The sample Pearson Correlation Coefficient for classification models and identifying platform economy businesses from web-scraped data

Gubbels, Luuk W.G.

Award date:
2023

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Statistics Group

The sample Pearson Correlation Coefficient for classification models and identifying platform economy businesses from web-scraped data

Master Thesis

Luuk W.G. Gubbels

Committee:

Prof. dr. Piet J.H. Daas	(supervisor)
dr. Marco J.H. Puts	(supervisor)
dr.ir. Erik Quaeghebeur	(committee member)
C.Riddlesden	(advisory committee member)

Eindhoven, January 2023

Abstract

Statistics Netherlands (CBS) set out to classify platform economy businesses and estimate the number of platform economy businesses using web-scraped data. This is a problem with elements of natural language processing (NLP) and extreme rare event detection. In classification problems, it is common practice to predict binary labels. This thesis sets out to use probabilities instead. This, in combination with NLP and extreme rare event detection, causes for a unique problem. Measuring the performance of classification models on such a problem requires a new approach that comes in the form of the sample Pearson Correlation Coefficient (sPCC). Derivations for binary and multiclass classification, relations between the sPCC and other metrics and test cases using the sPCC will be presented here in order to show its effectiveness as a metric. Ultimately, solutions to the original problem of the CBS will be presented and tested using the sPCC.

Preface

This thesis first started as an assignment regarding improving classification models on the web-scraped platform economy dataset from CBS. In the first few months, several strategies were tested and methods were applied, such as the Bayesian Calibration method, developed by Puts and Daas. As the Bayesian Calibration had not been applied to the platform economy dataset, it could provide a solution. A combination of difficulties lead to a new problem that required solving before the platform economy dataset could be tackled: there was no metric that could sufficiently describe performances. And so, a new part of the thesis was born: the use of the sample Pearson Correlation Coefficient for classification models.

I would like to thank several people. First of all, I would like to thank my supervisors, Prof. dr. Piet J.H. Daas and Dr. Marco Puts, for their help during the entire project, helping me with understanding the mathematical importance in machine learning, understanding the connections machine learning has to other fields of research, understanding the problems of data acquisition and bearing some of the 5 hour long brainstorming sessions. I would like to thank CBS for giving me the opportunity to work on this project.

Contents

0.1 Introduction

The CBS states that it their goals is to "publish trustworthy and coherent statistical information, that plays into the needs of the society" (from CBS' website). One of such pieces of statistical information is the distribution of types of companies. In other words: how many companies are there of a certain type (flower shops for example). One of the approaches that has been explored is using web-scraped data (from company websites) and predicted probabilities instead of predicted labels formed by classification models. Studies have been conducted on this avenue for different types of companies: innovative companies [1], platform economy companies [2][3] and drone companies [4]. This thesis will focus on the platform economy company dataset. A property of this dataset is that it is a binary labeled set, with an extremely low number of positively labeled samples (less than 0.5%). Not only is this problem a natural language processing problem, it is also an extreme rare event detection problem. Combining this with the probability prediction, creates a very unique challenge. The aforementioned studies go into the natural language processing and extreme rare event detection part of the problem, but not into the additional challenges that occur when predicting probabilities instead of labels.

In this thesis, we will try to improve on the current state of the art model that exist for the platform economy dataset, while also developing tools to more accurately measure the performance of these models.

This thesis will consist of a few introductory section and two chapters. The introductory sections consist of this introduction (Section 0.1), the preliminaries (Section 0.2), where some background information is given regarding some machine learning algorithms and metrics, and a problem description (Section 0.3), where the full problem this thesis tackles will be discussed and split into two separate problems: i) the use of the sample Pearson Correlation Coefficient and ii) classification models on the platform economy dataset. Each of these problems will have their own chapter dedicated to them.

The first chapter (Chapter 1) will revolve around the use of the sample Pearson Correlation Coefficient (sPCC) as a performance measure (more commonly referred to as metrics [1] in machine learning) for classification models and will give a short overview of the background of the sPCC and its definition. In Section 1.1 we will provide a few cases as to why it could prove useful to use the sPCC as a metric for classification models. As with other classification problems, a binary classification problem is less complex than a multiclass classification problem. As such, we will first look at how the sPCC can be used in a binary case in Section 1.2. We will derive an expression fit for the binary classification context in Section 1.2.1, as well as go into the relations the sPCC has to other metrics in Section 1.2.2. We will finish Section 1.2 by including some small tests on synthetic and real-world data in Section 1.2.3 and discussing the results of these tests in Section 1.2.3. In Section 1.3 the multiclass case will be discussed. In Section 1.3.1, a derivation similar to the binary case will be shown, though some choices have to be made which will also be explained. Since the emphasis of this thesis lies on binary classification, we will only show a test on a synthetic dataset in Section 1.3.2. The final section of Section 1.3 is Section 1.3.2, where the results of the tests will be discussed. We conclude Chapter 1 with Section 1.4, where a small conclusion will be made regarding the use of the sPCC.

The second chapter (Chapter 2) will introduce the problem of the platform economy dataset: a set consisting of texts of several different websites owned by platform economy and non-platform economy businesses. The chapter will be opened with a small summary of the problem description. In Section 2.1 several methods and approaches will be presented. Each method has its own subsection dedicated to it. After the methods have been presented, they will be put to the test and their results will be discussed in Section 2.3.

The entire thesis will be wrapped up with two sections: an overall conclusion in Section 2.4 and future research and discussion in Section 2.5.

[1] This name might be misleading for some, as most machine learning "metrics" are not metric functions used in metric spaces

0.2 Preliminaries

Decision boundary

In classification problems, decision functions are used to classify data. Such a decision function is a real-valued function f that scores data based on a set of parameters. This set of parameters is often learned through a previously seen set of data (commonly referred to as a training set). What these decision functions look like depends on the type of machine that is used. In order to come to an actual decision on the label of a data point, a decision boundary or threshold has to be defined. These, again, depend on the type of machine that is used. Let us look at some different machine learning algorithms and their decision functions.

Support Vector Machines (SVM's) are a type of machine learning algorithm that use data points from the training set in order to define a boundary between classes. The decision function is effectively the distance a data point has to the boundary and the sign of this score depends on what side it lays: if a point lays on the side of class 0 (also called the negative class) the sign is -1, whereas if the point lays on the side of class 1 (also called the positive class) the sign is $+1$. This means that the decision boundary for this decision function for SVM's is 0.

Logistic regression is a natural probabilistic machine. This means that it does not decide on a label. Its decision function is a function that returns a probability that a data point is of class 1. In order to decide on a label, we can introduce a decision boundary. Usually the boundary is set at 0.5, which is similar to the decision boundary for SVM's in a way. However, class 0 might vary more in its features than class 1 does. This can cause the machine to be less sure of the class in which data points. This can cause the machine to be less sure of which class data points are. Picking a different boundary might be a good idea to solve this problem.

Let us define the decision boundary formally. Suppose we have a set of data X and a data point in this set $x \in X$. Suppose we have a trained machine learning algorithm M with decision function f. Let τ be the decision boundary. Let $M(x)$ be the label predicted by the machine and is defined as

$$M(x) = \begin{cases} +, & \text{if } f(x) > \tau \\ -, & \text{if } f(x) < \tau \end{cases}. \tag{1}$$

How to choose τ depends on the problem at hand. It might be more important to correctly label positive data points than it is to correctly label negative data points. We might choose a lower τ in order to reduce the misclassification of positives. This will cause more negative data points to be misclassified. We look at the distribution of the decision boundary function values of the different classes in order to find a decent τ. Where these two distributions intersect, will often give a decent τ.

Bayesian Calibration

Bayesian Calibration will play an important role in solving the platform economy classification problem. Here we will briefly go into this method. Bayesian Calibration is a method developed by M.J.H. Puts and P.J.H. Daas [5]. They state that when one wants to determine the proportion of positives in an unknown population using an estimator, one cannot guarantee that this estimator is unbiased. Bayesian Calibration is a bias correction method. This means that it aims to equalize the False Negatives and False Positives. In [6], it is mentioned that bias can be introduced when a model is trained on a set with a certain prevalence that differs from real-world data. Bayesian Calibration estimates this real-world prevalence and uses it to correct for this prevalence difference. This is explained in more detail in [7]. Here, we will briefly summarize the Bayesian Calibration method.

Let $M : \mathcal{X} \to [0, 1]$ be a binary probabilistic classifier (such as the Logistic Regression) trained on a training set $\mathbf{T} \subseteq \mathcal{X}$, where $\mathcal{X}$ is the feature space. Let $c_N \subseteq \mathcal{X}$ denote the set of data points that are of the negative class and let $c_P \subseteq \mathcal{X}$ denote the set of data points that are of the positive

class. If the model is ideal and has the probability that a data point $x \in \mathcal{X}$ is of the positive class as an output: $\mathbb{P}(c_P|x, \mathbf{T})$, we can then rewrite this probability using Bayes' Theorem:

$$M(x) := \mathbb{P}(c_P|x, \mathbf{T}) = \frac{\mathbb{P}(x|c_P, \mathbf{T})\mathbb{P}(c_P|\mathbf{T})}{\mathbb{P}(x|\mathbf{T})} \tag{2}$$

$$= \frac{\mathbb{P}(x|c_P, \mathbf{T})\mathbb{P}(c_P|\mathbf{T})}{\mathbb{P}(c_P)\mathbb{P}(x|c_P, \mathbf{T}) + \mathbb{P}(c_N)\mathbb{P}(x|c_N, \mathbf{T})} \tag{3}$$

Since we use the training set $\mathbf{T}$ all of these probabilities will be dependent on it. When $\mathbf{T}$ is representative of the total population (i.e. a small sampling error), $\mathbb{P}(x|c_P, \mathbf{T})$ (the probability that x shows features of the positive class) should be representative as well. We do not know whether or not the proportion $\mathbb{P}(c_P|\mathbf{T})$ (the sample prevalence) is also representative. In order to estimate this proportion, we denote the unknown proportion of positives in the data by π. The goal is to find a maximum likelihood estimate for this π. This can be done by looking at the entire feature space (using $x \in \mathcal{X}$). However, the dimensionality of this space can be arbitrarily large, thus increasing difficulty of the problem. If we instead look at the output of the model $M(x)$, we have an easier problem, since the dimensionality is a lot smaller. Note that this also works in case M is *not* an ideal classifier and does not have $\mathbb{P}(c_P|x, \mathbf{T})$ as an output, but merely some sort of pseudo-probability / probability score. These are not the real probability that a data point is of the positive class, but something that is (hopefully) related to it. Let us denote this non-ideal classifier with M'. We now use $\mathbb{P}(M'(x)|\pi, \mathbf{T})$ instead of $\mathbb{P}(x|c_P, \mathbf{T})$. The probability $\mathbb{P}(M'(x)|\pi, \mathbf{T})$ can be rewritten as:

$$\mathbb{P}(M'(x)|\pi, \mathbf{T}) = \pi\mathbb{P}(M'(x)|c_P, \mathbf{T}) + (1 - \pi)\mathbb{P}(M'(x)|c_N, \mathbf{T}) \tag{4}$$

If we use Eq. 4, $M'(x)$ and π in Eq. 2, we obtain the following:

$$\mathbb{P}(\pi|M'(x), \mathbf{T}) = \frac{\mathbb{P}(M'(x)|\pi, \mathbf{T})\mathbb{P}(\pi)}{\mathbb{P}(M'(x)|\mathbf{T})} = \frac{\mathbb{P}(M'(x)|\pi, \mathbf{T})\mathbb{P}(\pi)}{\mathbb{P}(c_P)\mathbb{P}(M'(x)|c_P, \mathbf{T}) + \mathbb{P}(c_N)\mathbb{P}(M'(x)|c_N, \mathbf{T})} \tag{5}$$

If we assume $\mathbb{P}(\pi)$ to be uniform and $\mathbb{P}(M'(x)|\mathbf{T})$ to be a normalization constant, we can formulate a likelihood function over the whole data set $\mathbf{B} := \{M'(x)|x \in \mathcal{X}\}$:

$$\mathcal{L}(\pi|\mathbf{B}, \mathbf{T}) = \prod_{M'(x)\in\mathbf{B}} \mathbb{P}(M'(x)|\pi, \mathbf{T}). \tag{6}$$

We find the maximum likelihood estimate for π by solving

$$\hat{\pi} = \text{argmax}_\pi \mathcal{L}(\pi|\mathbf{B}, \mathbf{T}). \tag{7}$$

Using this estimate, we can properly estimate the probability that a data point x is positive, by using $\hat{\pi}$ instead of $\mathbb{P}(c_P|\mathbf{T})$ in Eq. 2.

Note that we applied Bayes' Calibration to $M'(x)$ instead of x. This means that the bias correction provided by this method is only as good as the model M. If there is already a bias present in M that has not been caused by the difference in prevalence, an error will be present in $\mathbf{B}$ and so there will also be an error in $\hat{\pi}$.

0.3 Problem Description

The web-scraped platform economy dataset from CBS is a set that consists of the text that has been scraped from the websites of platform economy and non-platform economy companies from the Netherlands. The text is pre-processed:

1. words with less than 3 characters are removed
2. websites that have less than 10 words are removed
3. a language feature (dutch/english) is added
4. a TF-IDF matrix is created

This TF-IDF matrix is the data that will be used in order to train machines and predict labels/probabilities. Our goal is to develop a method that can classify the samples in the data and estimate the number of platform economy companies in the data. Various other people have worked on this problem [2][3] or on related problems [1], which we will build upon.

This dataset has 1396 positively labeled samples (platform economy companies) and 593961 negatively labeled samples (non-platform economy companies). This means that only approximately 0.23% of the data is positively labeled. As such, we are dealing with extremely imbalanced data. Extreme rare event detection problems usually come in the forms of time series [8] or images [9] and are tackled using auto-encoders or PU learning methods.

Unlike other classification problems, we are less interested in predicting the label of every data point and more in estimating the total number of platform economy companies. To this end, we will use the definition of the estimated value, which is the sum of probabilities of a company being a platform economy company. Such (estimations of) probabilities can be found using machine learning algorithms. Labels can also still be predicted and used to estimate the total number of platform economy companies, akin to a counting process.

The combination of the use of text as data and extremely imbalanced data causes a unique problem. Because of this the previously mentioned methods (PU learning / auto-encoders) will not work. These methods will, for example, learn features of a language and not of the non-platform economy companies. Another problem is estimating the total number of platform economy companies. Calculating this estimate using probabilities causes various commonly used metrics (such as the accuracy or balanced accuracy) to not work. These only accept labels, not probabilities, and would require us to set a decision boundary. Transforming these probabilities to labels through the use of a decision boundary still would not capture the full performance of the model, especially when trying to estimate the total number of positively labeled samples. As such, an approach has to be found in order to properly capture the performance of these models.

To this end, we split this thesis into two parts: in Chapter 1 we will focus on finding measures that properly measure the performance of models in this context and in Chapter 2 we will focus on finding models that properly estimate the total number of positively labeled samples.

We will be using the terms class 0 and the negatively labeled data interchangeably for the non-platform economy class and class 1 and the positively labeled data interchangeably for the platform economy class.

0.4 A word of warning

Before we start, we must proceed with some level of caution. In this thesis we will make use of the scikit-learn module for Python. The machines in this module use a class method called `predict` in order to predict labels. In order to predict "probabilities", a class method called `predict_proba` is used. Naturally probabilistic machines, such as Logistic Regression (under the right conditions), produce probabilities naturally (as the name suggests). Decision machines, such as a Support Vector Machine, naturally produce labels and not probabilities. In order to generate probability scores from these labels, a method called Platt Scaling [10] is used (developed by J.C. Platt). Platt tries to **estimate** the probability of a sample being positively labeled:

$$\mathbb{P}(y = 1|x) = \frac{1}{1 + \exp(\alpha f(x) + \beta)},$$

where y is the true label, x is a sample, $f(\cdot)$ is some real-valued function and α, β are scalar parameters in order to properly estimate the probability. For $f(\cdot)$, we assume that it is some real-valued function. Such a function is referred to as a decision function. Essentially a logistic regression model is applied. However, by applying such a logistic regression model, we assume that the set obtained through the decision function f is normally distributed. In general, we cannot guarantee that the scores obtained through f are normally distributed. As such we cannot guarantee that the estimated probabilities are actual probabilities. Additionally, Platt scaling can turn perfectly calibrated machines into uncalibrated machines [11].

Another problem arises when applying Platt Scaling: the parameters α, β have to be estimated using stratified cross-validation. There are other possibilities of estimating these parameters, but Platt suggests using cross-validation. For extremely imbalanced data, cross-validation is by definition not ideal: there is already a small number of positively labeled samples available and a fraction of these (Platt suggests 1/3) will be used in order to find the parameters α, β. Platt claims that training a Support Vector Machine in such a way takes approximately 2.2 times more times than a single Support Vector Machine [10].

We must note that the Bayesian Calibration method does produce actual probabilities, regardless of the distribution of the data / classes.

From here on out, we will use Platt Scaling for decision machines, even though it has significant problems, as it is the standard probabilistic calibration method implemented in scikit-learn and it is out of the scope of this thesis to form another calibration method or fix/correct the Platt Scaling method. This means that one must be wary whenever probabilities are presented in this thesis.

Chapter 1

Pearson's Correlation Coefficient

In this chapter, we will try to find measures that properly measure the performance of classifiers in the context of the platform economy dataset. Most of the focus of this chapter will lay on the use of the sPCC as such a metric. In Section 1.1 we will discuss why other metrics will not suffice for the problems stated in Section 0.3. In Section 1.2 we will derive an expression for the sPCC to be used in a binary classification context. Additionally, a few small case studies will be held in order to show its effectiveness where other metrics might run into problems. Section 1.3 is almost analogue to Section 1.2, but goes into multiclass classification problems and explains choices that have to be made for the (non-trivial) extension. The chapter is wrapped up with a conclusion / summary in Section 1.4.

1.1 Why is a new metric needed?

As mentioned in Section 0.3, this problem is unique through its combination of the use of text and extreme rare event detection. We want to use probabilities in order to properly estimate the total number of positives. In Section 0.4 we saw that the current default calibration method (Platt scaling) is a poor method when applied to generally distributed data. To improve this, we would like to use a method developed by M.J.H. Puts and P.J.H. Daas [5], called Bayesian Calibration (see Section 0.2). This method applies a transformation on the predicted probabilities returned through Platt scaling. Comparing Platt scaling to Bayesian Calibration in extreme rare event detection requires measures that are able to not only properly capture the difference between the methods, but are also able to handle the extreme imbalance. Additionally, we are also interested in measuring the performance of the methods on individual sample level: how well are the models able to predict individual labels/probabilities. Note that being able to estimate the total number of positives well does not mean the machine is able to predict labels/probabilities well.

For estimating the total number of positives, we use a metric called bias (this metric is introduced in [7]). It measures the size of the error a machine makes in estimating the total number of positives (or negatives) in a certain set relative to the size of the set. It is defined as

$$bias := \frac{FP - FN}{n}$$

where FP are the estimated number of False Positives, FN the estimated number of False Negatives and n the number of points in the data set. When predicting labels, the false positives and negatives are easily defined: the number of negatively labeled samples that are predicted positive and vice versa for the false negatives. These can be estimated when using probabilities. For the false positives, we sum over the probabilities that non-platform companies are labeled as platform companies and vice versa for the false negatives. If the bias is positive, the machine will overestimate the number of positives, while a negative number will underestimate the number of positives.

Let us now focus on measures for classifying data. Metrics for imbalanced data that are usually considered good are: the balanced accuracy (BA), as it tries (implicitly) to compensate for the imbalance and calculate the appropriate accuracy afterwards; the Matthew's Correlation Coefficient (MCC), as it tries to find a correlation between the true labels and the predicted labels; the AUROC, which looks at the predicted probabilities and determines how well a classifier performs when shifting the decision boundary τ (see Section 0.2); and the recall, precision and F1-score, which look at how well the model classifies the positive class, the negative class and the F1-score is the harmonic mean of those two.

The BA is defined as

$$BA = \frac{TPR + TNR}{2} = \frac{1}{2}\left(\frac{TP}{P} + \frac{TN}{N}\right)$$

where TPR is the True Positive Rate, TNR the True Negative Rate, TP the number of True Positives (positively labeled points that have been predicted to be positively labeled), TN the number of True Negatives, P the number of total Positives and N the number of total Negatives.

In essence, when calculating the BA, we calculate at which rate the machine predicts positives correctly and negatives correctly. We then take the arithmetic mean of these two rates in order to obtain the BA. As seen in Section 0.2, this predicting of positives and negatives is heavily influenced by the decision boundary τ. As a reminder, τ is here chosen to be at the intersection of the probability distributions of the two classes. Note that these probability distributions do take prevalence (proportion of positives in a population) into account. The BA effectively sets this prevalence at 0.5, such that both classes weigh equally as heavy in order to "balance the accuracy". However, τ is not changed after changing this prevalence. This means there will be a slight error in the BA, as we will not actually measure what we want. This becomes very apparent in the case of extremely imbalanced data sets. The prevalence will have to change drastically and so might τ. When applying Bayesian Calibration, the probability distributions are changed and therefore, τ will also be moved. Calculating the BA now and comparing the results of before and after the calibration, might not make sense. We change τ and the probability distributions, meaning that we measure different things.

Another metric that is used for imbalanced data is the Matthew's Correlation Coefficient (MCC). It calculates the correlation between the true labels and the predicted labels and is defined as:

$$MCC = \frac{TP \cdot TN - FN \cdot FP}{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}.$$

Although the MCC is a better metric than the BA in regards to the extreme rare event detection problem, it still does rely on τ. As such, we face similar problems to the BA.

The AUROC is a metric that measures the Area Under the Receiver Operating Characteristic curve (ROC curve for short). This curve is calculated by gradually shifting τ from 0 to 1 and calculating the TPR and FPR. Since a finite set of data is used, these two rates form the coordinates associated with every τ. Connecting the points of subsequent values for τ forms the (discrete) ROC curve. Due to the combination of shifting τ and the use of the two rates, the AUROC performs really well on imbalanced data. A major problem for us is has to do with the AUROC's insensitivity to linear transformations (on the predicted probabilities) [12]. We would like to measure the performance of Bayesian Calibration. This is a transformation, but not necessarily linear. There are cases where it *is* a linear transformation: if the negative probability distribution is monotonic decreasing and the positive probability distribution monotonic increasing. As we will see in Section 1.2.3, the AUROC is also heavily influenced by sampling errors and is a poor metric to use when trying to say something about the external validity of the model: how well the model performs outside of your test and training set. This heavy influence is a problem when trying to measure the difference in performance, since we cannot know if the difference is caused by sampling errors or by a non-linear transformation.

Finally, the recall, precision and F1-score. The recall is equal to the True Positive Rate: the fraction of positives that is labeled correctly. The precision measures the number of samples that

are labeled as positive past the decision boundary (TP over TP+FP) and effectively measures how poorly the model labels negatives. The F1-score is the harmonic mean of these two metrics. In an extreme imbalanced case it is easy for the precision to become very poor. Let us have a dataset with 0.5% positives. If all positives are labeled correctly and 0.5% of negatives incorrectly, we would have a precision of approximately 0.5 and a recall of 1.0. This results in a F1-score of 0.667. This is a very pessimistic method of scoring the machine and would not accurately describe how well the machine is performing. Just using recall does not give any meaningful information about the entire model. Using either precision or F1-score in combination with the recall will impact the score in a large way. We can conclude that this set of metrics are less useful to use than other metrics and we will not use them in the proceedings of this thesis.

To summarize, we need a metric that

1. can handle extreme imbalance
2. can accept probabilities (or any real-value)
3. is not heavily influenced by sampling errors

The MCC is a great metric, except that it does not accept probabilities. We could estimate the measures in the definition of the MCC, similarly to what we did for the bias, but we choose not to take this approach. This does not mean that it is a poor metric. In Appendix A we can see that the MCC is identical to the sample Pearson Correlation Coefficient (sPCC) in a binary classification case. Let us look further into this sPCC. The Pearson Correlation Coefficient (often also referred to as the Pearson ρ) is a well-known statistic that measures how strongly two sets correlate. For two sets $X, Y \subset \mathbb{R}$, ρ is defined by:

$$\rho = \frac{Cov(X,Y)}{\sqrt{\mathbb{V}(X)\mathbb{V}(Y)}}$$

where $\mathbb{V}(X)$ is the variance of X. The sPCC will be used from here on out, since we do not know the distribution of the predicted probabilities. For two sets $x = \{x_1, ..., x_n\}, y = \{y_1, ..., y_n\}$, the sPCC is denoted with r_{xy} and is defined as the fraction of the estimator for the covariance of x and y over the square root of the product of their variances s_x^2 and s_y^2:

$$r_{xy} = \frac{q_{xy}}{\sqrt{s_x^2 s_y^2}} \tag{1.1}$$

$$\text{where} \qquad q_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \tag{1.2}$$

$$\text{and} \qquad s_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{1.3}$$

Here $\bar{x}$ is the mean of x and s_y^2 is defined analogue to s_x^2. Much like the MCC, the sPCC assumes values in the range $[-1, 1]$. Note that the sPCC is not unbiased [13], where smaller sample sizes lead to a larger bias.

1.2 The binary classification case

In a binary classification setting, two classes are present: a positive and a negative class. The true labels of a dataset of n samples is defined as $y = (y_1,...,y_n) \in \{0,1\}^n$. A classification model tries to predict these labels and these predictions are denoted by $\hat{y} = (\hat{y}_1,...,\hat{y}_n) \in \mathbb{R}^n$. A sample i is of the positive class if its true label is equal to one, $y_i = 1$, or of the negative class if its true label is equal to zero, $y_i = 0$. Putting these predicted labels as elements of a real-valued space might seem odd as most machine learning algorithms for classification problems are decision machines (with some exceptions [14]) and produce probabilities [1] by scaling the predicted labels into a probability measure [10][15]. For more information regarding this, see Section 0.4. This would mean that we would only have to look at values in the range of $[0,1]$. However, most machines use a decision function to come to a prediction. These functions are real-valued and we will show that the sPCC can also be applied to these "decision scores".

1.2.1 Derivation of the sPCC for binary classification

Define the subset of predicted labels which all have a positive true label as $\hat{y}^P = \{\hat{y}_i \in \hat{y} | y_i = 1\}$ and the subset of predicted labels which all have a negative true label as $\hat{y}^N = \{\hat{y}_i \in \hat{y} | y_i = 0\}$, such that $P = |\hat{y}^P|$ and $N = |\hat{y}^N|$. Our goal is now to derive the sPCC to a form which is more intuitive than its formal definition. We start by simplifying the sample covariance q between y and $\hat{y}$ (denoted by $q_{y\hat{y}}$). First we multiplying both sides of the equation (Eq. 1.4) by $(n-1)$ to make the derivation clearer. Let $\bar{\hat{y}}$ denote the average predicted label.

$$q_{y\hat{y}} = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}}) \tag{1.4}$$

$$(n-1)q_{y\hat{y}} = \sum_{i=1}^{n}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}}) \tag{1.5}$$

$$= \frac{N}{N+P}\left[\sum_{i=1}^{P}\hat{y}_i^P - \bar{\hat{y}}\right] + \frac{P}{N+P}\left[\sum_{i=1}^{N}\hat{y}_i^N - \bar{\hat{y}}\right] \tag{1.6}$$

$$= \frac{N}{N+P}\sum_{i=1}^{P}\hat{y}_i^P - \frac{N}{N+P}P\bar{\hat{y}} - \frac{P}{N+P}\sum_{i=1}^{N}\hat{y}_i^N + \frac{P}{N+P}N\bar{\hat{y}} \tag{1.7}$$

$$= \frac{NP}{N+P}\left[\frac{1}{P}\sum_{i=1}^{P}\hat{y}_i^P\right] - \frac{NP}{N+P}\left[\frac{1}{N}\sum_{i=1}^{N}\hat{y}_i^N\right] \tag{1.8}$$

$$= \frac{NP}{N+P}(\bar{\hat{y}}^P - \bar{\hat{y}}^N) \tag{1.9}$$

Let us now look at the sample variance of both sets, $s_y^2, s_{\hat{y}}^2$. Since nothing is known about the distribution of the predicted labels without making assumptions about the distribution of the original dataset and/or the used model, it cannot be evaluated here and instead will be denoted by $s_{\hat{y}}^2$. Contrary to the predicted labels, the true labels have a distribution that can be expressed in terms of N and P. A quick way to simplify the sample variance of the true labels is to note

[1] Note that, while these are actual probabilities, we cannot guarantee that these are the probabilities that data points are of the positive class.

that it is equal to q_{yy} and so we can use the derivation that we made earlier (Eq. 1.4-1.9):

$$\begin{aligned}(n-1)s_y^2 &= (n-1)q_{yy} \\ &= \frac{NP}{N+P}(y^P - y^N) \\ &= \frac{NP}{N+P}(1-0) \\ &= \frac{NP}{N+P}\end{aligned}$$

Putting these expression together we get the simplified expression for the sPCC:

$$\begin{aligned}r_{y\hat{y}} &= \frac{q_{y\hat{y}}}{\sqrt{s_y^2 s_{\hat{y}}^2}} \\ &= \left(\frac{NP}{(N+P)(N+P-1)}\right)^{1/2} \frac{\bar{\hat{y}}^P - \bar{\hat{y}}^N}{s_{\hat{y}}}\end{aligned} \tag{1.10}$$

And so the sPCC tells us something about how much distance there is between the average predicted probability of the positive samples and the average predicted probability of the negative samples compared to the total variance in the data times some constant depending on the class distribution. In other words: how well separable are the predictions between the positive and negative data relative to the total variance of the predictions times some constant?

1.2.2 The sPCC and its relation to other metrics

One of the motivations for using the sPCC is because of its relation to the MCC (as seen in Appendix A). It is important to note that the MCC is related to other metrics: accuracy [16], F1 score [16], balanced accuracy (BA)[17], bookmaker informedness (BM)[17] and markedness (MK) [17]. There are other metrics that the sPCC is related to in a binary case however, as discussed below.

Discriminality index d'

The discriminality index (also known as the sensitivity index or the detectability index), is a dimensionless statistic from the field of signal detection theory. This index is first mentioned by Peterson, Birdsall and Fox in [18]. Later Swets, Tanner and Birdsall expanded on this in [19]. Typically it is used to measure how well noise can be separated from an actual signal plus noise (as seen in its definition below). For two univariate distributions, X, Y (need not be normal), with means μ_X, μ_Y respectively and the same standard deviation σ, the index is denoted by d' and is defined as

$$d' = \frac{|\mu_X - \mu_Y|}{\sigma}$$

which measures the distance between the mean of the two distributions relative to their variance. At first glance, this index seems to want to measure the same thing as the sPCC. This is per definition not the case, as the variance in the denominator in the sPCC is the *total* variance, whereas σ is the within group variance which is the same for X and Y. In the case of binary classification, we cannot guarantee that the within group variance is the same for both the positive and the negative labelled data. Several different indices exist for unequal variances. We will cover three indices here: the Bayes discriminality index d'_b, the Root Mean Square standard deviation discriminality index d'_a (or RMS sd discriminality index for short) and the average standard deviation discriminality index d'_e (or average sd discriminality index for short). Denote the within group standard deviation for X and Y with σ_X, σ_Y respectively.

Bayes discriminality index The Bayes discriminality index, introduced by Das and Geisler [20] is based on the amount of overlap between two normal distributions with equal variances. Even though Das and Geisler go into more detail for normally distributed random variables, an approach for generally distributed random variables is still provided. Define two conditional probabilities:

$$\mathbb{P}(A|a) = \int_{-\infty}^{0} f_a(l)dl = F_a(0)$$
$$\mathbb{P}(B|b) = \int_{0}^{\infty} f_b(l)dl = 1 - F_b(0)$$

where F_a and f_a denote the cumulative and probability density function of the first random variable X respectively. Analogue for F_b and f_b for the second random variable Y. The Bayes discriminality index is then given by

$$d'_b = 2\,Z\left(\frac{\mathbb{P}(A|a) + \mathbb{P}(B|b)}{2}\right)$$

where $Z(\cdot)$ is the z-score. For any two distributions, the Bayes discriminality index corresponds to the accuracy of the point on the ROC curve that is farthest away from the diagonal.

RMS sd discriminality index The RMS sd discriminality index d'_a is a (closed-form) index that (as the name suggests) takes the root mean square of the standard deviations: $\sigma_{RMS} = \sqrt{1/2(\sigma_X^2 + \sigma_Y^2)}$. This leads to the following definition:

$$d'_a = \frac{|\mu_X - \mu_Y|}{\sqrt{1/2(\sigma_X^2 + \sigma_Y^2)}}$$

This index is related to the ROC curve as it is $\sqrt{2}$ times the z score of the Area Under ROC Curve (AUROC) (for a single criterion) [20]. Let $X = \hat{y}^P$ and $Y = \hat{y}^N$, we then can rewrite the index in terms of our original problem:

$$\begin{aligned} {d'}_a^2 &= \frac{2(\bar{\hat{y}}^P - \bar{\hat{y}}^N)^2}{(s^2_{\hat{y}^P} + s^2_{\hat{y}^N})} \\ &= \frac{2s^2_{\hat{y}}}{A(s^2_{\hat{y}^P} + s^2_{\hat{y}^N})} r^2_{y\hat{y}} \end{aligned}$$

where $A = NP/((N+P)(N+P-1))$. This means that d'_a is equal to the sPCC times some factor (dependent on the class distribution) times the square root of the total variance over the sum of the within group variances. We can write how the sPCC is related to the AUROC score:

$$r_{y\hat{y}} = \frac{\sqrt{A(s^2_{\hat{y}^P} + s^2_{\hat{y}^N})}}{s_{\hat{y}}} \cdot Z(AUROC)$$

Since a relation exists between the sPCC and the AUROC, we might wonder why we cannot just use the AUROC instead of the sPCC. As seen before (in Section 1.1), the AUROC has several problems, but the most important problem in this case is its insensitivity to transformations on the probabilities that preserve their ranks [12]. Since one of the end goals is to measure the effect of the Bayes Calibration, the AUROC will be a bad measure to use in this case.
As shown by Hanley and McNeil, the AUROC is related to the Mann-Whitney U statistic in the following way [21]:

$$AUROC = \frac{W_{XY}}{N_X \cdot N_Y}$$

where W_{XY} is the Mann-Whitney U statistic for X, Y and N_X, N_Y are the number of samples in group X and Y respectively.

Average sd discriminality index The average sd discriminality index d'_e is similar to the RMS sd discriminality index as it also takes some average over the standard deviations. It is defined as:

$$d'_e = \frac{2|\mu_X - \mu_Y|}{\sigma_X + \sigma_Y}$$

where the arithmetic average of the standard deviations is used in order to come to a discriminality index. Similarly to the RMS sd discriminality index, we can relate the average sd discriminality index to the sPCC. Let $X = \hat{y}^P$ and $Y = \hat{y}^N$.

$$d'_e = \frac{4(\bar{\hat{y}}^P - \bar{\hat{y}}^N)^2}{(s_{\hat{y}^P} + s_{\hat{y}^N})^2} = \frac{4s^2_{\hat{y}^P}}{A(s_{\hat{y}^P} + s_{\hat{y}^N})^2} r^2_{y\hat{y}}$$

where $A = NP/((N+P)(N+P-1)$. This closely relates to what we have seen before with the RMS sd discriminality index. The only difference is the denominator and the constant in the numerator.

1.2.3 Test cases

We would like to know how the sPCC behaves in comparison to other metrics in a binary classification case. To this end, we present several binary classification problems:

- normally distributed data (synthetic)
- non-normally distributed data (synthetic)
- two class subset (8,9) of the Fashion MNIST dataset (real world)
- two class subset (0,6) of the Fashion MNIST dataset (real world)

We will provide a histogram of the distribution of the data for the synthetic case. In addition to different problems, we also use different machine learning algorithms and one neural network. The used models are:

- Support Vector Machine (linear kernel) (SVM)
- Logistic Regression (LogReg)
- Random Forest Classifier (RFC)
- Neural Network (NN)

The neural network will be a very simple neural network consisting of 5 layers: an appropriately sized input layer, three dense layers with five neurons each and an output layer with one neuron. Every layer uses a ReLU function as activation function, except for the output layer, which uses a sigmoid as activation function. This might not be a great network for some of the provided problems, but the goal in this section is not to find the best model. Note that the neural network does not use the Platt scaling discussed in Section 0.4. For each problem, we will show histograms of the distribution of the predicted probabilities and a table with the following metrics:

- sPCC
- accuracy
- balanced accuracy (BA)
- MCC
- AUROC
- bias

These metrics have been chosen since the accuracy and AUROC are often the default and BA and MCC are often used when dealing with imbalanced data. Note that the accuracy, BA and MCC only take binary inputs. To this end, we choose to use the default decision boundary τ of 0.5 for the non-calibrated machines. We would also like to look at the effects of applying Bayesian Calibration on the various models and problems. Note that Neural Networks are not deterministic [2] and therefore we will not use the results of Neural Networks to look at the effect of Bayesian Calibration. The calibrated models innately define a more optimal τ than the previously chosen 0.5, so we will use this new τ for the accuracy, BA and MCC. Let us redefine the bias we have seen in Section 1.1 in a more formerly manner, appropriate for predicting probabilities.

$$bias = \frac{\sum_{i=1}^{n} \hat{y}_i \mathbb{I}\{y_i = 0\} - (1 - \hat{y}_i)\mathbb{I}\{y_i = 1\}}{n},$$

which can be interpreted as the estimated number of False Positives minus the estimated number of False Negatives based off the predicted probabilities over the number of samples present.

Normally distributed data

For this problem we generate two normally distributed sets in 1 dimension. Let class 1 be normally distributed with mean $\mu_1 = 0$ and standard deviation $\sigma_1 = 0.5$ and let class 2 be normally distributed with mean $\mu_2 = 4$ and standard deviation $\sigma_2 = 0.5$. Let class 1 have 4500 samples and class 2 500 samples. In Figure 1.1 the class distributions are shown through sampling. Note that the train and test sets will be sampled from the same distributions with the same number of samples per class. From this figure, one can confidently guess that the various algorithms will perform well on this dataset, as the classes are very well separated.

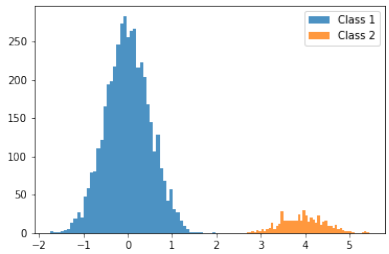
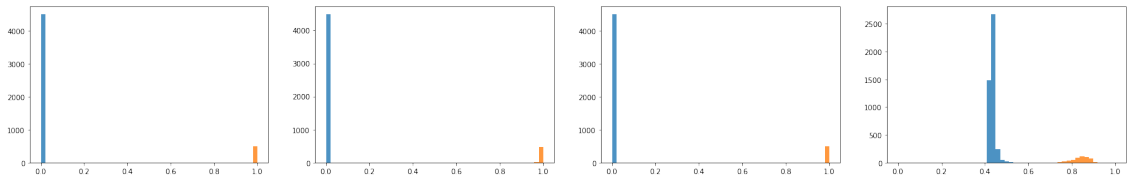
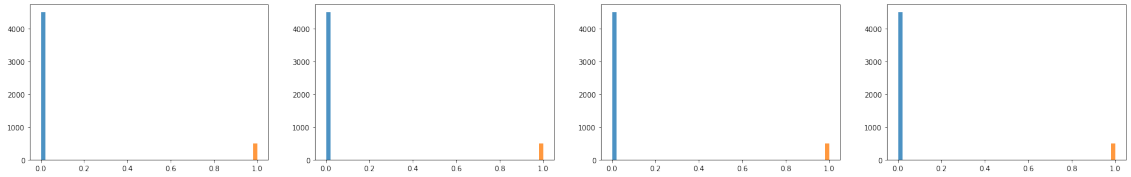


Figure 1.1: Histogram showing the distributions of two classes. Class 1 (in blue) is distributed as a normal distribution with mean $\mu_1 = 0$ and standard deviation $\sigma_1 = 0.5$, while class 2 (in orange) is distributed as a normal distribution with mean $\mu_2 = 4$ and standard deviation $\sigma_2 = 0.5$.

[2] I tried fixing the random seed for these machines, however, this seemed to not affect these machines



(a) Support Vector Machine (b) Logistic Regression (c) Random Forest Classifier (d) Neural Network



(e) Support Vector Machine (cal.) (f) Logistic Regression (cal.) (g) Random Forest Classifier (cal.) (h) Neural Network (cal.)

Figure 1.2: Histograms showing the distribution of the predicted probabilities per machine for normally distributed synthetic data. The predicted probabilities of class 1 are shown in blue, while the predicted probabilities of class 2 are shown in orange. The (cal.) postfix denotes the calibrated version of the machine.

And from Figure 1.2 we can conclude that our guess is indeed correct, as there are clearly two peaks for every machine. However, for the case of the (non-calibrated) neural network, the two peaks are not as distanced as for the other machines. As such, it will have a slightly harder time differentiating between classes.

Table 1.1: Performance of various machines measured with various metrics on normally distributed data. The (cal.) postfix denotes the calibrated version of the machine.

	SVM	SVM (cal.)	LogReg	LogReg (cal.)	RFC	RFC (cal.)	NN	NN (cal.)
sPCC	1.0	1.0	0.9999	1.0	1.0	1.0	0.9909	1.0
Accuracy	1.0	1.0	1.0	1.0	1.0	1.0	0.9946	1.0
BA	1.0	1.0	1.0	1.0	1.0	1.0	0.997	1.0
MCC	1.0	1.0	1.0	1.0	1.0	1.0	0.9711	1.0
AUROC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Bias	0.0001	0.0	-0.0	0.0	0.0	0.0	0.3755	0.0

Table 1.1 confirms our observations, as all the metrics are (nearly) one. The Neural Network shows here that, even though it scores fairly well on predicted the labels, it scores poorly on estimating the total number of positives. Applying Bayesian Calibration resolves this problem.

Non-normally distributed data

For this problem we generate data with classes distributed as weighted sums of normal distributions. Let class 1 have the distribution of $X_{11}\mathbb{I}\{U > 0.05\} + X_{12}\mathbb{I}\{U < 0.05\}$, where $U \sim Unif([0,1])$, $X_{11} \sim \mathcal{N}(0, 0.5)$ and $X_{12} \sim \mathcal{N}(3, 0.5)$. Let class 2 have the distribution of $X_{21}\mathbb{I}\{U < 0.1\} + X_{22}\mathbb{I}\{U > 0.1\}$, where $X_{21} \sim \mathcal{N}(1, 0.5)$ and $X_{22} \sim \mathcal{N}(4, 0.5)$. In Figure 1.3 the class distributions are shown through sampling.

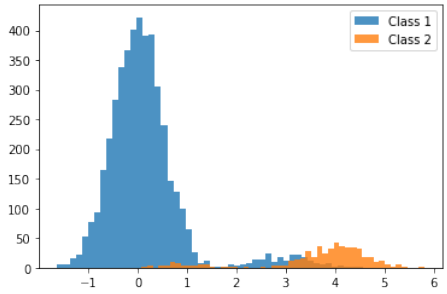
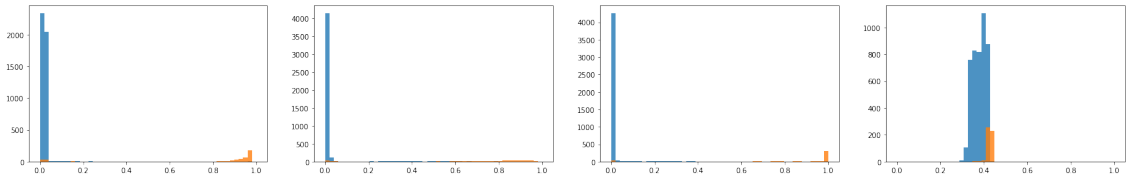
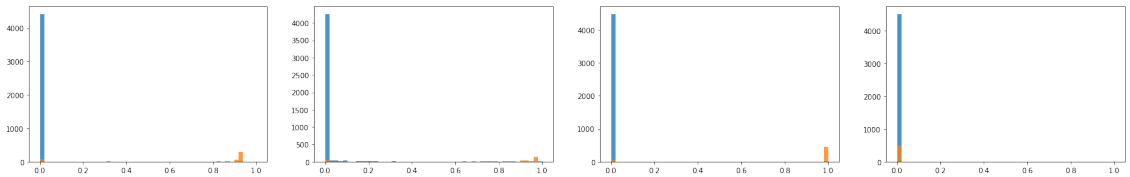


Figure 1.3: Histogram showing the distributions of two class. Class 1 (in blue) is distributed as the weighted sum of two normally distributed random variables with means $\mu_{11} = 0, \mu_{12} = 3$, standard deviations $\sigma_{11} = 0.5, \sigma_{12} = 0.5$ and weights $w_{11} = 0.95, w_{12} = 0.05$ respectively. Class 2 (in orange) is distributed as the weighted sum of two normally distributed random variables with means $\mu_{21} = 1, \mu_{22} = 4$, standard deviations $\sigma_{21} = 0.5, \sigma_{22} = 0.5$ and weights $w_{21} = 0.1, w_{22} = 0.9$ respectively.



(a) Support Vector Machine (b) Logistic Regression (c) Random Forest Classifier (d) Neural Network



(e) Support Vector Machine (cal.) (f) Logistic Regression (cal.) (g) Random Forest Classifier (cal.) (h) Neural Network (cal.)

Figure 1.4: Histograms showing the distribution of the predicted probabilities per machine for non-normally distributed synthetic data. The predicted probabilities of class 1 are shown in blue, while the predicted probabilities of class 2 are shown in orange. The (cal.) postfix denotes the calibrated version of the machine.

From Figure 1.4 we can see that the machines have some difficulties differentiating the two classes: class 1 samples are often classified as a class 0 samples. We can see this clearly in the SVM case (both non-calibrated and calibrated), where some class 2 samples are binned in the same bin as most of the class 1 samples (see Figure 1.4a and 1.4e). Though the reverse is not true: class 0 samples are rarely classified as class 1 samples (visually). The best performing machine is difficult to determine, though it looks like that the SVM and RFC perform better than the rest. Even though there is barely any difference, Bayesian Calibration does seem to improve to performance of some of the machines. The non-calibrated neural network predicts the two classes two be very close to each other. Hence why the calibrated version piles all probabilities together and is unable to distinguish the classes.

Table 1.2: Performance of various machines measured with various metrics on non-normally distributed data. The (cal.) postfix denotes the calibrated version of the machine.

	SVM	SVM (Cal.)	LogReg	LogReg (Cal.)	RFC	RFC (Cal.)	NN	NN (Cal.)
sPCC	0.856	0.8559	0.8343	0.8498	0.917	0.9267	0.4529	0.0000
Accuracy	0.9712	0.97	0.9662	0.9702	0.987	0.987	0.9000	0.9068
BA	0.8924	0.9122	0.9208	0.8919	0.9457	0.9466	0.5000	0.5340
MCC	0.8329	0.8319	0.8181	0.8275	0.9261	0.9261	0.0000	0.2482
AUROC	0.9368	0.9545	0.9792	0.9662	0.9652	0.965	0.9792	0.5000
Bias	0.0185	-0.0021	-0.0141	-0.0116	-0.0516	-0.0081	2.8372	-0.1000

Table 1.2 somewhat confirms what can be seen in Figure 1.2: the RFC performs best, though the SVM actually scores similarly to the LogReg. When looking at the effects of Bayesian Calibration, we can see that for the LogReg the bias barely changes (from -0.0141 to -0.0116), but for the other machines there is a significant change. The bias for the SVM changes from 0.0185 to -0.0021, for RFC from -0.0516 to -0.0081 and for NN from 2.8372 to -0.1000. This supports the claim that the Bayesian Calibration improves a machine's ability to estimate the total number of positives. The calibrated Neural Network scores poorly in both the calibrated case and the non-calibrated case on an individual level. On a population level (estimating the total number of positives), we see that Bayesian Calibration actually improved its performance a lot. Even though the BA and MCC are poor measures when setting τ at 0.5, in this case the metrics hold up quite well (except for the Neural Network). The more important metrics are the AUROC and the sPCC. Especially in the case of the non-calibrated Neural Network, which has a harder time separating the two classes. The AUROC in this case (0.9407) is significantly higher relative to the sPCC (0.4529). While it makes sense that the AUROC is high when looking at Figure 1.4d, it is important to note that the AUROC is heavily influenced by sampling errors. If we have two class distributions that do overlap, poor sampling might give us two class sampling distributions that overlap significantly more or significantly less. It is possible to correct for this sampling error by producing multiple test sets and taking the (arithmetic) mean AUROC score. Preferably these test sets would be independent and identically distributed. If these sets are not independent, the influence of some samples would be used multiple times and would skew the actual AUROC score. Note that this is not always possible, as sometimes the amount of data present is limited (finite population). The sPCC compensates for this sampling error by estimating the basic shape of the class distributions (mean and variance) and using these instead of the actual data. Note that the sPCC is still affected by sampling errors. Here, the sPCC estimates that the overlap is worse than the AUROC thinks it is and, in turn, gives us a better measure of performance in terms of external validity. This, however, does not come without problems. The sPCC only takes the mean and variance into account, but not any of the higher moments. Most notably, the skewness. For example, let $X \sim LN(0, 1)$ be log-normally distributed with mean 0 and standard deviation 1. The probability that $X = 0$ is zero. We take X as the distribution for class 1 and $-X$ as the distribution for class 0 (see Figure 1.5), both consisting of 10000 samples.

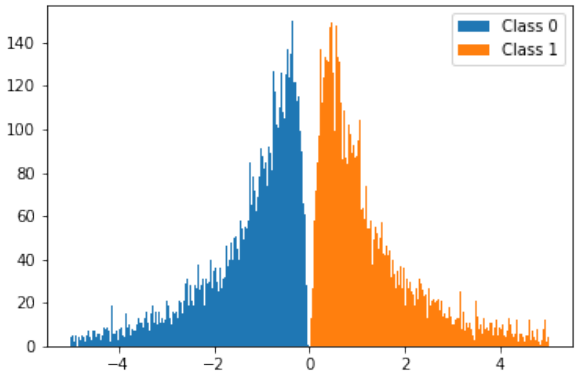


Figure 1.5: The sampled distributions of two log-normally distributed random variables with mean 0 and standard deviation 1, X (class 1) and $-X$ (class 0), where $-X = -1 \cdot X$.

For the distributions sampled in Figure 1.5, the sPCC is calculated at 0.6111, while the AUROC is 1.0. In actuality, the probability that these two distributions overlap is zero. The AUROC is correct here: a machine that would produce these distributions would be able to perfectly separate the two classes. If one were to use the sPCC, one must look at the distributions of the probabilities before a conclusion can be made about the validity of the use of the sPCC.

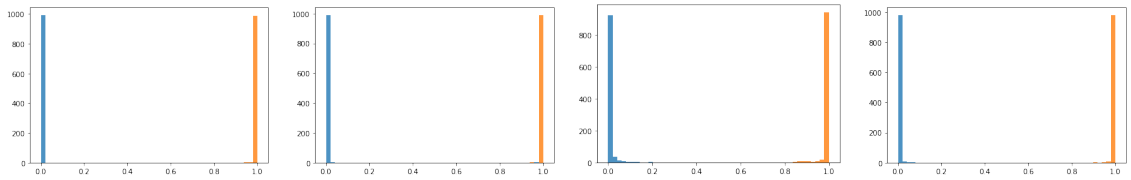
Fashion MNIST dataset

The Fashion MNIST dataset [22] is a set of pictures of clothing articles, such as bags, shirts and dresses, and is an alternative to the popular MNIST dataset (a dataset consisting of handwritten digits) [23]. The set is perfectly balanced and consists of a training and a test set. The training set contains 60000 pictures, with 6000 pictures per class, while the test set contains 10000 pictures, with 1000 pictures per class. Since the set is balanced, the accuracy is a good measure of performance, contrary to the previous (synthetic) datasets. The pictures have a resolution of 28 by 28 pixels and in Figure 1.6 one can see a sample of every class.

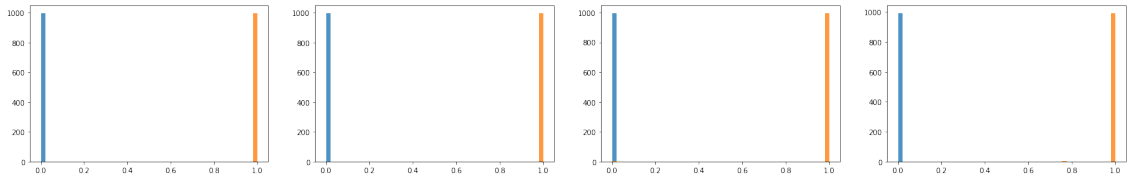
Figure 1.6: A sample per class of the Fashion MNIST dataset.

We make two separate problems: bags versus ankle boots and T-shirts/tops versus shirts. The

first problem takes bags and ankle boots. These are vastly different (see Figure 1.6) and will therefore be easily recognized as different articles by most classifiers. The second problem takes T-shirts/tops and shirts. These are quite similar (see Figure 1.6) and will therefore cause problems with classifiers. Let us start with the bags and ankle boots. Here we denote the class of bags with class 1 and the class of ankle boots with class 2.



(a) Support Vector Machine (b) Logistic Regression (c) Random Forest Classifier (d) Neural Network



(e) Support Vector Machine (cal.) (f) Logistic Regression (cal.) (g) Random Forest Classifier (cal.) (h) Neural Network (cal.)

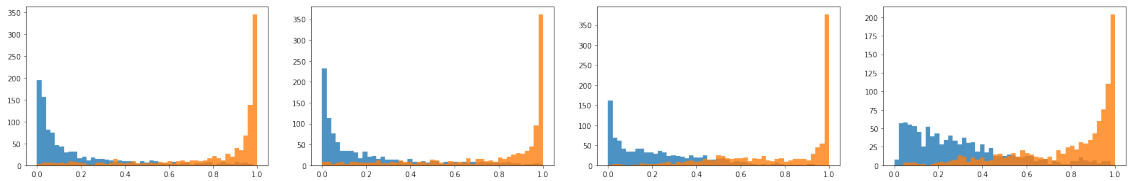
Figure 1.7: Histograms showing the distribution of the predicted probabilities per machine for the Fashion MNIST dataset with the bag and ankle boot classes. The predicted probabilities of class 1 (bag) are shown in blue, while the predicted probabilities of class 2 (ankle boot) are shown in orange. The (cal.) postfix denotes the calibrated version of the machine.

Much like in the normally distributed synthetic dataset, two clear separable peaks can be seen for all machines in Figure 1.7. It is obvious from this figure that all machines will perform very well (if not perfectly).

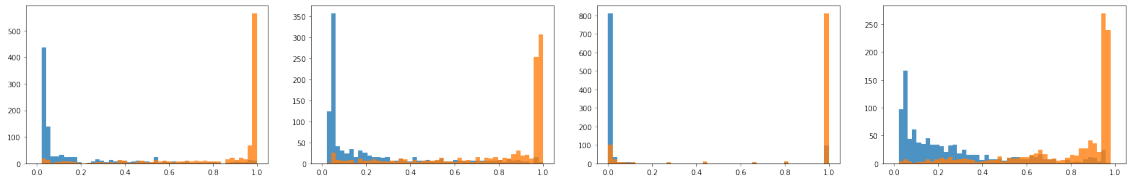
Table 1.3: Performance of various machines measured with various metrics on the Fashion MNIST dataset with the bag and ankle boot classes.

	SVM	SVM (Cal.)	LogReg	LogReg (cal.)	RFC	RFC (cal.)	NN	NN (cal.)
sPCC	0.9959	0.996	0.9955	0.9949	0.9949	0.993	0.9945	0.9938
Accuracy	0.998	0.998	0.9975	0.9975	0.9965	0.997	0.9965	0.9965
BA	0.998	0.998	0.9975	0.9975	0.9965	0.997	0.9965	0.9965
MCC	0.996	0.996	0.995	0.995	0.993	0.994	0.993	0.993
AUROC	0.9999	0.9985	0.9999	0.9984	1.0	0.9995	0.9995	0.9985
Bias	0.001	0.001	0.0007	0.0013	0.0004	-0.0005	0.0005	0.0003

According to Table 1.3, all machines do indeed perform very well. There is not much to note here, other than that the sPCC performs in accordance with the other metrics. The Bayesian Calibration also seems to not significantly change the behaviour of the machines, so we cannot conclude anything from these tests.
Let us proceed with the case of T-shirts/tops versus shirts. Here we denote the class of T-shirts/tops with class 1 and the class of shirts with class 2.



(a) Support Vector Machine (b) Logistic Regression (c) Random Forest Classifier (d) Neural Network



(e) Support Vector Machine (cal.) (f) Logistic Regression (cal.) (g) Random Forest Classifier (cal.) (h) Neural Network (cal.)

Figure 1.8: Histograms showing the distribution of the predicted probabilities per machine for the Fashion MNIST dataset with the T-shirt/top and shirt classes. The predicted probabilities of class 1 (T-shirt/top) are shown in blue, while the predicted probabilities of class 2 (shirt) are shown in orange. The (cal.) postfix denotes the calibrated version of the machine.

In Figure 1.8 we can see that there is a lot of overlap in class distributions in the non-calibrated machines. From this figure, we see that the calibrated machines perform a lot better than the non-calibrated in terms of the histograms. The calibrated machines showcase here what the Bayesian Calibration does in terms of the probability distributions. Since classes are not known to the machine when predicting, all probabilities are shifted in a similar fashion. Essentially, lower probabilities are pushed even lower and vice versa for the higher probabilities (see Figure 1.8b versus Figure 1.8f). How low "lower" is, depends on the data set. What happens precisely in the middle of the range $[0, 1]$ is difficult to know for an arbitrary dataset and we cannot precisely see what happens in Figure 1.8. Since there is some overlap between the probability distributions, one can also tell that the machines will not score perfectly.

Table 1.4: Performance of various machines measured with various metrics on the Fashion MNIST dataset with the T-shirt/top and shirt classes.

	SVM	SVM (Cal.)	LogReg	LogReg (cal.)	RFC	RFC (cal.)	NN	NN (cal.)
sPCC	0.7796	0.7765	0.7335	0.7292	0.778	0.7349	0.7206	0.7209
Accuracy	0.862	0.859	0.8335	0.831	0.862	0.862	0.8305	0.8305
BA	0.862	0.859	0.8335	0.831	0.862	0.862	0.8305	0.8305
MCC	0.7241	0.7182	0.6672	0.6621	0.7246	0.7246	0.6612	0.6622
AUROC	0.9393	0.9392	0.9179	0.918	0.942	0.9263	0.9107	0.9107
Bias	0.0035	0.0108	-0.0012	0.0085	0.0044	-0.024	0.0418	0.0302

Table 1.4 shows that the machines still perform fairly well. Taking the range difference into account shows that the sPCC score lower than the AUROC, but higher than the other metrics. All metrics agree that the SVM and RFC perform best, while the Logistic Regression and Neural Network perform worst. However, the difference between all machines is small.

Results

From the tests performed, we can conclude that the sPCC acts like we expected it to and compares well to the other metrics. For the very separable cases (the normally distributed synthetic data

and the (8,9) Fashion MNIST dataset), the sPCC shows high scores, much like the other metrics. For the less separable cases (the non-normally distributed synthetic data and the (0,6) Fashion MNIST dataset), the sPCC shows lower scores, again, much like the other metrics. A particular case is the non-calibrated Neural Network on the non-normally distributed data, where the two probability distributions are not clearly separable (see Figure 1.4d). The accuracy, balanced accuracy and Matthew's Correlation Coefficient are all flawed here, since τ is set at 0.5 and all predicted probabilities are below τ. The accuracy therefore tells us the ratio of negative samples, the balanced accuracy tells us that all samples of one class are classified correctly, while all samples of the other class are classified incorrectly and the Matthew's Correlation Coefficient is 0 as all predicted labels (after applying the boundary) are 0. As such there is no correlation between the predicted labels and the true labels. The AUROC, however, is also flawed. It has a high score (0.9792). This is correct when looking at the samples, but we can estimate the distributions from the samples' probabilities and see that, proportionally to their variances, the distributions might overlap when laying so closely together. The sPCC keeps the basic shape of the distribution (mean and variance) in mind and so, the sPCC is a lot lower in comparison (0.4529). An important issue with sPCC arises in this problem as well: the skewness of distributions is not taken into account when calculating the separability. We have seen this problem with opposing two log-normal distributions (see Figure 1.5), where there is no overlap between them per definition, but the sPCC still scores poorly. This is something that has to be kept in mind whenever distributions are shown to be heavily skewed towards each other. We discussed that the sPCC trumps the AUROC when applying and wanting to measure transformations to the predicted probabilities: the AUROC is unaffected by such transformations, while the sPCC is not. Therefore this was tested in the (0,6) Fashion MNIST dataset. A major flaw of this case, however, is the uncertainty that exists in the tails of the probability distributions. This causes the AUROC to change through the transformation (even if it should not).
We summarize the findings above in Table using the three properties we are looking for in a metric, defined at the beginning of this chapter (Section 1.1). These three properties were that a metric

1. can handle extreme imbalance
2. can accept probabilities (or any real-value)
3. is not heavily influenced by sampling errors

Table 1.5: Checklist to show which properties the tested metrics have. Property 1 is "the ability to handle extreme imbalance". Property 2 is "the ability to handle probabilities". Property 3 is "to not be heavily influenced by sampling errors". Unclear is put there were we are unable to clearly conclude if that metric has that property.

	sPCC	Acc	BA	MCC	AUROC
1	✓	×	×	✓	unclear
2	✓	×	×	×	✓
3	✓	×	×	✓	×

We can see in Table 1.5 that the sPCC checks all the three properties. The MCC also checks most of the properties, except for the fact that it needs labels instead of probabilities (and is therefore also dependent on the chosen τ). It is unclear whether or not the AUROC is able to handle extreme imbalance from its definition and the tests provided in Section 1.2.3. We will later see in this report that in fact the AUROC is not able to handle extreme imbalance well in Section 2.3.

We can also conclude that applying Bayes Calibration (at least) improves machine's ability to estimate the total number of positives in a data set. Applying Bayesian Calibration does not necessarily mean that machines are better at predicting the labels of samples. This can be seen for the Neural Network and the non-normally distributed data. The non-calibrated machine scores better on all metrics than the calibrated machine, except on bias. There is a significant decrease

in performance on an individual level, while there is a significant increase in performance on a population level.

1.3 The multiclass classification case

We can extend the binary application of the sPCC to a multiclass case. We will not use this multiclass extension in the second part of this thesis, but such an extension is interesting nonetheless and as such, we will go into some detail here. We will not do a study on the relation between the multiclass sPCC and other statistics and we will only look at the real world dataset, the Fashion MNIST dataset.
we will only look at a synthetic dataset (normally distributed).

As the name suggests, in a multiclass case $k > 2$ classes are present. There are two main ways of defining the labels $y = (y_1,...,y_n)$: $y_i = l$ where $l \in \{1,...,k\}$ or $y_i = e_l$ where e_l is the basis vector for the l'th index (such that $(e_l)_j = 1$ if and only if $j = l$). The latter is called a one-hot encoded label and is what we will be using from here on out.

1.3.1 Derivation of the sPCC for multiclass classification

Given are a set of true labels $y = (y_1,...,y_n) \in \{0,1\}^{n\times k}$ and a set of predicted labels $\hat{y} = (\hat{y}_1,...,\hat{y}_n) \in \{0,1\}^{n\times k}$, with the number of samples in class i equal to n_i such that $\sum_{i=1}^{k} n_i = n$. There are two possible cases: one versus one (OvO) and one versus rest (OvR). We start with introducing some notation: let $\hat{y}^i = \{\hat{y}_l | y_l = e_i\}$ denote the subset of predicted labels of samples with true label i, let $\hat{y}^{-i} = \{\hat{y}_l | y_l \neq e_i\}$ denote the subset of predicted labels of samples not with true label i and let $\hat{y}^{ij} = \{\hat{y}_l | y_l = e_i \vee y_l = e_j\}$ denote the subset of predicted labels of samples with either true label i or j.

One versus one

First the one versus one case, which is similar to the binary case with a few exceptions. Since we are working with vectors we opt to compare vector components instead of comparing vectors. We would like to now define the sPCC for the l'th component of the predicted probabilities for two classes. Note that we can only use the derivation of the sPCC of the binary case if one of the two classes is class l, otherwise the sPCC will reduce to zero: the l'th component of every true label in this comparison will be zero as well as their mean. Let us define the OvO sPCC:

$$r_{ij}(y,\hat{y}) = \left(\frac{n_i n_j}{(n_i+n_j)(n_i+n_j-1)}\right)^{1/2} \frac{\bar{\hat{y}}_i^i - \bar{\hat{y}}_i^j}{s_{\hat{y}^{ij}}}. \tag{1.11}$$

We calculate this for every possible pair (even for the pairs (i,i)). This returns a $k \times k$ matrix with zeros on the diagonal, where the zeros are $r_{ii}(y,\hat{y})$. One can interpret this matrix as some sort of inverse confusion matrix, where every value shows how likely (not a probability) it is that the machine confuses two classes. Much like in the binary case, a value of 1 indicates that two classes are perfectly separable, a value of 0 indicates that two classes are perfectly indistinguishable and a value of -1 indicates that two classes are perfectly confused.

One versus rest

The one versus rest case is a bit simpler in terms of derivation compared to the one versus one case. Again, we would like to immediately use the binary derivation, but cannot because of the different vector components. When comparing class i to the other classes, looking at a component of the predicted probabilities other than the i'th component tells us nothing. When comparing the i'th component of all vectors, we expect that only class i samples have a high value in the i'th component. All other classes are expected to have a low value. As such, we will only look at the i'th component of the vectors when comparing class i to the rest:

$$r_i(y,\hat{y}) = \left(\frac{n_i(n-n_i)}{n(n-1)}\right)^{1/2} \frac{\bar{\hat{y}}_i^i - \bar{\hat{y}}_i^{-i}}{s_{\hat{y}}} \tag{1.12}$$

Calculating this for all valid pairs returns a vector of length k, where the i'th component corresponds with the i'th class. A value of 1 at the i'th position of this vector indicates that class i is perfectly separable from all other classes, a value of 0 indicates that class i is perfectly indistinguishable from the other classes and a value of -1 indicates that class i is perfectly confused with the other classes. Values of (approximately) 0 cause the most difference between OvR and OvO. In the OvO case, we are able to know which two classes are confused. In the OvR case, we only know whether a class is being confused with another class, but not where this confusion comes from. Having larger sample sizes reduces the bias of the sPCC. This means that per definition a One versus Rest approach is less biased than a One versus One approach when purely looking at the pairwise sPCC's.

Reducing multiclass sPCC to a single value

While a matrix (in the OvO case) or a vector (in the OvR) can convey a lot of information, it is difficult to compare different machines using such objects. One might suggest taking the (arithmetic) average of all the values, but this is impossible in the sense that correlation coefficients cannot be added together in a sensible manner. As such, we propose several different methods of combining these values. All these methods involve ignoring / not using the diagonal in the OvO case. Other methods of reducing the matrix/vector to one value might exist. We only look at these three methods as a more in-depth look is out of the scope of this thesis.

- take the minimum of the matrix
- take the geometric mean of the matrix
- first take the Fisher's z-transformation of all the values, then the arithmetic mean and finally use the inverse Fisher's z-transformation (which we call a Fisher's average here)

Minimum Taking the minimum provides us with a good measure to see where the machines performs worst. However, this does not show us where this minimum occurs. It has other problems as well: in the case one of the sPCC's is equal to -1, we will obtain a -1 as an indication of performance. This means that a model that only scores a -1 for the sPCC's and a model that only scores a 1 for the sPCC's, except for one where it scores -1, are the same model in terms of performance. This is a very pessimistic way of looking at the performance of a model. This does not mean that this is a poor way of reducing the sPCC's to one value. It can still be useful if one needs a model to perform well in all cases. We will not use it since the minimum does not describe the overall performance of the model.

Geometric mean Taking the geometric mean does not raise the same problems as taking the minimum does: one poor sPCC score does not mean the overall performance of the model is poor. A (very significant) problem that occurs when using the geometric mean approach is that one value might be negative, while the other are positive. It is not trivial that such a situation can exists, so consider the following example for the OvO case: Let $\bar{\hat{y}}^i$ the vector of average probabilities for samples from class i. Let $\bar{\hat{y}}^1 = (2/3, 1/3, 0)$, $\bar{\hat{y}}^2 = (1/2, 1/4, 1/4)$ and $\bar{\hat{y}}^3 = (0, 0, 1)$. Since the variance does not matter for the sign of the correlation coefficients, we will not define it. From these averages, one can see that all coefficients are positive, except for $r_{21}(y, \hat{y})$ (since $1/4 - 1/3 < 0$). As such, we are unable to use the geometric mean.

The Fisher's z transformation Before we look into how the Fisher's z transformation (also known as the Fisher transformation) is defined, we look at a paper that looks at the bias of applying the transformation, averaging and transforming back. In [24], Corey, Dunlap and Burke look into the averaging of sPCC's and into the Fisher's average and the bias of both methods. There is most certainly a bias in both methods. Note that Corey, Dunlap and Burke look into sPCC's of independent and identically distributed sets of data. This means that all the values they average measure the same thing and that we must be careful about what applying such an

averaging method means in our context. We do not plan on averaging sPCC's that measure the same thing. In our context, however, it still makes sense to average our sPCC's, as we would like to know the average performance of the machine per pair of classes.

The Fisher transformation is defined as

$$z(r) = \frac{1}{2}\log\left(\frac{1+r}{1-r}\right) = \operatorname{artanh}(r)$$

and is identical to taking the inverse hyperbolic tangent function of r. The reverse Fisher transformation is defined as

$$r(z) = \frac{\exp(2z)-1}{\exp(2z)+1} = \tanh(z)$$

and is identical to taking the hyperbolic tangent function of z. Using these two definition, we can derive an expression for the average of the OvO case and the OvR case, respectively:

$$\bar{r} = \tanh\left(\sum_{i,j;i\neq j} \operatorname{artanh}(r_{ij}(y,\hat{y}))\right)$$

$$\bar{r} = \tanh\left(\sum_{i} \operatorname{artanh}(r_{i}(y,\hat{y}))\right)$$

We choose to use the Fisher's average from here on out, as it is a far better than using the minimum in our case.

1.3.2 Test cases

Contrary to the binary case, we will only look at one dataset: normally distributed data (synthetic). This dataset will be enough to show the behaviour of the multiclass sPCC. Again, we use some machine learning algorithms and a neural network:

- Logistic Regression
- Random Forest
- Neural Network

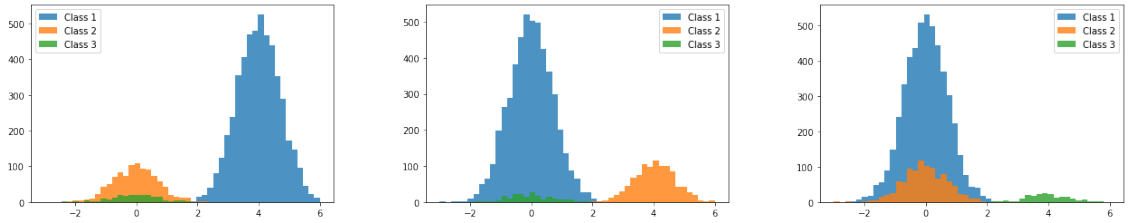
We do not use an SVM here. Training such a machine requires a lot more time than the other machines and as such it will not be used here. The neural network will be the same as for the binary case: an appropriately sized input layer (dependent on the number of features), three dense layers with five neurons each and an appropriate output layer (dependent on the number of classes). This might not be a great network for some of the provided problems, but the goal is not to find the best model.

In addition to the OvO and OvR versions of the sPCC, we use the same metrics as we did in the binary case. We will slightly alter the AUROC, as it also has the option for OvO and OvR. We will use both variants. We will also only be presenting the metrics and omitting the histograms of the predicted probabilities as this would require 9 histograms per test case. We will only present such histograms if needed.

Normally distributed data

For this set we generate normally distributed data in 3 dimensions. We take the example of the normally distributed data from the binary case (see Section 1.2.3), but transform it to a 3 dimensional problem instead of a 1 dimensional one. We want to create the binary problem in every dimension. This gives us that class i is multivariate normally distributed with mean $\mu_i = 4 \cdot e_i$ (with e_i the i'th basis vector) and standard deviation $\sigma_i = \sigma = (0.5, 0.5, 0.5)$. We create an imbalanced data set by making class 1 have 5000 samples, class 2 1000 samples and class 3 200

samples. In Figure 1.9 the class distributions are shown through sampling per dimension. From this figure, we can see that there is at least one dimension for every class that separates it from the other classes and as such, we can confidently guess that the various algorithms will perform well on this dataset.



(a) Dimension 1 (b) Dimension 2 (c) Dimension 3

Figure 1.9: Histograms showing the distributions of three classes, one for each dimension. Class i is multivariate normally distributed with mean $\mu_i = 4 \cdot e_i$ and a standard deviation $\sigma = (0.5, 0.5, 0.5)$.

Table 1.6

	Logistic Regression	RandomForestClassier	Neural Network
sPCC-OvR	0.9999	0.999	0.7871
sPCC-OvO	0.9998	0.9986	0.8351
Accuracy	1.0	1.0	0.8069
BA	1.0	1.0	0.3343
MCC	1.0	1.0	0.0451
AUROC-OvR	1.0	1.0	0.771
AUROC-OvO	1.0	1.0	0.771

As predicted, all machines perform well on this test case. The most important thing about Table 1.6 is the difference in the sPCC-OvR and sPCC-OvO: the OvO variant scores higher than the OvR variant. Let us look into this for the neural network as the difference there is the largest. The correlation coefficients calculated for both variants are presented in Table 1.7.

Table 1.7: The sPCC's per class (pair) calculated through the OvR method and the OvO method for a Neural Network.

	sPCC - OvR
Class 1 vs. rest	0.9237
Class 2 vs. rest	0.9145
Class 3 vs. rest	0.0231

	sPCC - OvO		
	Class 1	Class 2	Class 3
Class 1	0	0.9234	0.5793
Class 2	0.9436	0	0.9486
Class 3	-0.1916	0.8803	0

When looking at the values for class 1 and 2 in the OvR case, the neural network is capable of separating these two classes from the rest. This is also true for the OvO case: the scores are decently high for class 1 and 2, although it has some trouble separating class 3 from class 1 in the first component. The real difference is seen in class 3. In the OvR case, we see that the sPCC is near 0: no correlation between predictions and true labels. While in the OvO case, we see that in actuality, the neural network has significant trouble separating class 1 from class 3 in the third component. It is still good at separating class 2 from class 3. However, when summarizing these values, we see that the value of -0.19 (class 3 vs class 1) does not impact the value of the summarized sPCC for the OvO case (0.8351) as much as we would like it to. Similarly, in the OvR case, the summarized sPCC shows no sign of there being no correlation between class 3 and the other classes.

Results

We can now conclude that using the OvR method is a poor method as it does not properly capture the performance of the model. Using the Fisher's average is also a poor way of summarizing the values (for both the OvR and the OvO case), as low scores do not meaningfully impact the summarized sPCC. There are possible other ways of summarizing the OvO matrix or the OvR vector, but that is out of the scope for this thesis. For now, the OvO matrix is still a meaningful metric as it can be viewed as an inverse confusion matrix.

1.4 Conclusion Pearson Correlation Coefficient

To summarize this chapter: we have shown that the sample Pearson Correlation Coefficient can be used as a metric for binary classification. We have shown it is applicable to several types of machines, including natural probabilistic machines (Logistic Regression), decision machines (Support Vector Machine, Random Forest Classifier) and neural networks. A major problem of the sPCC are the higher moments of the probability distributions that are not taken into account. We have looked into one of these, the skewness. These higher moments cause the sPCC to calculate a score that is more pessimistic than is actually the case. The bias metric that has been introduced is also a great metric to measure the performance of machines regarding the estimation of positively labeled samples. We have also looked at the implementation of Bayesian Calibration. Applying Bayesian Calibration did not seem to increase the performance of the machines on a individual level, but did increase performance overall on a population level. Machines were better at estimating the total number of positively labeled samples with Bayesian Calibration compared to without Bayes Calibration. Lastly, we have looked at the multiclass extension of the sample Pearson Correlation Coefficient. We looked at two methods: the one versus rest case and the one versus one case. In the one versus rest case, we saw that when there is some confusion between two classes, it is difficult to see where this confusion occurs. In turn, the summarized value in the one versus rest case, is a poor metric to properly capture the behaviour of the machine. In the one versus one case, we were able to see where confusion stems from. However, the summarized value was, again, not able to properly capture the behaviour of the machine. We concluded that instead of summarizing the one versus one matrix, we should instead present the full one versus one matrix as some sort of inverse confusion matrix.

Chapter 2

Classifying platform economy companies from web scraped data

As mentioned before in Section 0.3, we would like to predict whether or not a company is a platform economy company based on the text retrieved from their website(s) and estimate the total number of platform economy companies. The dataset is very imbalanced (extreme rare event) and is based on language (natural language processing). The combination of estimating the total number of platform economy companies, extreme rare events and natural language processing causes for a very unique problem. Normal metrics will not work properly in this case (this has been discussed in Section 1.2.3). In this chapter, we will discuss several possible solutions in order to improve the current state-of-art methods and show their performance on the platform economy dataset both in terms of estimating the total number of platform economy businesses and in terms of classifying platform economy businesses.

2.1 Methods

In this section, we will discuss several methods that will be tested including the current state of the art. The methods that will be discussed are:

- Current state of the art
- Class weights
- Bootstrapping
- Ensembles

Recall that the dataset consists for 1396 out of positively labeled samples and for 593961 of negatively labeled samples (Section 0.3).

2.1.1 Current state of the art

The current state of the art method is a naive one and uses a set that is a very small subset of the total. It was developed by Piet Daas [25][26]. The set consisted of 1561 samples, 569 positives and 992 negatives, after processing. A stratified 80/20 split is used for training/testing, such that the class distribution is the same in the training and test sets. Several different machines have been tested and SVM with a linear kernel performed best overall. An SVM with an rbf kernel and Logistic Regression also performed very well. When testing the SVM on the full data set (∼600000 samples), execution time was at least a few days of processing per machine. Even when reducing the size of the set to only contain 10000 samples, execution still took too long (still a few days). Platt Scaling (see Section 0.4) is a contributor to this long execution time as well as the

large / high dimensional feature space that had to be handled. This problem did not occur when using Logistic Regression. We will be using this instead in the remainder of this thesis as it does not perform a lot worse than linear SVM's (as seen in [25][26]).

2.1.2 Changing class weights

Since the data is extremely imbalanced, a naive approach is to reduce this imbalance artificially. We change the class weights as to simulate having more samples of a certain class. As we have seen in the previous section (Section 2.1.1), the current state of the art uses Logistic Regression. One set of hyperparameters of a Logistic Regression is the set of class weights: weights that determine how heavily each class should impact the location and orientation of the decision boundary. Since we know that class 1 (the platform economy companies) are extremely rare, increasing the impact of this class can create a decision boundary that will more likely correctly predict samples of class 1. The way we change these class weights is by starting at $10^{0.1}$ and finishing at 10^2, taking 20 logarithmic steps and training 10 machines every step as to average their performance. A obvious problem with this approach is that sampling errors and outliers in terms of difficulty (difficult to classify samples) are amplified. This can cause the machine to overfit on these errors. It may try to learn a different feature distribution than is actually present in the test set. This leads to poor external validity.

2.1.3 Bootstrapping

Bootstrapping is a method developed by Bradley Efron in 1979 based on jackknife [27] and is used to approximate a statistic such as the mean or median of a dataset. Suppose we have a dataset $X = (x_i)_{i=1}^n$. In order to determine a statistic we take B bootstrap sample sets by sampling n data points *with replacement* from X. We then calculate the statistic of each bootstrap sample. This set of statistics forms a sampling distribution of the desired statistic.

This description of bootstrapping makes sense when looking at statistics, but not when looking at machine learning. Our end goal is to determine the label (or probability) for every sample in a test set X^*. Let us form a set of B bootstrap sample sets $\mathcal{B} = \{\mathcal{B}_i | i = 1, ..., B\}$ from our training set X by sampling n times from X with replacement for every $\mathcal{B}_i$. We now train a machine m_i on every set $\mathcal{B}_i$ in order to create a set of machines $\mathcal{M} = \{m_i | i = 1, .., B\}$. Instead of now having a sampling distribution of a statistic (often a single number), we have a sampling distribution of a function defined by the set $\mathcal{M}$: a function that maps a sample $x \in X^*$ to a label / probability. We would like to estimate the true label of such a sample $x \in X^*$ and to this end we take the average value of $\mathcal{M}(x) = \{m_1(x), ..., m_B(x)\}$. Note that this might look like the regular bagging (bootstrap aggregating) approach for classification. However, we do not use a majority vote in order to determine a label. Rather, we use the average of the votes, which resembles a more regression like approach in order to find desired scores.

Our data is extremely imbalanced. If we were to apply bootstrapping blindly, it is likely that we will train some machines that will have never seen any samples from the positive class. While these bootstrapping samples then perfectly describe the class distribution of the original training set on average, the machines that have never seen a positive sample will learn vastly different features than the ones that have seen such a sample. In order to be sure that every machine has at least seen both classes, we slightly change the way we apply bootstrapping. We apply bootstrapping to only the positive classes and apply normal sampling (without replacement) to the negative class. Instead of now trying to represent the class distributions in every training subset, we try to represent the feature space, such that every machine has a better view of what needs to be classified. This is another difference between this approach and bagging.

We now come to the number of bootstrap samples B that we have to take: what value does B have to be in order to properly approximate the labels? Wilcox [28] suggests using 599 bootstrapping samples (or machines in our case) as it is a good choice. He states that theory suggests a 0.95 confidence interval requires $0.95 \times (B + 1)$ to be equal to an integer, where B is the number

of bootstrapping samples[1]. This means that other choices for B are also possible, though the confidence interval of the corresponding obtained statistic might suffer/improve. This begs the question: does this still hold for our case, the machine learning case? The answer is no. An important difference between our case and the one discussed by Efron and Wilcox is that they are interested in estimating the sampling distribution of a random variable, such as the mean, exactly (this relates to Monte Carlo experiments). In our case, we are interested in estimating the mean of such a sampling distribution. Additionally, not all machines are deterministic in their training. This means that we also have to compensate for this behaviour. The law of large numbers then suggests that if we would like to estimate the mean, we are better off using a lot of bootstrapping samples. Let us therefore set B at 1000.

2.1.4 Ensembles

Similar to bootstrapping are ensembles: a collection of machines is trained on different parts of a given dataset and vote on the prediction for each sample. The idea of an ensemble is that machines trained on different parts of a dataset will learn different features and in turn have more expertise in certain parts of the data compared to other machines in the ensemble. Instead of taking an average of the predicted labels, we take a weighted vote, where each machine's weight is influenced by its accuracy on a validation set and how closely its learned features is related to that of a to be predicted data point. The training and test datasets are constructed as follows (this is an explanation of Algorithm 1): a set of data `X` is split into a training and test set, `X_train` and `X_test`. For every machine, we split the set `X_train` into `X_train'` and `X_val`. We now use `X_train'` for training the machine. Every machine also transforms this data to a TF-IDF matrix, such that we can obtain the observed features of the given training set for later use.
Afterwards we use `X_val` for obtaining a measure of internal validity (we will go into more detail about this topic in Section (2.2). Accuracy is used for this measure, but note that other measures or scores can be used depending on the problem. This measure of internal validity gives us some sort of weight to use for the voting system. This concludes the training part of the ensemble. We will continue with the prediction part of the ensemble.

Algorithm 1 Training an ensemble

Split X into X_{train} and X_{test}
Split corresponding y into corresponding y_{train} and y_{test}
for every machine M_i **do**
  Split X_{train} into $X'_{train,i}$ and $X_{val,i}$
  Split corresponding y_{train} into corresponding $y'_{train,i}$ and y_{val}
  Transform $X'_{train,i}$ to a TF-IDF matrix and store features F_i and transformation T_i
  Train machine M_i on $X'_{train,i}$ and $y'_{train,i}$
  Transform $X_{val,i}$ using transformation T_i
  Obtain predicted labels $y_{pred,i}$ of X_{val} by M_i
  Obtain the accuracy a_i by comparing the true labels $y_{val,i}$ and $y_{pred,i}$ for machine M_i
end for

In addition to this measure of internal validity, some sort of expertise weight has to be introduced in order to let the more knowledgeable machines on a certain sample have a heavier vote. We now use the features that we obtained earlier. Given a point `x`, obtain the features using the same approach as for the training sets. The size of the intersection of the set of features of the sample and that of a machine is divided by the number of features of the sample in order to give that machine a similarity score between the observed features and that machine's field of expertise. Since there is some randomness in the way text is stemmed, these scores are averaged

[1]There is a lot more to this, but it is out of the scope of this thesis. If you are interested, [29] goes in depth on the underlying theory and its connection to official statistics and its derivation from Monte Carlo experiments.

over 10 runs. This provides us with weights for the machines in the ensemble for every sample provided.

Algorithm 2 Predicting with an ensemble

for every machine M_i **do**
  Transform X_{test} using transformation T_i to $X_{test,i}$
  Obtain predicted labels $y_{test,i}$
end for
for j=1,j≤10,j++ **do**
  Transform X_{test} and obtain set of feature sets $F_{test,j,k}$
  for every machine M_i **do**
    for every sample k **do**
      Determine similarity score $W_{i,j,k} \leftarrow |F_{test,j,k} \cup F_i|/|F_{test,j,k}|$
    end for
  end for
end for
Average $W_{i,j,k}$ over j for average similarity score for machine M_i and sample k
Take the weighted average of $(y_{test,i})_i$ using weights $W_{i,k} \cdot a_i$

Here we make sure that there are always at least two partitions of the training set that have a non-empty intersection. This drives the probability of having just one machine be an expert on a part of the data to zero. We train 10 machines in order to properly determine the average predicted label for every sample in our testing set. To properly average the performance of this method, we train 20 ensembles and average their performance.

2.2 Testing

For all our own methods (class weight, bootstrap and ensembles), we create a differently sampled training set than for the current state of the art. For the current state of the art, the training set consists of 698 positively labeled samples and 698 negatively labeled samples. For the other methods, we increase the number of negatively labeled samples. We add these in order to increase the amount of possible information the machines are able to use. We do not want to create the same class distribution as in the real-world / test set, as such a set will contain a relatively small amount of information about the positive set. The training set for these methods consists of 698 positively labeled samples and 1628 negatively labeled samples, such that we get a 30/70 positive/negative split in terms of classes. The positives and negatives were split in different files, where the negatives were stored in files consisting of approximately 100000 samples. As such, we construct the test set by taking a 10% sample from three of such files to shape the negative part of the test set. We also take only 10% of the remaining positives, in order to create a test set that has a similar class distribution as that of the entire population; the test set consists for 0.23% out of positives.

We will be testing four variants of every single method: all combinations of using predicted labels or probabilities and using Bayesian Calibration or not. Note that for the current state of the art only the predicted labels and probabilities were used, but not Bayesian Calibration.

In order to measure the performance of all methods, we will be using the same metrics used in Section 1.2.3:

- bias
- sPCC
- Accuracy
- BA
- MCC
- AUROC

To provide some extra insight, we will also be providing the TP and positive estimate (Pos. Est.).

2.3 Results

In this section, we will look at the different methods and their performances. We will go through each method, one by one.

2.3.1 Current state of the art

We train and test the state of the art method 20 times in order to obtain the results presented in Table 2.1.

Table 2.1: The average performance of the four variants of the state of the art model. BayesCal indicates if Bayesian Calibration has been applied, Proba indicates if probabilities have been used.

BayesCal?	Proba?	TP	Pos. Est.	Bias	sPCC	Acc	BA	MCC	AUROC
No	No	69	2991	0.098344	0.1303	0.9012	0.9071	0.1303	0.9071
No	Yes	69	7656.731	0.255376	0.1409	0.9012	0.9071	0.1303	0.9694
Yes	No	69	637	0.019117	0.2053	0.9792	0.8088	0.2053	0.8088
Yes	Yes	69	636.540	0.019101	0.2254	0.9848	0.7610	0.2396	0.9693

We can see that applying Bayesian Calibration drastically improves the performances in terms of the bias. The estimate for the positives is still almost 10 times the actual size of the population for Bayesian Calibration. There are two things worth looking into here: the performance jumps in terms of the sPCC between the non-calibrated and calibrated variants and the difference in performance between the sPCC and the AUROC. Let us start with the jump in sPCC. First, let us look at the predictions of one of the 20 machines used. These figures can be found in Figure 2.1, where the top two figures show the predictions of the non-calibrated variant and the bottom two figures the predictions of the calibrated variant. On the left side, we have the normal (linear) scale on the y-axis, while on the right, we have a log scale on the y-axis.

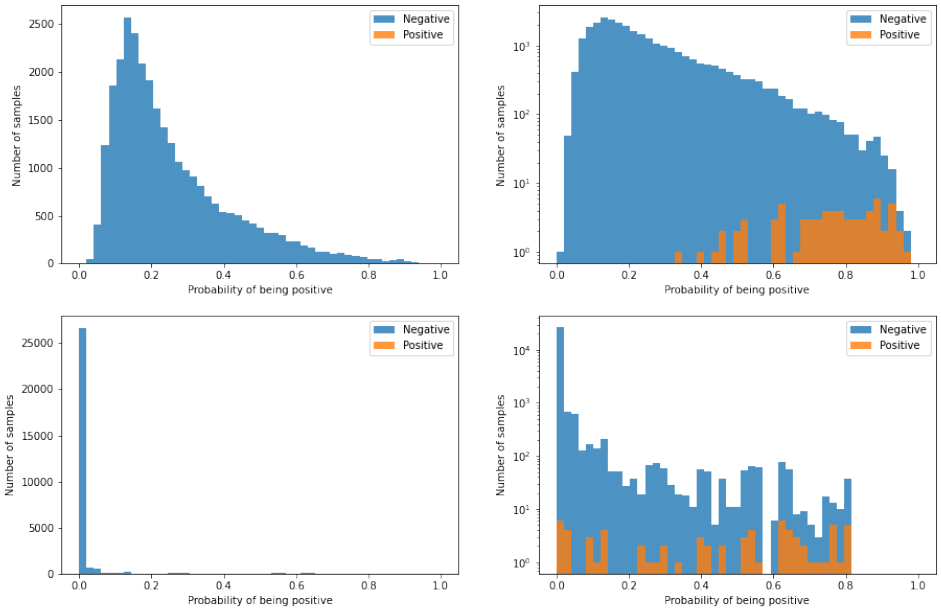
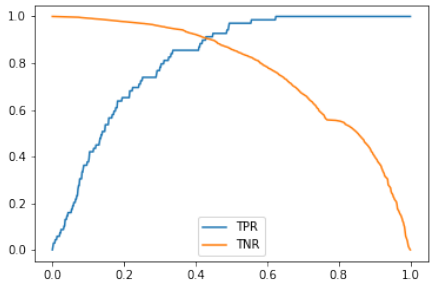
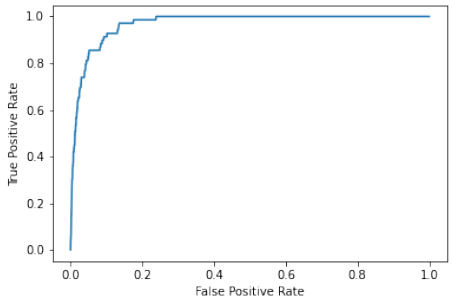


Figure 2.1: Four histograms of the predictions made by one of the current state-of-the-art models. The predictions of samples that belong to the negative class have been presented in blue, while the predictions of samples that belong to the positive class have been presented in orange. The top two figures present the non-calibrated predictions, while the bottom two figures present the calibrated predictions. The left two figures present the predictions with a normal (linear) scale on the y-axis, while the right two figures present the predictions with a log scale on the y-axis.

From these figures, we can see that the negative probability distribution is almost monotonic decreasing, after the obvious first increase. Almost, as there is a slight increase at the very end of the tail of the distribution (around 0.9). In this bump, the probability that a sample is from the negative class is larger than before the bump. In terms of Bayesian Calibration, $\mathbb{P}(y|c_0)$ (the probability that we have a certain probability y given it is from the negative class, c_0) is large, since a lot of the samples in the bump, around 0.9, are of the negative class. This probability will be smaller if we were to look at probabilities before the bump. Since the probability $\mathbb{P}(y|c_0)$ is larger for higher predicted probabilities, the upper bound of the domain of the predicted probabilities will be shifted down. Furthermore, this transformation will *not* be linear. Higher predicted probabilities (in the bump) will be shifted more toward the lower end of the domain than the slightly lower predicted probabilities (right before the bump). This entire shift in structure of the probability distributions causes a jump in the sPCC. So Bayesian Calibration does not only improve the performance of the model in terms of the bias, it also does so in terms of the sPCC. Let us now look at why the AUROC is such a high value if the sPCC is so low. The figures in Figure 2.1 are not really showing why the AUROC is as high as it is at a first glance. However, keep in mind that the AUROC uses the TPR and FPR. Implicitly the distributions get rescaled such that they have the same area. We can plot the True Negative Rate (1-FPR) and the True Positive Rate in order to visualize these rescaled distributions. Keep in mind that rather than the pdf (probability density function), the TPR and TNR look more like the cdf (cumilative density function). Additionally, we will plot the ROC curve.



(a) The True Positive Rate (in blue) and the True Negative Rate (in orange).



(b) The Receiving Operating Characteristic curve with the False Positive Rate on the x-axis and the True Positive Rate on the y-axis.

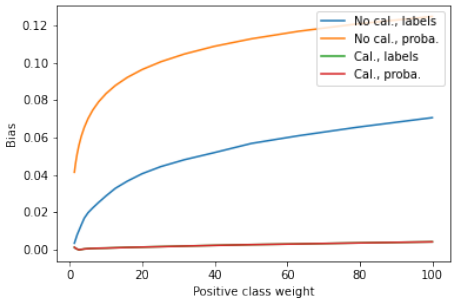
Figure 2.2: Two figures containing the True Positive and True Negative Rate (left) and the ROC curve (right).

From Figure 2.2a, we can see that there is a good reason the AUROC is as high as it is: there is a great candidate for separating the two classes (where the two lines intersect, around 0.45). The ROC curve reflects the high AUROC value as well (see Figure 2.2b). If Bayesian Calibration is a linear transformation, the AUROC and ROC cruve do not change (see Section 0.2). It seems that is coincidental here that the AUROC stays relatively the same before and after the calibration. A major problem with the AUROC, is the fact that prevalence is not taken into account. While the negative probability distribution tells us there is a low probability of negative samples being present in the tail of the distribution, there is overall a low probability of positive samples being present anywhere (also known as the prevalence). This means that if we find a sample with a predicted probability in the higher end of the domain (around 0.8), we cannot be as sure that it is a positive sample as Figure 2.2 would make you believe. Recall to Section 1.2.2, where we stated that it was unclear if the AUROC could handle extreme imbalance. Here we see that the AUROC is not able to handle it.

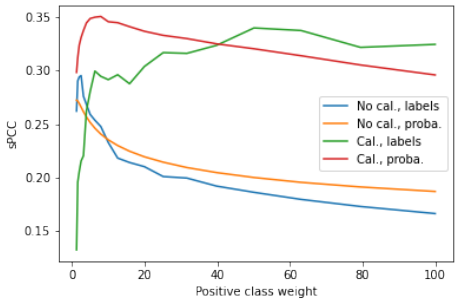
We can conclude that the bias and sPCC are the metrics that accurately describe the performance of the different variants. To complete this analysis, we can see that Bayesian Calibration improves the models performance in terms of both bias and sPCC. It does not matter if we predict probabilities or labels.

2.3.2 Changing class weights

We train and test this method 10 times per weight, looking at a total of 20 different weights, in order to obtain the average performance for each weight. Since there are effectively a total of 20 different methods, we will present their performances (in terms of the bias and sPCC) in graphs in Figure 2.3. We only present the bias and sPCC here, since there are a lot of methods/weights to compare and in Section 2.3.1 we found that all we really need to find a good machine / weight are the bias and the sPCC. All metrics for each of the different weights can be found in Appendix B.



(a) The bias over the different class positive weights per variant.



(b) The sPCC (y-axis) over the different positive class weights per variant.

Figure 2.3: The bias (left) and sPCC (right) over the different positive class weights. The legend indicates which line is calibrated (Cal.) or not (No cal.) and if labels or probabilities (proba.) are used.

From Figure 2.3a we can see that a lower weight is more preferable in terms of a better bias. We can also see that the bias and the sPCC clearly have a correlation to the positive class weight. Looking at Tables B.1 and B.2 (in Appendix B), we could choose the class weight that provides us with a bias of 0. However, this means we are optimizing on our test set. We do not want to do this, since this cannot be done when deploying such a model on real-world scenario's. To this end, we will continue with a positive class weight 100. This does not mean that this class weight is the best model out of all class weights, but it gives us some insight on the workings of this model. From Figure 2.3b, we can see that the same holds for the sPCC. From Figure 2.3b, we can also see that there is a jump in performance in terms of the sPCC. This same behaviour happened in the state of the art method. The jump here is smallest for smaller positive class weights. To see if this jump is again caused by a "bump" in the predictions, we plot these predictions in Figure 2.4 in a similar fashion to Figure 2.1 for positive class weight equal to 100.

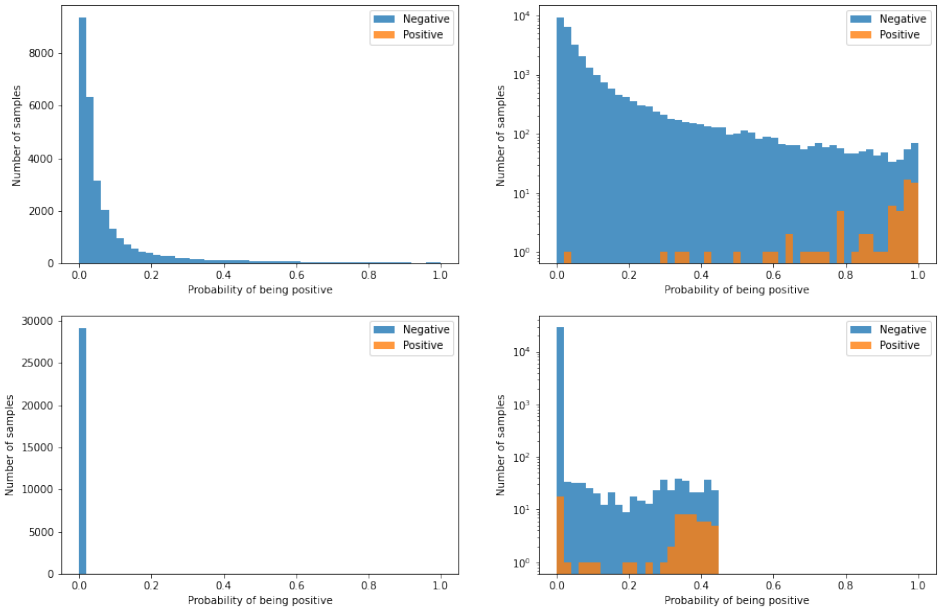


Figure 2.4: Four histograms of the predictions made by a machine using the class weight method, using positive class weight 100.0. The predictions of samples that belong to the negative class have been presented in blue, while the predictions of samples that belong to the positive class have been presented in orange. The top two figures present the non-calibrated predictions, while the bottom two figures present the calibrated predictions. The left two figures present the predictions with a normal (linear) scale on the y-axis, while the right two figures present the predictions with a log scale on the y-axis.

In the figures in Figure 2.4, we can see similar behaviour to that of the current state of the art model (seen in Figure 2.1). We see another bump near the end of the domain (near a probability of 0.95). This again will cause a shift in the probabilities when applying Bayesian Calibration. This time the shift is more extreme than in the state of the art model as the maximum probability is now only around 0.45. Since we have put τ at 0.5, the accuracy will be 1 minus the prevalence, the BA will be 0.5 and the MCC will be 0.0. In fact, this is the case for all the different positive class weights (see Appendix B). This means that this large shift occurs in all different models and therefore such a bump is present in all of them as well.

2.3.3 Bootstrapping

We train and test this method using 1000 bootstrap samples/machines in order to obtain the results presented in Table 2.2.

Table 2.2: The average performance of the four variants of the bootstrapping method. BayesCal indicates if Bayesian Calibration has been applied, Proba indicates if probabilities have been used.

BayesCal?	Proba?	TP	Pos. Est.	Bias	sPCC	Acc	BA	MCC	AUROC
No	No	69	830	0.025613	0.1870	0.9728	0.8201	0.1870	0.8201
No	Yes	69	4147.237	0.137259	0.1688	0.9728	0.8201	0.1870	0.9651
Yes	No	69	307	0.00801	0.2163	0.9895	0.7272	0.2163	0.7272
Yes	Yes	69	306.222	0.007984	0.2197	0.9925	0.6275	0.1536	0.9646

Again, we look at the bias and sPCC in order to determine how well the model performs. The Bayesian Calibrated variants perform a lot better than the calibrated current state of the art model: the bias of the bootstrap models is roughly half. The sPCC is not significantly higher or lower than that of the state of the art model. Note that there is still a difference between the non-calibrated and calibrated models in terms of the sPCC. However, this difference is a lot smaller than for the other models. We also see that there is not a large difference between using probabilities or labels, both in terms of the bias and the sPCC.

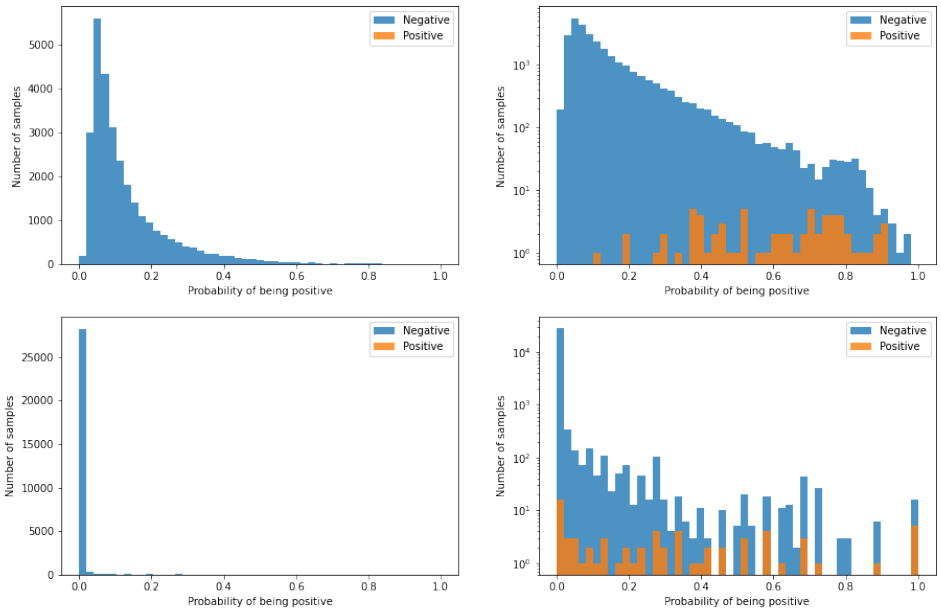


Figure 2.5: Four histograms of the predictions made by a machine using the bootstrapping method. The predictions of samples that belong to the negative class have been presented in blue, while the predictions of samples that belong to the positive class have been presented in orange. The top two figures present the non-calibrated predictions, while the bottom two figures present the calibrated predictions. The left two figures present the predictions with a normal (linear) scale on the y-axis, while the right two figures present the predictions with a log scale on the y-axis.

We see a behaviour in Figure 2.5 similar to that of Figure 2.4: a bump can be seen in the tail of the negative class distribution. There is an important difference between the two models: the size of the tail for bootstrapping, which is a lot larger in the case of changing the class weights. This can be seen by the slope of the negative distribution in the top right graph (for both Figures 2.4 and 2.5). These two differences should mean that this model should perform better than the

class weight method. However, in Figure 2.5 we can also see that the distribution of the positive class is very noisy (both before and after calibration) and has a high variance. There are also some negative samples that have a higher score than all the positives. While Bayesian Calibration might be able to improve the bias, it will have a difficult time increase the classification capabilities of this method.

2.3.4 Ensemble

We train and test this method 20 times, where each ensemble consists of 10 models, in order to obtain the results presented in Table 2.3.

Table 2.3: The average performance of the four variants of the ensemble method. BayesCal indicates if Bayesian Calibration has been applied, Proba indicates if probabilities have been used.

BayesCal?	Proba?	TP	Pos. Est.	Bias	sPCC	Acc	BA	MCC	AUROC
No	No	69	0	-0.00232	0	0.9977	0.5	0	0.5
No	Yes	69	120.5289	0.00173	0.0638	0.9977	0.5	0	0.5301
Yes	No	69	0	-0.00232	0	0.9977	0.5	0	0.5
Yes	Yes	69	6.6905	-0.00210	0.0469	0.9977	0.5	0	0.4436

From Table 2.3 we can see that this method is a very poor approach to this problem. There is barely any correlation between the true labels and the predicted labels, in both the non-calibrated and calibrated case. All variants also perform poorly in terms of the bias / positive estimate. For this particular problem we do not want to underestimate the total number of positives. This means that only the non-calibrated variant using probabilities is still useful, were it not for its poor performance in terms of the sPCC.

2.3.5 Summary

Here we will compare all tested methods and compare their results. We can see that the calibrated variant using probabilities is the best out of the four variants (sometimes the labels are equally as good). We can use these probabilities to not only estimate the number of positive, but we can also use it to estimate the standard deviation in this estimate. The results are shown in Table 2.4.

Table 2.4: The positive estimate (Pos. Est.) and its standard deviation (sd) and the bias per different method using Bayesian Calibration and probabilities.

	TP	Pos. Est.	sd	Bias
State of the art	69	636.540	19.449	0.019101
Class Weight (100.0)	69	214.770	11.665	0.004905
Bootstrap	69	306.222	13.389	0.007984
Ensemble	69	6.690	2.581	-0.00210

From Table 2.4 we can see that all methods are biased. According to Serwin and Perkins [30], there are three (main) reasons for bias in algorithms:

I input bias: inputs are "non-representative, lack information, historically biased, or otherwise "bad" data."

II training bias: "Training bias can arise where input data is mis-categorized or where outputs are inappropriately assessed."

III programming bias: "biases engrained in the design of the algorithm, such as subjective rules programmed into an algorithm."

We can recognize and relate these to our situation:

I input bias: due to the extreme imbalance in classes, it was necessary to use a slightly less imbalanced training set in order to properly present the features of the positive class. As a consequence, the input data is non-representative. Additionally, it might be that the classes overlap in the feature space in an unbalanced manner. Such that one of the classes is harder to distinguish from the other compared to the other way around.

II training bias: we average our performance over multiple machines. As such, these machines see different aspects of the feature space / data set. We can then assume that training bias is minimized here.

III programming bias: while we do not explicitly set up rules for the machines, we do implicitly so by choosing a Logistic Regression as our machine. We assume there is a linear decision boundary / hyperplane in the feature space. There might not be such a "simple" boundary and as such bias can be introduced.

Logistic Regression is a linear model. This means that it is only able to partition the feature space through a hyperplane. Programming bias occurs when the classes are not able to be optimally separated with the use of hyperplane, but rather with a more complex non-linear hypersurface. If the model uses a hyperplane while the decision boundary is more complex, we are effectively missing out on information. Linear combinations on features parallel to the hyperplane are not taken into account and are effectively reduced to noise or unexplainable variance in the data. We can see this especially clear in the bootstrapping case. Input bias occurs when classes are similar up to a certain degree: there are samples from the negative class that resemble the positive class more than some samples from the positive class. Suppose there is an optimal decision boundary. Bias would then exist if the overlap is not symmetric: the number of FP is not equal to the number of FN. Bayesian Calibration seeks to compensate for this. However, it does so using the scores obtained by the Logistic Regression (in our case) and, in turn, the information the Logistic Regression obtains from the feature space. However, this information is not complete as we have just discussed. Therefore, we cannot completely remove the bias caused by programming bias through Bayesian Calibration. Additionally, input bias occurs when the training set is not representative of the testing set / real-world population. We chose for this because of the extreme imbalance in classes and to ensure the positives were represented enough compared to the negatives. However, this should be solved by Bayesian Calibration.

From Table 2.4, we can also see that the Class Weight method (with a weight of 100) works best: it has the lowest bias and standard deviation. The bootstrapping method also performs fairly well. In this case it looks like the class weight method is the best out of all of them. However, there are some arguments in favor of and against using this method versus the bootstrapping method. Finding the right positive class weight requires some sort of hyperparameter optimization, something that is not required by bootstrapping. After finding the right positive class weight, the class weight method is a lot more efficient and quicker than the bootstrapping method. Only one machine has to be used and stored instead of possibly 1000. The class weight method also had the added benefit of being a lot more transparent: it is easy to see which features are important and which are not. The bootstrapping method is not as transparent, but it does not overfit as easily as the class weight method. When changing the (positive) class weight, we do not only increase the importance of data points that give models a good idea of what features to pick, it also increase the importance of data points that do the exact opposite. This means that the class weight models can latch onto bad features. Bad features might occur in some of the bootstrap models, but since we take enough bootstrap samples, these features are a lot less likely (compared to the class weight method) to influence the final result. A final argument in favor of bootstrapping is the fact that there is a lot more documentation and that it is a statistical method. We have yet to find a clear method of picking the right positive class weight.

2.4 Conclusion

In this thesis, we were looking for a method that is able to estimate the number of platform economy companies in an extremely imbalanced set of web-scraped data. The first step to finding such a method was using Bayesian Calibration. In order to test the influence of Bayesian Calibration, a close look had to be taken at commonly used metrics. The most notable outcome of this part is that the AUROC is a poor metric to use on extremely imbalanced populations, as it does not take this imbalance into account and rather assumes that both classes are balanced in the total population.

An alternative for the AUROC was proposed, the sample Pearson Correlation Coefficient, that does take this prevalence into account. The sPCC did not come without problems, however. It only takes the first two moments of a distribution into account, but none of the higher moments. This causes the sPCC to be a very pessimistic measure, scoring the separability lower than it actually is.

Whilst the sPCC was used to score how well machines were at classifying, the bias was introduced to score how well machines were at estimating the number of platform economy companies. We saw that Bayesian Calibration improved the bias significantly. It also increased the sPCC, meaning that Bayesian Calibration is also useful for improving classification models.

The proposed methods, bootstrapping and changing the positive class weight in combination with Bayesian Calibration, showed to be poor at classifying data. We can conclude that none of the proposed models are useful for this goal. These methods, however, were good at estimating the total number of platform economy companies. Even though Bayesian Calibration was used, bias was still present in these methods. This was caused by the simplicity of the used model, Logistic Regression. It could not capture the complexity of the most optimal decision boundary between the two classes. As we saw in Section 0.2, this causes the calibration to poorly estimate the prevalence and is therefore unable to properly remove the bias. According to Puts and Daas [6], estimators have to be unbiased for a use in official statistics. Because these estimators are not unbiased, we cannot use these for official statistics.

We can conclude that the methods proposed here show that using web-scraped data to estimate and classify platform economy companies is a promising avenue to explore.

2.5 Discussion and Future Research

Here we will discuss some of the problems that we have encountered and recommendations for future research.

We will start by looking into the sPCC. Whilst relations to other metrics and tests have been shown, more research and benchmarking should be done to prove that this is a proper measure. Additionally, other correlation coefficients could prove to be useful as classification metrics, such as the Spearman Rho or the Kendall Tau. As noted in Section 1.2.3, the sPCC only uses the first two moments of distributions. This causes distributions with high higher moments (such as a high skewness or high kurtosis) to skew the outcome of the sPCC. In order to solve this, steps could be made to compensate for this behaviour. One could for example change the sPCC to take these moments into account or use the decision function scores instead of the predicted probabilities.

We saw that the AUROC and other metrics do not work properly on extremely imbalanced populations, since they do not take prevalence into account. A recommendation is to look into ways to compensate for this behaviour by learning the prevalence as a weight or a hyperparameter and using this weight in the calculation of these metrics, for example.

Of the methods presented here, changing the positive class weight is the one that stood out the most as there is not a lot of research on how effective this method actually is for solving extreme imbalance. The approach here showed that the positive class weight is a hyperparameter that requires a validation set for optimization. However, such a method for optimizing the positive class weight can still be developed and tested.

Finally, more complex models should be used in order to solve the platform economy problem. We have seen that the proposed methods are a step in the right direction, but fail to capture the complexity of the feature space. Support Vector Machines (using an rbf kernel), Random Forest Classifiers and Deep Learning methods might be able to capture this complexity better and would be worth exploring / are recommended methods to explore. This will not only help reduce the bias, but also increase the ability to separate classes. A problem with these methods is the computing time. Logistic Regression is very quick in predicting scores since it is a such a simple model. More complex models might give better results, but will require a lot more time and memory.

Bibliography

[1] Piet J.H. Daas and Suzanne Van der Doef. Using website texts to detect innovative companies. *Center for Big Data Statistics - Working paper*, January 2021. 1, 4

[2] Tim De Jong and Piet Daas. Optimzing the hyperparameters for platform-economy classification. *Center for Big Data Statistics - Working Paper*, March 2022. 1, 4

[3] Piet Daas, Wolter Hassink, and Bart Klijs. On the validity of statistical text mining as a sampling method: An application to detect digital platforms, 2021. 1, 4

[4] P.J.H Daas, Blanca De-Miguel-Molina, and Mar'ia De-Miguel-Molina. Identifying drone web sites in multiple countries and languages with a single model. *Journal of Data Science*, pages 1–14, 01 2023. 1

[5] Marco J.H. Puts and Piet J.H. Daas. Unbiased estimations based on binary classifiers: A maximum likelihood approach. 02 2021. 2, 7

[6] Marco Puts and Piet Daas. Machine learning from the perspective of official statistic. *The Survey Statistician*, 84:12–17, 2021. 2, 42

[7] Marco Puts and Yvonne Gootzen. Unbiased estimations based on classifiers: an bayesian ideal classification approach. *Journal of Machine Learning Research 1 (2000) 1-48*, 2023 (in preparation). 2, 7

[8] Gary Weiss. Learning to predict extremely rare events. *AAAI Workshop on Learning from Imbalanced Data Sets*, 05 2000. 4

[9] Ryuhei Hamaguchi, Ken Sakurada, and Ryosuke Nakamura. Rare event detection using disentangled representation learning. pages 9319–9327, 06 2019. 4

[10] John Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Adv. Large Margin Classif.*, 10, 06 2000. 5, 10

[11] Meelis Kull, Telmo M. Silva Filho, and Peter Flach. Beyond sigmoids: How to obtain well-calibrated probabilities from binary classifiers with beta calibration. *Electronic Journal of Statistics*, 11(2):5052 – 5080, 2017. 5

[12] C. Ferri, Peter A. Flach, and José Hernández-Orallo. Modifying roc curves to incorporate predicted probabilities. 01 2005. 8, 12

[13] R. A. Fisher. Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika*, 10(4):507–521, 1915. 9

[14] Michael E. Tipping. Sparse bayesian learning and the relevance vector machine. *J. Mach. Learn. Res.*, 1:211–244, sep 2001. 10

[15] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 625–632, New York, NY, USA, 2005. Association for Computing Machinery. 10

[16] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21, 01 2020. 11

[17] Davide Chicco, Niklas Tötsch, and Giuseppe Jurman. The matthews correlation coefficient (mcc) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioData Mining*, 14, 02 2021. 11

[18] W. Peterson, T. Birdsall, and W. Fox. The theory of signal detectability. *Transactions of the IRE Professional Group on Information Theory*, 4(4):171–212, 1954. 11

[19] John Arthur Swets, WP Tanner Jr, and TG Birdsall. The evidence for a decision-making theory of visual detection. Technical report, MICHIGAN UNIV ANN ARBOR DEPT OF ELECTRICAL ENGINEERING, 1955. 11

[20] Abhranil Das and Wilson S Geisler. A method to integrate and classify normal distributions, 2020. 12

[21] J.A. Hanley and Barbara Mcneil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143:29–36, 05 1982. 12

[22] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. 18

[23] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 18

[24] David Corey, William Dunlap, and Michael Burke. Averaging correlations: Expected values and bias in combined pearson rs and fisher's z transformations. *Journal of General Psychology - J GEN PSYCHOL*, 125:245–261, 07 1998. 24

[25] P.J.H. Daas. Tekst classificatie van platformeconomie websites. *CBS Internal report*, March 31 2020. 29, 30

[26] P.J.H. Daas. Tekst classificatie van platformeconomie websites (deel 2). *CBS Internal report*, August 28 2020. 29, 30

[27] B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1 – 26, 1979. 30

[28] Rand Wilcox. *Fundamentals of Modern Statistical Methods: Substantially Improving Power and Accuracy*. 01 2001. 30

[29] Russell Davidson and James G. MacKinnon. Bootstrap tests: how many bootstraps? *Econometric Reviews*, 19(1):55–68, 2000. 31

[30] Ken Serwin and Alfred Jr. H. Perkins. Algorithmic bias: A new legal frontier. 40

Appendix A

The equivalence of the MCC and sPCC for binary sets

Suppose we have true labels $y = (y_1,...,y_n) \in \{0,1\}^n$ and corresponding predicted labels $\hat{y} = (\hat{y}_1,...,\hat{y}_n) \in \{0,1\}^n$. For the definition of the sPCC, let us first define the mean of the true labels $\bar{y}$ and the mean of the predicted labels $\bar{\hat{y}}$ in terms of the 2×2 contingency table:

$$\bar{y} = \frac{TP + FN}{n} \qquad\qquad \bar{\hat{y}} = \frac{TP + FP}{n} \tag{A.1}$$

Let us now start to prove that the sPCC is equivalent to the MCC for the given true and predicted labels. Let $\mathcal{A}_{TP} = \{i | y_i = 1, \hat{y} = 1\}$ the subset of data that is classified as True Positive and define analogous sets for FP, FN, TN. We first start with deriving the sample covariance between the true and predicted labels:

$$\begin{aligned}
(n-1) \cdot q_{y\hat{y}} &= \sum_{i=1}^{n} (y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}}) \\
&= \sum_{i \in \mathcal{A}_{TP}} (1 - \bar{y})(1 - \bar{\hat{y}}) \\
&+ \sum_{i \in \mathcal{A}_{FN}} (1 - \bar{y})(0 - \bar{\hat{y}}) \\
&+ \sum_{i \in \mathcal{A}_{FP}} (0 - \bar{y})(1 - \bar{\hat{y}}) \\
&+ \sum_{i \in \mathcal{A}_{TN}} (0 - \bar{y})(0 - \bar{\hat{y}}) \\
&= TP(1 - \bar{y})(1 - \bar{\hat{y}}) + FN(1 - \bar{y})(0 - \bar{\hat{y}}) \\
&+ FP(0 - \bar{y})(1 - \bar{\hat{y}}) + TN(0 - \bar{y})(0 - \bar{\hat{y}})
\end{aligned}$$

We use the definition of the average $\bar{y}$ seen in Eq. A.1 and the fact that $n = TP+FP+FN+TN$.

$$\begin{aligned}(n-1)\cdot q_{y\hat{y}} &= TP\left(1-\frac{TP+FN}{n}\right)\left(1-\frac{TP+FP}{n}\right)\\ &+ FN\left(1-\frac{TP+FN}{n}\right)\left(-\frac{TP+FP}{n}\right)\\ &+ FP\left(-\frac{TP+FN}{n}\right)\left(1-\frac{TP+FP}{n}\right)\\ &+ TN\left(-\frac{TP+FN}{n}\right)\left(-\frac{TP+FP}{n}\right)\\ &= \frac{TP}{n^2}(TN+FP)(TN+FN) - \frac{FN}{n^2}(TN+FP)(TP+FP)\\ &- \frac{FP}{n^2}(TP+FN)(TN+FN) + \frac{TN}{n^2}(TP+FN)(TP+FN)\end{aligned}$$

Multiply both sides with n^2 and collect terms on the RHS.

$$\begin{aligned}n^2(n-1)\cdot q_{y\hat{y}} &= (TP\cdot TN + TP\cdot FN - FN\cdot TP - FN\cdot FP)(TN+FP)\\ &+ (TN\cdot TP + TN\cdot FP - FP\cdot TN - FP\cdot FN)(TP+FN)\\ &= n(TP\cdot TN - FN\cdot FP)\end{aligned}$$

This leaves us with the following derived expression for the covariance between y and $\hat{y}$:

$$(n-1)\cdot q_{y\hat{y}} = \frac{TP\cdot TN - FN\cdot FP}{n} \tag{A.2}$$

Let us now look at the variance of the true labels:

$$\begin{aligned}(n-1)\cdot s_y^2 &= \sum_{i=1}^{n}(y_i-\bar{y})^2\\ &= (TP+FN)\left(\frac{TN+FP}{n}\right)^2 + (TN+FP)\left(\frac{TP+FN}{n}\right)^2\\ &= \frac{(TP+FN)(TN+FP)}{n}\end{aligned}$$

Similarly, the variance of the predicted labels can be derived:

$$(n-1)\cdot s_{\hat{y}}^2 = \frac{(TP+FP)(TN+FN)}{n}$$

We now collect all the derived expressions to complete our proof:

$$\begin{aligned}r_{y\hat{y}} = \frac{q_{y\hat{y}}}{s_y s_{\hat{y}}} &= \frac{TP\cdot TN - FN\cdot FP}{(TP+FN)(TP+FP)(TN+FP)(TN+FN)}\\ &= MCC\end{aligned}$$

And so, the sample Pearson correlation coefficient is equal to the Matthew's correlation coefficient for binary true and predicted labels. Q.E.D.

Appendix B

Class Weight Metrics

Table B.1: Metrics per variant and positive class weight from 1.2589 through 6.3096.

Pos. class weight	BayesCal?	Proba?	TP	Pos. Est.	Bias	sPCC	Acc	BA	MCC	AUROC
1.2589	No	No	69.0	174.0	0.0035	0.2621	0.9938	0.7077	0.2621	0.7077
1.2589	No	Yes	69.0	1300.8988	0.0415	0.2723	0.9938	0.7077	0.2621	0.9822
1.2589	Yes	No	69.0	29.0	-0.0013	0.1328	0.9971	0.5431	0.1328	0.5431
1.2589	Yes	Yes	69.0	28.7089	-0.0014	0.2982	0.9977	0.5	0.0	0.9822
1.5849	No	No	69.0	231.0	0.0055	0.2903	0.9924	0.7648	0.2903	0.7648
1.5849	No	Yes	69.0	1442.8345	0.0462	0.2715	0.9924	0.7648	0.2903	0.9834
1.5849	Yes	No	69.0	45.0	-0.0008	0.1959	0.9969	0.5791	0.1959	0.5791
1.5849	Yes	Yes	69.0	44.5502	-0.0008	0.3119	0.9977	0.5	0.0	0.983
1.9953	No	No	69.0	304.0	0.0079	0.2939	0.9903	0.8072	0.2939	0.8072
1.9953	No	Yes	69.0	1586.303	0.0511	0.2692	0.9903	0.8072	0.2939	0.9842
1.9953	Yes	No	69.0	57.0	-0.0004	0.2056	0.9966	0.5935	0.2056	0.5935
1.9953	Yes	Yes	69.0	56.6204	-0.0004	0.3235	0.9977	0.5	0.0	0.9843
2.5119	No	No	69.0	375.0	0.0103	0.2952	0.9883	0.8423	0.2952	0.8423
2.5119	No	Yes	69.0	1729.7563	0.0559	0.2658	0.9883	0.8423	0.2952	0.9849
2.5119	Yes	No	69.0	69.0	0.0	0.2156	0.9964	0.6078	0.2156	0.6078
2.5119	Yes	Yes	69.0	68.4012	-0.0	0.3307	0.9977	0.5	0.0	0.9854
3.1623	No	No	69.0	464.0	0.0133	0.2759	0.9854	0.8553	0.2759	0.8553
3.1623	No	Yes	69.0	1872.2321	0.0607	0.2615	0.9854	0.8553	0.2759	0.9855
3.1623	Yes	No	69.0	75.0	0.0002	0.2205	0.9962	0.6149	0.2205	0.6149
3.1623	Yes	Yes	69.0	74.9512	0.0002	0.3372	0.9977	0.5	0.0	0.9855
3.9811	No	No	69.0	571.0	0.0169	0.2683	0.9821	0.8826	0.2683	0.8826
3.9811	No	Yes	69.0	2012.7578	0.0654	0.2567	0.9821	0.8826	0.2683	0.9859
3.9811	Yes	No	69.0	83.0	0.0005	0.2624	0.9962	0.6439	0.2624	0.6439
3.9811	Yes	Yes	69.0	82.6481	0.0005	0.3443	0.9977	0.5	0.0	0.9859
5.0119	No	No	69.0	657.0	0.0198	0.259	0.9793	0.8957	0.259	0.8957
5.0119	No	Yes	69.0	2150.9448	0.0701	0.2515	0.9793	0.8957	0.259	0.9863
5.0119	Yes	No	69.0	88.0	0.0006	0.2804	0.9962	0.6583	0.2804	0.6583
5.0119	Yes	Yes	69.0	87.8327	0.0006	0.3484	0.9977	0.5	0.0	0.9863
6.3096	No	No	69.0	734.0	0.0224	0.2536	0.9769	0.9089	0.2536	0.9089
6.3096	No	Yes	69.0	2286.6128	0.0746	0.2462	0.9769	0.9089	0.2536	0.9865
6.3096	Yes	No	69.0	92.0	0.0008	0.2994	0.9962	0.6728	0.2994	0.6728
6.3096	Yes	Yes	69.0	91.4968	0.0008	0.3496	0.9977	0.5	0.0	0.9865

Table B.2: Metrics per variant and positive class weights from 7.9433 through 100.0.

Pos. class weight	BayesCal?	Proba?	TP	Pos. Est.	Bias	sPCC	Acc	BA	MCC	AUROC
7.9433	No	No	69.0	822.0	0.0253	0.2477	0.9741	0.9219	0.2477	0.9219
7.9433	No	Yes	69.0	2419.8884	0.0791	0.2407	0.9741	0.9219	0.2477	0.9867
7.9433	Yes	No	69.0	95.0	0.0009	0.2945	0.9961	0.6727	0.2945	0.6727
7.9433	Yes	Yes	69.0	94.1778	0.0008	0.3503	0.9977	0.5	0.0	0.9867
10.0	No	No	69.0	926.0	0.0288	0.2328	0.9706	0.9202	0.2328	0.9202
10.0	No	Yes	69.0	2550.7917	0.0835	0.2353	0.9706	0.9202	0.2328	0.9869
10.0	Yes	No	69.0	97.0	0.0009	0.2914	0.996	0.6727	0.2914	0.6727
10.0	Yes	Yes	69.0	96.7064	0.0009	0.3454	0.9977	0.5	0.0	0.9869
12.5893	No	No	69.0	1047.0	0.0329	0.2183	0.9665	0.9181	0.2183	0.9181
12.5893	No	Yes	69.0	2679.8906	0.0879	0.23	0.9665	0.9181	0.2183	0.9871
12.5893	Yes	No	69.0	102.0	0.0011	0.296	0.9959	0.6799	0.296	0.6799
12.5893	Yes	Yes	69.0	101.2718	0.0011	0.3445	0.9977	0.5	0.0	0.9871
15.8489	No	No	69.0	1159.0	0.0367	0.2142	0.9628	0.9308	0.2142	0.9308
15.8489	No	Yes	69.0	2806.8936	0.0921	0.2247	0.9628	0.9308	0.2142	0.9872
15.8489	Yes	No	69.0	108.0	0.0013	0.2876	0.9957	0.6798	0.2876	0.6798
15.8489	Yes	Yes	69.0	107.237	0.0013	0.3408	0.9977	0.5	0.0	0.9872
19.9526	No	No	69.0	1278.0	0.0407	0.2103	0.959	0.9433	0.2103	0.9433
19.9526	No	Yes	69.0	2932.3862	0.0964	0.2195	0.959	0.9433	0.2103	0.9873
19.9526	Yes	No	69.0	113.0	0.0015	0.3037	0.9957	0.6942	0.3037	0.6942
19.9526	Yes	Yes	69.0	112.4084	0.0015	0.3365	0.9977	0.5	0.0	0.9873
25.1189	No	No	69.0	1391.0	0.0445	0.2011	0.9552	0.9414	0.2011	0.9414
25.1189	No	Yes	69.0	3055.9536	0.1005	0.2145	0.9552	0.9414	0.2011	0.9874
25.1189	Yes	No	69.0	120.0	0.0017	0.3166	0.9956	0.7086	0.3166	0.7086
25.1189	Yes	Yes	69.0	119.3281	0.0017	0.3327	0.9977	0.5	0.0	0.9874
31.6228	No	No	69.0	1500.0	0.0482	0.1997	0.9516	0.9541	0.1997	0.9541
31.6228	No	Yes	69.0	3178.8474	0.1047	0.2095	0.9516	0.9541	0.1997	0.9875
31.6228	Yes	No	69.0	129.0	0.002	0.3159	0.9954	0.7157	0.3159	0.7157
31.6228	Yes	Yes	69.0	128.4711	0.002	0.3297	0.9977	0.5	0.0	0.9875
39.8107	No	No	69.0	1612.0	0.0519	0.1922	0.9479	0.9522	0.1922	0.9522
39.8107	No	Yes	69.0	3300.3225	0.1088	0.2048	0.9479	0.9522	0.1922	0.9876
39.8107	Yes	No	69.0	140.0	0.0024	0.3234	0.9951	0.7301	0.3234	0.7301
39.8107	Yes	Yes	69.0	139.3451	0.0024	0.3248	0.9977	0.5	0.0	0.9876
50.1187	No	No	69.0	1759.0	0.0569	0.1864	0.943	0.957	0.1864	0.957
50.1187	No	Yes	69.0	3420.4353	0.1128	0.2002	0.943	0.957	0.1864	0.9876
50.1187	Yes	No	69.0	152.0	0.0028	0.3396	0.9949	0.7516	0.3396	0.7516
50.1187	Yes	Yes	69.0	151.0227	0.0028	0.3202	0.9977	0.5	0.0	0.9876
63.0957	No	No	69.0	1881.0	0.061	0.1798	0.9389	0.9549	0.1798	0.9549
63.0957	No	Yes	69.0	3539.8123	0.1168	0.1957	0.9389	0.9549	0.1798	0.9877
63.0957	Yes	No	69.0	163.0	0.0032	0.3372	0.9946	0.7587	0.3372	0.7587
63.0957	Yes	Yes	69.0	162.8329	0.0032	0.3137	0.9977	0.5	0.0	0.9877
79.4328	No	No	69.0	2017.0	0.0656	0.1732	0.9343	0.9526	0.1732	0.9526
79.4328	No	Yes	69.0	3657.9644	0.1208	0.1914	0.9343	0.9526	0.1732	0.9877
79.4328	Yes	No	69.0	179.0	0.0037	0.3215	0.9941	0.7585	0.3215	0.7585
79.4328	Yes	Yes	69.0	178.011	0.0037	0.3051	0.9977	0.5	0.0	0.9877
100.0	No	No	69.0	2168.0	0.0706	0.1666	0.9292	0.9501	0.1666	0.9501
100.0	No	Yes	69.0	3774.9968	0.1247	0.1872	0.9292	0.9501	0.1666	0.9878
100.0	Yes	No	69.0	196.0	0.0043	0.3243	0.9936	0.7727	0.3243	0.7727
100.0	Yes	Yes	69.0	195.9161	0.0043	0.2958	0.9977	0.5	0.0	0.9878