Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Classification on small datasets

Semester paper

Nino Courtecuisse

May 25, 2024

Advisors: Prof. Dr. Patrick Cheridito, Dr. Sebastian Becker

Department of Mathematics, ETH Zürich

Contents

Chapter 1

Introduction

In this project we will study a binary classification problem. Our goal is to give a mathematical description of binary classification with an application on a real-life problem.

In *binary classification*, we consider a feature space $\mathcal{X}$, two distinct categories labeled as $\mathcal{Y} = \{0, 1\}$ and we want to find a mapping $f : \mathcal{X} \to \mathcal{Y}$.
The space $\mathcal{X}$ will have the form $\mathcal{X} = \mathcal{X}_1 \times ... \times \mathcal{X}_d$ where $\mathcal{X}_j \subseteq \mathbb{R}$ for all $j = 1, ..., d$. An element $x \in \mathcal{X}_i$ will be called a *feature* or an (explanatory) variable and an element $y \in \mathcal{Y}$ will be called a *label* or a *target*.

In the following chapters, we will present different supervised learning algorithms to find meaningful mappings $f : \mathcal{X} \to \mathcal{Y}$.
Supervised learning means that we will choose a function f of a particular form, depending on parameters and we will adjust or 'learn' the parameters from a dataset $\mathcal{D} \subseteq \mathcal{X}$ to obtain an 'optimal' candidate. The meaning of optimal is not clear so far but will be discussed in the subsequent chapters. We will write $\mathcal{D} = \{(x_1^i, ..., x_d^i, y^i) \in \mathcal{X} \times \mathcal{Y} \mid i = 1, ..., m\}$ where m is the number of data.

One area where binary classification can be put into application is credit risk analytics. The goal is to assess the risk of according a loan to a given person. Therefore, based on a set of data, we want to classify customers in two categories : the non-defaulters, i.e. a person who will pay back the loan, and the defaulters.

From now on, label $y = 0$ corresponds to a non-defaulter and label $y = 1$ corresponds to a defaulter. The class of defaulters will sometimes be referred as the minority class.
This problem has two interesting characteristics that will be discussed in the next chapter : the imbalancedness and the small number of data.
Imbalancedness means that the number of defaulters in our dataset is much smaller than the number of non-defaulters, causing our models to have a

poor training on defaulters cases.
Since a lot a features might influence the creditworthiness of a person, our dataset will typically be small compared to the number of features, and we will therefore need to find a smaller subset of relevant features.

We will first start by presenting solutions to deal with these issues and two different classification models. Then, we will run some empirical tests on real world data. This second part was made possible thanks to the company Emprex, who provided the data. Emprex is building a digital lending system with a cutting-edge machine learning credit scoring engine to assess risk in ways that have never before been possible in Brazil, to offer fair and affordable consumer loans.

Chapter 2

Data processing for binary classification

In this chapter, we will focus on data processing before training a model. We will first present data encoding, i.e. encode techniques to transform raw data into usable datasets for classification algorithms. Secondly, we will see different processes to deal with imbalanced datasets.

2.1 Features encoding

We distinguish between two types of features, namely numerical and categorical ones.

▷ Numerical feature
A numerical feature is a variable taking numerical values.

  ▷ Discrete : a discrete numerical feature is a variable taking finitely many values (e.g. age in years)

  ▷ Continuous : a continuous numerical feature is a variable taking values in an interval (e.g. precise age)

▷ Categorical feature
A categorical feature is a non-numerical variable representing different levels within a category. For example possible levels for a categorical feature ′education′ can be ′graduate′ or ′postgraduate′

  ▷ Ordinal : an ordinal categorical feature is a variable where the levels have a natural ordering (e.g. age coded as young, middle-aged, old)

  ▷ Nominal : a nominal categorical feature is a variable where the levels don′t have a natural ordering (e.g. country of origin)

All the models considered in this project use numerical features as inputs. We thus need methods to transform categorical into numerical features. More generally we study in this section how to encode variables as inputs for classification algorithms.

2.1.1 Categorization

Categorization is the process of grouping several levels of a categorical variable together. Categorization will therefore reduce the number of levels and hence potentially the number of parameters a model will have to estimate, making the model easier to train and more robust.
This process is particularly meaningful on levels with low frequency as they have weak statistical power.

2.1.2 Categorical features encoding

We will illustrate three important encoding techniques with an example of a categorical feature representing the education with three levels: basic, graduate and postgraduate.

One-hot encoding

Given a categorical feature C with categories $\{\mathrm{Cat}_1, ..., \mathrm{Cat}_N\}$, we create N discrete numerical features $C_1, ..., C_N$. We set $C_i = 1$ if $C = \mathrm{Cat}_i$ and 0 otherwise.

Id	Education
1	basic
2	basic
3	graduate
4	postgraduate

$\xrightarrow{one-hot}$

Id	basic	graduate	postgraduate
1	1	0	0
2	1	0	0
3	0	1	0
4	0	0	1

One-hot encoding is very easy to implement but can have serious pitfalls.

(1) Using one-hot encoding for a categorical variable with N categories results into N discrete numerical variables. So this encoding adds a lot of sparse variables to encode a single categorical feature.

(2) The second problem is the linear relationship between the columns. Indeed, we can easily predict the values of one column given the others. This might lead to multicolinearity.

To address (1), we can first apply categorization to reduce the number of levels, hence reducing the number of variables after one-hot encoding.

To address (2), we can consider the first N-1 encoded discrete variables as then we can deduce the last one. This is only meaningful when we train a model that considers all the features simultaneously at each training step, for example in logistic regression, but not in tree-based methods.

Weight of evidence

In weight of evidence (WOE), we use information about the target to assign a numerical value to each level.
For each level, we compute $WOE = \ln\left(\frac{\text{Distribution of non-defaulters in this level}}{\text{Distribution of defaulters in this level}}\right)$.

Education	Non-defaulters	Distr. non-defaulters	Defaulters	Distr. defaulters	WOE
Basic	640	49.2%	120	52.2%	-5.9%
Graduate	560	43.1 %	90	39.1%	9.7%
Postgraduate	100	7.7%	20	8.7%	-12.2%
Total	1300		230		

Id	Education
1	basic
2	basic
3	graduate
4	postgraduate

$\xrightarrow{WOE}$

Id	WOE
1	-5.9
2	-5.9
3	9.7
4	-12.2

Weight of evidence is one solution to encode the predictive power of a variable in relation to the target variable.
For example a level containing 50% of defaulters and 50% of non-defaulters has a very poor predictive power, and will therefore have a weight of evidence of 0.
On the other hand, a level where the distribution of non-defaulters is much higher than the distribution of defaulters will have a highly positive weight of evidence.

As seen in the example above, weight of evidence encoding results in only one variable. It can therefore be a good solution for categorical variables with too many levels to perform one-hot encoding.
On the other hand, weight of evidence encoding uses information about the targets and one should therefore be cautious to avoid overfitting.

Label encoding

Given an ordinal categorical feature, label encoding assigns to each level a number such that the resulting sequence respect the levels ordering.

Id	Education
1	basic
2	basic
3	graduate
4	postgraduate

$\xrightarrow{label}$

Id	Education
1	1
2	1
3	2
4	3

For an ordinal categorical variable with N categories, we only need one discrete numerical variable for the label encoding.

Label encoding can also be applied to nominal categorical variables but, by assigning numbers to categories, label encoding introduces a relation between the categories. This effect must be studied on a case by case basis.

2.2 Imbalanced data

In credit analytics, data are often heavily imbalanced. This means that the number of observations without default is much higher than the number of observations with default.
A consequence of imbalancedness is that non-default observations will be too important for the model and it will tend to classify all observations as non-default.

Undersampling and oversampling

Undersampling consists in removing non-default observations from the training set to make it more balanced.
Similarly oversampling consists in duplicating default observations in the training set.

Id	Variables	Label
1	...	1
2	...	0
3	...	0
4	...	0
5	...	0
6	...	1
7	...	0

$\xrightarrow{Undersampling}$

Id	Variables	Label
1	...	1
2	...	0
5	...	0
6	...	1
7	...	0

Id	Variables	Label
1	...	1
2	...	0
3	...	0
4	...	0
5	...	0
6	...	1
7	...	0

$\xrightarrow{Oversampling}$

Id	Variables	Label
1	...	1
1	...	1
2	...	0
3	...	0
4	...	0
5	...	0
6	...	1
6	...	1
7	...	0

In practice, one can combine undersampling and oversampling.

Synthetic Minority Oversampling Technique (SMOTE)

Instead of only replicating existing observations, SMOTE artificially creates new samples from the minority class using the existing ones.
Given a dataset $\mathcal{D} = \{(x_1^i, ..., x_d^i, y^i) \in \mathcal{X} \times \mathcal{Y} \mid i = 1, ..., m\}$, SMOTE works as follows.

1. Choose a sample $(x^i, y^i) \in \mathcal{D}, i \in \{1, ..., m\}$, from the minority class, i.e. with label $y^i = 1$, and consider its k (hyperparameter) nearest neighbors in the minority class.

2. Depending on the oversampling rate we need, randomly select one or more of the neighbors.

3. For each selected neighbor, consider the segment line between x^i and the selected neighbor. Randomly draw a point on this segment. This gives one new synthetic feature.

For example, take $k = 5$ and an oversampling rate of 100%, i.e. we want to create as many synthetic observations as the number of current observations in the minority class.
For each sample $i \in \{1, ..., m\}$ such that $y^i = 1$, compute the 5-nearest neighbors of sample $(x^i, y^i) \in \mathcal{D}$ in the minority class, and randomly choose one of them, denoted as x^{i_1}. Draw a uniform random number U_i on $(0, 1)$ and construct the synthetic observations as $\tilde{x}^i = x^i * U_i + x^{i_1} * (1 - U_i)$. Label $\tilde{x}^i$ as defaulters and this creates an additional samples for our dataset.

Note that here we assumed to have only continuous variables, so that the sample space is euclidean. But in the original paper [5], they also constructed an extension called SMOTE-NC to perform SMOTE algorithm on mixed datasets of both categorical and continuous features.

Chapter 3

Features selection

In the past years, it has become easy to collect or buy large sets of features. It is therefore common to end up with a small dataset, in comparison to the number of available features. In such a case, features selection is a crucial part for any application of machine learning algorithms.
It improves machine learning applications in at least three ways

- ▷ Faster training speed
- ▷ More accurate predictions: eliminate irrelevant features
- ▷ Easier model interpretation: include only the most relevant features, therefore easier to interpret on which features the model is based

In this section, we will present three different features selection algorithms particularly useful for small datasets.

3.1 Maximum relevance and minimum redundancy feature selection (mRMR)

The goal of mRMR method is to find the features with the maximum relevance towards the target while keeping low redundancy among them. Low redundancy ensures that the selected features are a good representation of the whole features set.
Following [11], [9] and implementation [1], a first possibility is to use the mutual information to measure the relevance between a feature and the target.

Definition 3.1 (Mutual information) *Given two random variables* $X : \Omega \to \mathbb{R}^d, Y : \Omega \to \mathbb{R}$ *defined on the same probability space* $(\Omega, \mathcal{F}, \mathbb{P})$*, with joint density* $p(x,y)$ *and marginal densities* $p(x), p(y)$*, we define the mutual information between*

X and Y as

$$I(X,Y) = \int_{\mathbb{R}} \int_{\mathbb{R}^d} p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)}\right) dxdy$$

Now given a non empty set of features $S \subset \{X_1, ..., X_d\}$ and $X_i \notin S$, $i \in \{1, ..., d\}$, we measure the importance of X_i via the mutual information quotient (MIQ)

$$f^{MIQ}(X_i) = \frac{I(Y, X_i)}{\frac{1}{|S|}\sum_{X_s \in S} I(X_s, X_i)}$$

The intuition behind MIQ is to balance the relevance and the redundancy. We want to select a new variable X_i, $i \in \{1, ..., d\}$, if it has a large mutual information with the target, i.e. a large $I(Y, X_i)$, while having a low mutual information with the already selected variables, i.e. a low $\frac{1}{|S|}\sum_{X_s \in S} I(X_s, X_i)$. Overall, we want to maximize the MIQ, and we deduce the mRMR algorithm.

Algorithm 1 MIQ based mRMR

```
Require: Number of desired features $d'$
  $S = \emptyset$
  while $|S| < d'$ do
      $X = \arg\max_{X_i \notin S} f^{MIQ}(X_i)$
      $S = S \cup \{X\}$
  end while
```

In practice, for continuous variables, the densities can be computationally hard to estimate and therefore the mutual information hard to compute. Mimicking the balancing between relevance and redundancy, we can choose other measures, easier to compute in practice.

Let $\mathcal{D} = \{(x_1^i, ..., x_d^i, y^i) \in \mathcal{X} \times \mathcal{Y} \mid i = 1, ..., m\}$ be our dataset of i.i.d. samples from $(X_1, ..., X_d, Y)$, where, as in definition 3.1, $X = (X_1, ..., X_d) : \Omega \to \mathbb{R}^d$ and $Y : \Omega \to \mathbb{R}$ are defined on the same probability space $(\Omega, \mathcal{F}, \mathbb{P})$.
For $j \in \{1, ..., d\}$, we will denote by $\mathbf{x}_j = (x_j^i)_{i=1}^m$ the samples from the j-th feature and correspondingly $\mathbf{y} = (y^i)_{i=1}^m$.

For the relevance measure, we choose the following statistic between feature $j \in \{1, ..., d\}$ and the target

$$F(\mathbf{y}, \mathbf{x}_j) = \frac{\sum_{k=1}^{2} m_k(\bar{\mathbf{x}}_j^k - \bar{\mathbf{x}}_j)}{\sigma^2}$$

where m_k is the number of samples in class k

$$\bar{\mathbf{x}}_j^k = \frac{1}{m_k}\sum_{i=1}^{m} x_j^i \mathbb{1}_{y^i=k} \quad \text{, is the mean of feature } j \text{ on class } k$$

$$\bar{\mathbf{x}}_j = \frac{1}{m}\sum_{i=1}^{m} x_j^i \quad \text{, is the mean of feature } j$$

$$\sigma^2 = \frac{1}{m-2}\sum_{k=1}^{2}(m_k-1)\sigma_k^2 \quad \text{, is the pooled variance of feature } j$$

$$\sigma_k^2 = \frac{1}{m_k-1}\sum_{i=1}^{m}(x_j^i - \bar{\mathbf{x}}_j^k)^2 \mathbb{1}_{y^i=k} \quad \text{, is the variance of feature } j \text{ on class } k$$

For the redundancy measure, we use the Pearson correlation. This gives the F-test correlation quotient (FCQ), defined as follows for $j \in \{1, ..., d\}$

$$f^{FCQ}(\mathbf{x}_j) = \frac{F(\mathbf{y}, \mathbf{x}_j)}{\frac{1}{|S|}\sum_{\mathbf{x}_s \in S} \rho(\mathbf{x}_s, \mathbf{x}_j)}$$

where $\rho(\mathbf{x}_s, \mathbf{x}_j)$ is the Pearson correlation.
We can now perform mRMR algorithm based on f^{FCQ} to select features.

This feature selection method is very general because it is model free. It can be applied regardless of the final classification model we choose.

3.2 Model based feature selection

If we know in advance which model we want to train, we can use a model specific feature selection.

In chapter 4 and 5 we present two models together with measures of feature importance. After training, feature importance assign a score to each feature indicating how useful this feature is for the model. We can therefore train the chosen model on the whole features set, rank the features according to their importance and choose the top ones.

This method is particularly meaningful when using tree-based method, as these algorithms are less sensitive to the effect of highly correlated features, see chapter 5.
For logistic regression, see chapter 4, the algorithm considers the features all at once and highly correlated features can have a significant impact on the model's performance. Therefore this method may be hard to apply in presence of a dataset with a very large number of features.

3.3 Iterative random feature selection (IRFS)

This method is a variation of the previous one. The idea is to iterate the following steps

(1) Randomly pick a subset of features

(2) Split the selected dataset into training and test set

(3) Train the chosen model on the training set

(4) Evaluate the performance on the test set, see chapter 6

After the iteration, we consider all feature subsets where the performance is higher than a certain threshold. Among these we can select the most important features.

Chapter 4

Logistic regression

In this section, we will present our first classification algorithm. Logistic regression is a statistical model used for binary classification, we describe the model below.

4.1 The model

Let $X \in \mathbb{R}^d$ be a vector of features and Y a binary outcome with conditional distribution $Y|X \sim Ber(p(X))$.
In this section we focus on logistic regression using a linear classifier, i.e. we model the probability $p(X)$ via

$$p(X) = \psi(\beta_0 + \beta_1 X_1 + ... + \beta_d X_d)$$

where $\beta \in \mathbb{R}^{d+1}$ is a parameter, $\psi(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}$ is the logistic function.

We now motivate the choice of the logistic function $\psi : \mathbb{R} \to (0,1)$, plotted in Figure 4.1, over other functions from $\mathbb{R}$ to $[0,1]$.

The function ψ is a bijection with a well-know inverse $y \mapsto ln(\frac{y}{1-y})$, called the logit function.
This gives

$$logit(p(X)) = ln\left(\frac{p(X)}{1-p(X)}\right) = ln\left(\frac{P[Y=1|X]}{P[Y=0|X]}\right) = \beta_0 + \beta_1 X_1 + ... + \beta_d X_d$$

Even if both models have similarities, it is important to note that logistic regression and linear regression are different on at least two key points : the non-gaussian noise and the non-constant variance of the outcome. Both statements are clear from the conditional distribution of Y.

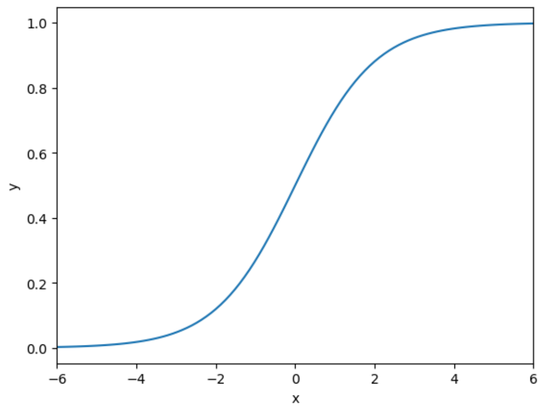


Figure 4.1: The logistic function on [-6, 6]

The linear dependency of the log-odds on the features gives an interpretation of the model.
Suppose that for some $j \in \{1, ..., d\}$, feature X_j increases by 1 unit. Then the log-odds increase by β_j and the odds are multiplied by a factor e^{β_j}. So,

- if $\beta_j > 0$, then the odds increase with X_j
- if $\beta_j < 0$, then the odds decrease with X_j

From this interpretation, we define the feature importance as the absolute value of the coefficient. We can therefore rank the features by importance, the top one being X_{j_0} where $j_0 = \arg\max_{j=1,...,d} |\beta_j|$ and so on. This will be useful in chapter 7.

From the probability distribution of Y, we have that $\mathbb{P}[Y = 1 \mid X] = p(X)$. Therefore the logistic regression model is a function $p : \mathcal{X} \to (0, 1)$ where, in the context of credit analytics, $p(x)$ is the estimated probability that a customer with features $x = (x_1, ..., x_d) \in \mathcal{X}$ default on his loan.
We will see in chapter 6 how to turn the estimated probabilities into classification decisions.

4.2 Training

The model described in the previous section depends on a parameter $\beta \in \mathbb{R}^{d+1}$. We use the maximum likelihood method to estimate this parameter.

Let $\{y^i \mid i = 1, ..., m\}$ be i.i.d. realization of $Y \sim Ber(p)$ for some $p \in (0, 1)$. Then the log-likelihood is given by

$$\mathcal{L}(p|y^1, ..., y^m) = ln\left(\prod_{i=1}^{m} p^{y^i}(1-p)^{1-y^i}\right) = \sum_{i=1}^{m} y^i ln(p) + (1-y^i) ln(1-p)$$

Let $\{(x_1^i, ..., x_d^i, y^i) \mid i = 1, ..., m\}$ be i.i.d. realization of $(X_1, ..., X_d, Y)$.
We set $x_0^i = 1$ for $i = 1, ..., m$ and we write $p_i = \psi({x^i}^T \beta)$.
We then want to solve the following optimization problem

$$\begin{aligned}
&\min_{\beta \in \mathbb{R}^{d+1}} -\sum_{i=1}^{m} y^i ln(p_i) + (1 - y^i) ln(1 - p_i) \qquad (4.1)\\
&= \min_{\beta \in \mathbb{R}^{d+1}} \sum_{i=1}^{m} y^i ln\left(\frac{1 - p_i}{p_i}\right) - ln(1 - p_i)\\
&= \min_{\beta \in \mathbb{R}^{d+1}} \sum_{i=1}^{m} -y^i {x^i}^T \beta + ln(1 + e^{{x^i}^T \beta})
\end{aligned}$$

This problem has no closed-form solution but, as proved below, the objective is convex in β, so we can use stochastic gradient descent to approximate a solution of (4.1)

Lemma 4.1 *Let $x \in \mathbb{R}^{d+1}$, then the map $\beta \mapsto ln(1 + e^{x^T \beta})$ is convex.*

Proof We check that the Hessian is positive semi-definite.
We have $\frac{\partial f}{\partial \beta_j}(\beta) = \frac{x_j e^{x^T \beta}}{1 + e^{x^T \beta}}$ and $\frac{\partial^2 f}{\partial \beta_i \partial \beta_j}(\beta) = x_i x_j \frac{e^{x^T \beta}}{(1 + e^{x^T \beta})^2}$ for $i, j \in \{1, ..., d\}$.
Since $\frac{e^{x^T \beta}}{(1 + e^{x^T \beta})^2} \geq 0$, it is enough to check that the matrix $M \in \mathbb{R}^{(d+1) \times (d+1)}$ given by $M_{i,j} = x_i x_j$ is semi-positive definite.
Let $y \in \mathbb{R}^{d+1} \setminus \{0\}$, then

$$y^T M y = \sum_{i,j=0}^{d} y_i x_i x_j y_j = \left(\sum_{i=0}^{d} y_i x_i\right) \left(\sum_{j=0}^{d} y_j x_j\right) \geq 0 \qquad \square$$

For prediction tasks the MLE will overfit the data. Instead we consider regularized versions of (4.1), i.e. we introduce a bias to reduce the variance.
We consider

$$\min_{\beta \in \mathbb{R}^{d+1}} \sum_{i=1}^{m} \left\{-y^i {x^i}^T \beta + ln(1 + e^{{x^i}^T \beta})\right\} + \lambda \sum_{j=0}^{d} \left\{\alpha \beta_j^2 + (1 - \alpha)|\beta_j|\right\} \qquad \text{(Elasticnet)}$$

where $\lambda \in \mathbb{R}, \alpha \in [0, 1]$ are hyperparameters.
The particular case $\alpha = 1$ is usually called "ridge regularization" and $\alpha = 0$ "lasso regularization".

The objective of the minimization problem is still convex in β.
Ridge regularization is more handy in practice because the objective is differentiable, so we can use stochastic gradient descent to approximate the solution. But lasso regularization tends to set some parameters β_j to 0 during training, therefore selecting explanatory variables.

We will use logistic regression on a practical example in chapter 7.

Chapter 5

Decision trees and gradient boosting

In this section, we develop our second classification algorithm: gradient boosted trees. We will start by presenting general decision trees and then move on to gradient boosting.

5.1 Decision trees

Given a data set $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$, decision trees aim at building a function of the form

$$f(x) = \sum_{i=1}^{k} v_i \mathbb{1}_{P_i}(x) \tag{5.1}$$

for some $x \in \mathcal{X}$, $k \in \mathbb{N}$, some values $\{v_i, i = 1, ..., k\}$ and a partition $\{P_i, i = 1, ..., k\}$ of $\mathcal{X}$.

To construct such a function, we will perform a serie of yes/no decisions which can be summarized in a binary tree.

Definition 5.1 (Binary tree and operations) *A binary tree is any subset $T \subset \bigcup_{l \in \mathbb{N}_0} \{0,1\}^l$ where each $t \in T$ represents a node.*
Given two nodes $t \in \{0,1\}^l$ and $t' \in \{0,1\}^{l'}$, we define

- *$t^{flip} \in \{0,1\}^l$ such that $t_i^{flip} = t_i$ for $i < l$ and $t_l^{flip} = 1 - t_l$*
- *$t^{cut} \in \{0,1\}^{l-1}$ such that $t_i^{cut} = t_i$ for $i < l$*
- *$t \prec t' \iff l < l'$ and $t_i = t'_i$ for $i = 1, ..., l$*

We will denote by $\{()\} \subset \bigcup_{l \in \mathbb{N}_0} \{0,1\}^l$ the set containing the empty sequence. To describe the set of functions of the form 5.1, we need the following definition.

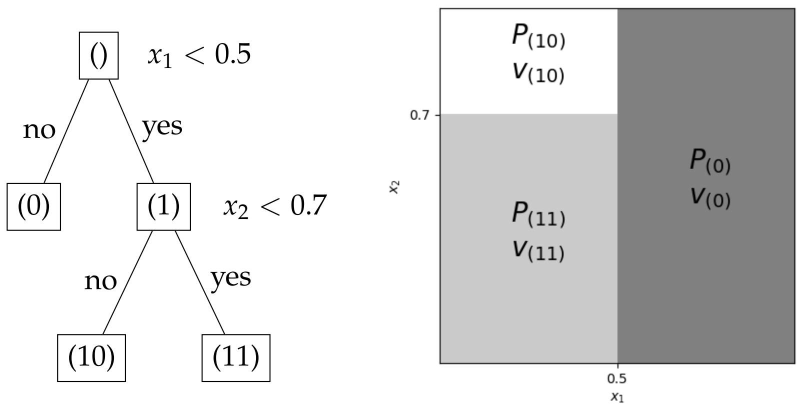


Figure 5.1: Example of a standard binary tree on $\mathcal{X} = [0,1]^2$. The underlying binary tree with the yes/no questions, or 'splits', are represented on the left and the corresponding partition on the right. The leaves are $\{(0),(10),(11)\}$.

Definition 5.2 (Standard binary tree) *A standard binary tree is a triplet (T,P,V) where*

- $T \subset \cup_{l\in\mathbb{N}_0}\{0,1\}^l$ *is a non-empty finite set such that for all $t \in T$, $t^{flip} \in T$ and $t^{cut} \in T$*
- $P = \{P_t \mid t \in \mathcal{T}\} \subset \mathcal{P}(\mathcal{X})$ *is such that $\cup_{t\in\mathcal{T}} P_t = \mathcal{X}$, $P_t = P_{t^{cut}} \cap \{x \in \mathcal{X} \mid x_i \in C_i\}$ for a dimension $i \in \{1,...,d\}$ and a certain half-space $C_i \subset \mathcal{X}_i$*
- $V = \{v_t \mid t \in \mathcal{T}\}$

Definition 5.3 (Leaves of a tree) *Given a standard binary tree (T,P,V), we define its set of leaves as $\mathcal{T} = \{t \in T : t \not< t' \text{ for any } t' \in T\}$*

Figure 5.1 represents a standard binary tree.
Given a standard binary tree (T,P,V), we define its decision function on $\mathcal{X}$ via $f = \sum_{t\in\mathcal{T}} v_t \mathbb{1}_{P_t}$.

5.1.1 The CART Algorithm

Classification and regression trees (CART) algorithm is a growing tree algorithm, i.e. an algorithm to fit a decision tree to data. The intuitive idea is to find a tree that splits the data into purer and purer subsets.
This can be formalized by defining the of impurity of a node. We first need the following quantities.

Definition 5.4 (Estimated probabilities of a tree) *Let (T,P,V) be a standard binary tree, a leaf $t \in \mathcal{T}$, a dataset $\{(x^i,y^i) \in \mathcal{X} \times \mathcal{Y}, i = 1,...,m\}$ and $y \in \mathcal{Y}$. We*

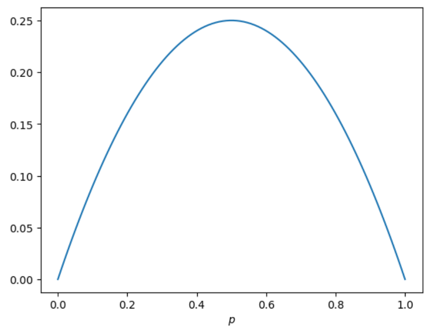


Figure 5.2: Gini impurity for binary classification

introduce the estimated probability that $x \in P_t$

$$p(t) = \frac{\#\{i \in \{1,...,m\} : x^i \in P_t\}}{m}$$

and the estimated probability of being classified as y *given that* $x \in P_t$

$$p(y \mid t) = \frac{\#\{i \in \{1,...,m\} : x^i \in P_t, y^i = y\}}{\#\{i \in \{1,...,m\} : x^i \in P_t\}} \tag{5.2}$$

We can now define the Gini impurity of a leaf, and extend it to the impurity of a tree.

Definition 5.5 (Gini impurity) *Let* (T, P, V) *be a standard binary tree, a leaf* $t \in \mathcal{T}$ *and a dataset* $\{(x^i, y^i) \in \mathcal{X} \times \mathcal{Y}, i = 1, ..., m\}$ *and* $y \in \mathcal{Y} = \{1, ..., K\}$ *for* $K \in \mathbb{N}$.
The Gini impurity of is the map $I : \mathcal{T} \to [0, 1]$ *given by*

$$I(t) = \frac{1}{K}\sum_{k=1}^{K} p(k \mid t)(1 - p(k \mid t))$$

In the context of binary classification, since $p(1 \mid t) + p(1 \mid t) = 1$, this reduces to $I : \mathcal{T} \to [0, \frac{1}{4}]$ given by

$$\begin{aligned} I(t) &= \frac{1}{2}p(0 \mid t)(1 - p(0 \mid t)) + \frac{1}{2}p(1 \mid t)(1 - p(1 \mid t)) \\ &= p(1 \mid t)(1 - p(1 \mid t)) \end{aligned}$$

Suppose that leaf t only contains samples from one class, e.g. only labeled as $y = 0$, then $p(0 \mid t) = 1$ and $p(1 \mid t) = 0$. This gives $I(t) = 0$ and the leaf has the lowest possible impurity.
If leaf t contains an equal number of samples from each class, then $p(0 \mid t) = 0.5$ and $p(1 \mid t) = 0.5$. This gives $I(t) = \frac{1}{4}$, the highest impurity for binary classification.

Definition 5.6 (Total Gini impurity) *Let (T, P, V) be a standard binary tree, a dataset $\{(x^i, y^i) \in \mathcal{X} \times \mathcal{Y}, i = 1, ..., m\}$ and $I : \mathcal{T} \to [0, 1]$ the Gini impurity. We define the impurity of the tree as*

$$\bar{I}(T) = \sum_{t \in \mathcal{T}} p(t) I(t)$$

Before presenting CART algorithm, we need the following notation. Given a tree (T, P, V), $t \in \mathcal{T}$, $j \in \{1, ..., d\}$ and $C_j \subseteq \mathcal{X}_j$ we define $T^{(t,j,C_j)}$ to be the set of nodes corresponding to a tree obtained by splitting leaf t along C_j, i.e. a tree with nodes $T \cup \{t0, t1\}$ and partition $P \cup \{P_{t0}, P_{t1}\} \setminus \{P_t\}$ where $P_{t0} = P_t \cap \{x \in \mathcal{X} : x_j \notin C_j\}$ and $P_{t1} = P_t \cap \{x \in \mathcal{X} : x_j \in C_j\}$.

CART

Given a dataset $\{(x^i, y^i) \in \mathcal{X} \times \mathcal{Y}, i = 1, ..., m\}$, a regularization parameter $\alpha > 0$ and the impurity function I, the algorithm goes as follows:

(1) Start with an empty tree $T = \{()\}, P_{()} = \mathcal{X}$

(2) Iterate over all leaf $t \in \mathcal{T}$:

   a) Find an optimal split candidate $(j, C_j) = \arg \min_{\substack{j' \in \{1,...,d\} \\ C_{j'} \subseteq \mathcal{X}_{j'}}} \bar{I}(T^{t,j',C_{j'}})$, where the optimization is done only over half-spaces $C_{j'} \subseteq \mathcal{X}_{j'}$

   b) If the split candidate is such that $\bar{I}(T^{(t,j,C_j)}) < \bar{I}(T) - \alpha$, then we grow the tree along this split i.e. we set
      $T = T \cup \{t0, t1\}$
      $P_{t0} = P_t \cap \{x \in \mathcal{X} : x_j \notin C_j\}$
      $P_{t1} = P_t \cap \{x \in \mathcal{X} : x_j \in C_j\}$

      If not the we stop splitting this leaf, i.e. we stop considering this leaf during future iterations of 2.

(3) New leaves where potentially added to the tree during the iteration, so restart (2) as long as there are leaves to consider.

Assigning values

CART algorithm will result in the tree structure (T, P) but we still need to assign values $V = \{v_t \mid t \in \mathcal{T}\}$ to each leaf. One possibility to directly have a model that decides on a class is to use majority voting,

$$y_t = \arg \max_{c \in \mathcal{Y}} \{i \in \{1, ..., m\} : x^i \in P_t, y^i = c\}, t \in \mathcal{T}$$

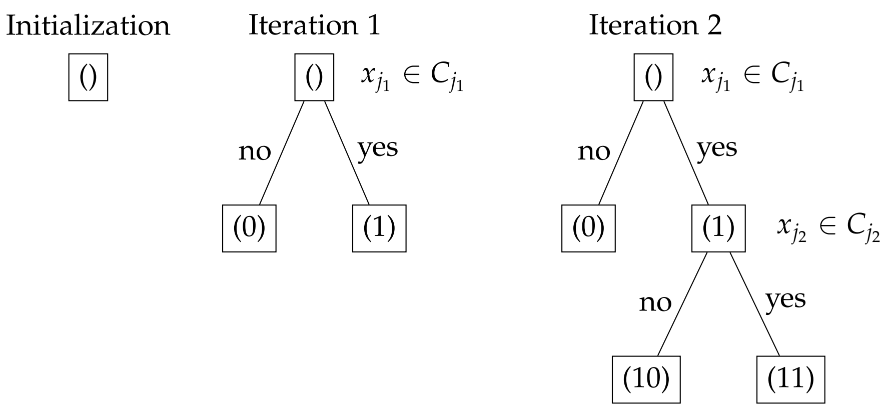


Figure 5.3: Two iterations of the CART algorithm. It starts with an empty tree $T = \{()\}$. During Iteration 1, there is only one leaf $()$. The optimal split is given by (j_1, C_{j_1}) and it satisfies the regularization condition (2)b so we grow the tree. During Iteration 2, there are 2 leaves to check $(0), (1)$. The optimal split over (0) does not satisfy the regularization condition (2)b, so we don't split and (0) will not be considered in the next iteration. The optimal split over (1) is given by (j_2, C_{j_2}) and satisfies (2)b so we perform the split.

For the particular case of binary classification, we are more interested in the default probabilities and we will therefore assign the estimated probabilities $p(1 \mid t), t \in \mathcal{T}$ to each leaf.
Our model of decision trees therefore outputs probabilities and we will see in chapter 7 how to turn the estimated probabilities into classification decisions.

5.2 Gradient boosted trees

In this section we describe gradient boosted trees, following the XGBoost implementation [2] and [6]. As usual, we consider a dataset $\{(x^i, y^i) \in \mathcal{X} \times \mathcal{Y}, i = 1, ..., m\}$ where $\mathcal{Y} = \{0, 1\}$.

5.2.1 The additive model

The idea is to sequentially train several decision trees and to use a greedy approach, similar to gradient descent, for training.

Let $K \in \mathbb{N}$, we consider the model

$$f = \sum_{k=1}^{K} f_k$$

where each f_k is a decision tree (T_k, P_k, V_k).

5.2.2 Training

As mentioned before, we want to train sequentially each $f_k, k = 1, ..., K$. This is precised and described in what follows.

For $i \in \{1, .., m\}$ denote by $\hat{y}^{i,(s)} = \sum_{k=1}^{s} f_k(x^i)$ the prediction at the s^{th}-step. We set $\hat{y}^{i,(0)} \in (0,1)$ to be an initial guess and then we have

$$\begin{aligned}
&\hat{y}^{i,(1)} = f_1(x^i) = \hat{y}_i{}^{(0)} + f_1(x^i) \\
&\hat{y}^{i,(2)} = f_1(x^i) + f_2(x^i) = \hat{y}^{i,(1)} + f_2(x^i) \\
&...
\end{aligned}$$

Since $\hat{y}^{i,(0)}$ is already specified, we only need to specify the induction step of the sequential training.
Let $s \geq 1$ and assume we trained the first $(s-1)$ summands of the model, so that we can compute $\hat{y}^{i,(s-1)}$.

Given a loss function $\ell : \mathcal{Y} \times (0,1)$, we want to optimize the following regularized objective over f_s

$$\begin{aligned}
Obj^{(s)} &= \sum_{i=1}^{m} \ell(y^i, \hat{y}^{i,(s)}) + \sum_{k=1}^{s} \Omega(f_k) \\
&= \sum_{i=1}^{m} \ell(y^i, \hat{y}^{i,(s-1)} + f_s(x^i)) + \sum_{k=1}^{s} \Omega(f_k) \qquad (5.3)
\end{aligned}$$

where the second term is a regularization, given by

$$\Omega(f_k) = \gamma|\mathcal{T}_k| + \frac{1}{2}\lambda \sum_{t \in \mathcal{T}_k} {v_t^{(k)}}^2$$

with $\mathcal{T}_k$ the leaves of $f_k = (T_k, P_k, V_k)$ and $\{v_t^{(k)} \mid t \in \mathcal{T}_k\}$ the set of values V_k.

For binary classification, we can use the negative cross-entropy $\ell : \mathcal{Y} \times (0,1)$ defined by

$$\ell(y, \hat{y}) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

Gradient boosting step

To optimize (5.3), we want to use an approach similar to gradient descent while remaining in the space of sums of decision trees. One possibility is to do a second order approximation of ℓ as follows

$$Obj^{(s)} \simeq \sum_{i=1}^{m} \left[\ell(y^i, \hat{y}^{i,(s-1)}) + g_i f_s(x^i) + \frac{1}{2} h_i f_s(x^i)^2 \right] + \sum_{k=1}^{s} \Omega(f_k)$$

where $g_i = \partial_{\hat{y}} \ell(y^i, \hat{y})_{|\hat{y}=\hat{y}^{i,(s-1)}}$ and $h_i = \partial_{\hat{y}}^2 \ell(y^i, \hat{y})_{|\hat{y}=\hat{y}^{i,(s-1)}}$

After removing all the constants, we obtain our new objective for step s

$$\begin{aligned} Obj^{(s)} &= \sum_{i=1}^{m} \left[g_i f_s(x^i) + \frac{1}{2} h_i f_s(x^i)^2 \right] + \Omega(f_s) \\ &= \sum_{i=1}^{m} \left[g_i f_s(x^i) + \frac{1}{2} h_i f_s(x^i)^2 \right] + \gamma|\mathcal{T}| + \frac{1}{2}\lambda \sum_{t\in\mathcal{T}} v_t^2 \end{aligned} \tag{5.4}$$

where $\mathcal{T}$ is the set of leaves and $V = \{v_t \mid t \in \mathcal{T}\}$ are the assigned values of the tree corresponding to f_s.

Let us further denote P the partition set corresponding to f_s and for $t \in \mathcal{T}$, we introduce $I_t = \{i \in \{1, ..., m\} \mid x^i \in P_t\}$, the set of indices of data points assigned to leaf $t \in \mathcal{T}$.
Then we can rewrite (5.4) as

$$Obj^{(s)} = \sum_{t\in\mathcal{T}} \left[\left(\sum_{i\in I_t} g_i \right) v_t + \frac{1}{2} \left(\sum_{i\in I_t} h_i + \lambda \right) v_t^2 + \gamma \right] \tag{5.5}$$

Solving the optimization problem

To solve the optimization problem (5.5), we are now left with two questions :

(1) How to find an optimal structure for the tree f_s, i.e. (T, P)?

(2) How to find optimal values to assign to the leaves, i.e. V?

First assume we solved (1), then we can solve (2) as follows:
The objective (5.5) is a quadratic form in $\{v_t \mid t \in \mathcal{T}\}$. First and second order conditions directly give the optimal parameters

$$v_t^* = -\frac{\sum_{i\in I_t} g_i}{\sum_{i\in I_t} h_i + \lambda} = -\frac{G_t}{H_t + \lambda}, t \in \mathcal{T}$$

where $G_t = \sum_{i\in I_t} g_i$ and $H_t = \sum_{i\in I_t} h_i$

The optimal objective value is then

$$Obj^{(s)^*} = -\frac{1}{2} \sum_{t\in\mathcal{T}} \frac{G_t^2}{H_t + \lambda} + \gamma|\mathcal{T}| \tag{5.6}$$

We can now solve (1) by considering two subproblems:

(1.1) How to find optimal splits?

(1.2) When to stop splitting?

Let $\tilde{t} \in \mathcal{T}$, how to find an optimal split at $\tilde{t}$?
Consider $j \in \{1, ..., d\}$, $C_j \subseteq \mathcal{X}_j$ an half-space and assume we choose this

split.
Let $I_L = \{i \in \{1, ..., m\} \mid x^i \in P_{\tilde{t}0}\}$ and $I_R = \{i \in \{1, ..., m\} \mid x^i = P_{\tilde{t}1}\}$ be the sets of indices of data points assigned to the two new leaves.

From (5.6), the objective of leaf $\tilde{t}$ is

$$Obj_{\tilde{t}}^{(s)^*} = -\frac{1}{2}\frac{G_{\tilde{t}}^2}{H_{\tilde{t}} + \lambda} + \gamma$$

and so we define the gain of this split by

$$gain(j, C_j, \tilde{t}) = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{G_{\tilde{t}}^2}{H_{\tilde{t}} + \lambda}\right] - \gamma \tag{5.7}$$

This definition is such that the following lemma holds

Lemma 5.7

$$Obj_{after\ split}^{(s)^*} = Obj_{before\ split}^{(s)^*} - gain(j, C_j, \tilde{t})$$

Proof

$$\begin{aligned}
Obj_{\text{before split}}^{(s)^*} - gain(j, C_j, \tilde{t}) &= \sum_{t \in \mathcal{T}} \left(-\frac{1}{2}\frac{G_t^2}{H_t + \lambda} + \gamma\right) \\
&\quad - \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{G_{\tilde{t}}^2}{H_{\tilde{t}} + \lambda}\right] + \gamma \\
&= \sum_{\substack{t \in \mathcal{T} \\ t \neq \tilde{t}}} \left(-\frac{1}{2}\frac{G_{\tilde{t}}^2}{H_{\tilde{t}} + \lambda} + \gamma\right) + 2\gamma \\
&\quad - \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda}\right] \\
&= Obj_{\text{after split}}^{(s)^*}
\end{aligned}$$

□

Therefore, for all $t \in \mathcal{T}$ we need solve

$$(j, C_j) = \arg \max_{\substack{j' \in \{1, ..., d\} \\ C_{j'} \subseteq \mathcal{X}_{j'}}} gain(j', C_{j'}, t)$$

where the optimization is done only over half-spaces $C_{j'} \subseteq \mathcal{X}_{j'}$.

If $\mathcal{X}_i$ has an natural order then we can simply order increasingly $\{x_j^i : i = 1, ..., m, x^i \in P_t\}$, say $x_j^{(1)} \leq ... \leq x_j^{(n_{t,i})}$, and a brute force search will take at most $n_{t,i}$ many steps.

Finally, to answer (1.2), we can follow CART algorithm 5.1.1 using the gain (5.7) instead of the impurity and only splitting if the gain is positive.

Feature importance

Using XGBoost as described above, we simply define the importance of a feature as the average gain across all splits where this feature was used. This will be useful for chapter 7.

Chapter 6

Diagnostic measures

To assess the performance of a model we need diagnostic measures.
Both models presented before output the estimated probability that a given sample $x \in \mathcal{X}$ is classified as a defaulter, denoted as $\hat{p}(x)$ in this section.
To take a classification decision based on such models, one can proceed as follows. We choose a decision threshold $c \in (0, 1)$ and set $\hat{y}^i = 1$ if $\hat{p}(x^i) \geq c$ and 0 otherwise.

In this section, we will use the following terminology, a sample $(x^i, y^i) \in \mathcal{D}$ is

- ▷ positive (P) if it is a defaulter, i.e. $y^i = 1$
- ▷ negative (N) if it is a non-defaulter, i.e. $y^i = 0$
- ▷ true positive (TP) if $\hat{y}^i = 1$ and $y^i = 1$
- ▷ false positive (FP) if $\hat{y}^i = 1$ and $y^i = 0$
- ▷ true negative (TN) if $\hat{y}^i = 0$ and $y^i = 0$
- ▷ false negative (FN) if $\hat{y}^i = 0$ and $y^i = 1$

A straightforward diagnostic measure is the accuracy, given by $A = \frac{TP+TN}{P+N}$, i.e. the proportion of correctly predicted classes.
However, when dealing with imbalanced datasets the accuracy is not the best measure since a model can reach very high accuracy by predicting all samples to be in the majority class.
We therefore need a diagnostic measure with an emphasis on the minority class.

6.1 AUC

One possibility is to consider the performance on the majority and on the minority class separately. This can be done using the true positive rate (TPR) and the true negative rate (TNR), defined as follows

$$TPR = \frac{TP}{P} \qquad TNR = \frac{TN}{N}$$

We therefore want to find a model with a high TPR and a high TNR.

Instead of considering the TNR, we will instead use the false positive rate (FPR) defined as

$$FPR = \frac{FP}{N} = 1 - TNR$$

We therefore want a model with a high TPR and a low FPR.
It is natural to consider the parametric curve (TPR, FPR) as the decision threshold c ranges in $[0, 1]$ and compute the area under this curve (AUC).
A large area would show that the model can achieve a high TPR with a low FPR.

Chapter 7

Application

7.1 Dataset presentation

In this section we use the data provided by the company Emprex to empirically test some of the techniques presented before.

One of the major difficulty is that the raw dataset is very small compared to the number of features, it consists of 1874 loan samples for 1624 features.
The features we use range from financial information about the borrower (such as last incomes) to personal information (such as marital status or employment type). In addition we used derived data (such has geographical data or information about phone devices).
Each sample is labeled according to our convention, 1 for a defaulter and 0 for a non-defaulter.

We start by encoding the categorical features. For this we will use either one-hot or label encoding. After encoding, our dataset has 1658 features.
We will also need to deal with imbalancedness since our dataset contains around 73% of non-defaulters and 27% of defaulters.

7.2 Empirical comparison

In this section we use our loans dataset to compare the methods presented earlier, namely logistic regression and gradient boosted trees on the different features selection methods.
After each feature selection, we have two different pipelines for each model: with or without using SMOTE.

After feature selection, we want to train and assess the performance of our models using the ROC and its AUC, see Chapter 6. To have robust estimates on both the test and the train sets, we use stratified K-Fold cross validation, a variation of K-Fold cross validation where the folds are made by preserving

Table 7.1: Average AUC scores on the test set (maximum value of each column is highlighted)

Model	Logistic regression			XGBoost		
# Features	50	100	300	50	100	300
Model based	0.706	0.728	**0.736**	0.704	0.706	0.697
Model based + SMOTE	0.706	0.724	0.731	0.700	**0.721**	**0.710**
IRFS	0.710	0.714	0.699	0.685	0.692	0.696
IRFS + SMOTE	0.703	0.704	0.684	0.687	0.693	0.707
mRMR	**0.734**	**0.733**	0.702	0.707	0.704	0.678
mRMR + SMOTE	0.733	0.730	0.695	**0.711**	0.714	0.706
All features	0.579			0.685		

Table 7.2: Average AUC scores on the train set

Model	Logistic regression			XGBoost		
# Features	50	100	300	50	100	300
Model based	0.750	0.801	0.883	0.902	0.926	0.928
Model based + SMOTE	0.746	0.793	0.874	0.873	0.897	0.910
IRFS	0.752	0.784	0.848	0.878	0.883	0.930
IRFS + SMOTE	0.747	0.777	0.839	0.843	0.859	0.912
mRMR	0.770	0.797	0.839	0.844	0.878	0.907
mRMR + SMOTE	0.768	0.794	0.827	0.824	0.858	0.893
All features	0.967			0.929		

the proportion of defaulters.
We train our model on each fold, compute the ROC and measure the AUC on both the training and the test set of this fold. We then average the ROC and the AUC over each fold. Figure 7.1 shows the output of this process on the test set, with the ROC's of each fold and the mean ROC.

In tables 7.1 and 7.2 we reported the average AUC scores on the test and the train set respectively, for different feature selection methods, with a different number of selected features and with or without using SMOTE after features selection.

Overall, we could reach a higher AUC score on the test set using logistic regression. Our XGBoost models have a significantly higher AUC on the train set, indicating that these models overfit the train set compared to our logistic regression models. This may cause a poorer generalization to unseen data, explaining the smaller AUC on the test set. Since XGBoost is a more

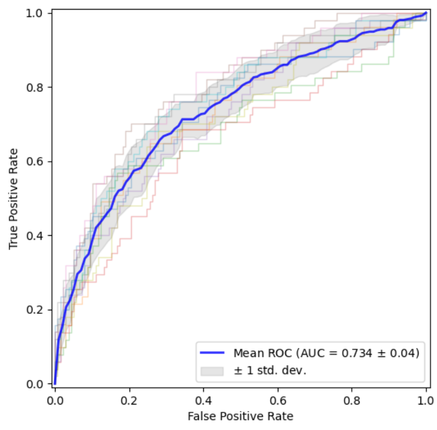


Figure 7.1: Logistic regression ROC using 50 features, selected by mRMR

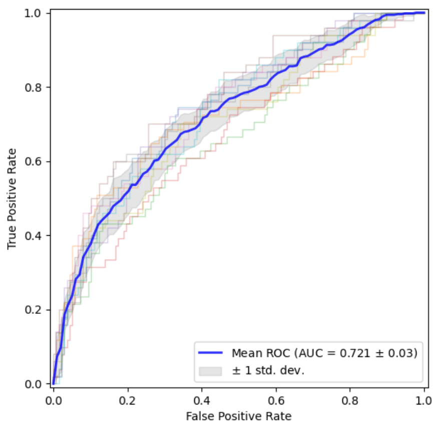


Figure 7.2: XGBoost ROC using 100 features, selected by XGBoost and using SMOTE

complex model in terms of parameters, it was harder to train and especially to prevent overfitting.

Looking at table 7.2, we see that the utilization of SMOTE reduces the AUC on the train set. When using logistic regression models this is accompanied by a reduction of AUC on the test set. But for XGBoost, we generally see an improvement in the AUC on the test set.

In the following, we will focus on the best performing models. For logistic regression, the maximum we could reach is 0.736 AUC using 300 features. Since this number of features is too big to interpret the model and because the AUC using only 50 features is 0.734, we will use the latter.
For gradient boosted trees models, we could reach an AUC of 0.721 using 100 features. This is still a fairly large number but our model performance decreased significantly when using less features.

Figures 7.1 and 7.2 show the mean roc's of these models, all roc's computed during cross validation, the mean auc and its variance.

These curves are steep in the beginning but flatten significantly for higher false positive rates. This means that for a given target of true positive rate, we will need to tolerate a relatively high false positive rate.

To compare the features selected by the different selection methods, figures 7.3 and 7.4 show the correlation between the top 10 selected features by our two best models and the correlation between the features and the target variable (denoted as Y in the plots below).

We can see that the top 1 feature is always highly correlated to the target variable. We also see that the top 10 features for logistic regression are less

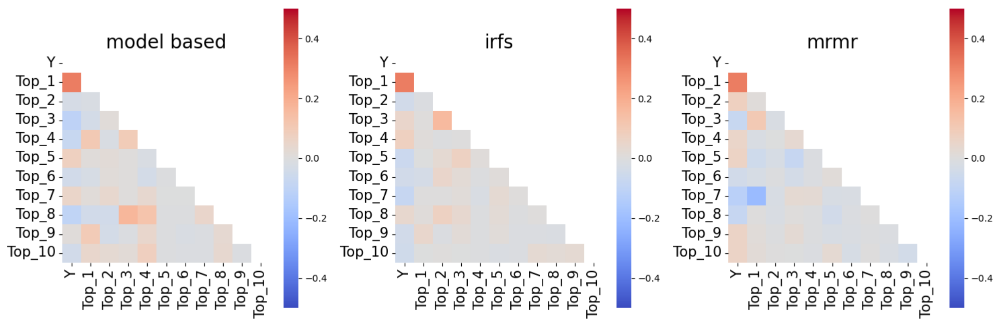


Figure 7.3: Correlation heatmaps of top 10 features and target variable, using logistic regression with 50 features

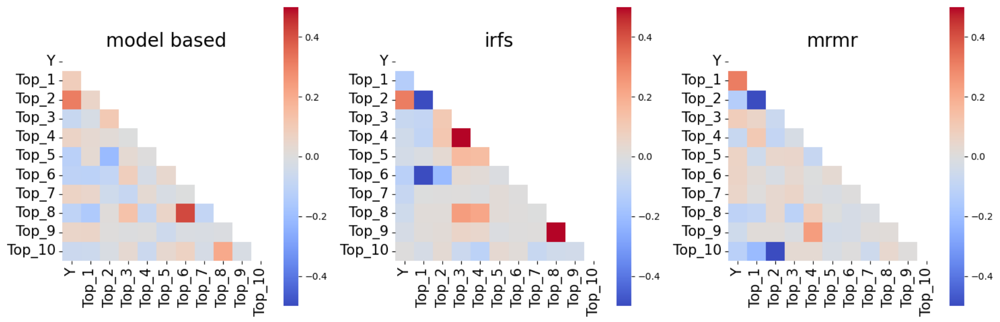


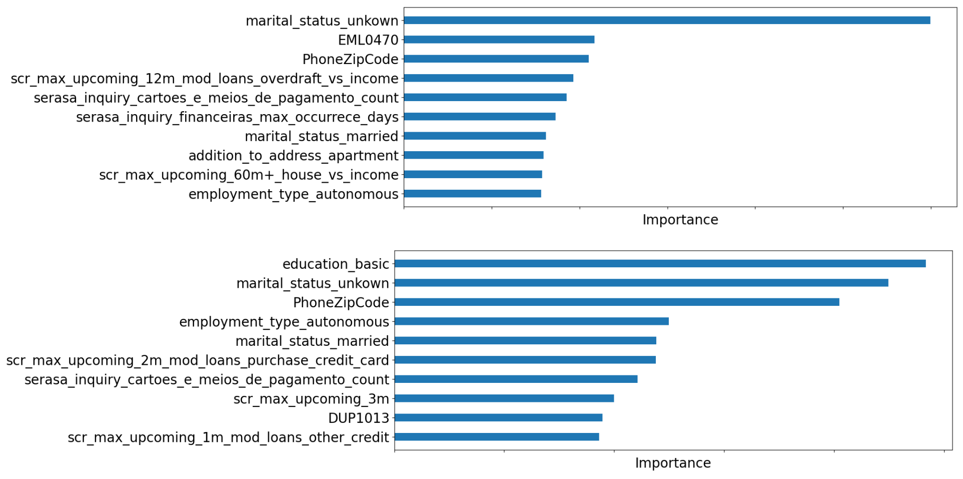
Figure 7.4: Correlation heatmaps of top 10 features and target variable, using XGBoost with 100 features

correlated among each other than the top 10 for XGBoost where we still have 1 to 2 pairs with very high correlations.

When looking at Figure 7.5, we can see that the feature named `'marital_status_unknown'` has a very high importance. We can retrospectively understand this because Emprex modified the form to get borrower information to have the marital status as mandatory. So if a customer still has its marital status set to unknown it means that the customer did not use the platform since this update and will likely never come back to pay its loan.

Overall, these feature selection methods are good to reduce drastically the number of features. In our case we could go from 1568 to 50 features. The impact of resampling using SMOTE was only seen using XGBoost and even for this model often not significant. In this example the feature selection mRMR seems to work especially well with logistic regression, where XGBoost feature selection seems more adapted to an XGBoost model.

Figure 7.5: Top 10 features of the best performing logistic regression model (top) and top 10 features of the best performing XGBoost model (bottom)



Chapter 8

Conclusion

This project started with the desire to learn more about machine learning and especially with a real world application. Having a background focused on more abstract topics in mathematics, I learned a lot about the difficulties that may arise when going from a theoretical model to a real-life training.
It was also my first academic work on such an applied topic so it was interesting to experience the difficulties one can encounter when sharing experimental results.

Acknowledgments

I am grateful to Prof. Dr. Patrick Cheridito who accepted to host my semester paper and always gave me prompt feedback on my work. I would also like to thank my supervisor, Dr. Sebastian Becker, for his guidance, his advice and of course for his time to answer all my questions. As a co-founder, Sebastian was the link to Emprex, and among other things, he introduced me to the data. Finally, I want to thank Emprex and Silvan Roth, co-founder and CEO, to allow me using their data for my semester paper.

Bibliography

[1] Mrmr, github public repository. https://github.com/smazzanti/mrmr. Accessed: 2024-05-15.

[2] Xgboost documentation. https://xgboost.readthedocs.io/en/stable/tutorials/model.html. Accessed: 2024-03-24.

[3] Bart Baesens, Daniel Roesch, and Harald Scheule. Credit Risk Analytics.

[4] L. Breiman, J. Friedman, R.A. Olshen, and C.J. Stone. *"Classification And Regression Trees"*. Chapman and Hall/CRC, 1984.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, August 2016.

[7] Patrick Cheridito. Lecture notes 'machine learning in finance and insurance. *ETHZ*.

[8] Gregory W Corder and Dale I Foreman. Nonparametric Statistics.

[9] C. Ding and H. Peng. Minimum redundancy feature selection from microarray gene expression data. In *Computational Systems Bioinformatics. CSB2003. Proceedings of the 2003 IEEE Bioinformatics Conference. CSB2003*, pages 523–528, 2003.

[10] Victor Panaretos. Lecture notes on linear models. *EPFL*.

[11] Zhenyu Zhao, Radhika Anand, and Mallory Wang. Maximum relevance and minimum redundancy feature selection methods for a marketing machine learning platform, 2019.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

CLASSIFICATION ON SMALL DATASETS

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
COURTECUISSE	NINO

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 26.05.2024	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.