning behind the choice of algorithms, and which modeling steps are most important to focus on for each particular algorithm.

There are a wide variety of different machine learning algorithms and choosing the right one for the task is based on the dataset, the objective and experimentation. In an attempt to ease the process of algorithm choosing, some requirements that the algorithms should comply to are discussed. This is followed by a discussion of the chosen algorithms and the reasoning behind these choices.

3.6.1 Algorithm Requirements

Based on the objective set in section 1.2 and the challenges discussed in section 3.4, the algorithms should meet the following requirements:

- They must be suitable for binary classification;
- They must produce explanatory predictions; so that the feature contributions of a particular prediction can be retrieved. This will improve the process of claim evaluation for the fraud experts.
- They must contain parameters to deal with class imbalance.

Furthermore, following the discussion in section 2.1 on the types of machine learning models, a choice is made for using supervised learning algorithms. A supervised learning algorithm tries to predict a ground truth that it trained on with the use of labeled data, whereas an unsupervised model tries to learn a better representation or structure of the data without any ground truth as guidance. The reason to choose for supervised learning is because it provides more control over what the algorithms are trying to classify. Supervised learning allows to set the target to distinguishing between fraudulent and non-fraudulent claims, whereas unsupervised algorithms

might choose to separate claims based on other patterns than the fraudulence of a claim.

The machine learning algorithms that seem most appropriate to achieve the objective and that comply with the requirements above are: Logistic Regression, Random Forest and LightGBM. The sections below will discuss in more detail why these algorithms are chosen and which things one should focus on during model training.

3.6.2 Logistic Regression

Logistic Regression is probably the most standard classification algorithm that exists in the world of machine learning, and is also used for fraud detection [22], [23]. It is in fact so standard that it is common practice to always include a Logistic Regression model for binary classification.

As mentioned in section 2.3.2, Logistic Regression is prone to overfitting, which can be controlled with regularization (section 2.3.3). The Sci-kit learn implementation of Logistic Regression implements regularization, for which both the type of regularization (L1, L2) and its strength can be set. Furthermore, class weighting is available to deal with class imbalance. It also allows to set a weight for each sample separately, however in the case of binary classification there are no specific weights that need to be added to claims other than the class weights.

3.6.3 Tree-based Models

Since the data is provided with scaled features (section 3.3.1), it is very likely that some of the features were scaled out of proportion due to outliers—squeezing most values in a very narrow region with a few outliers with far higher values. Since the features are kind of black box, tree-based models are an obvious choice, because they are generally more robust to outliers and data transformations than other algorithms.

The final model setup contains two algorithms that are based on decision trees, namely, Random Forest (see section 2.4.3 for more background) and LightGBM. LightGBM is an implementation of gradient boosted decision trees (see section 2.4.2 for more background). Modeling for each follows a similar approach. Given the nature of tree-based algorithms, the key aspect of modeling is to control overfitting; section 2.4 explains why overfitting is such a big issue with tree-based models and the techniques that are applied to control it.

Both algorithms rely on combining trees to create a model with low bias and low variance; section 2.4.1 discusses the terms bias and variance. A Random Forest classifier grows many specialized trees on different subsets of the training data. Separately, each specialized tree has a low bias and high variance, however, by creating a forest of trees that uses a combination of many trees for a prediction, a model is created with low bias and low variance. LightGBM takes the opposite approach of Random Forest and uses trees with a high bias and low variance, so-called weak learners. LightGBM recursively

grows weak learners that reduce the error of the ones before it, to end up with a low bias and low variance.

Random Forest

Sci-kit learn's implementation of Random Forest is used. As mentioned before, algorithms based on decision trees are prone to overfitting, therefore modeling is mainly about controlling tree growth. Tree growth can be controlled by setting the maximum tree depth, minimum amount of samples per leaf and the maximum amount of features that can be used for each tree. To further reduce the variance, the number of trees can be specified.

As a decision criterion, talked about in section 2.4, Gini impurity is preferred over Entropy. Especially because Gini is faster to compute and in only 2% of the cases Entropy gives different results [15].

Lastly, just like is the case for the other models, the class weight is again tuned to make up for the class imbalance.

LightGBM

From the many implementations of gradient boosting trees, Microsoft's LightGBM is chosen for its speed and high level of configurability [24]. Modeling is mainly focused on choosing the right class weights and finding a balance between tree growth and generalization. Just like Random Forest, LightGBM enables the modeler to control tree growth by setting the maximum tree depth, minimum amount of samples per leaf and the maximum amount of features that can be used for each tree. Apart from the standard functionality of boosting trees, described by section 2.4.2, LightGBM also implements bagging. Bagging allows for the subsampling of features and/or data samples, to help improve generalization.

Furthermore, LightGBM supports a training mechanic called *early stopping* that can be used during the training phase to help prevent overfitting. It works by continuously validating performance[5] on a validation set during the training phase.[6] Whenever performance on the validation set stagnates, and performance is still increasing for the primary training data; overfitting is likely happening. After multiple concurrent training iterations without an increase in performance on the validation set, training will stop, and LightGBM will take the iteration with the highest performance for future predictions.

Its speed and configurability make LightGBM one of the most versatile algorithms that is suitable for any problem, hence the reason that is also applicable to fraud detection. Moreover, LightGBM being a tree-based model also allows for relative easy derivation of the feature contributions.

[5] Weighted binary logloss is used as performance metric for early stopping.

[6] To prevent performance bias on the actual validation set, a split of the training set is used as a validation set for early stopping

3.6.4 Stacking Classifier

As mentioned in section 2.5, the job of a Stacking classifier is to improve generalization while maintaining the performance of the other models [20], hence the reason that a Stacking classifier is included in the final model lineup. For this research, the Stacking classifier is actually a Logistic Regression model that takes the predictions of the other three models (Logistic Regression, Random Forest and LightGBM) as its input.

The modeling steps that were previously mentioned for Logistic Regression also apply to the Stacking classifier. The main difference is that no preprocessing steps will be used to transform the input space of the Stacking classifier. Class weighting, on the other hand, is still necessary to obtain a balanced prediction output.

Chapter 4

Model Evaluation and Interpretation

4.1 Overview

First, section 4.2 describes the model evaluation process. After the discussion on model evaluation, section 4.3 recaps how the training and prediction steps relate to each other. The last part of the approach is described in section 4.4, which is a discussion on how the predictions of linear and tree-based models can be interpreted, to prepare the predictions for display to the fraud experts.

4.2 Evaluation

In the evaluation stage the predictive performance of the models is measured and evaluated. This includes comparing different parameter versions of the same model, comparing models based on different algorithms and lastly comparing models to the baseline. Model evaluation for classifiers on imbalanced data can be done with many different metrics [25]. Section 4.2.1, introduces the basic performance metrics, followed by the establishment of the baseline performance in section 4.2.2. Section 4.2.3, discusses the most widely used performance metrics and why or why not they are suitable to an imbalanced learning problem. Sections 4.2.4 and 4.2.5, follow this up with a description of the approaches used for model to model comparisons and comparisons to the baseline. Lastly, in section 4.2.6, an approach to improve the quality of the dataset is discussed.

4.2.1 Basic Performance Metrics: Confusion Matrix, Recall, Precision and Specificity

A confusion matrix, is a tool to visualize the classification performance of machine learning models, by presenting the four base classification statistics in a tabular fashion. Table 4.1, shows an example of a confusion matrix. Below follows an explanation of the four base statistics that are present in the confusion matrix, and all other metrics are based on:

Table 4.1: Confusion matrix, a tabular representation of the intersection between actual labels and predicted values

		Actual	
		Positive (P)	Negative (N)
Predicted	Positive	TP	FP
	Negative	FN	TN

- **TP**, true positive, number of labeled positives (*labeled fraud*) that are correctly classified as positive (*predicted fraud*)
- **FN**, false negative, number of labeled positives (*labeled fraud*) that are incorrectly classified as negative (*predicted non-fraud*)
- **FP**, false positive, number of labeled negatives (*labeled non-fraud*) that are incorrectly classified as positive (*predicted fraud*)
- **TN**, true negative, number of labeled negatives (*labeled non-fraud*) that are correctly classified as negative (*predicted non-fraud*)

The relationship of these statistics is described by equation (4.1).

$$\begin{aligned} FN &= P - TP \\ FP &= N - TN \end{aligned} \tag{4.1}$$

With P the number of real positive cases, and N the number of real negative cases in the data. From this base, three metrics (recall, precision and specificity) are defined that break down classification performance, for a complete overview.

Recall (also called Sensitivity) indicates the fraction of labeled positives (*fraud*) that are identified as positive, see equation (4.2).

$$\text{recall} = \frac{TP}{TP + FN} \tag{4.2}$$

For fraud detection, recall is useful to measure the fraction of fraudulent claims that are identified by a machine learning model.

Precision is the fraction of correctly classified labeled positives (*fraud*) among all classified positives, see equation (4.3).

$$\text{precision} = \frac{TP}{TP + FP} \tag{4.3}$$

A low precision indicates a high false positive ratio, meaning that many instances classified as positive are actually negative. For fraud detection, precision indicates how many of the flagged claims are actually fraud, and thus says something about the amount of claims fraud experts need to check before they find fraud.

Specificity measures the fraction of labeled negatives that are correctly identified as negative, see equation (4.4).

$$\text{specificity} = \frac{TN}{TN + FP} \tag{4.4}$$

Specificity indicates the ratio of negatives that has been correctly classified as negative, basically recall on the negative instead of positive class. Since, fraud detection is all about catching fraud (positives), recall is a more intuitive metric than specificity to evaluate a fraud detection algorithm.

4.2.2 Baseline Performance

Baseline performance is constructed using the legacy rule-based system (discussed in more detail in section 3.2) that is currently installed at the collaborating insurance company to detect fraudulent car insurance claims. Baseline performance can be measured in many different ways and the choice of metrics is dependent on the application. For fraud detection, one is usually interested in increasing the amount of fraud caught and reducing the effort to find fraud.

From the metrics specified in section 4.2.1, the two metrics that seem most suitable for the application are:

- **Recall**, the percentage of known labeled frauds that a model is able to classify as fraudulent. Recall is be able to keep track of the amount of fraud that gets caught.
- **Precision**, the percentage of correctly classified claims that are flagged as fraudulent. It is the ratio between true positives and false positives. Precision indicates the quality of the predicted fraud, in other words, how much effort it takes to find a fraudulent claim between all the claims that are predicted as fraud.

Having decided on the metrics, the next important step is to create a baseline to which the machine learning models can be compared and evaluated. Using *fraud score*—awarded to every incoming claim by the rule-based system (section 3.3)—as a threshold to classify claims, one can calculate the precision and recall for each of the fraud score thresholds. Every claim with a fraud score similar or higher than that of the fraud score threshold, is flagged as fraud. The obtained labels are then compared to the actual labels, after which, precision and recall for the particular threshold can be calculated. Table 4.2, lists the obtained performance metrics of the rule-based system for three fraud score thresholds.

Table 4.2: Precision and recall of rule-based system for multiple fraud score thresholds on the TrainVal A set

	Recall	Precision	Flagged (%)
Fraud Score: >= 1	0.94	0.01	93.66
Fraud Score: >= 2	0.44	0.04	8.17
Fraud Score: >= 3	0.15	0.06	1.86

One can see that a fraud score threshold of 1 catches almost all currently known frauds (94%, 0.94 recall), however due to the very low precision, more than 93% of claims get classified as fraud. This leaves the fraud experts with

almost all claims being suspicious, which is not very helpful. A fraud score threshold of 2 produces more useful results. Investigating 8% of all incoming claims results in catching 44% of all currently known fraud. However, more than half of the fraudulent claims are left undetected, indicating that many fraudulent claims get detected due to human intervention while processing the claim. With a fraud score threshold of 3, about 15% of currently known fraud gets detected and only 2% of all incoming claims needs to be checked. Making it useful for easy catches, but most of the fraudulent claims stay untouched.

In practice, fraud experts consider a *fraud score* of 2, on an incoming claim, to be a good threshold, and anything with a lower score is considered not to be worth investigating. Therefore, the performance metrics belonging to a fraud score threshold of 2, are used to evaluate the final performance of the machine learning models.

4.2.3 Alternative Performance Metrics

As mentioned in section 4.2.2, precision and recall provide the best overview to evaluate the ability of a model to detect fraud. Recall keeps track of the amount of fraud caught, and precision indicates with what kind of efficiency fraud is caught. Recall serves as quantity control, to guarantee a certain amount of fraud being detected. Precision serves as quality control, a higher precision will mean more fraudulent claims among all claims classified as fraud, and thus leaving less work for the fraud experts.

Precision and recall can both be computed for the baseline, so it is possible to compare models directly with precision and recall. However, even though these two metrics give a complete picture, from a comparison point of view, it is desirable to have a single metric to describe model performance. With a single metric, performance is well defined—a higher score indicates better performance. With two metrics on the other hand, performance evaluation is not as clear-cut. It is obvious that a model has better performance when both metrics are higher, however, the comparison becomes ill defined whenever one metric is higher and the other one is lower.

Below follows an explanation and analysis of the performance metrics that are most widely used in the context of imbalanced learning [26]. The metrics are tested for their viability of replacing precision and recall with a single metric estimation on highly imbalanced data (0.03 ratio). In the dataset, fraud belongs to the positive class and non-fraud belongs to the negative/majority class. Therefore, classification performance on the positive/minority class is more important than on the negative class, after all, one wants to evaluate how good a model is at classifying fraud. This is an important consideration, since many metrics are biased towards the majority class (non-fraud), when dealing with imbalanced data. That is, for increasing data imbalance, the minority class performance has a decreasing impact on the metric score. Therefore, each metric is reviewed for its behavior on imbalanced data and whether it produces viable output scores.

A metric is considered viable when it summarizes performance

on the positive class in a way that no situation can occur where a model's performance converges to a point that yields poorer fraud detection performance than the baseline i.e. quantity and/or quality-wise worse than baseline.

To find a metric that summarizes precision and recall for easier performance comparisons. Heat maps—e.g. figure 4.1—are created with metric scores for all possible combinations of precision and recall on balanced and imbalanced data. The heat maps will give an impression of how a metric summarizes precision and recall, and how it is affected by data imbalance.

Accuracy

Accuracy is the fraction of correctly classified instances among all instances, see equation (4.5).

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.5}$$

Intuitively, accuracy seems to be a good metric to estimate the predictive performance of a model. However, it weighs correctly classified minority and majority instances equally, which poses a problem when dealing with (highly) imbalanced data. As the following example illustrates, accuracy can give very distorted performance indications for imbalanced learning problems; hence accuracy is considered not to be applicable to this problem.

> **Example: Accuracy failing on imbalanced learning problem**
> Given an imbalanced dataset with a minority ratio of 1%. In case a model predicts every instance to be part of the majority class, one ends up with an accuracy of 99%. This hints at great predictive performance, however, the prediction accuracy on the minority class is 0%. For fraud detection, this means that an accuracy of 99% can still mean that 0 fraudulent claims are being identified.

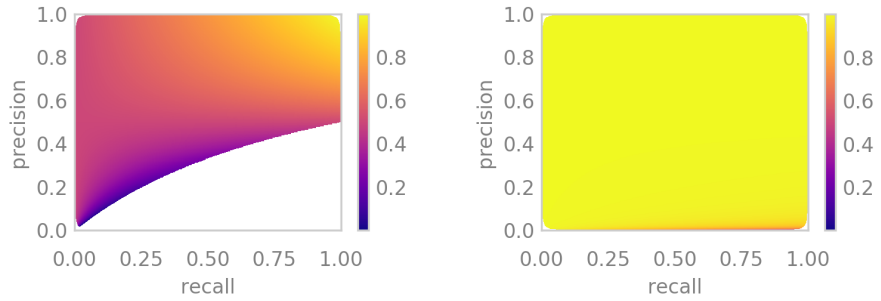
Figure 4.1, confirms that precision and recall are not reflected in the accuracy score for highly imbalanced data. The majority class is simply too big to be affected by performance changes in the minority class.

F-score, b-varied F-score

F-score [27] is a metric that attempts to summarize the quality of classification on (mainly) the positive class, because F-score is based on the two metrics (precision and recall) that describe the positive classification performance. Standard F-score (F_1 score) is the harmonic mean[1] of precision and recall, given by equation (4.6).

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.6}$$

[1] Because one is dealing with fractions, it makes sense to use the harmonic mean, as the harmonic mean can be described as: the reciprocal of the arithmetic mean of reciprocals.



(a) Balanced data: Accuracy is sensitive for value changes of both precision and recall.

(b) Imbalanced data (ratio 0.003): Accuracy is insensitive to value changes of either precision or recall.

Figure 4.1: Accuracy scores for all possible combinations of precision and recall on both balanced (a) and imbalanced (b) data.

Even though F-score is focused on the positive class, it is still able to also summarize model classification performance for both classes on a balanced dataset. Since, precision is directly linked to the amount of *true negatives (TN)* via the *false positives (FP)* (equation (4.3)). For highly imbalanced data, however, the relationship between *TN* (relatively large) and *FP* (relatively small) gets out of proportion, biasing F-score more towards the positive class.

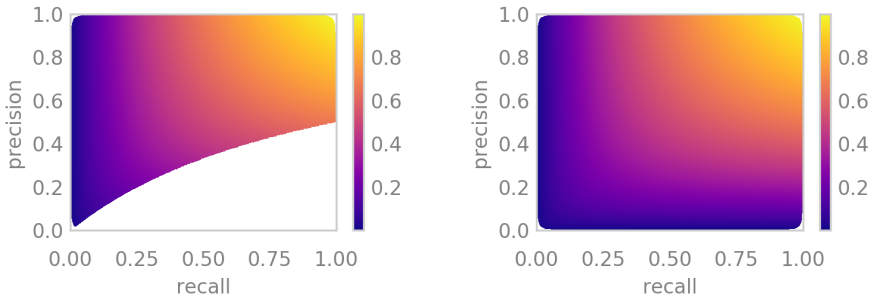
Since F-score is only dependent on precision and recall (equation (4.6)), it is obvious that F-score shows the same relationship to precision and recall, regardless of class imbalance; illustrated by figures 4.2a and 4.2b. F-score is able to optimize performance when precision and recall are considered equally important. However, the equal treatment of precision and recall poses a problem for the detection of fraud, because for a similar F-score one might obtain a model with either *high recall with low precision* or *low recall with high precision*. In the first case a lot of fraud is detected with a low efficiency, and in the second case little fraud is detected with high efficiency. This makes it difficult to guarantee better performance (higher precision and recall) than the baseline for a higher F-score.

A more general defined variant of F-score is β-varied F-score, which contains functionality to alter the relative importance between precision and recall; with β controlling this behavior, see equation (4.7).

$$F_\beta = (1+\beta) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{4.7}$$

- For $\beta < 1$, F-score becomes more biased towards precision
- For $\beta > 1$, F-score becomes more biased towards recall

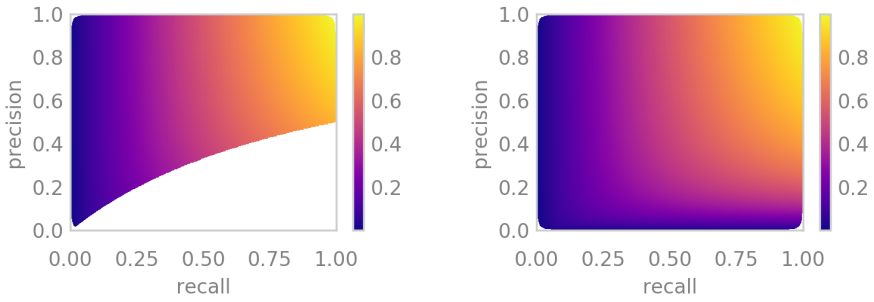
Figure 4.3, illustrates how a β of 2 transforms the score space. This extension of F-score helps to overcome the previously stated problem, of F_1. However, choosing β is not an exact science, so it is difficult to decide how large β should be.



(a) Balanced data: F_1 is equally sensitive to value changes of both precision and recall.

(b) Imbalanced data (ratio 0.003): F_1 is not affected by class imbalance.

Figure 4.2: F_1-scores for all possible combinations of precision and recall on both balanced (a) and imbalanced (b) data.



(a) Balanced data: F_2 is biased more to recall than to precision.

(b) Imbalanced data (ratio 0.003): F_2 is biased more to recall than to precision and not affected by class imbalance.

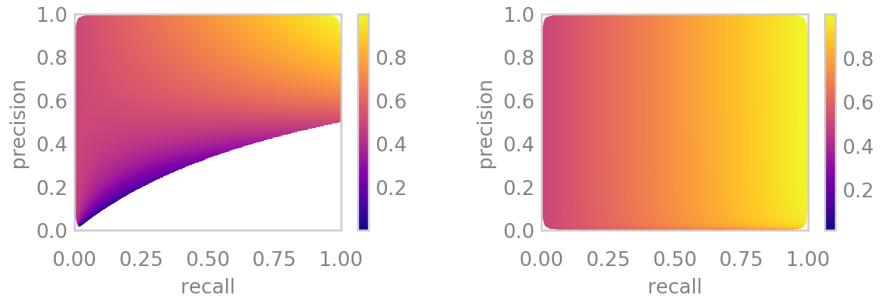
Figure 4.3: F_2-scores for all possible combinations of precision and recall on both balanced (a) and imbalanced (b) data.

Balanced Accuracy

Balanced accuracy [28] describes the average accuracy of both classes, by taking the average of sensitivity and specificity, see equation (4.8).

$$\text{balanced accuracy} = \frac{\text{sensitivity} + \text{specificity}}{2} = \frac{\frac{TP}{TP+FN} + \frac{TN}{TN+FP}}{2} \tag{4.8}$$

Balanced accuracy produces similar output as classical accuracy whenever the obtained accuracy is the same for both classes. However, unlike classical accuracy, balanced accuracy is able to reflect a change in either the positive or negative class accuracy, regardless of class imbalance. Figure 4.4, confirms this, because recall (accuracy of minority class) affects the output—the score is increasing along the recall axis—for both the balanced and imbalanced case. This is the main advantage over conventional accuracy, since conventional accuracy does neither reflect precision or recall in its output when dealing



(a) Balanced data: Balanced accuracy is, like conventional accuracy, sensitive to both precision and recall.

(b) Imbalanced data (ratio 0.003): Balanced accuracy is only sensitive to recall and negligible small values of precision.

Figure 4.4: Balanced accuracy scores for all possible combinations of precision and recall on both balanced (a) and imbalanced (b) data.

with imbalanced data; illustrated by figure 4.1. Despite this advantage, balanced accuracy does not reflect precision in its output for the imbalanced data case—figure 4.4b shows a gradient along the recall axis. In order for precision to have an effect on balanced accuracy, it needs to become negligible small. Because of this, balanced accuracy is unable to correctly summarize model performance for the fraud detection problem.

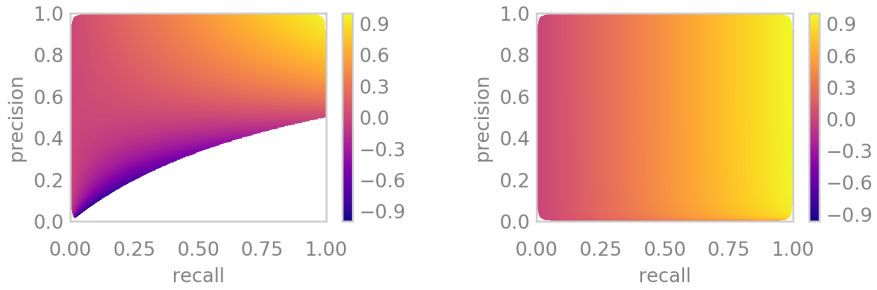
Youden's Index

Youden's index (also indicated by γ, [29]) is another attempt to summarize performance with a single metric. The index tells something about the proportion of combined misclassified instances for both classes, and is described by both sensitivity and specificity, see equation (4.9).

$$\gamma = \text{sensitivity} - (1 - \text{specificity}) = \frac{TP}{TP + FN} + \frac{TN}{TN + FP} - 1 \tag{4.9}$$

Its output ranges from -1 to 1, with 0 as a baseline when similar proportions of both classes are classified as positive. An output of 1 is awarded when no false positives or negatives are present, i.e. the model has perfect performance; for -1 vice versa.

Youden's index equally weights false positives and negatives, and thus it is impossible to conclude from the index score which class has more misclassifications. This becomes a serious issue with imbalanced data, illustrated by figure 4.5. Figure 4.5a, shows that Youden's index reflects both precision and recall in its output for the balanced data case. However, as one can see in figure 4.5b, for highly imbalanced data Youden's index only contains information about the recall performance of a model. Precision has to become negligible small in order to be reflected in the output, meaning that the quality of the classification on the positive class is not reflected in Youden's index for highly imbalanced data. Making Youden's index not a good metric to summarize the performance of fraud detection models.



(a) Balanced data: Youden's index is sensitive to both precision and recall.

(b) Imbalanced data (ratio 0.003): Youden's index is only sensitive to recall and negligible small values of precision.

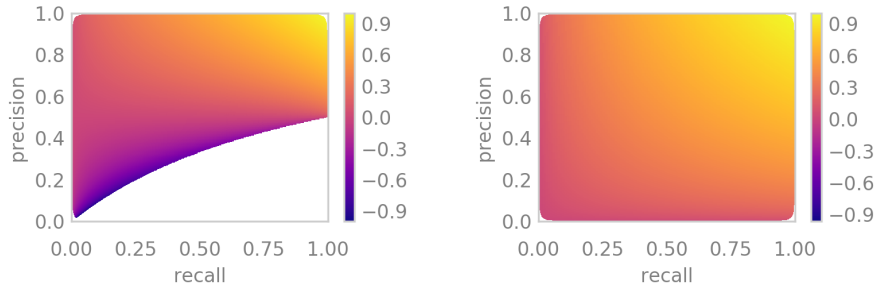
Figure 4.5: Youden scores for all possible combinations of precision and recall on both balanced (a) and imbalanced (b) data.

Matthews correlation coefficient (MCC)

Matthews Correlation Coefficient (MCC) [30] is a single performance metric that measures the quality of a binary classification. It is regarded as a measure that is less impacted by imbalanced data, because it considers all aspects of the confusion matrix.[31] Essentially, MCC is a correlation coefficient between the actual and predicted classification , which is returned as an outcome between 1 and -1; see equation (4.10). A value of 1 indicates a perfect prediction, 0 similar performance as random, and -1 everything wrongly predicted.

$$MCC = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP)\,(TP + FN)\,(TN + FP)\,(TN + FN)}} \tag{4.10}$$

Even though a confusion matrix can never be fully replaced by a single number, MCC is regarded to be one of the best singular assessment metrics, especially when dealing with imbalanced data.[32] Figure 4.6, helps to identify the strengths and weaknesses of using MCC, it illustrates the MCC scores for all combinations of precision and recall on highly imbalanced data (minority class of 0.3%). As one can see, MCC's score increases equally for both precision and recall, and shows no distortion induced by imbalanced data. For some cases this might be desired behavior, however, for the fraud detection case, MCC gives a wrong perception of the actual performance. After all, a *high recall with low precision* is very different from *low recall with high precision*. In the first case, a lot of fraud is detected with a low efficiency In the second case little fraud is detected with a high efficiency. MCC is therefore unable to guarantee that an MCC score higher than the baseline, always means that neither precision or recall are lower than the baseline. This violates the previously defined viability requirement, and thus MCC is not applicable to summarize the performance of fraud detection models.



(a) Balanced data: MCC is equally sensitive to value changes of both precision and recall.

(b) Imbalanced data (ratio 0.003): MCC is hardly affected by class imbalance.

Figure 4.6: MCC scores for all possible combinations of precision and recall on both balanced (a) and imbalanced (b) data.

Receiver Operating Characteristic (ROC) Curve

A receiver operating characteristic curve, i.e. ROC curve [33], is a graphical tool to evaluate model performance on a binary classification problem for a varying classification threshold.

A ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) for each classification threshold. Recall and sensitivity are alternative names for true positive rate, and TPR is therefore specified by equation (4.2). The false positive rate is specified as (1 − specificity), and more elaborately by equation (4.11).

$$FPR = 1 - \text{specificity} = \frac{FP}{FP + TN} \tag{4.11}$$

Saito et al. [34] shows that the visual interpretability of ROC curves in the context of imbalanced datasets can be deceptive with respect to conclusions about the reliability of classification performance. Section 4.2.3, contains an example that illustrates why a ROC curve can be deceptive with imbalanced data.

Precision-Recall Curve (PRC)

A precision-recall curve (PRC) is a graphical representation of a model's classification performance on the positive class, expressed by precision/recall pairs. By varying the classification probability threshold, it is possible to obtain many precision/recall pairs that can be plotted, to form a curve with recall on the x-axis and precision on the y-axis. PRC provides a way to judge the classification performance of a model over the entire recall space, instead of just a single probability threshold.

Davis et al. [35] states that one can better use a precision-recall curve than a ROC curve for imbalanced data. To illustrate this, an example is given that shows the ROC and PRC scores measured on a single point for two different models on an imbalanced dataset.

Example: PRC is more sensitive to positive class than ROC for imbalanced data
Model 1: 100 retrieved claims, 90 are fraudulent
Model 2: 2000 retrieved claims, 90 are fraudulent
Model 2 has significantly worse performance, since the workload is 20 times higher for the same amount of fraud. The scores for the ROC curve become:
Model 1: TPR: 0.9, FPR: 0.00001
Model 2: TPR: 0.9, FPR: 0.00191 (difference: 0.0019)
The scores for PRC become:
Model 1: Recall: 0.9, Precision: 0.9
Model 2: Recall: 0.9, Precision: 0.045 (difference: 0.855)

As one can see, the score difference between the models is significantly higher in the case of the precision-recall curve; the FPR shows only a slight change. Obviously, these are single point scores, however, when this imbalance persists across various classification thresholds, the precision-recall curve is a better choice than the ROC curve for imbalanced datasets.

For easy comparison of precision recall curves, one could consider using the area under curve method.

Area Under Curve (AUC)

Area under curve (AUC) converts a curve to a single metric approximation, by approximating the area underneath a curve; simplifying the comparison between multiple instances of a graphical metric. Intuitively, a larger AUC usually indicates better performance, since a bigger area indicates that the overall values of the curve are higher.

AUC is computed with the trapezoidal rule that tries to estimate the definite integral. The trapezoidal rule works by approximating the area under the curve as the area of a trapezoid. By partitioning the curve and fitting multiple trapezoids, the approximation becomes more accurate. The smaller the partition size, the more accurate the approximation becomes. The trapezoidal rule is defined by

$$\sum_{k=1}^{N} \frac{f(x_{k-1}) + f(x_k)}{2} \Delta x_k \tag{4.12}$$

with Δx_k the horizontal distance between points x_{k-1} and x_k—in other words Δx_k is the width of partition k.

In many cases AUC is used in combination with ROC, however, ROC is found to be an unreliable measure to compare model performance in an imbalanced data setting. Therefore, AUC will also yield unreliable results when used to compare ROC curves. Fortunately, AUC is still a viable metric to compare other more meaningful curves, such as the precision-recall curve.

4.2.4 Model Optimization and Comparison

With model optimization one compares the same algorithm with different parameter setups and/or preprocessing pipelines, to find the configuration with the best predictive performance. Model comparison involves comparing models based on different algorithms. Again, to see which model has the best predictive performance. In this thesis model optimization and comparison are done in the same way.

Section 4.2.3, discusses the most widely used metrics to measure predictive performance for imbalanced learning problems. From this discussion, three metrics can be distilled that summarize precision and recall, and hold up with data imbalance; F-score, Matthews correlation coefficient (MCC) and the area under curve of the precision-recall curve. F-score and MCC are both point estimates, measured for a single classification threshold, whereas the precision-recall curve covers the performance of the entire prediction space.

Section 4.2.3 already described that the standard version of F-score, F_1, and MCC both treat precision equally to recall. From this follows that both measures can optimize a model to both a *high recall with low precision* or a *low recall with high precision*, for the same metric score. In the first case a lot of fraud is detected with a low efficiency, and in the second case little fraud is detected with high efficiency. This layer of abstraction removes transparency for the modeler, and might cause a model to be wrongly optimized. This same problem partially applies to the area under curve of the precision-recall curve, after all, the area under curve can be the same for a peak in precision for either a high or low recall. However, by showing the precision-recall curve in combination with the AUC score, it is immediately obvious what the precision and recall scores are like.

Furthermore, since the models are going to be used in a production environment, in which model requirements might change. It is good to evaluate a model on its overall prediction performance for all the classification thresholds, in case that the classification threshold needs to be moved to change a model's performance—e.g. get a higher recall. The area under curve of the precision-recall curve summarizes the overall performance better than the other two metrics.

The precision-recall curve in combination with the area under curve, maintains transparency through visual feedback, while also allowing for easy comparison with a single metric estimation of the entire prediction space.

4.2.5 Baseline Comparison

The last step in the evaluation process is to compare the models to the baseline. Baseline comparison is useful to estimate the actual performance improvements that the models bring about. The increase in predictive performance is evaluated by comparing the performance of the models on the validation sets[2] and the test set[3] against the baseline performance.

[2]There are multiple validation sets, because cross-validation is used.

[3]The test set is only used for the final model comparison, to prevent bias.

Baseline performance (section 4.2.2) is defined as the classification performance of the rule-based system for a *fraud score* higher than 2, for which multiple metrics can be computed; see table 4.2. Since the baseline performance is expressed in both precision and recall, it is clear that a model should have at least a recall and precision that is higher than the baseline. However, beyond this requirement, it is difficult to decide which one of the metrics should be considered more important. After all, a higher precision will lower the workload and a higher recall will increase the amount of fraud caught; both are important.

By moving a model's classification threshold, it is possible to favor either precision or recall, creating a range of possible thresholds. Therefore, the baseline will be defined as two performance boundaries that constrain this range of possible thresholds:

- **Maximum effort**
  The effort fraud experts currently spend on checking all claims that are classified as fraud by the baseline.
- **Minimum recall**
  The level of recall, or the current amount of fraud currently caught by the baseline.

These two boundaries are expressed in recall and effort. Effort is given by equation (4.13), and is incorporated into precision, see equation (4.14). Effort is not really a metric in the sense that it depends on the size of the dataset, hence the reason that a precision-based metric (precision-recall curve) is still used for model to model comparisons between different subsets of the data.

$$\text{effort} = TP + FP \tag{4.13}$$

$$\text{precision} = \frac{TP}{TP + FP} = \frac{TP}{\text{effort}} \tag{4.14}$$

Models are compared to the baseline by computing the *fraud increase* and *effort reduction*, given by equations (4.15) and (4.16).

$$\text{fraud increase} = \frac{\text{recall}_{\text{model}} - \text{recall}_{\text{baseline}}}{\text{recall}_{\text{baseline}}} \cdot 100\% \tag{4.15}$$

$$\text{effort reduction} = \frac{\text{effort}_{\text{baseline}} - \text{effort}_{\text{model}}}{\text{effort}_{\text{baseline}}} \cdot 100\% \tag{4.16}$$

By computing the above metrics for the classification thresholds that lie between the boundaries, one obtains two curves for fraud increase and effort reduction. These curves are then normalized on the x-axis, so that only the height of both *fraud increase* and *effort reduction* count towards the quality of the model. Then, the area under curve is computed for each model separately. And to obtain the final performance estimation the two AUCs are averaged, as given by equation (4.17).

$$\text{AUC}_{\text{total}} = \frac{\text{AUC}_{\text{fraud increase}} + \text{AUC}_{\text{effort reduction}}}{2} \tag{4.17}$$

4.2.6 Dataset Improvement through Claim Analysis

In an attempt to improve claim classification, wrongly classified claims are discussed with a fraud expert to gain valuable insights on what might have caused misclassification. There are two types of misclassifications:

- *Fraud* is classified as *non-fraud*; *false negatives*.
- *Non-Fraud* is classified as *fraud*; *false positives*.

Misclassified *fraud* is considered to be more valuable to be analyzed further, by a fraud expert, than misclassified *non-fraud*. Firstly, because the dataset is highly imbalanced and the main goal is to detect fraudulent claims. Secondly, for misclassified *non-fraud*, it is difficult to judge if a claim has really been misclassified, due to the fact that suspicious claims are labeled as legitimate claims in the dataset (section 3.3). Claims chosen for further analysis can help in the following ways:

- Finding new features that better profile a *fraudulent* claim; fraud might be profiled by specific features that are not always present in the current dataset.
- Label or feature value correction; in some cases labels or values were wrongly put into the system and have to be corrected.

As mentioned in section 3.3, for each processed claim, the so-called *research cost* are available. These cost indicate the amount of money that was spent on fraud investigation and claim handling for a specific claim, where higher cost usually indicate more suspicious. Figure 4.7, shows an example of a research cost plot.

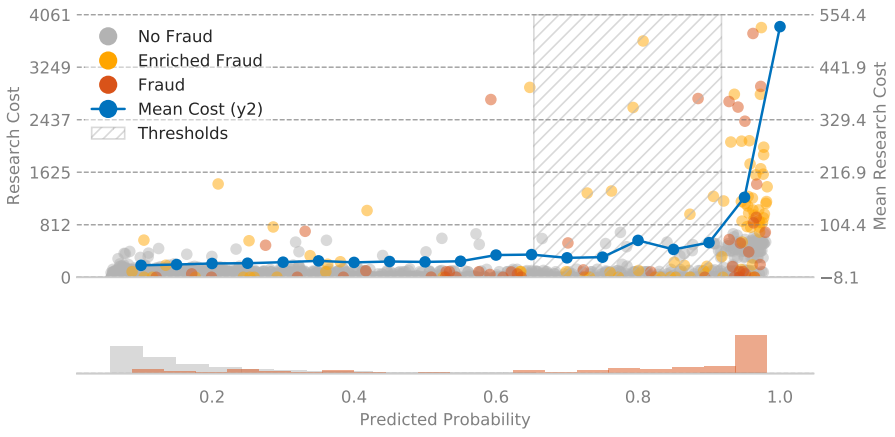


Figure 4.7: Example of a research cost plot. With the predicted probability that a claim is fraudulent on the x-axis and the research cost on the y-axis. The mean cost line indicates the average research cost per bin and uses the right y-axis.

In the plot each claim is represented by a dot and is positioned according to its predicted probability and research cost. The plot differentiates between

non-fraud, originally labeled fraud and claims that received a fraud label after label enrichment (section 3.5.1). The *mean research cost* line represents the mean cost of all claims in each bin. Ideally, fraudulent claims reside mainly on the right side and non-fraudulent claims reside mainly on the left side, because a higher predicted probability means that a claim is more suspicious.

This visualization helps to learn more about the dataset in collaboration with a fraud expert, to improve the predictive performance of the machine learning models. The plot regions that are most helpful to learn more about the dataset are:

- Fraudulent claims with low probability:
  - and low research cost;
    the model thinks the claim is non-fraudulent, which means the model is likely wrong. The model did not receive a lot of attention by the fraud experts, because the research cost are low. This means that the claim is either clearly fraudulent or the label is wrong. When the claim is clearly fraudulent, it probably contains new features that the model did not train with that can help profile fraud.
  - and high research cost;
    the model thinks the claim is non-fraudulent, which means the model is likely wrong. The research cost are high, thus the claim received attention from the fraud experts. Moreover, the claim has been proven to be fraudulent. The claim can therefore probably provide the model with new features to profile fraudulent claims.
- Enriched fraudulent claims with low probability:
  - and high research cost;
    the model thinks the claim is non-fraudulent, which matches with the original label. However, the claim has been enriched, because in the past the claim received attention by the fraud experts. So, there should be a feature that triggered them into thinking the claim is fraudulent. If the model is right, future research cost can be saved.
  - and low research cost;
    the model thinks the claim is non-fraudulent, which matches with the original label. The research cost are low, which means that the claim has a suspicious fraud score history, but did not receive a lot of attention from the fraud experts. It is therefore plausible that this claim has been wrongly enriched.

The new features and label cleanups are applied in the data acquisition and preprocessing stage.

4.3 Final Pipeline Design

This section gives an overview of how the previously discussed stages of the machine learning work flow relate to each other and how data is turned into

trained models and predictions.

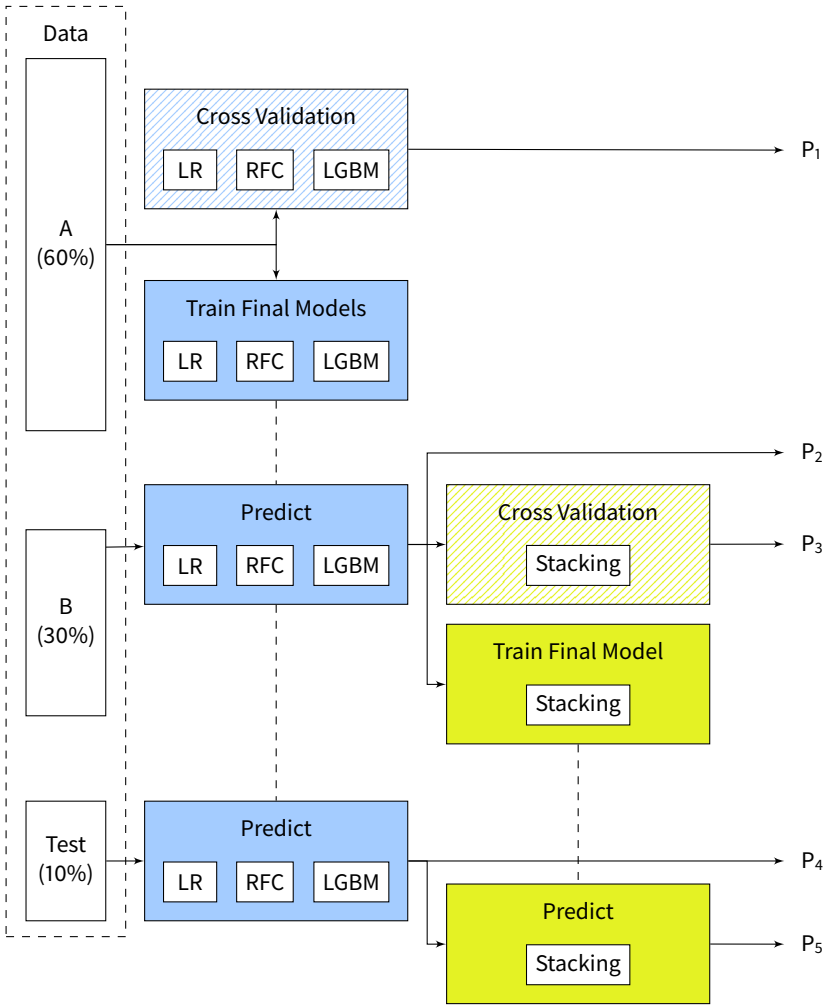


Figure 4.8: An abstract visual representation of the process to go from data to predictions. The different colors separate the classical machine learning models from Stacking. Moreover, a hatched block indicates that it is used for model optimization and determining the final model configuration.

Figure 4.8, gives an abstract visualization of the data that is used to train certain models. The different colors in the diagram separate the classical machine learning models from the Stacking classifier. Moreover, blocks with a hatched background determine the final configuration of a model, whereas the solid colored block use the final configuration for either training or predicting.

Starting with the classical models, every model starts with cross validation. Here, models are optimized and the final model configuration is determined. With the final configuration ready, the models are trained on the entire train set, which happens to be partition A (TrainVal A) for the classi-

cal models. The trained models are then tested on partition B (TrainVal B) and the Test partition. The visualization shows that Stacking uses the predictions of the other models as its input. Also Stacking is first optimized with cross validation to find the final parameter configuration. This configuration is then used to train Stacking on the predictions of the other classical models. The trained model of Stacking is then tested on the Test partition, using the predictions of the other models as its input.

To clarify the steps described above, a more detailed version of all the steps is given by figure 4.9. Here, cross validation is depicted more explicit and also the resample and preprocessing steps are shown. Technically, resampling is part of preprocessing, but it is important to note that resampling only happens on the training input of models, it is not done on input that is going to be predicted. It is important to note that only the classical machine learning models contain preprocessing steps. Stacking does not contain this, because its input is already nicely formatted. Namely, three features with values between 0 and 1, after all, its input are the predicted outputs of the tree models: Logistic Regression, Random Forest and LightGBM.

Lastly, it is interesting to talk about the use of each prediction indicated by P_i in the diagrams. Predictions P_1 and P_3 both contain the predicted output of cross validation. This output is mainly used during the optimization of a model, to help with parameter tuning, and create the final configuration. Moreover, cross validation is used to compute the standard deviation of a model, so a judgement can be made on how well a model generalizes to other parts of the data, and whether certain predictive performance is representative. Prediction P_2, is the predicted output of the classical models on the B set (TrainVal B). The results of P_2 will be used to be able to compare the classical models more fairly to the Stacking performance of P_3, since they are both tested on the same dataset. Predictions P_4 and P_5 contain the predictions of the classical models and Stacking on the Test set. This is the fairest comparison, as both Stacking and the other models are tested on exactly the same set. Therefore, it will be used to draw conclusions about the predictive performance of the models.

4.4 Interpreting Model Predictions

An important difference between machine learning models and the legacy rule-based system is that the latter is transparent. For each claim it shows the most important rules that lead to its classification; helping fraud experts to quickly verify any suspicions. In order to turn machine learning models into a viable solution, it is required that fraud experts are given clues on the features that are most important for a specific prediction. The following sections explain how to find the top most contributing features of a claim for different types of algorithms.

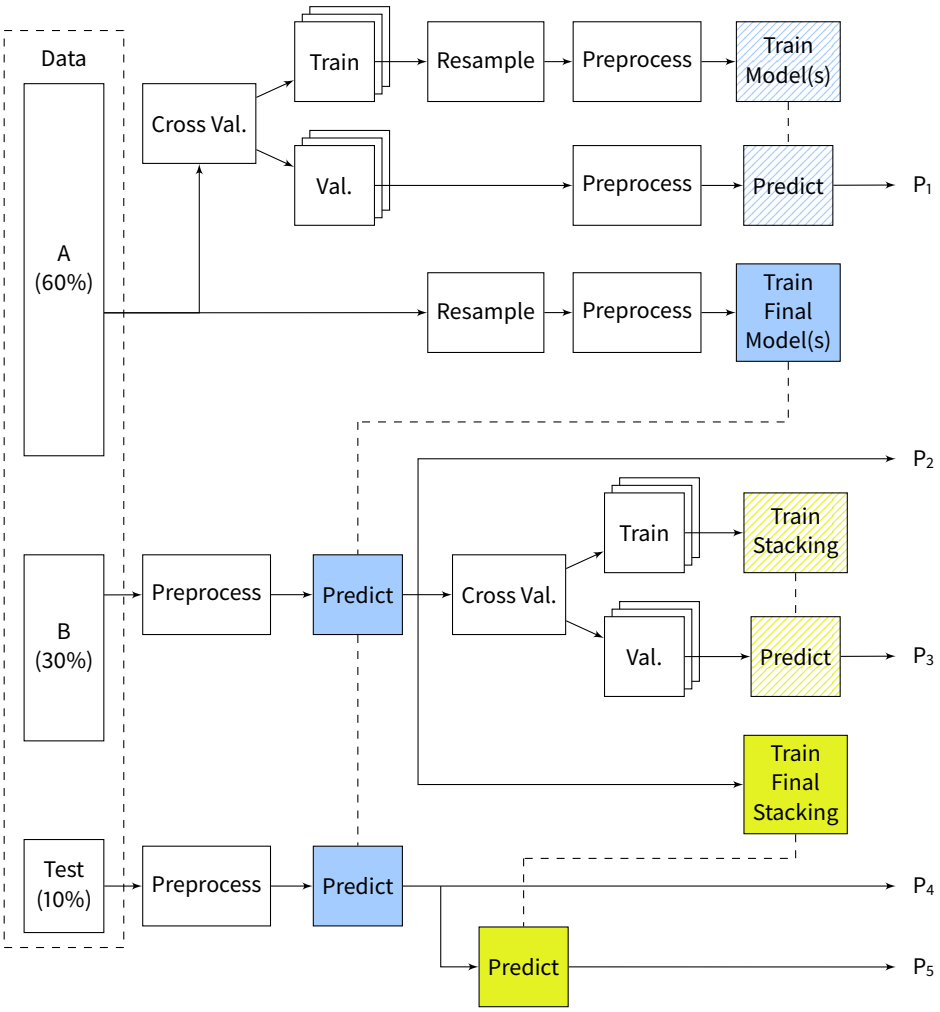


Figure 4.9: A detailed visual representation of the process to go from data to predictions. The different colors separate the classical machine learning models from Stacking. Moreover, a hatched block indicates that it is used for model optimization and determining the final model configuration.

4.4.1 Interpretation of Logistic Regression

As mentioned in section 3.3.1, standard scaling has been applied to all scalar features to make their values anonymous. As an added benefit all the features have the same scale, which simplifies the process of computing the feature contributions for Logistic Regression.

Because standardized data is used, the predictors already all have the same scale, which allows for the direct comparison of the regression coefficients. The standardized coefficients represent the mean change in the response given a one standard deviation change in the predictor.

Multiplying the values of the predictor with their corresponding regres-

sion coefficients, gives the contribution for each feature for the predictor. The features with a high positive contribution will be the top contributing features to why a claim is classified as positive/fraudulent, whereas the features with a high negative contribution will be the top contributing features as to why a claim is classified as negative/non-fraudulent.

4.4.2 Interpretation of Decision Trees

One possible way of getting insights into a decision tree is to compute its feature importances, this functionality is offered by Scikit-learn. Feature importance basically describes how prominent a feature is in the classification tree. How much a feature contributes to the predictions overall, but not specifically to a single prediction. A general idea of which features contribute most to a prediction helps to get a feel if the model bases its predictions on sensible features. However, it does not provide the granularity to tell the feature contributions of a single prediction. Below an approach is described on how to find the feature contribution for each prediction separately.

Each decision in a tree corresponds with a feature, which either adds or subtracts from the value of the parent node. A prediction is defined as the sum of feature contributions plus the *bias*[4], given by equation (4.18).

$$f(\mathbf{x}) = c_{bias} + \sum_{j=1}^{J} \text{contrib}(\mathbf{x}, j) \tag{4.18}$$

Where c_{bias} is the value at the root of the node, J is the number of features and *contrib* the contribution of feature j for instance $\mathbf{x}$.

Figure 4.10, provides visual aid to understand how the contribution of features is computed. The tree represents a fitted decision tree model with an input space that consists of two features J_1 and J_2. During training the tree has 100 samples available, 50 for each class; (n_0, n_1) shows the class label distribution for each node, with n_0 positive samples and n_1 negative samples. To learn more about how a decision tree is trained, see section 2.4. The values between brackets [p_0 , p_1], show the probability for an instance to belong either to the positive p_0 or negative p_1 class. These probabilities are then used to compute the contribution of a feature to the branching decision, by subtracting the probability of the previous node from the probability of the current node; resulting in $\{c_0, c_1\}$. The contribution values of a node represent the contribution of the feature belonging to the previous node.

Now, with all the tree data available, the feature contributions for a prediction can be computed. As an example, take $\mathbf{x}$ with feature values $j_1 > t_1$ and $j_2 < t_2$. The instance is predicted to first take the right branch because it has a value bigger than t_1, the feature contribution value of j_1 now becomes 0.32. Thereafter, at the second decision point, the instances is directed to take the left branch; resulting in a contribution value of 0.07 for j_2. The predicted probability that $\mathbf{x}$ belongs to the positive class now becomes 0.89, with feature contribution values for j_1 and j_2 of 0.32 and 0.07 respectively.

[4]The ratio of a class at the parent node as determined during training.

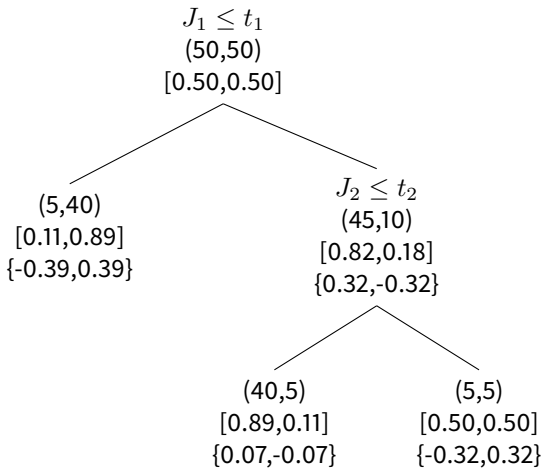


Figure 4.10: A simple decision tree for an input space with two features J_1 and J_2. The space is partitioned into 3 regions. Each leaf shows its class label distribution (n_0, n_1), with n_0 positive samples and n_1 negative samples. [p_0 , p_1], indicates the probability for an instance to belong to either class. $\{c_0, c_1\}$ are the contributions of a feature to the branching decision for either class.

According to equation (4.18), the predicted probability is the bias plus the feature contributions, which gives $0.5 + 0.32 + 0.07 = 0.87$, this is indeed correct.

Interpretation of Random Forest

Section 4.4.2, describes an approach to compute the contribution each feature makes towards a final prediction, to aid the user of the prediction with further analysis. This approach can be extended upon for random forests, as random forests rely on decision trees to make predictions. Instead of using a single decision tree, random forests uses a forest of trees that together count towards the final prediction, see section 2.4.3 for more background.

In order to move from a decision tree to a forest of trees, one should sum and take the average of the tree predictions. The predicted output for random forests is given by:

$$F(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} f_m(\mathbf{x})$$

with M the number of trees and $f_m(\mathbf{x})$ the function for the prediction value of a decision tree. By replacing $f(\mathbf{x})$ with equation (4.18), the prediction value for random forests can be written as:

$$F(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} c_{bias,m} + \sum_{j=1}^{J} \left(\frac{1}{M} \sum_{m=1}^{M} \text{contrib}_m(\mathbf{x}, j) \right) \tag{4.19}$$

with J the number of features and contrib_m is computed according to the approach described in section 4.4.2. By summing and averaging the feature

contributions for each path taken in each tree, one is able to compute the contribution of each feature for each prediction of random forests separately. The feature contribution for instance $\mathbf{x}$ and feature j is given by equation (4.20).

$$\text{feature contribution}(\mathbf{x}, j) = \frac{1}{M}\sum_{m=1}^{M} c_{bias,m} + \frac{1}{M}\sum_{m=1}^{M} \text{contrib}_m(\mathbf{x}, j) \quad (4.20)$$

Interpretation of LightGBM

Section 4.4.2 lays the foundation for computing feature contributions for decision trees. This can be extended upon to compute the feature contributions for LightGBM. There is a fundamental difference between bagging trees (Random Forests) and boosting trees (LightGBM). In order to maintain equation (4.18), where "prediction = bias + sum of the feature contributions". Each bagged tree, maps from bias to target, and the bagged ensemble is the average vote of all bagged trees together. Hence the prediction of Random Forests, see equation (4.19), is computed by dividing the bias and feature contributions by the number of trees. Each boosted tree on the other hand, maps from residual to target, and the boosted ensemble maps from bias to target. Therefore, the prediction is given by

$$F(\mathbf{x}) = \sum_{m=1}^{M} f_m(\mathbf{x})$$

with M the number of trees and $f_m(\mathbf{x})$ the function for the prediction value of a decision tree. By replacing $f(\mathbf{x})$ with equation (4.18), the prediction value for LightGBM can be written as:

$$F(\mathbf{x}) = \sum_{m=1}^{M} c_{bias,m} + \sum_{j=1}^{J}\left(\sum_{m=1}^{M} \text{contrib}_m(\mathbf{x}, j)\right)$$

with J the number of features and $contrib_m$ is computed according to the approach described in section 4.4.2. By summing the bias and feature contributions for each path taken in each tree, one is able to compute the final bias and feature contributions for every prediction separately. The feature contribution for instance $\mathbf{x}$ and feature j is given by equation (4.21).

$$\text{feature contribution}(\mathbf{x}, j) = \sum_{m=1}^{M} c_{bias,m} + \sum_{m=1}^{M} \text{contrib}_m(\mathbf{x}, j) \quad (4.21)$$

4.4.3 Interpretation of Stacking Classifier

The computation of feature contributions for the Stacking Classifier is similar to that of section 4.4.1, since the Stacking classifier is a Logistic Regression model that uses the predictions of the other three models—Logistic Regression, Random Forest, LightGBM—as its input. For a prediction, the feature contributions of each input model are computed

separately. After this, the feature contributions of the input models are multiplied by their respective (stacking) regression coefficients. This results in three contribution values for each feature, these values are summed to end up with the final feature contribution values. Again, the high positive feature contributions are contributing to the probability that a claim belongs to the positive/fraudulent class, whereas the large negative feature contributions are contributing to the probability that a claim is negative/non-fraudulent.

Chapter 5

Discussion and Results

5.1 Overview

This chapter lists and discusses the results that are obtained by following the approach as described in chapter 4. The main goal is to find the best performing model and see how it compares to the baseline; to get a feel for the predictive performance that can be expected when the model goes live. Furthermore, it is interesting to see if the chosen evaluation metrics are able to capture the results well, and if they hold up in an imbalanced data setting. The following four models are featured in this chapter: Logistic Regression, Random Forest, LightGBM and a Stacking classifier based on Logistic Regression.

Before discussing the results, it helps to understand the partitioning of the dataset, and the role each partition plays in the process from algorithm to prediction. For a reference of the data partition names, see figure 3.4. Section 3.3 provides more background on the reasoning behind the use of multiple data partitions.

Apart, from the partitioning of the dataset, it is also valuable to get familiar with the term *classification threshold region*, since it is used extensively throughout the entire results chapter. The classification threshold region, depicted in some figures, indicates the possible probabilities to put the classification threshold, in order to still comply with the performance boundaries set in section 4.2.5. For reference, the baseline performance is set by a fraud score higher than 2. With the two boundary constraints being:

- **Maximum Effort**: the model should not classify more claims as fraudulent than the baseline, thus it should comply to a certain amount of effort (number of claims) that it flags as potentially fraudulent.
- **Minimum Recall**: the model should at least catch a similar amount of fraudulent claims as the baseline, thus it should have a recall that is similar or higher than the baseline.

The thresholds of a model are the mean of the thresholds that comply with the boundaries described above when cross validating the model. Section 4.2 contains more information on the evaluation metrics used and

how classification thresholds are chosen to match the performance with the baseline.

5.2 Probability Density

This section studies how each model is separating fraudulent claims from non-fraudulent ones. A visualization that can help with studying this behavior is the probability density plot. The probability density plot illustrates the distribution of predicted probabilities for both fraud and non-fraud.

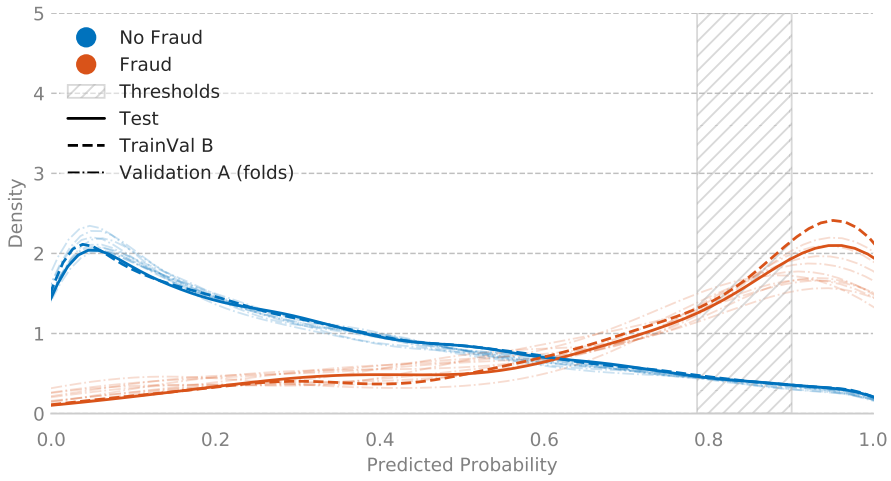


Figure 5.1: Probability density distribution plot of Logistic Regression. The fraudulent claims have relatively high probabilities, whereas the non-fraudulent claims have relatively low probabilities.

Figure 5.1, depicts the density of the predicted probability for Logistic Regression. It is interesting to note that, the spread (or variance) of the probability density lines obtained through cross validation is higher for fraud than for non-fraud. This is likely due to the small amount of fraud cases that are present in the dataset; catching a few extra fraud cases can already significantly change the probability density graph. Furthermore, the threshold region, indicated by the dashed hatch, shows the possible probabilities where a classification threshold can be placed to stay within the performance boundaries mentioned before.

The probability density distribution of Random Forest is shown by figure 5.2. Compared to Logistic Regression, the peak of non-fraudulent claims is twice as high. In return, the fraudulent claims are almost evenly distributed across the entire probability space. However, this more aggressive approach pushes most non-fraudulent claims in the low probability region. Leaving Random Forest with slightly more separation than that of Logistic

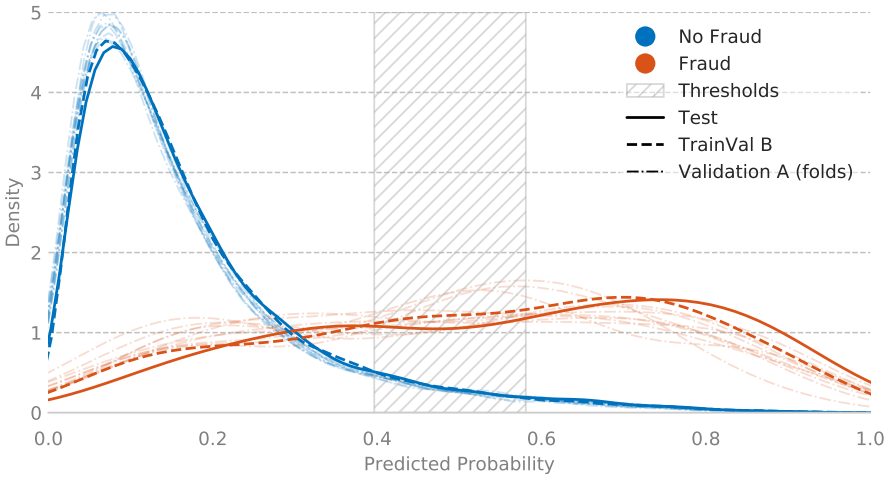


Figure 5.2: Probability density distribution plot of Random Forest. Most non-fraudulent claims have a low predicted probability, whereas the fraudulent claims have a more equal spread of probabilities.

Regression and thus a wider threshold region.[1,2] The variance for non-fraud is again relatively small as compared to fraud.

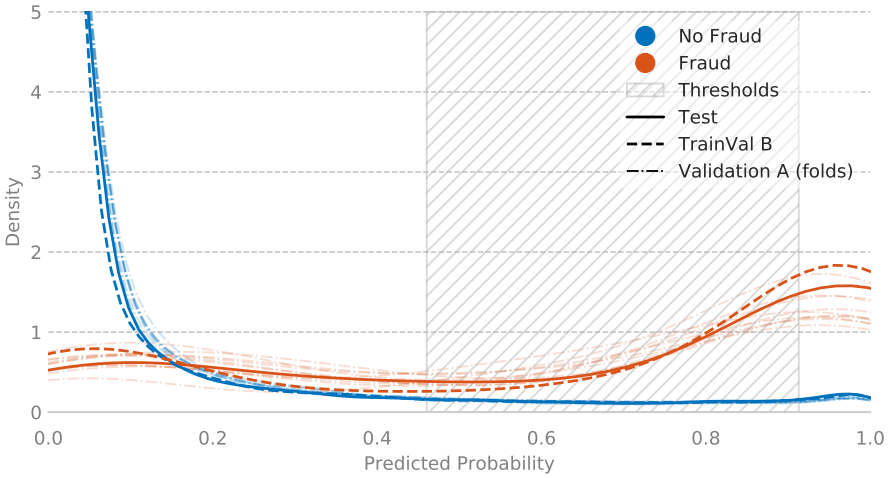


Figure 5.3: Probability density distribution plot of LightGBM. There is aggressive separation of the claims, with a high concentration of non-fraudulent claims in the low probability range.

From figure 5.3, one can conclude that LightGBM has the most aggressive

[1]It is important to note that a wider threshold region is not necessarily better, it just indicates that the claims are separated more, making it easier to pick a threshold that will generalize well to other data partitions.

[2]Interesting to note is that Random Forest is the only one of all tested models that actually has a classification threshold region around the conventional 0.5 probability mark.

separation of all trained models. The probability density peak of non-fraudulent cases is twice as high as the one of Random Forests and four times as high as the one of Logistic Regression. The fraudulent claims, in the meantime, get pushed towards the higher end of the probability space. However, part of the fraudulent claims end up in the low probability area; this is likely due to the aggressive separation, which causes a few misclassifications. The probability density graph of LightGBM seems to borrow characteristics of the fraud peak of Logistic Regression and the no fraud peak of Random Forest. Yet again, the variance for the fraudulent claims is relatively high as compared to the non-fraudulent claims; something that the Stacking classifier aims to solve.

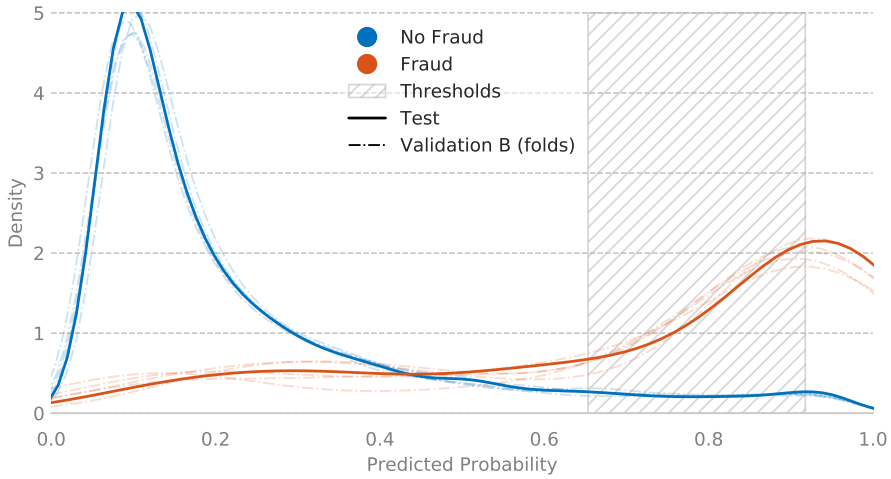


Figure 5.4: Probability density distribution plot of the Stacking classifier. It blends characteristics of the other probability distributions.

Figure 5.4, depicts the probability density distribution for the Stacking classifier. Since the Stacking classifier uses the predictions of the other three models as its input, it is no surprise that its probability density distribution shows a lot of resemblance to the other models. The distribution seems to have inherited the separation strategy from LightGBM, with two clear peaks on either side of the graph. Moreover, the low probability part of the fraud distribution is similar to that of Logistic Regression; removing the bump that is present in the distribution of LightGBM. Apart from this, the most important takeaway is that by building an ensemble of models, the variance in the fraud distribution is visibly smaller.

5.3 Research Cost

The research cost plot enables a modeler to quickly spot interesting cases based on the predicted probability and research cost. Section 4.2.6, elaborates more on how this plot helps improve data quality and discusses the regions of the plot that are most interesting to explore. Each claim in a research cost

plot is represented by a dot that is positioned according to its corresponding research cost and predicted probability. The research cost are cost that were spend on claim handling and fraud investigations for a specific claim. Therefore, higher cost generally mean more suspicious. Figure 5.5, shows the

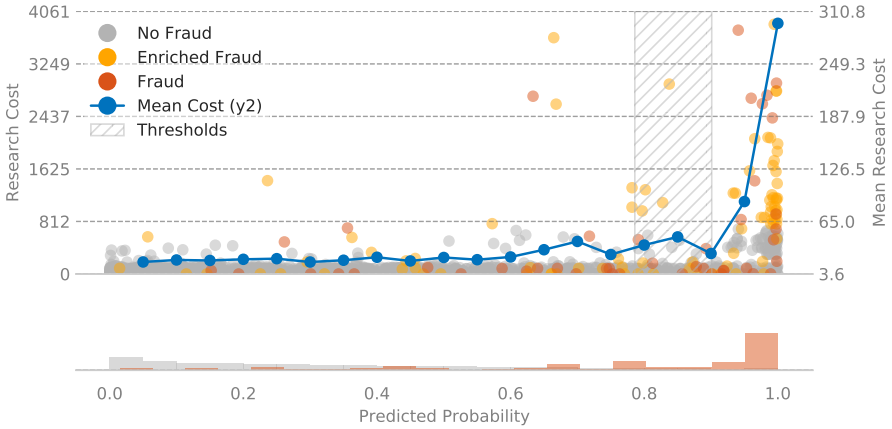


Figure 5.5: Research cost plot of Logistic Regression. The bottom density plot is normalized separately for fraud and non-fraud, this means that the plot indicates what percentage of claims is gathered in a specific region.

research cost plot of Logistic Regression. The mean research cost only shows a firm increase in the last 10% probability. This is good, as it shows that most of the fraudulent cases are pushed towards the top 10%, beyond the threshold region. There are only a few fraudulent claims with high research cost floating in front of the threshold region. Most misclassified fraudulent claims have low research cost. These are interesting to look at with further research, as low research cost might mean that these cases have been wrongly flagged as fraud in the dataset, as they have not had any investigations. On the bottom right side of the graph, there is a fairly large blob of non-fraudulent cases, however they have higher research cost than their counterparts that are evenly distributed among the bottom of the graph. It might be very plausible that those non-fraudulent claims with relatively high expert costs are actually suspicious and have thus received the correct classification.

As indicated by the probability density plot of Random Forests (figure 5.2) and what is confirmed by its research cost plot (figure 5.6), for Random Forest the fraudulent claims are evenly spread across the entire probability space. The non-fraudulent cases are mainly packed at the lower end of the probability space. The threshold region of Random Forests is shifted more to the beginning of the graph and because of this the claims on the right hand side are spread out more. Even though it does not look similar to that of Logistic Regression, in classification terms the result is similar. The blob of non-fraudulent claims with higher research cost that was identified in the research cost plot of Logistic Regression, is spread out more with Random Forest. More of these claims reside in the lower end of the probability space before the classification threshold region. This means that Random

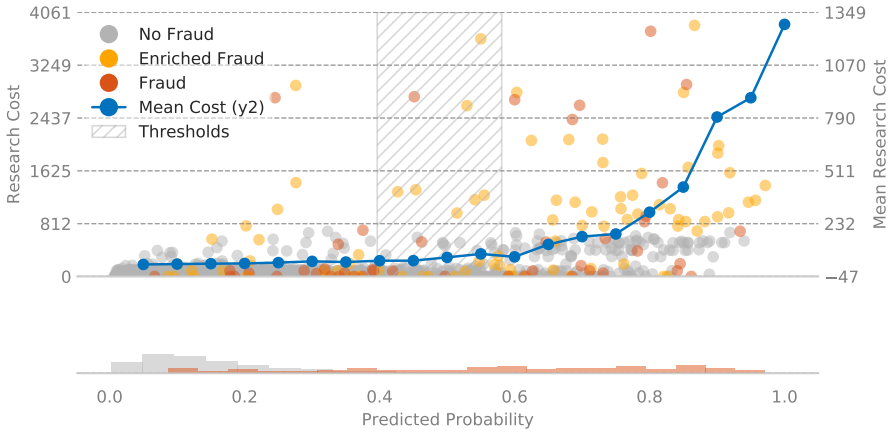


Figure 5.6: Research cost plot of Random Forest. The bottom density plot is normalized separately for fraud and non-fraud, this means that the plot indicates what percentage of claims is gathered in a specific region.

Forest finds these claims less suspicious than for example Logistic Regression. In performance terms this is good, since those claims have the non-fraud label, however, in real-life it might turn out that those claims are actually suspicious; which would be in favor of Logistic Regression.

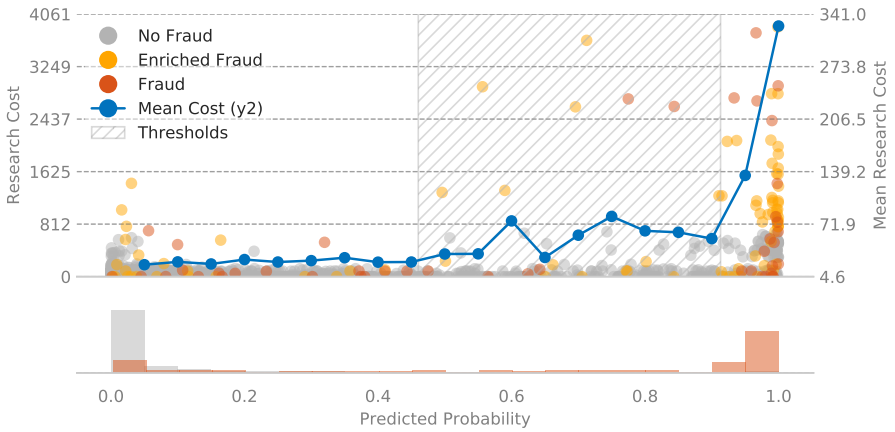


Figure 5.7: Research cost plot of LightGBM. The bottom density plot is normalized separately for fraud and non-fraud, this means that the plot indicates what percentage of claims is gathered in a specific region.

Figure 5.3, shows that LightGBM has aggressive separation and this is reflected in its research cost plot, see figure 5.7. Most fraudulent claims are pushed towards the far right side of the graph. However, due to the aggressive separation, also quite a few fraudulent claims reside on the far left side of the graph; Logistic Regression handles these cases better. The mean research cost for LightGBM increases steeply towards the end of the probability space.

Around a predicted probability of 0.8 there is a small peak in mean cost, this is likely caused due to the little claims that are present due to the aggressive separation, making the mean cost more prone to outliers. A thing to note is that the non-fraudulent claims with a relatively high research cost are split in two by the LightGBM model. A part of those claim reside on the far left side of the graph, and the rest resides on the far right side of the graph. This behavior is similar to that of RFC and differs from that of Logistic Regression. Furthermore, fraudulent cases with research cost higher than 1600 all have a probability higher than the lowest threshold boundary, something that is not the case with the other models.

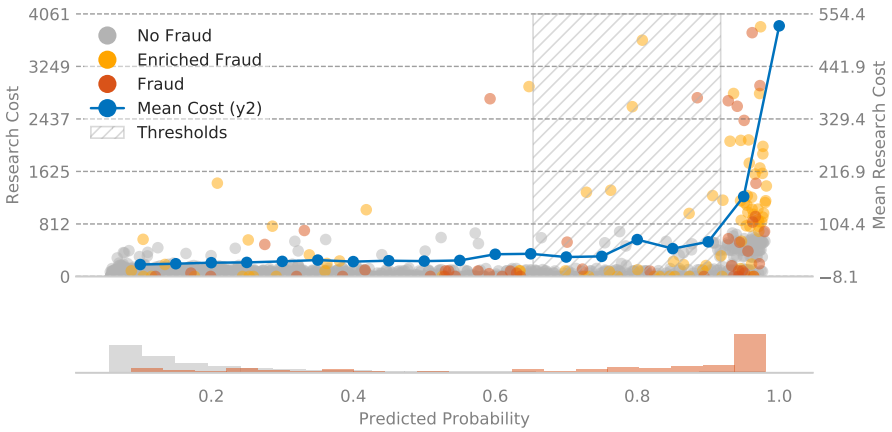


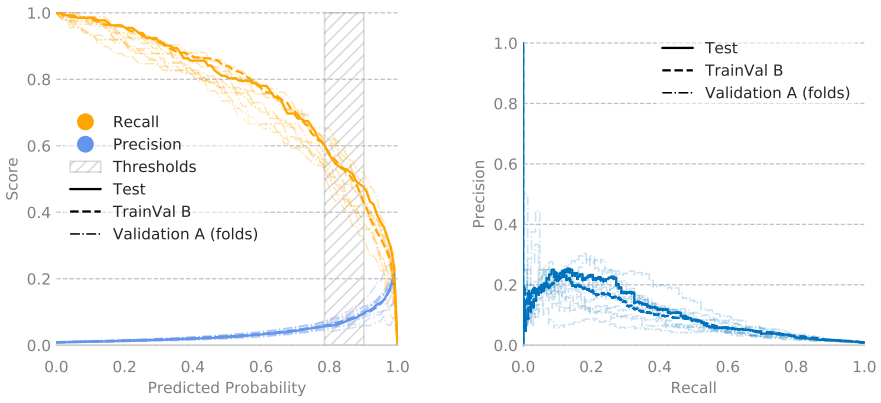
Figure 5.8: Research cost plot of the Stacking classifier. The bottom density plot is normalized separately for fraud and non-fraud, this means that the plot indicates what percentage of claims is gathered in a specific region.

In the research cost plot of the Stacking classifier (figure 5.8), one expects to see the same hybrid performance as in figure 5.4; borrowing characteristics of the other trained models. The fraudulent claims that are present on the far left side of the graph of LightGBM, are spread out more in the graph of the Stacking classifier. Basically the graph shows the pattern of LightGBM with some extra probability spread induced by Logistic Regression and Random Forest. This also shows with the mean research cost, where the peaks of LightGBM have been dampened by the other algorithms.

5.4 Precision and Recall

As mentioned in section 4.2.1, precision and recall are metrics that enable the modeler to measure the performance of a model for a single classification threshold. However, by recursively moving the classification threshold along the entire probability space and measuring the precision and recall for each of those thresholds, one can create a *precision and recall plot* and a *precision-recall curve*. Figure 5.9a, shows the precision and recall plot for Logistic Regression, which illustrates the precision and recall scores (y-axis) for the

entire predicted probability range (x-axis). This plot differs from the more conventional precision-recall curve (figure 5.9b), where precision is plot on the y-axis and recall is plot on the x-axis. The precision and recall plot is useful to check the variance on precision and recall separately, whereas the precision-recall curve is mainly interesting for model comparison, because in combination with the area under curve it is able to summarize the entire predictive performance into a single metric.



(a) Precision and recall plot

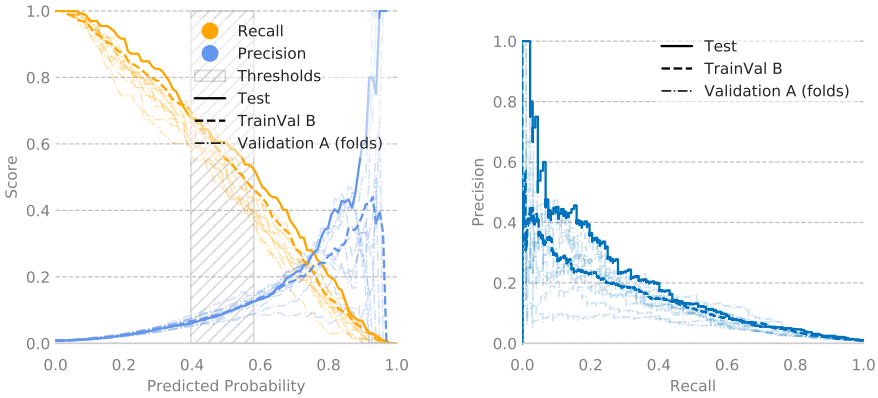
(b) Precision-recall curve

Figure 5.9: Precision and recall plot (a), and the precision-recall curve (b) of Logistic Regression.

From figure 5.9a, one can conclude that the recall performance on the Test set is a good representation of the average recall inside the threshold region, because the lines lie between the other fold lines. Furthermore, the lower recall becomes, the higher the variance within precision, because with lower recall less fraudulent claims are present. The peak at the beginning of the precision-recall curve (figure 5.9b) is caused by the claim with the highest probability being fraudulent. In this case, precision becomes 100% for a recall with a value equal to only one claim.

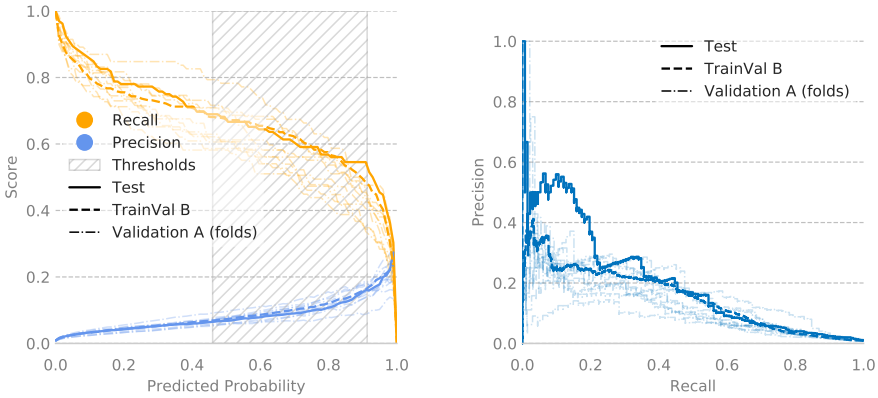
The recall line of Random Forests, see figure 5.10, falls of quicker than that of Logistic Regression. This is likely due to the fact that Random Forest classifies most non-fraudulent cases at the start of the probability space, and thus part of the fraudulent claims got also classified in this region. However, because of this strategy, the precision rises also faster than that of Logistic Regression. The variance for recall at the end of the threshold region is also lower for Random Forest. Furthermore, after surpassing a recall of 0.2, the values for precision become noisy just like with Logistic Regression. However, in this case the effect is more profound, because the recall line is less steep in this area. With Logistic Regression this noise is pushed to the border of the graph.

Figure 5.11a, illustrates the precision and recall plot of LightGBM. An interesting thing to note when looking at figure 5.11a, is that recall immediately dips to about 0.8. The reason for this is the high amount of



(a) Precision and recall plot (b) Precision-recall curve

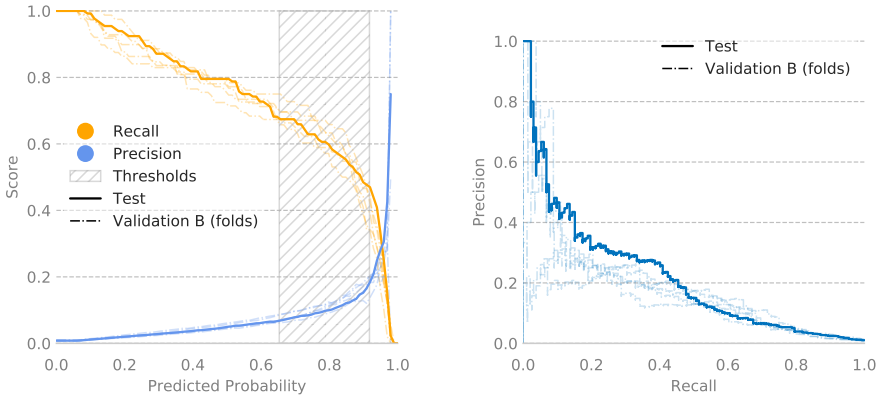
Figure 5.10: Precision and recall plot (a), and the precision-recall curve (b) of Random Forest.



(a) Precision and recall plot (b) Precision-recall curve

Figure 5.11: Precision and recall plot (a), and the precision-recall curve (b) of LightGBM.

claims that reside at the beginning of the probability space. LightGBM is not able to separate all the fraudulent claims from the non-fraudulent ones, and because of this a part of the fraudulent claims received a low probability, hence the immediate dip in recall. Another thing that is apparent in the graph, is that the variance on recall is larger as compared to the recall of Logistic Regression and Random Forest. A likely reason for this is the small claim count that is present in the middle of the probability space, the variance is likely to be higher when there is a small amount of claims present. Again, the smaller recall becomes, and thus the lesser amount of fraudulent cases are caught, the more variance precision has.



(a) Precision and recall plot

(b) Precision-recall curve

Figure 5.12: Precision and recall plot (a), and the precision-recall curve (b) of the Stacking classifier.

Figure 5.12, shows the precision and recall plot, and the precision-recall curve of the Stacking classifier. The variance in the precision and recall plot (figure 5.12a) is lower as compared to that of the other models. Moreover, recall does not show the same drop as LightGBM. It is also interesting to see that recall follows a constant decent as is also present in the plots of Random Forest and LightGBM, whereas the recall plot of Logistic Regression has more of a curving descent. The precision-recall curve (figure 5.12b) shows a line of similar performance as the ones from Random Forests (figure 5.10b) and LightGBM (figure 5.11b). However, with slightly better variance, especially when compared to the variance in the precision-recall curve of Random Forest.

5.5 Model Comparison

As mentioned in section 4.2.4, model comparison is done using the area under curve metric measured on the precision-recall curve of each model; see figures 5.9b, 5.10b, 5.11b and 5.12b. This evaluation technique is used to compare the performance of the same model for differently tuned parameters, and also to find the best performing model. The models presented here have already been tweaked and tuned to its fullest, hence the comparison is mainly about which one of the different models performs best. Table 5.1, presents the AUCs for each model on each partition that the models are evaluated on.

The table shows that the Stacking classifier has the highest AUC on both the validation folds and the test set. However, Random Forest (RFC) and LightGBM (LGBM) only have marginally lower performance. Especially when the cross validated performance of Stacking on the B partition (Validation B (folds)) is compared to the performance of Random Forest and LightGBM on the B partition (TrainVal B), the difference is marginal. Taking into account the standard deviation, even though Stacking has the highest mean, its value range is almost entirely overlapped by that of Random Forest and LightGBM.

Table 5.1: Area under curve values measured on precision-recall curves for different models on different partitions. For the cross validation folds the AUC is given as the mean and the standard deviation, with 1.0 the maximum value of AUC. The values between "()" are the standard deviations of the cross validation folds.

Data Partition	Model	AUC
Validation A (folds)	LR	0.09 (0.025)
	RFC	0.13 (0.034)
	LGBM	0.13 (0.034)
Validation B (folds)	Stacking	0.16 (0.031)
TrainVal B	LR	0.09
	RFC	0.14
	LGBM	0.15
Test	LR	0.10
	RFC	0.20
	LGBM	0.20
	Stacking	0.22

The Test set yields the same results in terms of differences. Moreover, it seems like the Test set favors Random Forest, LightGBM and Stacking, as their AUC is significantly higher than on the other sets, where the AUC of Logistic Regression hardly changes.

The main goal of using Stacking is to maintain the performance of the best performing model and reduce its variance. The models with the best performance (highest AUC) are Random Forest and LightGBM, and indeed Stacking has a similar or even higher performance, and a slightly smaller standard deviation, however, this difference is again only marginal. Logistic Regression on the other hand, has the smallest standard deviation but also the lowest AUC. For low recall scores, the precision-recall curves become noisy, this likely overshadows the rest of the curve performance, thus the standard deviation of each model is relatively similar.

One could argue that judging a model on the entire recall space is generally a good idea. However, with large class imbalance and a small amount of fraudulent claims, the performance measurements become unstable for small values of recall. The higher recall part of the precision-recall curve of Stacking seems to have less variance than the other models, especially when compared to LightGBM. This leads to the conclusion that the tree-based models and Stacking definitely outperform Logistic Regression. Moreover, there are signs that Stacking is able to maintain or even improve performance with a lower variance; especially for higher values of recall.

5.6 Models vs. Baseline

Table 5.2, shows the performance of each model at the boundaries of the threshold regions previously mentioned in sections 4.2.5 and 5.1. The two main things to look at are the *fraud increase for maximum effort* and the *effort reduction for minimum recall*. On the cross validation folds, Stacking is the best performer in terms of both fraud increase and effort reduction. Also its standard deviation is lower than that of the other models for both boundaries. However, with some further investigation, one can see that the performance of the tree-based models on the TrainVal B set is nearly identical to that of Stacking on the cross validated B set. The performance improvement of Random Forests on the B partition is not completely fair though, as its effort reduction for maximum effort is -10%, which means that its effort exceeds the boundary limit in order to achieve a fraud increase of almost 80%. The reason for this is explained below.

Table 5.2: The effort reduction and fraud increase for both performance constraints for each model on various data partitions. The values between "()" are the standard deviations of the cross validation folds.

Partition	Model	Maximum Effort		Minimum Recall	
		Effort Reduction (%)	Fraud Increase (%)	Effort Reduction (%)	Fraud Increase (%)
Validation A (folds)	LR	0.20 (0.94)	41.75 (12.11)	54.97 (11.92)	0.55 (1.43)
	RFC	-0.17 (0.63)	56.04 (10.25)	65.61 (5.74)	-0.39 (1.49)
	LGBM	0.12 (0.26)	63.15 (23.03)	72.52 (7.24)	-0.56 (1.30)
Validation B (folds)	Stacking	0.56 (0.45)	78.77 (8.05)	77.25 (3.49)	-3.21 (1.79)
TrainVal B	LR	-6.86	53.70	52.01	11.11
	RFC	-10.33	79.63	61.98	19.14
	LGBM	-4.23	75.31	69.40	23.46
Test	LR	-3.22	61.22	53.00	26.53
	RFC	-9.74	83.67	59.66	38.78
	LGBM	-2.64	85.71	67.64	44.90
	Stacking	4.03	81.63	74.89	28.57

Before moving on to evaluating the performance on the test set, it is important to check the *effort reduction column for maximum effort* and the *fraud increase column for minimum recall*. Under ideal circumstances both these columns should be zero. After all, the effort reduction at maximum effort should be zero and the fraud increase for minimum recall should also be zero. However, the numbers in these columns fluctuate and this is because the classification thresholds are determined in the cross validation partitions and do not generalize perfectly to the other data partitions. As one can see, the numbers are near zero for the cross validation partitions and change

value for the TrainVal B and Test partitions. Sections 4.2.5 and 5.1, elaborate more on how the classification thresholds are chosen. In a nutshell, during cross validation, for each fold separately, the two classification thresholds that match the boundary are computed. After that, the final two boundary classification thresholds are computed by taking the mean over the thresholds computed in the folds. These thresholds are then used to benchmark the boundary performance on the other partitions.

Coming back to the performance on the Test partition. Random Forest and LighGBM both have a higher fraud increase for maximum effort than the Stacking classifier. However, the Stacking classifier has positive effort reduction, whereas the tree-based models have a negative one. Obviously, with more effort, fraud increase also increase, so one can conclude that the three models have the same range of performance for the maximum effort threshold boundary. The Stacking classifier outperforms the other models in terms of effort reduction for the minimum recall. However, again, the boundary threshold does not generalize perfectly to the Test partition. So, LightGBM and Random Forest have a lower effort reduction, but a higher fraud increase, which is also beneficial for the performance. Thus, one could argue that also for this threshold boundary the performance is similar on the Test partition.

Table 5.3: The AUCs computed according to the approach described in section 4.2.5 of each model on various data partitions. The values between "()" are the standard deviations of the cross validation folds.

Partition	Model	$AUC_{\text{effort reduction}}$	$AUC_{\text{fraud increase}}$	AUC_{total}
Validation A (folds)	LR	28.70 (7.46)	23.31 (20.46)	26.00 (11.94)
	RFC	37.33 (3.67)	30.95 (20.71)	34.14 (10.80)
	LGBM	37.26 (4.08)	39.15 (27.22)	38.21 (13.53)
Validation B (folds)	Stacking	38.33 (2.94)	54.96 (11.81)	46.65 (6.32)
TrainVal B	LR	23.45	34.42	28.94
	RFC	30.90	46.93	38.91
	LGBM	32.63	56.90	44.76
Test	LR	25.34	42.85	34.09
	RFC	30.50	62.39	46.44
	LGBM	32.90	64.02	48.46
	Stacking	39.10	59.88	49.49

Table 5.3, shows the results of an attempt to measure the performance difference of each of the models over their entire respective threshold region. Section 4.2.5, elaborates more on the approach that is being used to end up with this comparison. The sections before already talked about effort reduction and fraud increase. To extend upon this, one can compute the effort reduction and fraud increase for many thresholds within the threshold region. This allows for an effort reduction curve and a fraud increase curve to

be created, see figure 5.13 for a visual presentation of these curves on the Test partition. To allow for the comparison of these curves, the area under curve is measured for both curves ($AUC_{\text{effort reduction}}$, $AUC_{\text{fraud increase}}$), after which, they are averaged to end up with AUC_{total}.[3] AUC_{total} basically describes the quality of the threshold region of each model. The higher effort reduction and fraud increase are in the entire threshold region, the higher AUC_{total} will be.

Now looking at table 5.3, one can see that for effort reduction, Stacking is the best performer with the lowest standard deviation. For fraud increase, Stacking is outperformed by LightGBM and also Random Forest on the Test partition, however, Stacking does have the lowest standard deviation. Finally, the averaged total of the two curves results in Stacking to be the highest performer with the lowest standard deviation. However, LightGBM and Random Forest have similar performance, but worse generalization. Another interesting thing to note, is that the standard deviation for effort reduction is lower than that of fraud increase, while their values are of the same order of magnitude. A reason for this is that effort reduction is influenced by both the fraudulent and non-fraudulent claims, see equation (4.13), whereas recall solely measures the positive claims. Because of the large class imbalance, effort reduction is "buffered" by the negative claims, thus has a lower standard deviation.

Table 5.4: Final results for each model on the Test set. The effort reduction and fraud increase are presented for both performance constraints. AUC_{total} displays the average of the AUCs shown in figure 5.13

Model	Maximum Effort		Minimum Recall		AUC_{total}
	Effort Reduction (%)	Fraud Increase (%)	Effort Reduction (%)	Fraud Increase (%)	
LR	-3.22	61.22	53.00	26.53	34.09
RFC	-9.74	83.67	59.66	38.78	46.44
LGBM	-2.64	**85.71**	67.64	**44.90**	48.46
Stacking	**4.03**	81.63	**74.89**	28.57	**49.49**

Table 5.4, summarizes the final results on the Test set, for which figure 5.13 gives a visual representation. Using tables 5.2 and 5.3, one can see how well the numbers presented in table 5.4 generalize. The main takeaway is that the Stacking classifier matches the performance of LightGBM and in some scenarios even has better performance, with better generalization—lower standard deviation. Random Forest comes close to the performance of Stacking and LightGBM, but also with a standard deviation higher than that of Stacking. Logistic Regression is not able to match the performance of the tree-based models, then again, in some scenarios it is able to contain its variance a bit better, which might have helped the Stacking classifier in the end.

[3] It is important to realize that the x-axis is normalized, thus only the values of fraud increase and effort reduction contribute to the AUC, not the width of the threshold region.

The final performance evaluation by computing the area under curve of the curve shown in figure 5.13, is merely to confirm that the Stacking classifier is at least on par with the best performance whilst improving generalization. The AUC values consequently show that Stacking is the best performer, but given the standard deviation measured on the cross validation folds this conclusion cannot be drawn with full certainty.

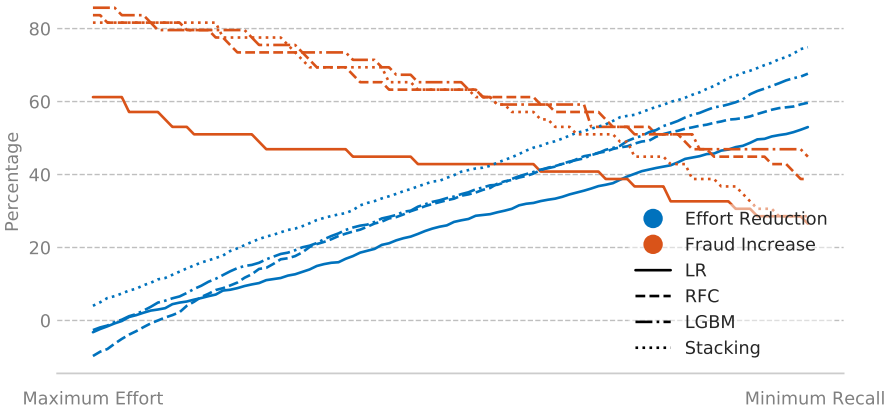


Figure 5.13: Final baseline to model comparison. The diagram illustrates the overall quality of a model as compared to the baseline within the objective constraints. The trend of the effort reduction and fraud increase are shown over the course of the normalized threshold regions belonging to each model.

Chapter 6

Conclusion & Evaluation

6.1 Main Objective

The main objective to outperform the currently installed rule-based fraud detection system has been achieved. The final model is able to find more fraudulent insurance claims with a higher efficiency than the currently installed rule-based fraud detection system. Moreover, the classification threshold of the model can be chosen in such a way to either favor more fraud or a smaller workload, while still respecting the boundaries of the objective. Compared to the rule-based system, the final model can be tuned to roughly a 70-80% increase in extra fraud caught or a 75% reduction in effort; see section 5.6. It is up to the collaborating insurance company to decide which parameter they find more important. The extra fraud caught will obviously increase the amount of real fraudulent claims that the fraud experts get to see. The smaller workload leads to an increase in capacity, enabling the fraud experts to spend more time on relevant claims, which increases the chance to catch fraudsters of suspicious claims.

The final model is a Stacking classifier based on Logistic Regression that combines three other models. The three models used for the Stacking classifier are: Logistic Regression, Random Forest (bagging tree) and LightGBM (boosting tree). These three models are trained on the imbalanced learning problem by using a combination of undersampling (random and Edited Nearest Neighbors), oversampling (SMOTE) and class weighting.

6.2 Evaluation

From the things discussed in this thesis some things worked out well and others did not work out exactly as expected. This section recaps the most important lessons.

The first thing that stands out is that the evaluation of the models was rather complicated due to the small population of fraudulent claims. This gave the results a rather high variance and made it more difficult to draw reasonable conclusions. It also complicated model optimization, as it is difficult to judge whether a performance improvement is due to a lucky split

or an actually better model. Label enrichment, as described in section 3.5.2, can help to mitigate this problem.

The high data imbalance also adds difficulty to drawing conclusions on the performance of the Stacking classifier. From the models to baseline comparison (section 5.6) can be concluded that the Stacking classifier matches or improves upon the predictive performance of the top performing models, and combines this with a lower variance/standard deviation. However, with the model to model comparisons, the standard deviation of Stacking was relatively similar, which gives reasons to think that the variance is not always smaller for Stacking. Section 5.5 gives an explanation as to why the variance might have been similar in this case.

Furthermore, bounding performance by the two constraints, as discussed in the objective statement, proves to be a good method to somewhat restrict the area of performance on which the models were evaluated. However, because models contain variance, the chosen thresholds did not always stay within the constrained performance. One should take this into account when choosing for a similar approach to compare models to the baseline.

6.3 Additional Contributions

Part of this thesis is heavily focused on how to evaluate predictive performance; for both model to model comparisons and model to baseline comparisons. The insights that were derived are described below.

This thesis introduces a new way of looking at classification output by adding an extra dimension in the form of research cost to the end results of the predictions; see section 4.2.6. The plot makes it easy to differentiate between non-fraudulent and suspicious/fraudulent claims and provides a quick overview of the overall quality of the classification. Claims that reside in the wrong probability regions can be forwarded to a fraud expert to check if there are any features missing in the dataset that profile this particular claim. This type of plot can help with any classification problem, as long as a second dimension is available that correlates with the class types that one tries to predict.

Generally, for classification problems, baseline performance is either a machine learning model, or a single metric value. When the baseline consists of a single metric value, it is tempting to convert the predictive performance of the, to be compared, machine learning models to a single value as well. However, this makes it difficult to draw a conclusion on the overall quality of the models, since they are only compared on one specific classification threshold. What this thesis tries to show is that whenever one has a baseline that is made up of single performance metrics, one can likely shape the problem into two constraints rather than one. By evaluating models on a space rather than a single point, more confidence can be gained in the overall quality of a model.

Lastly, the most widely used metrics for measuring predictive performance on (imbalanced) classification problems [25] have been investigated; see section 4.2. To see how these metrics are influenced by imbalanced data

and to judge their ability to summarize the predictive performance on the positive/minority class. The main highlights of this investigation are:

- A confusion matrix is the most transparent way of looking at imbalanced data (precision and recall mainly).
- F-score and Matthews Correlation Coefficient hold up well as single classification threshold measurements, however, they have little advantage over just using precision and recall.
- Using the popular ROC curves for imbalanced data gives a skewed view of performance when dealing with imbalanced data, especially on the positive/minority class.
- The precision-recall curve in combination with the area under curve, maintains the transparency of the confusion matrix through visual feedback, while also allowing for easy comparison with a single metric estimation.

6.4 Real World Performance

During this research multiple extra small scale tests have been carried out to check the predictions of the models with the opinion of the fraud expert. It is important to note that these tests do not have enough scientific foundation to draw any conclusions. The results that will be presented are merely to take note of how much real world performance can differ from the theoretical performance measured using the data labels.

The first test was carried out to get a feel for how many suspicious claims are likely to be in the dataset, and how much the actual performance is different from the measured performance. For this test, 5 claims were taken that were labeled as non-fraudulent in the dataset, but that were classified as fraud by the model. 4/5 claims happened to be suspicious, which gives reason to think that the real world performance is way better than the measured performance. After all, according to the labels the model misclassified all of these claims.

The second test was done to find out if the computed feature contributions helped the fraud expert to find fraud more easily. This time 8 claims—labeled as non-fraudulent in the dataset but marked as suspicious/fraudulent by the model—were tested. 6/8 claims happened to be suspicious and from these 6 suspicious claims the feature contributions helped 5 times to point the fraud expert in the right direction. There was one particular case, where the fraud expert confirmed that the algorithm made a misclassification and that the non-fraud label in the dataset was indeed correct. However, after providing the fraud expert with the most important features that triggered the high fraud probability, the claim turned out to be highly suspicious after all. With the feature clues, the fraud expert was able to direct his investigation to the important features and found that the same person filed 3 very similar claims in one year. Separately the claims looked normal, but together they raised a lot of suspicion.

Again, these tests can only be used as an indication. Nonetheless, it is interesting to see how many suspicious claims there actually are in the dataset. From the $5 + 8 = 13$ misclassifications that were tested, $4 + 6 = 10$ turned out to be correctly classified as suspicious/fraudulent. Further tests are needed to investigate the exact difference in performance, but the results project a promising boost in performance once the model goes live.

6.5 Recommendations

Following the conclusion and evaluation of the results, several recommendations are shared that can bring improvements to both the performance and performance measurements.

Raw Data Values

As mentioned in section 3.3.1, all features were provided anonymized with encoded values. The scalar features were encoded by removing the mean and scaling to unit variance; also called *standard scaling*. Due to the scaling, it is difficult to combine features of different magnitudes into a new feature to make their relationship more explicit—e.g. combine *length* and *width* into *area*. Moreover, some features only contain values for some claims, as for most claims this data is not available. These empty values, however, are saved as infinity values in the dataset, after which the entire feature is scaled to unit variance. Due to the many infinity values that are present, the distribution of the feature becomes distorted. Lastly, data cleanup of for example duplicate categories (e.g. "None", "none", "NONE") can only be done before they are anonymized. The usage of raw untransformed feature values will therefore allow for a more thorough analysis of the dataset and potentially increase the quality of the input space.

Class Label Improvements

One of the challenges of this thesis, was trying to cope with the fact that only *proven fraud* is labeled as fraudulent in the dataset, meaning that suspicious claims have the same labels as other non-fraudulent claims. However, since the ultimate goal of the fraud detection system is to flag both suspicious and fraudulent claims, supervised learning algorithms and performance evaluation are compromised by "wrongly" labeled suspicious claims.

Label enrichment, as discussed in section 3.5.1, partially deals with this issue, but the process of flagging claims needs to be changed at the respective insurance company. Especially, because the small scale tests with the fraud experts as mentioned in section 6.4 show that the predictive performance of the models might be significantly better than what is indicated by the performance measurements that were done according to the data labels.

The fraud experts should get better tools and be more methodical when labeling claims. There should be a clear distinction between proven fraud, suspicious claims and seemingly normal claims. This will lead to

a performance indication that is closer to the real world performance, and supervised learning algorithms will also benefit from more accurate labels.

New Viable Algorithms

There are two things that can open up a wide variety of new algorithms to function as a fraud detector. Firstly, unsupervised learning becomes more viable when the output of a model can be checked directly by a fraud expert, instead of having to rely on labels for performance evaluation. Since the labels do not completely separate the claims correctly, it might be possible for an unsupervised model to have relatively poor predictive performance according to the labels, while it is actually fitting very well to the underlying structure. Secondly, according to the algorithm requirements set in section 3.6.1, algorithms should be interpretable[1], which means that it should be possible to retrieve the contribution of each feature for a certain prediction. Since many algorithms, e.g. neural networks, are not designed to be interpretable, a requirement like this narrows down the algorithm space significantly. Therefore, new approaches like LIME [36] that turn black box models into white box models should be investigated. Technologies like these allow for many other algorithms to be investigated, which opens up many new avenues to research.

[1] Interpretability is of great importance, especially when people need to interface with the output of a machine learning model.

Bibliography

[1] A. Manes, "Insurance crimes," *Journal of Criminal Law and Criminology (1931-1951)*, vol. 35, no. 1, pp. 34–42, 1945, ISSN: 08852731. [Online]. Available: http://www.jstor.org/stable/1138134.

[2] I. Europe, "The impact of insurance fraud," *Brussels: Insurance Europe*, 2013.

[3] L. Caron and G. Dionne, "Insurance fraud estimation: More evidence from the quebec automobile insurance industry," in *Automobile Insurance: Road Safety, New Drivers, Risks, Insurance Fraud and Regulation*, G. Dionne and C. Laberge-Nadeau, Eds. Boston, MA: Springer US, 1999, pp. 175–182, ISBN: 978-1-4615-4058-8. DOI: 10.1007/978-1-4615-4058-8_9. [Online]. Available: https://doi.org/10.1007/978-1-4615-4058-8_9.

[4] R. J. Bolton and D. J. Hand, "Statistical fraud detection: A review," *Statistical science*, pp. 235–249, 2002.

[5] D. R. A., "Insurance fraud," *Journal of Risk and Insurance*, vol. 69, no. 3, pp. 271–287, DOI: 10.1111/1539-6975.00026. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1539-6975.00026. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1539-6975.00026.

[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[7] I. Tomek, "Two modifications of cnn," *IEEE Trans. Systems, Man and Cybernetics*, vol. 6, pp. 769–772, 1976.

[8] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009, ISSN: 1041-4347. DOI: 10.1109/TKDE.2008.239.

[9] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 3, pp. 408–421, 1972.

[10] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical review E*, vol. 69, no. 6, p. 066138, 2004.

[11] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[12] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, "A comparison of optimization methods and software for large-scale l1-regularized linear classification," *Journal of Machine Learning Research*, vol. 11, no. Nov, pp. 3183–3234, 2010.

[13] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[14] B. Leo, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and regression trees," *Wadsworth International Group*, 1984.

[15] L. E. Raileanu and K. Stoffel, "Theoretical comparison between the gini index and information gain criteria," *Annals of Mathematics and Artificial Intelligence*, vol. 41, no. 1, pp. 77–93, 2004.

[16] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[17] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[18] T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised learning," in *The elements of statistical learning*, Springer, 2009, pp. 485–585.

[19] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[20] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[21] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[22] E. Kirkos, C. Spathis, and Y. Manolopoulos, "Data mining techniques for the detection of fraudulent financial statements," *Expert systems with applications*, vol. 32, no. 4, pp. 995–1003, 2007.

[23] R. Maranzato, A. Pereira, A. P. do Lago, and M. Neubert, "Fraud detection in reputation systems in e-markets using logistic regression," in *Proceedings of the 2010 ACM symposium on applied computing*, ACM, 2010, pp. 1454–1455.

[24] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017, pp. 3146–3154.

[25] M. Bekkar, H. K. Djemaa, and T. A. Alitouche, "Evaluation measures for models assessment over imbalanced datasets," *Iournal Of Information Engineering and Applications*, vol. 3, no. 10, 2013.

[26] M. Bekkar, H. K. Djemaa, and T. A. Alitouche, "Evaluation measures for models assessment over imbalanced datasets," *Iournal Of Information Engineering and Applications*, vol. 3, no. 10, 2013.

[27] N. Chinchor, "The statistical significance of the muc-4 results," in *Proceedings of the 4th conference on Message understanding*, Association for Computational Linguistics, 1992, pp. 30–50.

[28] V. Garcia, R. A. Mollineda, and J. S. Sanchez, "Index of balanced accuracy: A performance measure for skewed class distributions," in *Iberian Conference on Pattern Recognition and Image Analysis*, Springer, 2009, pp. 441–448.

[29] W. J. Youden, "Index for rating diagnostic tests," *Cancer*, vol. 3, no. 1, pp. 32–35, 1950.

[30] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[31] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using matthews correlation coefficient metric," *PLOS ONE*, vol. 12, no. 6, pp. 1–17, Jun. 2017. DOI: 10.1371/journal.pone.0177678. [Online]. Available: https://doi.org/10.1371/journal.pone.0177678.

[32] Z. Ding, "Diversified ensemble classifiers for highly imbalanced data learning and its application in bioinformatics," AAI3486649, PhD thesis, Atlanta, GA, USA, 2011, ISBN: 978-1-267-04661-1.

[33] R. M. Centor, "Receiver operating characteristic (roc) curve analysis using microcomputer spreadsheets," in *Proceedings of the Annual Symposium on Computer Application in Medical Care*, American Medical Informatics Association, 1985, p. 207.

[34] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PLOS ONE*, vol. 10, no. 3, pp. 1–21, Mar. 2015. DOI: 10.1371/journal.pone.0118432. [Online]. Available: https://doi.org/10.1371/journal.pone.0118432.

[35] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 233–240.

[36] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 1135–1144.