UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Learning with invariants

Author:
Vladislav NIKOLOV

Supervisor:
Dr. Oriol PUJOL

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

June 30, 2022

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Learning with invariants

by Vladislav NIKOLOV

The current machine learning paradigms consider no further information apart from the data samples when trying to learn a model that can represent the data and be used later on to infer information about new samples. During the training process, the current methods try to minimize the error with respect to the original data, thus searching the function that best fits the data in an infinite space of functions. However, the training data has statistical information that can help reduce the searching space to a region of it, allowing also to find functions that better fit the data. A recently proposed data-driven learning paradigm called LUSI tries to take advantage of this information by preserving some invariants containing interesting statistical properties of the data. Despite offering good results, the invariants are problem dependent, prior knowledge of the problem is required when selecting the invariants and it is limited to preserve the information of only one class at the time. Hence, in this work we propose and study the use of random projections and random hyperplanes as general use invariants. We observe that the random projections offer results similar to the original invariants although are a bit limited, whereas the random hyperplanes fall quite short compared to the other types of invariants. Also, we successfully extend the LUSI paradigm to multiple classes using the Error Correcting Output Codes framework, enabling its use in multiclass classification problems.

Acknowledgements

First and foremost, I would like to thank Dr. Oriol Pujol for giving me the opportunity to work along with him on this project and supervising my work. Also, I would like to thank him for constantly encouraging me, giving me new ideas to try when I got stuck on some problem, for his constant support and for helping me with other academic stuff. It has been a real pleasure working along with him on this project from the beginning to the end of it.

Second, I would like to thank my parents for unconditionally supporting me and always motivating me to keep studying the things that I like.

I would also like to thank Pablo Álvarez, who had previously worked on a similar project and whose work helped me when I got stuck with some issues.

Next, I would like to thank my friends Álvaro, Arseniy and Noah for their constant help, support and motivation throughout this year. I feel very lucky for having the opportunity to work with them on different projects throughout the academic year.

Finally but not least, I would like to thank my dear friend Irene for all her help, motivation and unconditional support, as well as the tips and help that she provided me with to better present the results of this work and for constantly encouraging me to keep pushing forward while I was working on this thesis. And my dear friend Otis, for making me take things easier, for his unconditional help and support and for making my working environment much better in general. And to both of them, thank your for sticking with me while I was working on this document and for helping me with the experiments.

Contents

Chapter 1

Introduction

1.1 Motivation and brief description of the project

The current machine learning paradigms consider no further information apart from the data samples when trying to learn a model that can represent the data and be used later on to infer information about new samples. During the training process, the current methods try to minimize the error with respect to the original data, thus searching the function that best fits the data in an infinite space of functions. However, the training data has statistical information that can help reduce the searching space to a region of it, allowing also to find functions that better fit the data.

A new learning paradigm that takes into account the statistical information of the training data in the form of statistical invariants has been recently proposed by Vapnik and Izmailov, 2019. Thanks to it, statistical information of the problem can be used in the learning process, which might be overlooked by most of the models because some relationships between variables are hard to spot or require prior knowledge that the model does not have access to. Nonetheless, it seems that this learning paradigm has not been fully explored or applied that much in practice.

Therefore, in this work we would like to further explore the possible applications of this paradigm and whether it can be made more general, without requiring prior knowledge of the problem.

1.2 Goals and objectives

In this thesis we aim to (i) understand and further explore the application of the invariants in the learning problem, (ii) propose new invariants that require no previous knowledge of the problem and thus can be applied to multiple domains, (iii) automatize the selection process of the most suitable invariants for a new problem and (iv) extend the learning paradigm so that it can be applied to multiclass classification problems.

In order to accomplish these goals, we propose a series of milestones that must be achieved first:

1. Understand the original paper and reproduce it, which implies implementing the proposed algorithms for learning with statistical invariants and reproducing some of the experiments and results. Because there is no source code available, we have to start from scratch.

2. Propose new invariants and apply them to the same problems as the ones in the paper to get an initial idea of how they work.

3. Build a wrapper around the previously defined binary classifier to enable its use in multiclass classification problems.

4. Experiment with multiclass classification problems to test the proposed invariants and compare them to a baseline to see whether they are actually helping or not during the learning process.

The expected outcome of this work is a small software package that contains a machine learning model that can be applied to classification problems.

1.3 Brief summary of the results

With the completion of this work, we have been able to achieve the following objectives:

- We have proposed new invariants that can be applied to different problems with no prior knowledge required.
- We have extended the LUSI paradigm to work with multiclass problems.
- We have created a small software module containing the extended version of LUSI as a machine learning model. This model has the same interface as a machine learning model from `scikit-learn`, which makes it fully compatible with the functionality provided by this module.

However, we have also noted that we did not have that much of a success in the following aspects:

- We were not able to automatize the selection process of the most suitable invariants for a problem. Thus, it still remains an open question.
- As we will see in later chapters, the invariants that we have proposed have not achieved the best results. The random projections were able to achieve similar but slightly worse results compared to the original invariants, and the random hyperplanes seem to provide little to no relevant information.

Nevertheless, we have been able to better understand the learning with invariants problem and made some contributions on the topic. All of the developed code and experiments can be found in this GitHub repository.

1.4 Layout

This thesis is structured as follows:

- Chapter 1 introduces this work, presenting the main goals that are expected to be achieved by the end of it and briefly discussing the obtained results.
- Chapter 2 explains the background work that has inspired this project, showing its main contributions and results.
- Chapter 3 proposes a series of invariants that aim to be more general and easy to apply to different problems and a method to expand this paradigm to multiclass classification problems.
- Chapter 4 studies how the proposed invariants and methods work in practice and what results can be achieved with them.
- Chapter 5 briefly discusses what conclusions can be drawn from this work and what future work can be done on the topic.

Chapter 2

Learning using statistical invariants

Given that this work intends to explore the applications of the invariants in the learning process, we first need to introduce the background work that proposed this new learning paradigm, which is called LUSI (Learning Using Statistical Invariants).

This chapter intends to provide the necessary background to understand the basis of this work and an overview of the most relevant aspects of the original paper that presented the LUSI paradigm, which was proposed by Vapnik and Izmailov, 2019. For further information and more details, please refer to the original paper.

2.1 Weak convergence and the LUSI paradigm

Supervised machine learning algorithms try to find the best estimate of some conditional probability function $P(y|x)$, i.e., given a data point x, we want to compute the probability that this point belongs to a particular class y.

Classical methods do this by using the strong mode of convergence in the Hilbert space. However, in the LUSI paradigm this estimation is obtained using the weak mode of convergence. Hence, it is important to understand the difference between this two modes of convergence and what role the weak mode of convergence plays in the LUSI paradigm.

2.1.1 Strong and weak modes of convergence

In a Hilbert space, the relationships between two functions $f_1(x)$ and $f_2(x)$ have two numerical properties:

1. The distance between functions

$$\rho(f_1, f_2) = \|f_1(x) - f_2(x)\|$$

   that is defined by the metric of the L_2 space and

2. The inner product between functions

$$R(f_1, f_2) = \langle f_1(x), f_2(x) \rangle$$

   that has to satisfy the corresponding requirements.

These two properties imply two different modes of convergence: a strong one and a weak one. Classical learning paradigms rely on the strong convergence mode (convergence in metrics), trying to find a sequence of functions $\{P_l(y=1|x)\}$[1] such that

$$\lim_{l\to\infty} \|P_l(y=1|x) - P(y=1|x)\| = 0 \quad \forall x$$

The weak mode of convergence (convergence in inner products) is given by

$$\lim_{l\to\infty} \langle P_l(y=1|x) - P(y=1|x), \psi(x)\rangle = 0 \quad \forall \psi(x) \in L_2$$

Note that this mode of convergence has to take place for *all* functions in the Hilbert space L_2.

It is known that the strong mode of convergence implies the weak one, although generally speaking, the reverse is not true.

2.1.2 The LUSI paradigm

Opposite to the classical learning paradigms, LUSI is based on the weak mode of convergence. It replaces the infinite set of functions with a set of functions $\mathcal{P} = \{\psi_1(x), \dots, \psi_m(x)\}$ called predicates, which describe some important properties of the desired conditional probability function and restrict the scope of weak convergence only to the set of functions $\mathcal{P}$. These properties are called invariants, and can be expressed as the following equalities:

$$\int \psi_s P(y=1|x)dP(x) = \int \psi_s dP(y=1,x) = a_s, \quad s = 1, \dots, m$$

where a_s is the expected value of the predicate $\psi_s(x)$ with respect to measure $P(y=1,x)$. These values are unknown but can be estimated using the training data $\{(x_i, y_i),\ i = 1, \dots, l\}$. Therefore, the previous expression can be rewritten as follows:

$$\frac{1}{l}\sum_{i=1}^{l} \psi_s(x_i) P_l(y=1|x_i) \approx a_s \approx \frac{1}{l}\sum_{i=1}^{l} y_i \psi_s(x_i), \quad s = 1, \dots, m \tag{2.1}$$

Simply put, the general idea of the LUSI paradigm is to find an approximation $P_l(y=1|x)$ of the real conditional probability function in the subset of functions that preserve the invariants associated to the set of predicates $\mathcal{P}$, reducing effectively the set of candidate functions to those that satisfy (2.1).

2.1.3 Predicate selection

In order to find this approximation of the conditional probability function, there must exist some kind of mechanism that allows us to determine which invariants should be used. Luckily, the authors propose a very simple way to sequentially selecting invariants. Given an approximation $P_l^m(y=1|x)$ using m invariants and a new predicate ψ_{m+1} which we would to know whether it should be considered or not. We can compute the following value before adding it:

$$\mathcal{T} = \frac{\left|\sum_{i=1}^{l} \psi_{m+1}(x_i) P_l^m(y=1|x_i) - \sum_{i=1}^{l} y_i \psi_{m+1}(x_i)\right|}{\sum_{i=1}^{l} y_i \psi_{m+1}(x_i)} \tag{2.2}$$

[1]We focus here in the binary problem setting, i.e. $y \in \{0,1\}$. Thus, $P_l(y=1|x)$ fully specifies the output probability distribution.

If $\mathcal{T} \geq \delta$ for some small threshold δ, the new invariant defined by predicate ψ_{m+1} is considered. Otherwise, the expression (2.1) is treated as an equality and the invariant is not considered in the approximation.

2.2 Statistical invariants

A *statistical invariant* is a specific realization of a predicate with statistical meaning. This means that it captures some sort of statistical information of the data that has to be conserved when selecting the best approximation of the conditional probability function.

There are different types of statistical invariants, each one of them providing different information about the data. In this case, the authors have considered two in particular: the zeroth order and first order moments of the conditional probability function $P(y = 1|x)$. We will briefly discuss each one of them and see what kind of information they provide.

2.2.1 Zeroth order invariant

Suppose that we are given a binary classification problem, in which the positive instances are labeled as 1 and the negative ones as 0. The zeroth order invariant would give us information about the ratio of elements of the positive class. It is defined as follows:

$$\psi_{z.o.}(x) = 1$$

The logic behind it is the following: the predicate is applied to each single sample in the dataset, which will yield the vector $(1, \dots, 1) \in \mathbb{R}^l$, where l is the number of samples. Taking into account expression (2.1), we can see that each element of this vector is multiplied by the predicted labels (left side) and the true labels (right side). These values are summed and then divided by l, which gives us the proportion of positive predicted elements on the left side and the proportion of true positive elements on the right side. Notice that the invariant is only taken into account for those elements whose predicted or true label is positive. Thus, the negative samples are not considered.

2.2.2 First order invariant

Suppose the same case scenario as in the previous subsection. If we apply the first order invariant to a dataset, we would get the mean or centroid of the positive class. Its mathematical expression is:

$$\psi_{f.o.}(x) = x$$

Same as before, when this predicate is applied to the dataset it will generate the vector $(x_1, \dots, x_l) \in \mathbb{R}^l$. Following expression (2.1) again, only the positive true or predicted elements will be considered. Therefore, their values will be summed and then averaged, yielding indeed the centroid of the positive class (both for the predicted and true labels).

2.3 Solving the learning problem

The authors show that in a specific type of Hilbert space called Reproducing Kernel Hilbert Space (RKHS) the estimate of the conditional probability function can be computed as

$$f(x) = A^T \mathcal{K}(x) + c$$

where $A \in \mathbb{R}^l$ is a vector of coefficients, $\mathcal{K}(x) = (K(x_1, x), \ldots, K(x_l, x))^T$ is a vector of functions determined by the kernel associated to the RKHS[2] and evaluated on the training data, and $c \in \mathbb{R}$ is the bias term.

Additionally, let $Y = (y_1, \ldots, y_l)$ be the labels of the training set, $K \in \mathbb{R}^{l \times l}$ the matrix with elements $K(x_i, x_j),\ i, j = 1, \ldots, l$, $\Phi_s = (\psi_s(x_1), \ldots, \psi_s(x_l))^T$ the vector obtained from evaluating the l points of the sample using predicate ψ_s, $1_l = (1, \ldots, 1) \in \mathbb{R}^l$ a vector of ones and $V \in \mathbb{R}^{l \times l}$ a matrix called the V-matrix, proposed by Vapnik and Izmailov, 2015, which captures some geometric properties of the data[3].

With all of this information, we can formulate and solve a minimization problem subject to the constraints (2.2) which has a closed-form solution. Using its Lagrangian, we can obtain that the coefficients A are given by:

$$A = (A_V - cA_c) - \left(\sum_{s=1}^{m} \mu_s A_s \right)$$

where

$$A_V = (VK + \gamma I)^{-1} VY$$
$$A_c = (VK + \gamma I)^{-1} V 1_l$$
$$A_s = (VK + \gamma I)^{-1} \Phi_s, \quad s = 1, \ldots, n$$

In this case, γ controls the amount of regularization applied so that the resulting matrix is not singular.

The values of c and the m coefficients μ_s can be obtained solving the following system of equations:

$$c[1_l^T VKA_c - 1_l^T V 1_l] + \sum_{s=1}^{m} \mu_s [1_l^T VKA_s - 1_l^T \Phi_s] = [1_l^T VKA_V - 1_l^T VY]$$
$$c[A_c^T K \Phi_k - 1_l^T \Phi_k] + \sum_{s=1}^{m} \mu_s A_s^T K \Phi_k = [A_V^T K \Phi_k - Y^T \Phi_k], \quad k = 1, \ldots, m$$

This algorithm, which uses m invariants, is called vSVM&I$_m$ if the V-matrix is used or SVM&I$_m$ in case it is not.

[2]In this case, the authors have considered the Gaussian Kernel, which is defined as

$$K(x, x') = \exp\{-\delta \|x - x'\|^2\},\ \delta > 0$$

[3]The most simple case considers that the V-matrix is equivalent to the identity matrix. Also, the authors found that using the V-matrix over the identity matrix didn't improve the results that much, but it was rather the use of invariants that made the difference. For the sake of consistency, we are going to keep the V-matrix in the expressions as this is how they are supposed to be written, although bear in mind that it can be substituted with the identity matrix.

2.4 Overview of the LUSI algorithm

Following the mathematical formulations from the previous sections, let us now present an overview of the LUSI algorithm.

Consider the following learning method given a training sample $(x_i, y_i),\ i = 1, \dots, l$ and $\psi_k(x),\ k = 1, \dots, m$ predicates:

Step 1: Construct SVM or vSVM estimate of conditional probability as described in section 2.3, without considering the predicates.

Step 2: Find the maximal disagreement value $\mathcal{T}_s$ as defined in (2.2) for vectors

$$\Phi_k = (\psi_k(x_1), \dots, \psi_k(x_l))^T, \quad k = 1, \dots, m$$

Step 3: If $\mathcal{T}_s > \delta$, add the invariant associated to the predicate ψ_s; otherwise stop.

Step 4: Find a new approximation of the conditional probability function and go back to **Step 2**; otherwise stop.

2.5 Main results and limitations

According to the original work, LUSI yields quiet good results overall, reducing the error and thus improving the accuracy of the models that use it compared to models that do not use statistical invariants. Moreover, the authors state that it can reduce the number of necessary training examples to obtain a good approximation of the conditional probability function. Therefore, this method could be very useful in cases in which the amount of available training data is small.

However, this new learning paradigm presents some important flaws:

1. The selected invariants are problem dependent, which means that it is hard to have general invariants that can both be applied to multiple problems without requiring any kind of previous knowledge and yield good results.

2. Often, the invariant selection is a "black-art" due to the fact that they can either be very esoteric or require a lot of knowledge about the specific problem which might be hard to obtain. Hence, some craftsmanship is required when selecting the invariants that are going to be used.

3. Considering expressions (2.1) and (2.2), we can clearly see that the invariants can only consider the statistical information of the positive class. The way that the values are computed make it hinder the application of the invariants to the negative class and to multiclass classification problems since there is no positive class in this kind of scenarios. This a very serious drawback of this method that needs to be further addressed.

Thus, even though the use of invariants can improve the obtained results, they also introduce some additional complexity to the task because they require extra knowledge that may not be accessible.

Chapter 3

Proposals

As we have already seen in Chapter 2, LUSI introduces a new data-driven learning paradigm which aims to find better approximations of the conditional probability function using statistical invariants. However, these invariants are often problem dependent and choosing the appropriate ones is not an easy task, as there are many possible ones and some of them are not straightforward to come up with. Also, the paradigm does not consider statistical information of the negative class in binary problems and has to be extended so that it can be applied to multiclass problems.

Thus, our main goal with this work is to create a series of new invariants which we expect to be of general use, as well as automatizing the selection process of the invariants that should be considered for a given problem and extending the LUSI paradigm to multiclass problems[1].

In this chapter we are going to present our proposals, which are a two new invariants based on randomness which aim to be more general than the original ones and an extension of the LUSI paradigm to multiclass problems using Error Correcting Output Codes (ECOC).

3.1 Random invariants

Our first proposal is a series of random invariants, which are invariants that have some sort of random process inside of them but that aim to preserve some sort of statistical information of the data. This way, we aim to greatly reduce the amount of necessary prior knowledge of the problem when choosing which invariants to use, treating it just like any other hyperparameter of a machine learning model. In this work, we propose an invariant based on random projections as well as another one based on random hyperplanes.

3.1.1 Random projections

Random projections have been frequently used in the machine learning field to perform dimensionality reduction in a faster and computationally less expensive way than other techniques (i.e., PCA) as studied in Dasgupta, 2000 and Bingham and Mannila, 2001.

In this case, the random projection invariant projects the data into a new one-dimensional space, as if it was viewed from a particular point in the original space. In some way, it is like having different compressed views of the same data. Intuitively, this can be observed in figure 3.1.

[1]Even though in the original paper the authors state that the method can be applied to multiclass problems, they do not go into too much detail of how this is done.

More formally speaking, consider a data point $x \in \mathbb{R}^d$ and a projection vector $p \sim \mathcal{N}(\mu, \Sigma)$, where the multivariate normal distribution has mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$. We can define the random projection invariant as follows:

$$\psi_{r.p.}(x) = xp \tag{3.1}$$

As we can see in expression (3.1), we are computing the dot product between the data point and the projection vector. When this invariant is used in expression 2.1 it will try to preserve the centroid of the positive class in the new projected space. Hence, it is a variation of the first order invariant. When using multiple random projections as invariants, we expect that the centroid of the positive class is preserved across different views of the data.

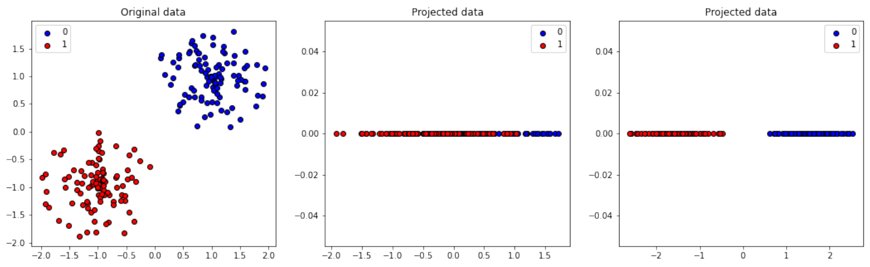

FIGURE 3.1: Example of a dataset and two random projections of it. In the first projection, we can see that the points from the positive and the negative classes are almost totally overlapped, whereas in the second projection they are completely separable.

3.1.2 Random hyperplanes

The second invariant that we propose is the random hyperplane. With it, we aim to create two partitions of the original data: the samples that are on the right side of the hyperplane and the ones on the left, which we will consider as the positive and the negative samples, respectively. Opposite to the previous invariant, which produced a real value when applied to a point x, this one produces a discrete value $\{0, 1\}$ based on the relative position of the point with respect to the normal vector of the hyperplane. Figure 3.2 shows an example of how this invariant works when applied to an example dataset.

Formalizing the previous explanation, consider an arbitrary point from the sample $x_c \in \mathbb{R}^d$. Let $n \sim \mathcal{N}(\mu, \Sigma)$ be the normal vector of the hyperplane that contains x_c, where the multivariate normal distribution has mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$. We can define the random hyperplane invariant as

$$\psi_{r.h.}(x) = \begin{cases} 1 & \text{if } (x - x_c)n \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Considering the previous expression, we can clearly see that this invariant will yield a vector of zeroes and ones when applied to a dataset. If we also take into account expression (2.1), we can deduce that this invariant will try to preserve the proportion of positive samples that fall on the right side of the hyperplane. Consequently, we can see that this is a variation of the zeroth order invariant. The main difference is that we are now trying to preserve the proportion of positive elements in a subspace of the original space (the subspace formed by the samples that are on the same side as the normal vector of the hyperplane), instead of trying to preserve it in the whole space.

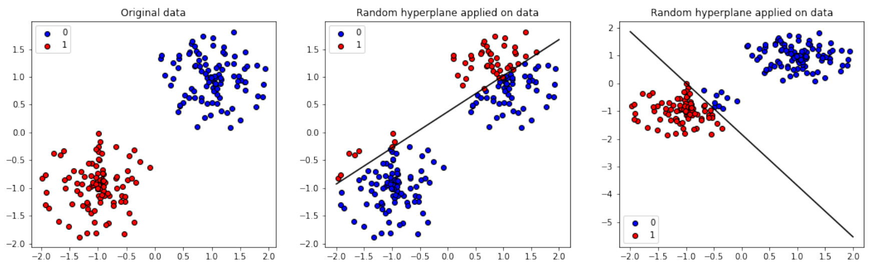

FIGURE 3.2: Example of the hyperplanes invariant. In the left image, we can see the original data. In the middle and right images we can see two hyperplanes that divide the data in elements which are on the right and the left of the hyperplane, labeled as 1 and 0 respectively. Note that these new labels do not necessarily match the original ones.

3.1.3 The LUSI algorithm with random invariants

Because we are introducing some random behavior in the LUSI algorithm, the previously defined algorithm has to be adapted so that it can operate with these new random invariants, especially the invariants selection process.

Algorithm 1 Modified version of the LUSI algorithm considering random invariants

```
1: function RandomInvariantsLUSI($N$, $M$, $tolerance$, $\delta$)
2:     Compute initial values of $A$ and $c$ without considering invariants
3:     $invariants \leftarrow$ EmptyArray()
4:     $tries \leftarrow 0$
5:     while length($invariants$) $< N$ and $tries < tolerance$ do
6:         $random\_invariants \leftarrow$ GenerateRandomInvariants($M$)
7:         Compute maximal disagreement value $\mathcal{T}_s$ of the invariants using 2.2
8:         if $\mathcal{T}_s > \delta$ then
9:             Add invariant $s$ to $invariants$
10:            $tries \leftarrow 0$
11:            Compute new approximations of $A$ and $c$ with equations from 2.3
12:        else
13:            $tries \leftarrow tries + 1$
14:        end if
15:    end while
16: end function
```

Suppose that we want to train a classifier using up to N random invariants. At each step, we can generate M random invariants of the same type (random projections or hyperplanes) and evaluate them using expression (2.2), selecting the one with the maximal value $\mathcal{T}_s$. If this value is greater than the value of δ, then the invariant is selected. Otherwise, we would repeat the previous process. Note that due to the stochasticity of the invariants generation process, we could end up in an infinite-loop if none of them have a greater value than δ. Thus, we introduce a parameter *tolerance* to control the number of times that we want this process to be repeated. Algorithm 1 offers an overview of this modified version of the LUSI algorithm.

3.2 Extending the LUSI paradigm to multiclass problems

As we have seen up until now, this learning paradigm can be easily applied to binary classification problems and solve them. Nevertheless, note that the invariants are defined specifically for just one of the classes. This ignores the invariant information that could be gathered by considering the other class. Moreover, in the most general setting of multiclass classification, the algorithm has to be extended. So, instead of restraining ourselves to a particular kind of problems, it seems reasonable trying to extend LUSI to multiclass problems so that we have a wider application scope.

One framework that can be used to easily deal with this kind of problems is Error Correcting Output Codes (ECOC), introduced by Dietterich and Bakiri, 1995. We can use it to treat multiclass classification problems as if they were binary. For each one of the N_c classes we create a codeword of length n. These codewords can be arranged as the rows of a matrix called the *coding matrix* $M \in \{0,1\}^{N_c \times n}$. Each column of the matrix is treated as an individual binary classification problem and a binary classifier is trained for each one of these problems using the corresponding encoding (the columns).

An example of this can be seen in tables 3.1 and 3.2, which show two different coding matrices for the same 5-class classification problem. The first one encodes the problem as five one-against-all problems, which means that in each problem there will be only one class that can be considered the positive one. In the second matrix, we can appreciate that some classes are encoded more than once as the positive class.

	h_1	h_2	h_3	h_4	h_5
C1	1	0	0	0	0
C2	0	1	0	0	0
C3	0	0	1	0	0
C4	0	0	0	1	0
C5	0	0	0	0	1

TABLE 3.1: Coding matrix for a multiclass classification problem with 5 classes and codewords of length $n = 5$. This matrix defines a special coding called one-against-all.

	h_1	h_2	h_3	h_4	h_5
C1	1	1	1	0	0
C2	0	1	1	0	1
C3	1	0	0	1	0
C4	0	1	0	1	0
C5	0	0	1	0	1

TABLE 3.2: Another coding matrix for the same 5-class classification problem.

Suppose that we have already trained a binary classifier for each one of the problems and we want to predict the class for a given data point x. Let us denote $f(x) = (f_1(x), \ldots, f_n(x))$ the vector containing the predictions for each one of the problems. In order to obtain the corresponding output class, we would need to decode this vector of predictions using the coding matrix that we have defined for the problem. This is usually done by using some kind of distance metric and selecting the class with the closest codeword according to the selected metric.

There are many distance metrics that we could use: Hamming distance, Euclidean distance, etc. In this particular case, we have considered the Euclidean distance, which we will denote as d_E and is computed as $d_E(x, x') = \|x - x'\|_2$. Let $M_r \in M$ be the r-th row of the coding matrix We can express the predicted class $\hat{y}$ as

$$\hat{y} = \underset{r}{\operatorname{argmin}}\, d_E(M_r, f(x)) \tag{3.3}$$

Thus, the predicted class will be the one associated to the row that is closest to the vector of predictions. Something important that we have to consider is that the binary classifiers for each problem should return the probability $P(y = 1|x)$ rather than the actual label, which means that no threshold should be applied during the prediction stage. This is because using the label directly might cause some errors when computing the closest codeword. For example, consider the coding matrix defined in 3.1 and that we are given a vector of predictions $(1, 1, 0, 0, 0)$ whose associated vector of probabilities is $(0.8, 0.55, 0.4, 0.1, 0.25)$. If we consider the binary vector of predictions, then both class 1 and class 2 are the closest ones, which will lead to randomly selecting among them and to a potential missclassification since class 1 is the most probable 1. However, if we consider the vector of probabilities, then the closest class in this case will be class 1, which is indeed the most probable one.

In consequence, in order to extend LUSI to multiclass classification problems we have to follow these steps:

Step 1: Create the coding matrix M of size $N_c \times n$.

Step 2: Use the columns of M to generate new binary labels for each one of the n binary classification problems.

Step 3: Train n binary classifiers that use LUSI with the previously generated binary labels.

Step 4: Given an input data point x, predict the vector of probabilities using the previous binary classifiers.

Step 5: Select the output class using expression (3.3).

Chapter 4

Experimentation and results

After having explained the necessary theoretical background and presented the proposals of this work, it is time that we put them into action to see how they perform. First, we are going to test our proposals on some toy datasets in order to better understand them. After that, we are going to try the proposed invariants as well as the multiclass extension with ECOC on real data to see how well they perform when comparing them to the original invariants proposed in Vapnik and Izmailov, 2019.

4.1 Experimenting with toy problems

In this section we will perform a set of simple experiments with some toy datasets. In this case, we have considered the circles and moons datasets, which can be seen in figure 4.1. Both of these problems are available in `scikit-learn`, which means that we will be able to easily generate our custom datasets with the available functions. When experimenting with these problems we aim to:

- Compare the original invariants with the ones that have been proposed in this work.
- Compare the original version of the LUSI algorithm with the ECOC version to see if there is any significant difference in a binary setting.
- Discover whether some types of invariants are more likely to be selected when considering all types of invariants.

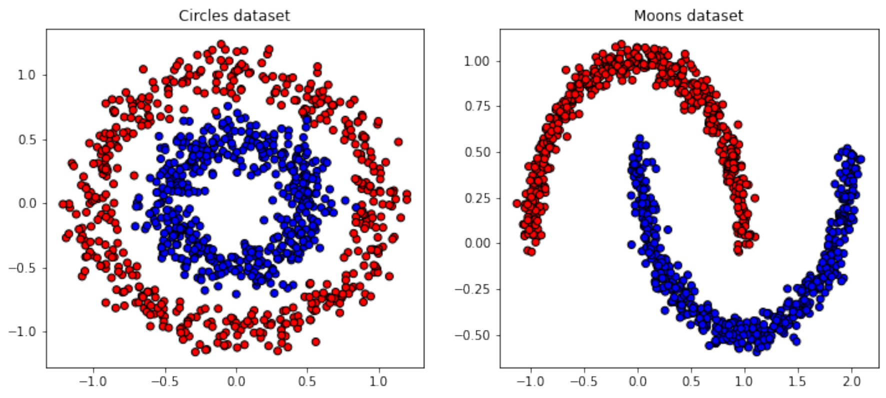

FIGURE 4.1: Toy datasets used in the experimentation.

4.1.1 Experimental settings

For this experimentation, we have generated one dataset for each problem with a fixed seed. The experimental parameter settings for each type of problem can be seen in table 4.1. In an experiment, we split the data in training and test, keeping 10% of the data in the training partition and the reamining 90% in the test partition. We have performed these experiments using Vapnik's invariants and the ones that we have proposed. As for Vapnik's invariants, we have considered the zeroth and first order invariants, which are

$$\psi_0(x) = 1, \quad \psi_1(x) = x_1, \quad \psi_2(x) = x_2$$

Problem	Parameter	Value
Circles	`n_samples`	1000
	`noise`	0.1
	`factor`	0.5
Moons	`n_samples`	1000
	`noise`	0.05

TABLE 4.1: Parameter settings of the toy problems.

4.1.2 Comparing the different invariant types and versions of LUSI

In order to compare the two versions of the LUSI algorithm and the different invariants we have trained a model for each version of LUSI and each invariant type on both problems with a fixed initial random state. We have visualized the decision boundaries in each case in order to see if there is any significant difference between them. Additionally, for the LUSI version we have reported the mean number of selected invariants of each type by repeating the experiment with 10 different initial states. We have limited the maximum number of invariants of each model to 3 because even though we can generate infinite new random invariants, we cannot do the same with the original invariants.

Figures 4.2 and 4.3 show the decision boundaries generated by the original version of the LUSI algorithm whereas figures 4.4 and 4.5 display the decision boundaries of the ECOC version. For the circles problem, we can see that all invariants produce very similar decision boundaries. In the case of ECOC version, this decision boundary seems to be much closer to the points of the blue class than in the case of the original algorithm, where the decision boundary is a bit wider. As for the moons problem, we can observe that the decision boundary is not perfect in any of the versions of the algorithm since some points fall into the region of the opposite class, probably caused by to the geometric shape of the both classes. For this problem, the decision boundary seems to be more accurate using the original algorithm. This is especially true in the case of the random hyperplanes, where the decision boundary is better adjusted in the case of the original algorithm, whereas it seems that it has been overfitted when using the ECOC algorithm because we can observe some "decision islands" for the blue class. Overall, it seems that Vapnik's invariants and the random projections produce very similar results regardless of which version of the algorithm is applied. Hence, if we use any of these two invariants, we could apply any version of the algorithm and get similar results in a binary classification problem. In the case of the random hyperplanes, there would be some difference between the results obtained with each version of the algorithm. As we have seen, depending on the problem, we could get similar results to the ones obtained using the other two types of invariants.

The idea that Vapnik's invariants and the random projections are similar can be further explored. Figure 4.6 show how many invariants have been selected for each problem on average. We can observe that the mean number of selected invariants is the same for Vapnik's invariants and the random projections, whereas the number of selected invariants is equal to the maximum number of invariants in the case of the random hyperplanes. Thus, it seems that the number of invariants that can be chosen when using Vapnik's invariants and the random projections is limited by the number of dimensions of the data, as selecting more does not provide any new information. In the case of the random hyperplanes, this is generally not true as we could keep adding more invariants of this type. This might be caused by the fact that there is a very large number of hyperplanes that separate the data in two partitions and that can be used to preserve the proportion of elements that fall on the right side of the hyperplane. Because of this, many invariants of this type can be selected.

4.1.3 Exploring the bias towards certain types of invariants

Now, we would like to study the scenario in which all types of invariants are considered to see if there is any kind of bias towards particular types of invariants. For this purpose, we have run a similar experiment to the previous one using the original version of the LUSI algorithm. We have fit 10 models on both problems using different random states and setting the maximum number of invariants to 50. In each experiment, the model could choose among all of the invariant types. For each run, we have computed how many invariants of each type were selected.

A summary of the results can be seen in figure 4.7, where the number of selected invariants has been averaged for each problem. We can see that the models have not selected any of Vapnik's invariants. On average, they have chosen 2 random projections per problem, which once again matches the number of dimensions that these problems have. The models have selected 48 random hyperplanes on average per problem, which gives more strength to the hypothesis that the number of hyperplane invariants that can be selected is very large, potentially infinite. Because of this, we can see that when considering all types of invariants at the same time, it is more likely that the hyperplanes invariant will be selected because it can contribute with more information.

4.2 Experimentation on real data

After testing our proposals on some toy problems and comparing them to the original work, it is time that we put them into action on real problems. With this experimentation we want to study the quality of the new invariants compared to Vapnik's proposals to see if we have achieved our goal of creating general purpose invariants that can be applied to multiple problems and that allow us to obtain quality results. In this section, we are going to present the experiments that we have performed as well as the obtained results.

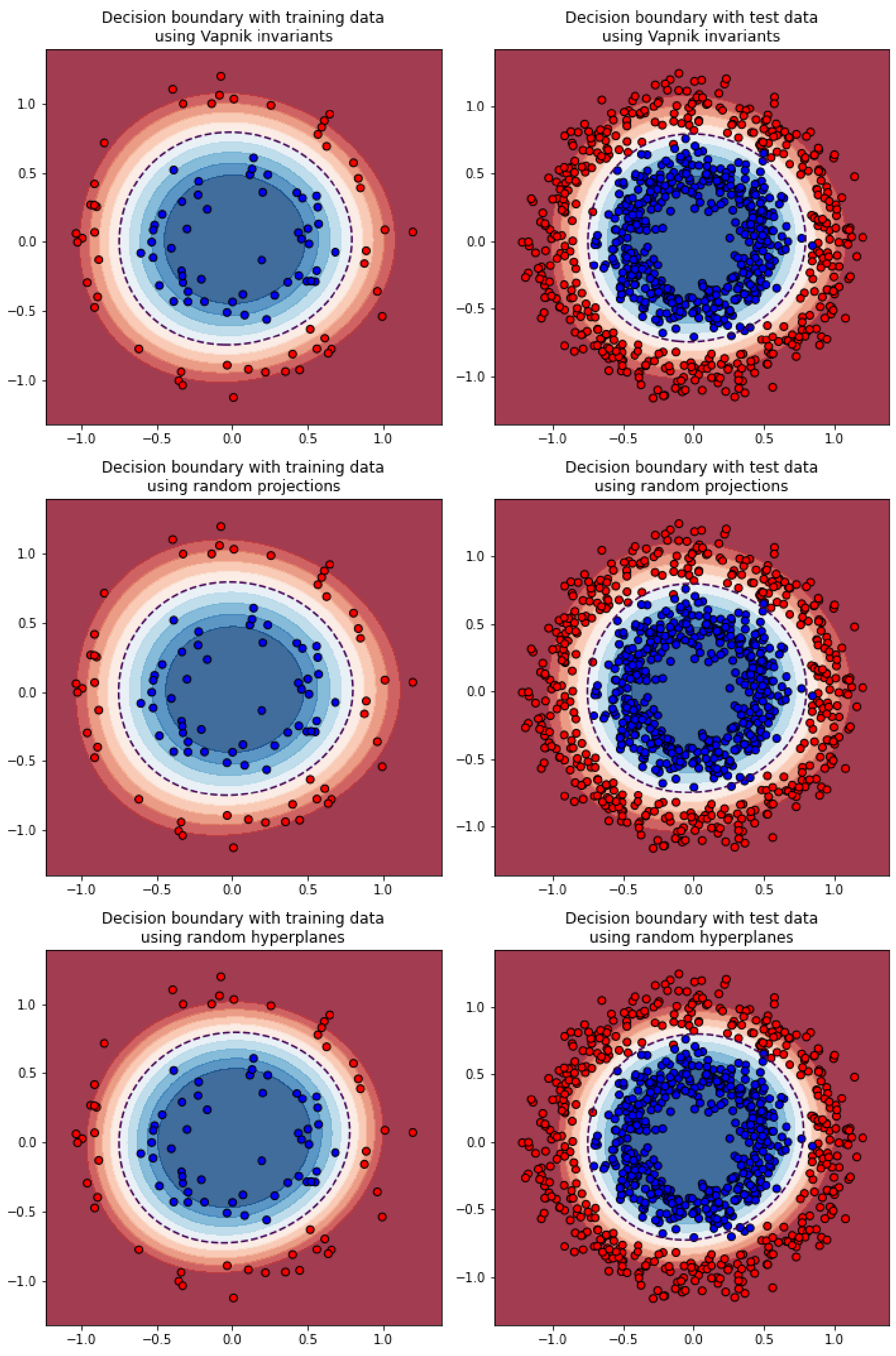

FIGURE 4.2: Decision boundaries in the circles problem using the original LUSI algorithm with each type of invariant on the training and test sets.

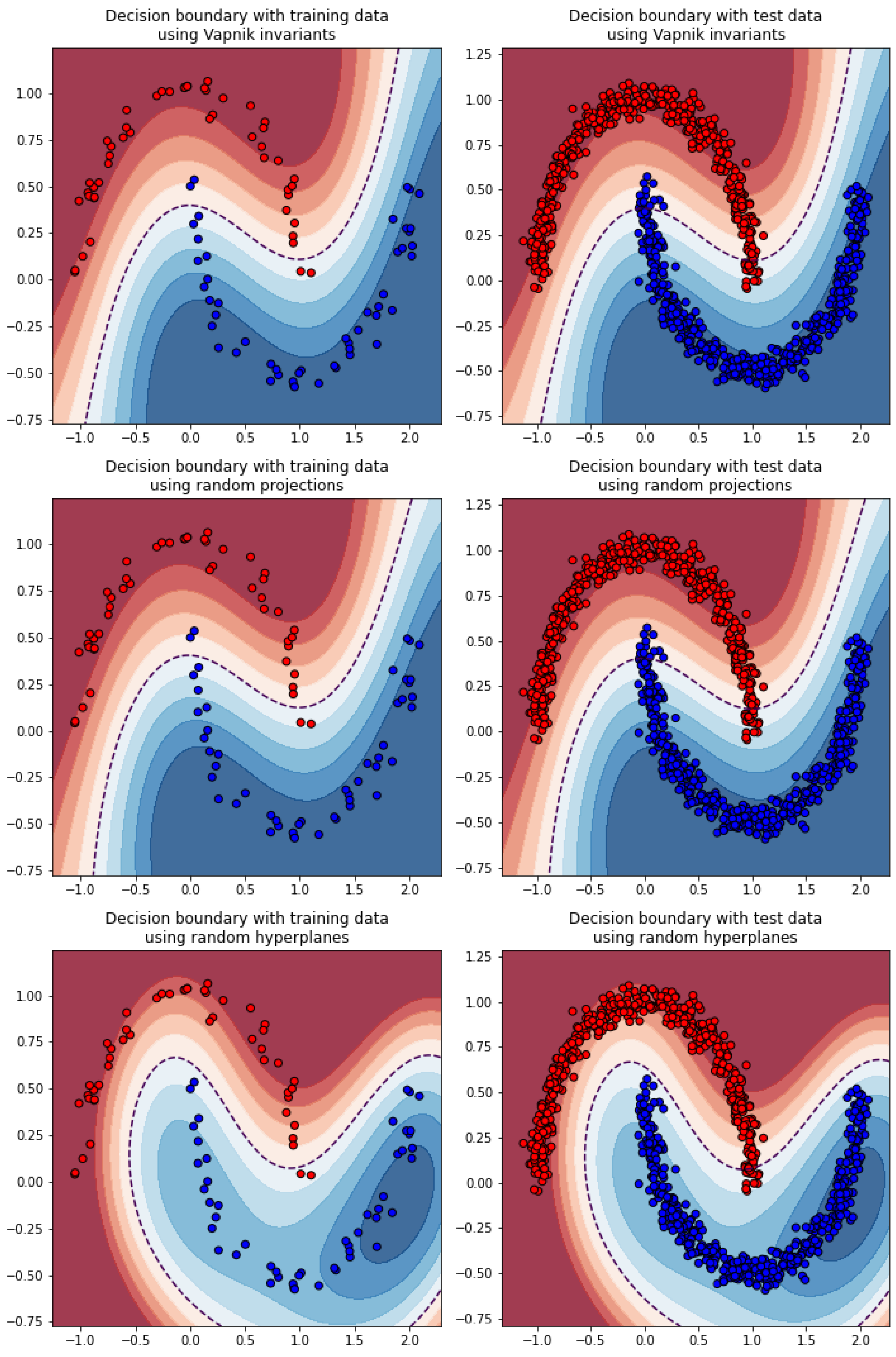

FIGURE 4.3: Decision boundaries in the moons problem using the original LUSI algorithm with each type of invariant on the training and test sets.

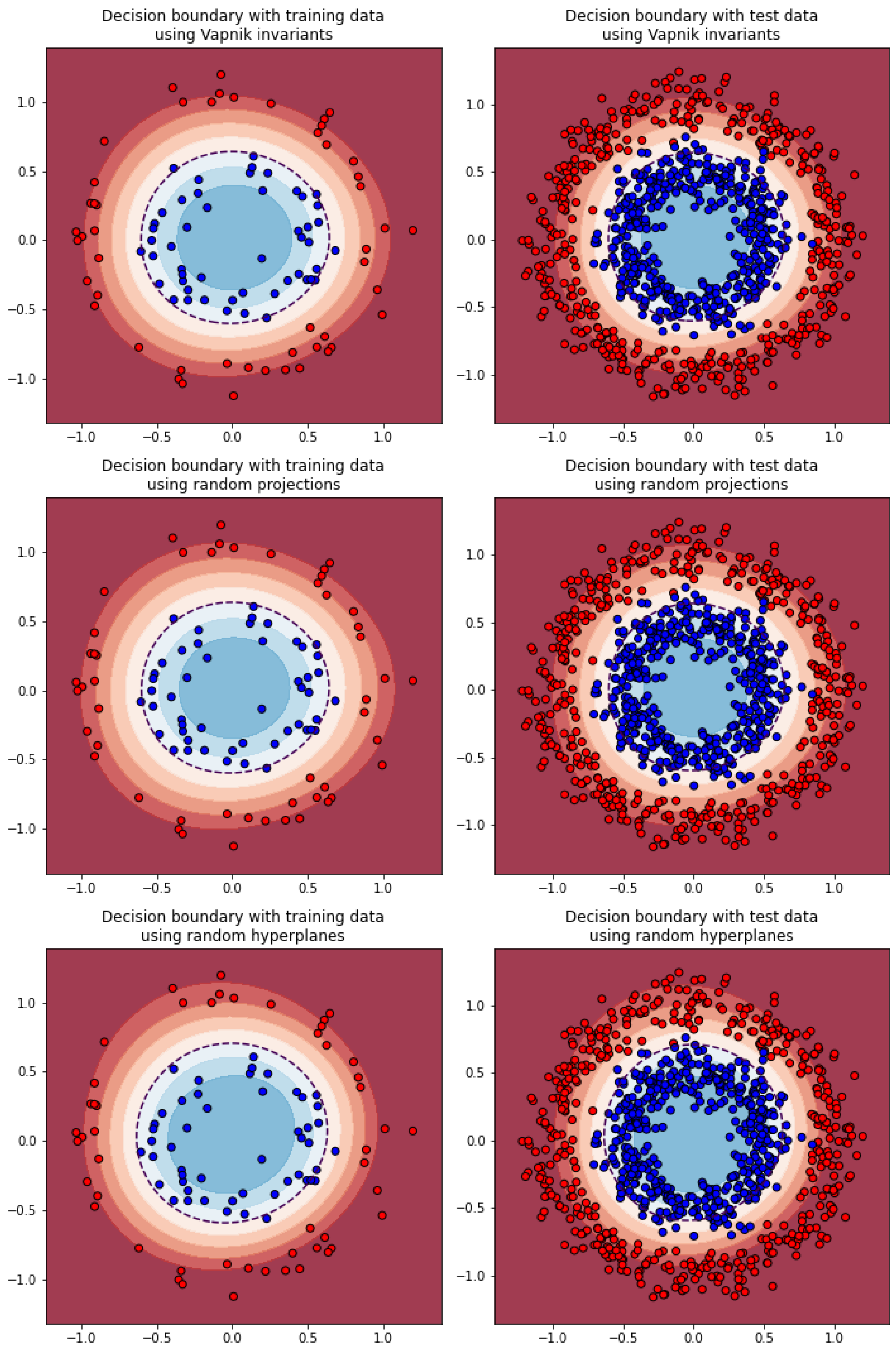

FIGURE 4.4: Decision boundaries in the circles problem using the ECOC version of the LUSI algorithm with each type of invariant on the training and test sets.

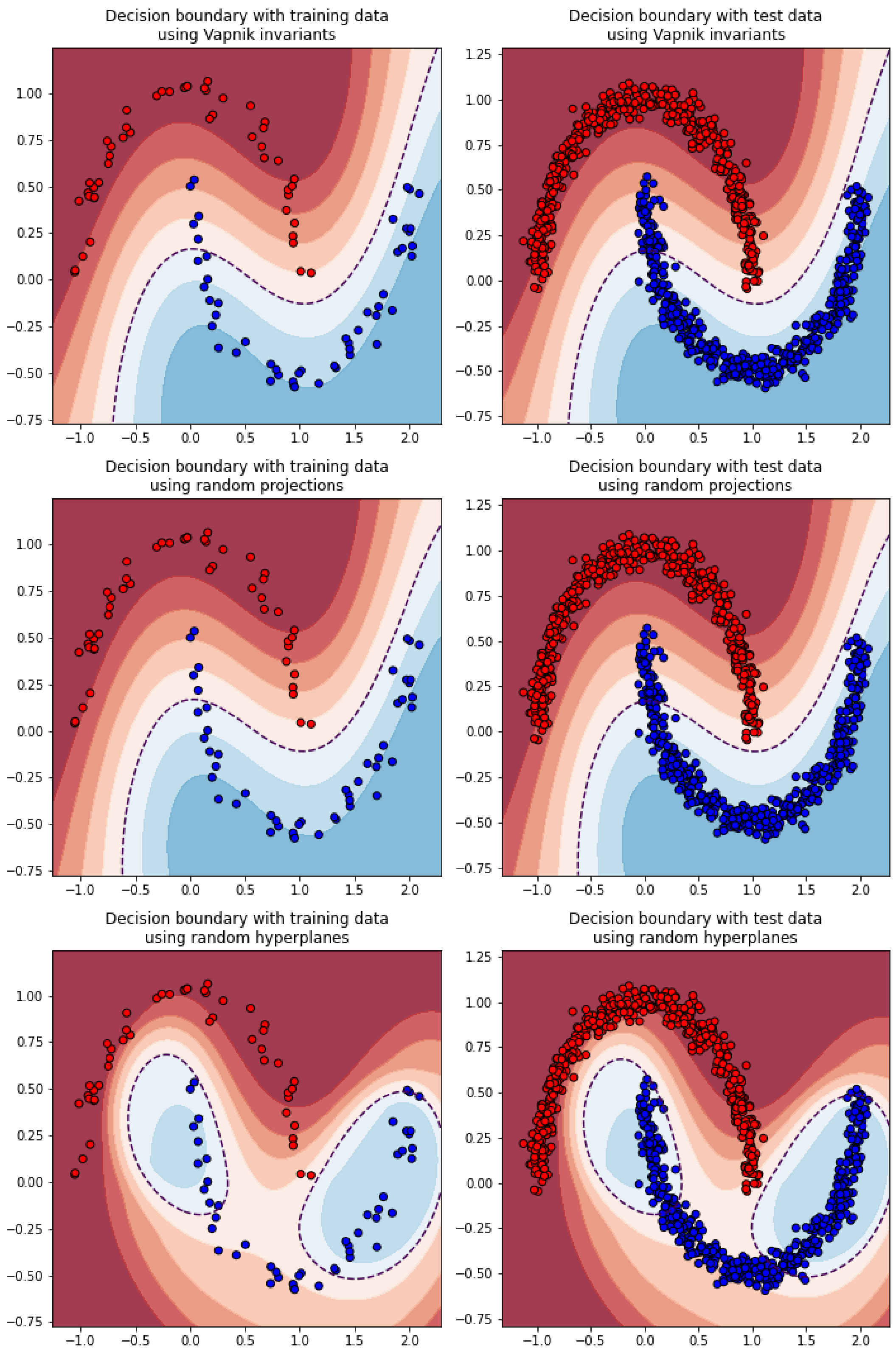

FIGURE 4.5: Decision boundaries in the moons problem using the ECOC version of the LUSI algorithm with each type of invariant on the training and test sets.

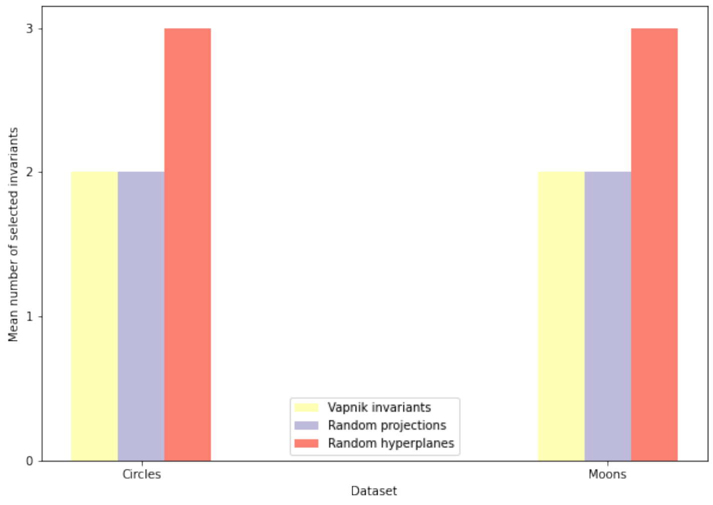

FIGURE 4.6: Mean number of selected invariants per problem. The maximum number of invariants was set to 3.

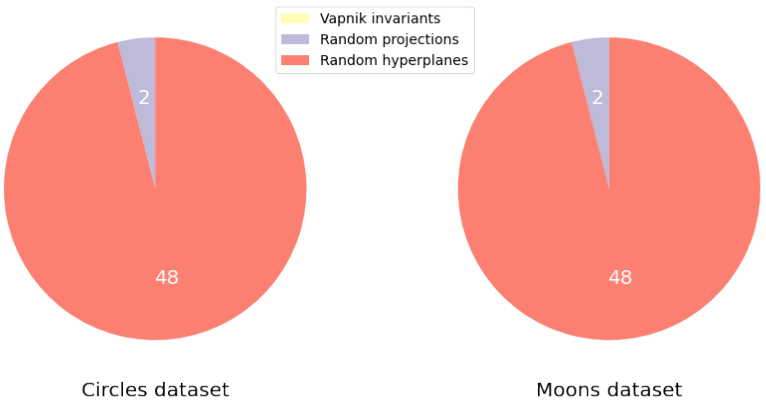

FIGURE 4.7: Mean number of selected invariants when the maximum number of invariants is 50 and the three types of invariants are considered simultaneously.

4.2.1 Experimental settings

First and foremost, we need to discuss what datasets we are going to use in the experimentation, what will be compared and how the experiments will be performed.

As for the data, we are going to use five datasets from the UCI repository (Dua and Graff, 2017). Originally, we planed on using more datasets, but due to strong time restrictions and the fact that each experiment took from a couple of hours up to a couple of days it did not seem worthwhile increasing the number of datasets. The selected datasets along with some basic information about them can be found in table 4.2. As we can see, all of them are multiclass classification problems, which means that we must use the ECOC-based extension that we have proposed in this work. We applied some preprocessing to the datasets before using them in our experiments, like transforming the categorical values using hashing and transforming the classes to numerical values so that we could more easily create the encoding matrices and apply the posterior decoding step.

Problem	**Num. examples**	**Attributes**	**Classes**
Balance Scale	625	4	3
Ecoli	336	8	8
Glass	214	9	7(6)[1]
Iris	150	4	3
Yeast	1484	8	10

TABLE 4.2: Information of the datasets used in the experimentation.

In these experiments, we will be comparing four different versions of LUSI using different types of invariants: (i) a baseline version with no invariants, (ii) Vapnik's invariants (zeroth and first order invariants), (iii) random projections and (iv) random hyperplanes.

In order to compare the different models, we have designed an experimentation methodology that we will briefly describe. We are going to create three different stratified partitions of each dataset using different seeds. In these partitions we are going to keep 80% of the data for training and the remaining 20% will be used to test the models. For each training partition, we are going to create three different stratified subsamples of it, keeping 100%, 50% and 10% of the data. Using each one of these subsamples, we are going to perform three grid searches using 3-fold cross validation over different combinations of hyperparameters for each one of the models, finding the best possible combination of hyperparameters in each case. Finally, we are going to evaluate these models using the test set from the corresponding partition and obtain a accuracy for that particular scenario. We are using the same test partition to evaluate the different models so that we can have a common ground that allows us to fairly compare the different types of invariants. A general overview of this experimentation methodology can be seen in figure 4.8, where an example is shown using the random projections invariant.

Finally, we need to clarify which hyperparameters we are going to be fine-tuning. For this experimentation and due to the previously mentioned time restriction as well as the hefty amount of hyperparameter combinations for each model, we have restricted ourselves to performing the grid search on the following hyperparameters:

- The maximum number of invariants that can be selected.
- The value of δ used in the invariants selection.
- The value of γ used when computing the kernel.

[1]According to the dataset's documentation, there are 7 classes. However, there is one that does not have any elements. Therefore, the actual number of classes for this problem is 6.

- The value of `C`, which is a regularization parameter that controls how much perturbation is applied to the product of the V and K matrices before computing the inverse so that it is not a singular matrix.

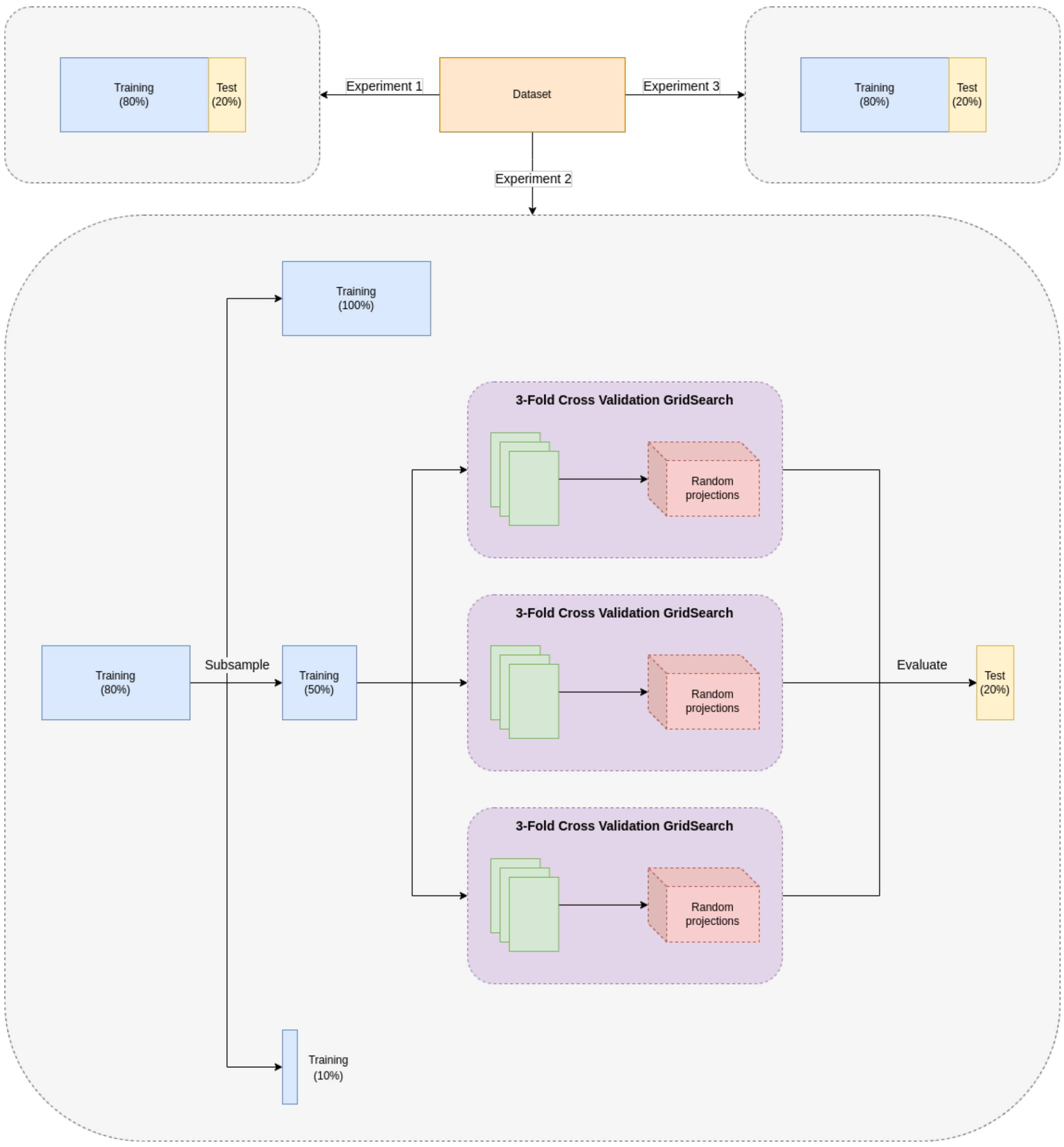

FIGURE 4.8: Overview of the experimentation methodology. In this example, the grid search is only applied to the random projections model for a subsample of 50% of the training partition, although in a real scenario it would be applied to each training subsample and to each type of model. Note that all models are evaluated using the same test partition.

4.2.2 Results

Let us now analyze the results that we have obtained from the experimentation. Tables 4.3 and 4.4 contain the mean accuracies and standard deviations for each dataset considering the different invariant types and training subsample sizes. We have only reported the mean accuracies achieved when using a subsample of 100% and 10% of the training partitions, as they represent two extremes of the learning problem: one in which there is a lot of available training data, and one where this data is scarce.

We can observe that the baseline model with no invariants almost completely outperforms the rest of the models which use some kind of invariants on both training sizes. Also, it is the one that has the least variation in the results. In second place we find the original invariants proposed by Vapnik, which offer good results overall, although a model that uses no invariants is capable of achieving better results. The random projections follow quite closely Vapnik's invariants, offering average results that are a bit worse than the original invariants. However, they are capable of achieving better results than Vapnik's invariants in the Balance Scale dataset using a big training size and in the Glass dataset when considering a small training dataset. The large deviation in the Iris dataset in the small problem is really curious, probably caused by a bad random seed. Finally, the random hyperplanes come in last place, offering the worst results in almost all problems irregardless of the training size.

One possible explanation for the poor results obtained by the random invariants may be that the random seeds that we have chosen are quite bad. Because these invariants heavily rely on stochastic processes, a bad initialization can greatly worsen the results. Thus, we can try to repeat these experiments with more carefully selected random seeds or by increasing the pool of initial seeds, which would mean that we would have to run more experiments, which as we have already mentioned, takes a very long time and computational resources.

This randomness also affects the variation of the results. As we can clearly see, the models that use the random invariants have larger standard deviations. Because the model that uses no invariants has no need to select invariants and the one that uses Vapnik's proposed invariants does not have to generate new invariants, the training process becomes more deterministic, which reduces the overall variability of the results.

Another result that must be highlighted is the poor performance of the random hyperplanes. A possible explanation of this might be that this invariant is a modification of the zeroth order invariant in which the proportion of elements is only kept in a region of the original space. Although the idea seemed interesting in the first place as we have potentially infinite hyperplanes that can separate the data in order to keep different proportions of it, in practice it seems that preserving this kind of statistical information might not be that useful as the models may be prone to overfitting, as we saw in the experimentation with the toy problems.

Dataset	**Baseline model**	**Vapnik invariants**	**Random projections**	**Random hyperplanes**
Balance Scale	$91.47 \pm 0.38\%$	$91.47 \pm 0.38\%$	$91.73 \pm 0.38\%$	$90.13 \pm 1.51\%$
Ecoli	$85.29 \pm 2.08\%$	$85.29 \pm 1.20\%$	$84.15 \pm 2.05\%$	$75.98 \pm 6.28\%$
Glass	$72.87 \pm 7.91\%$	$72.87 \pm 7.91\%$	$72.09 \pm 7.44\%$	$71.06 \pm 7.60\%$
Iris	$96.67 \pm 0.00\%$	$95.56 \pm 1.57\%$	$95.19 \pm 1.66\%$	$88.52 \pm 11.56\%$
Yeast	$53.76 \pm 1.04\%$	$53.87 \pm 1.20\%$	$52.53 \pm 5.44\%$	$50.39 \pm 5.33\%$

TABLE 4.3: Mean accuracy and standard deviation for each type of invariant in each problem considering the results for subsamples of 100% of the training data. The results are expressed as percentages.

We can delve into the previous results by comparing how many times each model obtains a better, worse or the same accuracy as the rest of the models for the same experiment. This information has been condensed in figures 4.9 and 4.10, where we can see the performance of the different models across all experiments.

As we can observe, the model with no invariants is capable of achieving more times greater accuracies than the rest of the models irregardless of the training size.

Dataset	**Baseline model**	**Vapnik invariants**	**Random projections**	**Random hyperplanes**
Balance Scale	$88.80 \pm 1.73\%$	$87.73 \pm 2.10\%$	$83.73 \pm 7.17\%$	$76.00 \pm 12.77\%$
Ecoli	$71.57 \pm 1.39\%$	$72.55 \pm 4.22\%$	$67.32 \pm 4.03\%$	$63.89 \pm 9.61\%$
Glass	$56.59 \pm 4.78\%$	$45.74 \pm 6.67\%$	$52.45 \pm 7.03\%$	$45.48 \pm 9.50\%$
Iris	$93.33 \pm 2.72\%$	$90.00 \pm 4.71\%$	$77.78 \pm 22.93\%$	$84.81 \pm 9.04\%$
Yeast	$47.92 \pm 2.08\%$	$47.92 \pm 2.14\%$	$45.68 \pm 4.60\%$	$37.82 \pm 7.92\%$

TABLE 4.4: Mean accuracy and standard deviation for each type of invariant in each problem considering the results for subsamples of 10% of the training data. The results are expressed as percentages.

When compared to the model that uses Vapnik's invariants, we can see that they achieve the same accuracy in a lot of experiments (approximately 50% of the times). This model offers worse results in only a couple of experiments, which are less than 20% of the times. Something similar happens when we compare it to the random projections, although the results are a bit more favorable for the model with no invariants in this case. When comparing it to the random hyperplanes, we can see that the latter one is totally outmatched.

If we compare now Vapnik's invariants to the random projections, we can see that they achieve very similar results overall, although they are more favorable in the case of Vapnik's invariants. When comparing these invariants with small training sizes, we can see that the difference between them is less significant. Hence, as we saw in the previous experimentation and in this one, we can state that these types of invariants are very similar. Probably, this is due to the fact that most of Vapnik's invariants are first order invariants, and the random projections are a variation of this type of invariant. With them, we can preserve the centroid of each class, which is an interesting information that should be kept when learning the bets approximation of the target function.

Finally, to no one's surprise, the random hyperplanes are the ones that perform the worst when compared to the rest of the models, as it is the invariant type that gets beaten the most frequently. This might be because the information provided by this variation of the zeroth order invariant is not as relevant as the one provided by the original zeroth order invariant or higher order invariants. Hence, its usefulness seems questionable even when a model that uses no invariants is capable of outperforming a model that uses this kind of invariant by far.

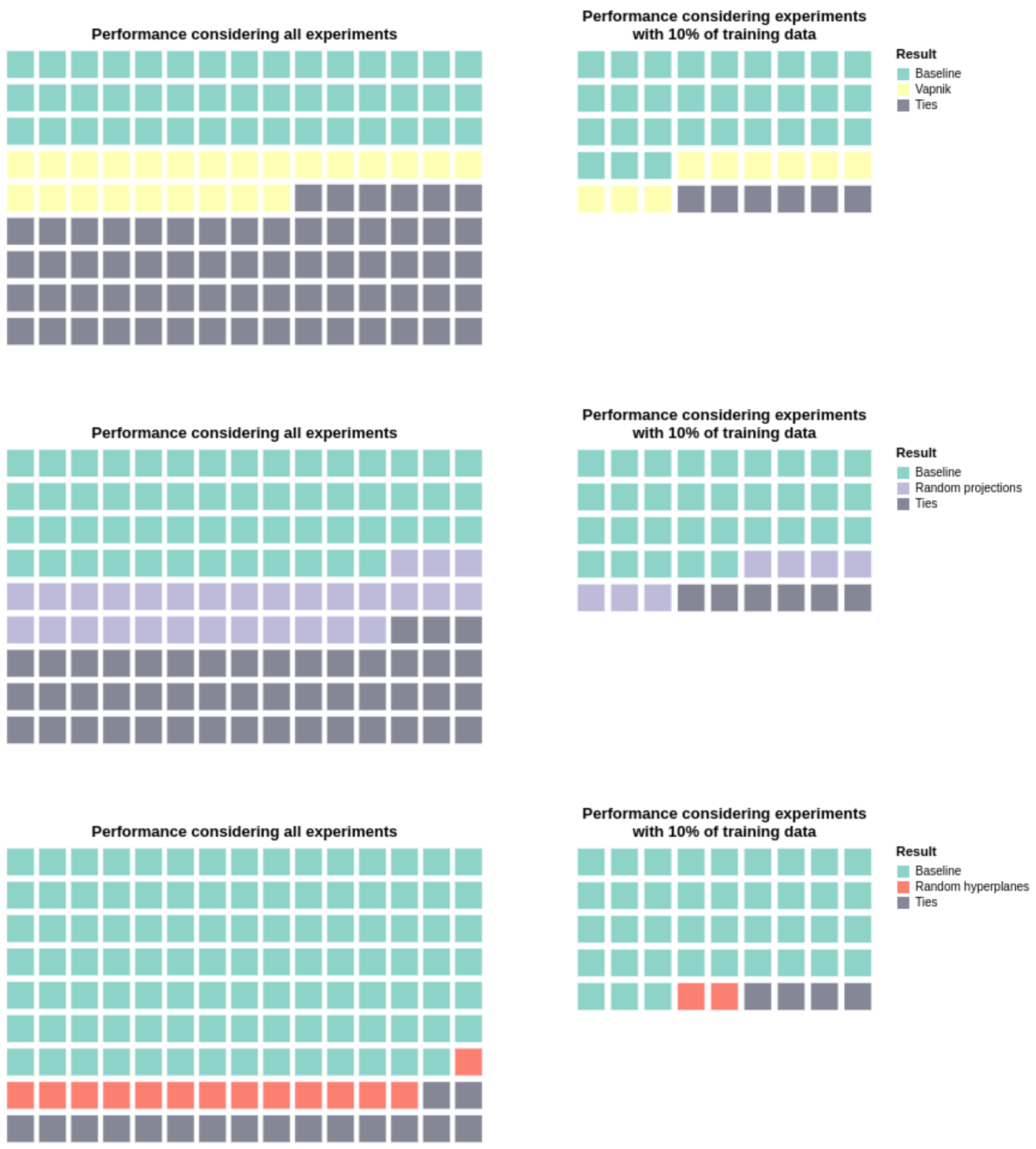

FIGURE 4.9: Number of times that the baseline model achieves a better, worse or the same accuracy when compared to the rest of the models in the same experiment, aggregated across all datasets. Left column shows the results obtained when the training size is disregarded. The right one shows the results only when considering the experiments where 10% of the training partition is used.

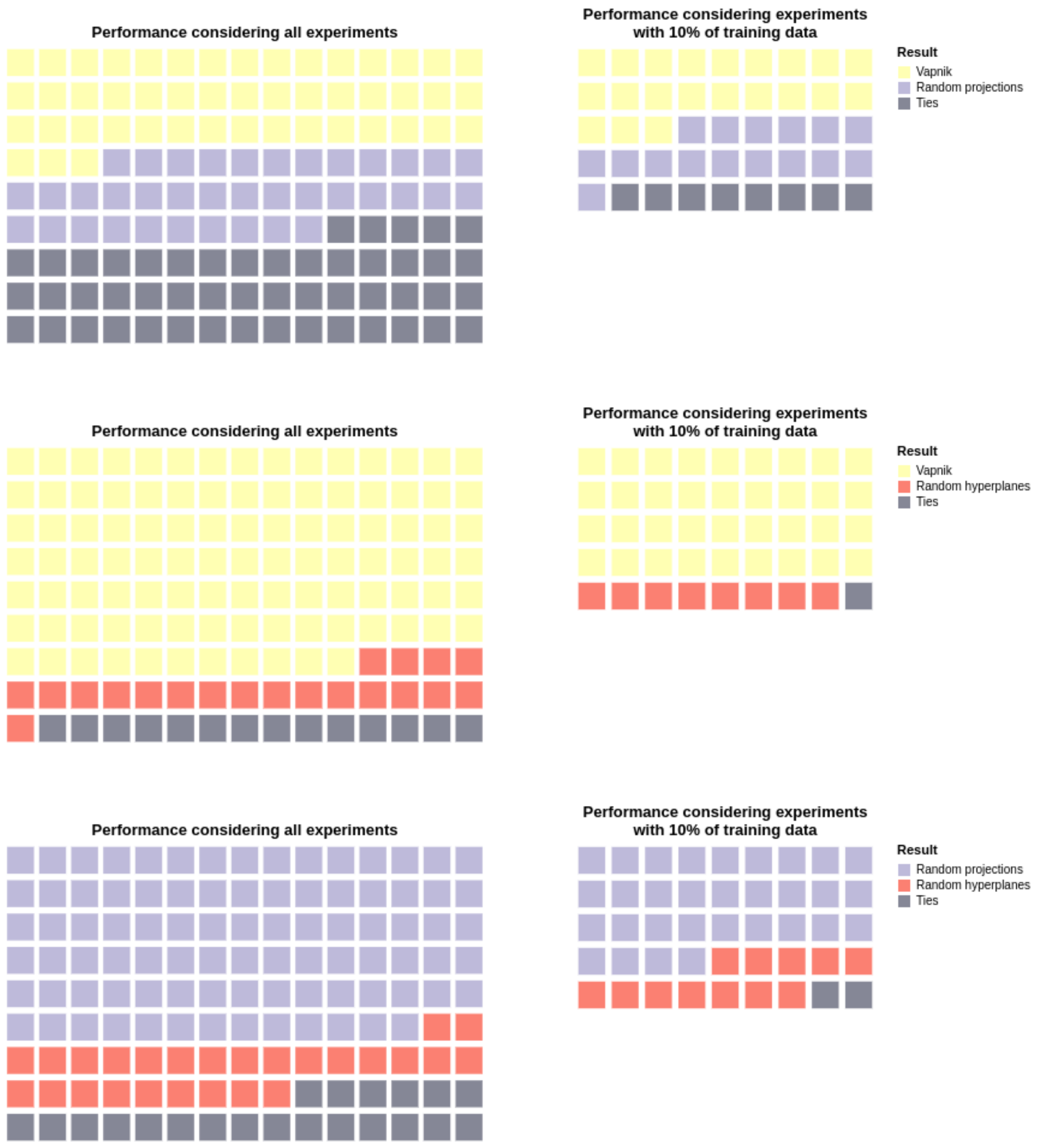

FIGURE 4.10: Number of times that the models using Vapnik's invariants, random projections and random hyperplanes achieve a better, worse or the same accuracy when compared one to another in the same experiment, aggregated across all datasets. Left column shows the results obtained when the training size is disregarded. The right one shows the results only when considering the experiments where 10% of the training partition is used.

Chapter 5

Conclusions

Summing up, in this work we have studied the application of statistical invariants to the learning problem through a new learning paradigm called LUSI. We have seen the theoretical background that it is based on as well as the potential benefits that it can bring and its limitations. We have seen that the invariants are problem dependent and choosing the most appropriate ones is a hard task which requires prior knowledge. Also, the original version of the LUSI algorithm can only consider one class at the time, which hinders its application to more complex problems.

With these limitations in mind, we have proposed two new invariants, which are random projections and random hyperplanes. These invariants use random processes in order to keep statistical information of the data, are of general use and require no previous knowledge of the problem. Also, we have proposed an extension of the original LUSI algorithm using ECOC so that it can be applied to any class in both binary and multiclass classification problems.

We have seen that the extended version of LUSI using ECOC has been successfully applied to both binary and multiclass classification problems without negatively impacting the final results.

However, after performing some experiments with the new invariants we have seen that the obtained results were not as good as expected. The random projections offered similar results to the original invariants, although they were slightly worse in general. As far as the random projections go, the results that we obtained using them were discouraging, as it was the type of invariant that performed the worst in almost all scenarios.

Also, even though it was one of the main goals of this thesis, we were not able to automatize the invariants selection process, as we still have to manually choose which invariants need to be applied to a given problem. Thus, it still remains an open topic.

In conclusion, we believe that this new version of LUSI using ECOC is an important step in making this data-driven paradigm more accessible and easy to apply for the machine learning community. Also, the random projections are an interesting proposal that can be applied to multiple problems and achieve overall good results, although there are other invariants that might work better. Nonetheless, it is an important example of how new invariants can be created.

5.1 Future work

Even though we have accomplished some of the goals that we had set at the beginning of this thesis, there is still a lot of room for improvement. Moreover, there are some other lines of work we wanted to explore but we couldn't because of the time limitations of this project or because they ended up being out of scope. Thus, here are some proposals of future work that we believe are of great interest:

- The current formulation of the LUSI algorithm as a system of equations is quite cumbersome. We could rewrite this formulation so that an iterative algorithm like SGD can be applied to it. This way, we could enable the application of this data-driven paradigm to different types of machine learning algorithms. We believe that this task is quite difficult and we would need to explore the possible limitations of this new formulation, or whether it can be done at all.
- The original work only proposed invariants up to the first order. It would be interesting to see whether it makes sense to apply higher order invariants to the learning problem and if they can be used to achieve better results.
- Related to the previous points, coming up with invariants that can be applied to text or images is a task that would further enable LUSI to be applied to new domain problems. However, it would first need to be reformulated.
- We believe that the random projections invariant can be improved by constructing an orthogonal space from random vectors and projecting the data in it. This could be done using an orthonormalization process like the Gram-Schmidt process.

Bibliography

Bingham, Ella and Heikki Mannila (2001). “Random Projection in Dimensionality Reduction: Applications to Image and Text Data”. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’01. San Francisco, California: Association for Computing Machinery, 245–250. ISBN: 158113391X. DOI: 10.1145/502512.502546. URL: https://doi.org/10.1145/502512.502546.

Dasgupta, Sanjoy (2000). “Experiments with Random Projection”. In: *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*. UAI’00. Stanford, California: Morgan Kaufmann Publishers Inc., 143–151. ISBN: 1558607099.

Dietterich, Thomas G. and Ghulum Bakiri (1995). “Solving Multiclass Learning Problems via Error-Correcting Output Codes”. In: *CoRR* cs.AI/9501101. URL: http://dblp.uni-trier.de/db/journals/corr/corr9501.html#cs-AI-9501101.

Dua, Dheeru and Casey Graff (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Vapnik, Vladimir and Rauf Izmailov (2015). “V-Matrix Method of Solving Statistical Inference Problems”. In: *J. Mach. Learn. Res.* 16.1, 1683–1730. ISSN: 1532-4435.

— (2019). “Rethinking statistical learning theory: learning using statistical invariants”. In: *Machine Learning* 108.3, pp. 381–423. ISSN: 1573-0565. DOI: 10.1007/s10994-018-5742-0. URL: https://doi.org/10.1007/s10994-018-5742-0.