UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Predicting venous thromboembolic events in patients with cancer using a new machine learning paradigm

Author:
Pablo Álvarez Cabrera

Supervisor:
Dr. Oriol Pujol Vila
Dr. José Manuel Soria

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

September 2, 2021

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Predicting venous thromboembolic events in patients with cancer using a new machine learning paradigm

by Pablo Álvarez Cabrera

The rise of machine learning in the last decade has facilitated great advances in fields such as medicine, where very powerful models have been developed, capable of predicting certain medical conditions with an accuracy never seen before.

The present work is focused on predicting one of the leading causes of death among patients with cancer: venous thromboembolic events (VTE). Over the years, several statistical models based on clinical/genetic data have been developed, and have made it possible to create some risk assessment tools, like the Khorana score [2]. However, none of them are based on machine learning.

In this way, we propose a new model that uses advanced machine learning techniques and is able to outperform all models currently available. Furthermore, the model is based on a very recent and promising learning paradigm that has barely been tested, hence it is a great opportunity for us to explore and evaluate it.

This breakthrough ultimately has an impact on the patient's quality of life, improving the ability to detect patients at high risk of developing a VTE, who would benefit from preventive treatment.

Acknowledgements

To my family and friends, to whom I can only express my most sincere appreciation for their support during this year.

I would especially thank my tutors, Oriol Pujol and José Manuel Soria, for their continuous guidance in the development of this work.

Contents

Chapter 1

Introduction

1.1 Problem description

The present work is focused on the prediction of venous thromboembolism events (VTE) in patients with cancer. A venous thromboembolism occurs when a blood clot forms in a deep vein, usually in the leg, groin or arm, and travels through the bloodstream, reaching the lungs and blocking the blood supply in some cases. Patients with cancer have a 7-fold increased risk of developing this potentially deadly medical condition, specially in the first months after diagnosis [1].

Prevention is critical in this area. If a patient with high risk of developing a VTE is detected, then doctors can apply thromboprophylaxis (a medical treatment to prevent the development of thrombosis), reducing the risk of death. Accuracy in detecting high-risk patients is also important, since this treatment should be applied only in necessary cases, as it may increase the risk of bleeding.

Different guidelines for assessing the risk of developing a VTE have been considered in the last years, with the Khorana score [2] being probably the most popular one. The patient receives a risk score based on some clinical variables such as the cancer type, body mass index (BMI) and hemoglobin level[1]. However, its accuracy has been called into question recently [3, 6]. With the aim of developing a better model, the authors in [4] proposed the TiC-Onco risk score, a model based on multivariate logistic regression analysis, which takes into account not only clinical information from the patient, but also genetic variables. Results showed that the TiC-Onco risk score was better at predicting VTE than the Khorana score. However, none of the tools discussed above use the potential of the recent artificial intelligence-based models, which could be very useful in this area.

Therefore, the development of an effective prediction tool, able to detect patients which are in high risk of suffering VTE is still open, and its deployment would suppose a breakthrough in preventive medicine.

1.2 Objectives

The objectives proposed at the beginning were the following:

- First, analyze and preprocess the data we have in order to generate the database we will work with.

[1] This interactive tool can be used to calculate the Khorana score.

- Second, be able to replicate the results of the TiC-Onco risk score reported in [4], that will be used as a baseline.
- Next, to explore several machine learning techniques that can be used for the problem and find the most suitable ones.
- Lastly, be able to test some models and improve the previous results, proposing a new tool for predicting VTE based on artificial intelligence.

1.3 Summary of the contributions

Once the project has been completed, we can identify the following contributions to each of the objectives described above:

- We have recreated the dataset that was used in [4], that can be used to compare the results obtained and also for further experiments.
- The results provided in [4] have been validated with our own methodology, demonstrating that the TiC-Onco score is better than the Khorana score in predicting VTE.
- A wide variety of machine learning models have been tested, including classical and innovative models. In particular, we have experimented with a new technique that promises to be a breakthrough in the next few years [8].
- We have been able to develop an artificial intelligence-based model that outperforms all the previous ones, obtaining a new and more accurate tool for predicting VTE.

1.4 Work outline

This document is organized into five chapters:

- The initial chapter provides an introduction of the problem and summarizes the objectives an contributions made.
- Chapter 2 covers the background concepts that are fundamental to understand the procedures and decisions we have made. The methods used in [4], the data processing, as well as the metrics and evaluation techniques used throughout the work are discussed.
- Chapter 3 presents the new machine learning technique that will be used to approach the problem. It includes a brief explanation of the idea and the most important technical details for implementing the algorithm, along with a few basic examples using this approach.
- Chapter 4 covers all the experiments done with machine learning models, explaining some important aspects as well as analyzing and comparing the results obtained.
- Finally, chapter 5 provides some final conclusions and some ideas that can be used in the future.

Chapter 2

Background

Before testing any model, let's analyze the problem and the data we have considered, as well as define the metrics and evaluation techniques that will be used throughout the work.

2.1 Problem statement: predicting venous thromboembolic events in patients with cancer

Patients with cancer should be periodically examined for VTE risk, as they have an increased risk of developing this medical condition. Here we will discuss one of the most recent risk assessment tools, the TiC-Onco risk score [4]. This score was proposed in 2018 to improve the results provided by the Khorana score [2], which is commonly recommended for evaluating the VTE risk.

The innovation of this tool is that it does not only uses clinical variables for the analysis (as the Khorana score does), but also genetic variables, under the hypothesis that a genetic component is involved in the apparition of VTEs, as stated in recent studies [5].

The TiC-Onco risk score will be the starting point of our work, so we will now briefly explain the methodology and the variables considered in [4], as well as the results they obtained with this tool.

The TiC-Onco risk score was developed in three steps:

1. Selection of clinical variables.

   The following clinical variables, considered as risk factors, were collected from a total of 391 patients at the time of cancer diagnosis: sex, age, diabetes, use of tobacco, family (first degree) history of VTE, body mass index (BMI), high blood cholesterol level, hypertension, primary tumour site, tumour node metastasis stage and haemoglobin, platelet and leukocyte concentration in blood, using the same cut-offs as for the Khorana score.

   Then, they performed a statistical analysis to determine the variables associated with the occurrence of VTE in the first 6 months after the diagnosis, selecting those variables that lead to a high risk of VTE ($p \leq 0.25$).

2. Development of a genetic risk score.

   All the patients were genotyped at the time of cancer diagnosis, considering the number of risk alelles in some specific genes that could be associated with the appearance of VTE: F5 rs6025, F5 rs4524, F2 rs1799963, F12 rs1801020,

F13 rs5985, SERPINA1 rs121909548, SERPINA10 rs2232698 and the A1 blood group.

After 6 months, multivariate logistic regression analysis was performed to determine and select the genetic variables associated with an increased risk of VTE ($p \leq 0.25$).

3. Development of the clinical-genetic model.

   Finally, they applied multivariate logistic regression analysis on the variables selected in the previous steps to determine the weight of each variable in the occurrence of VTE. Table 2.1 shows the selected variables and their associated p-values.

Variable	**p-value**
BMI >25	0.0658
Family history	0.1076
Primary tumour site:	
HR (lung)	0.3483
VHR (stomach, pancreas)	0.0033
Tumour stage > 3	0.0003
Genetic risk score:	0.0049
rs2232698	0.1460
rs6025	0.2064
rs5985	0.2003
rs4524	0.0396

TABLE 2.1: Variables considered in the TiC-Onco risk score and reported p-values [4].

Results were calculated for the Khorana and TiC-Onco risk scores using the bootstrap approach (described later in this chapter, section 2.4), considering 100 resamples from the original data. Table 2.2 summarizes the results, which clearly suggest that the TiC-Onco risk score can better predict high risk patients than the Khorana score. We will explain those metrics in section 2.3. It can be seen that there is still substantial progress to be made in this field, as some of the scores are still too low.

	Khorana	TiC-Onco
AUC (95% CI)	0.58 (0.51-0.65)	0.73 (0.67-0.79)
Sensitivity (95% CI)	0.23 (0.13-0.32)	0.49 (0.38-0.61)
Specificity (95% CI)	0.82 (0.78-0.86)	0.81 (0.77-0.86)
PPV (95% CI)	0.22 (0.12-0.31)	0.37 (0.27-0.47)
NPV (95% CI)	0.83 (0.78-0.87)	0.88 (0.84-0.92)

TABLE 2.2: Predictive capability of Khorana and TiC-Onco scores [4].

2.2 Dataset and data preprocessing

In order to be able to compare our results with those on [4], the authors have provided us with their data. Concretely, we were given a file containing data from 408 patients, with a total of 90 variables, including all the clinical and genetic variables mentioned previously.

Using the data provided, we have recreated the dataset that was used in [4]. The process we followed is described below.

2.2.1 Row selection

Some of the VTE have been dismissed by the authors after being included in the dataset. The variable `excluido` indicates whether a row is discarded or not. Thus, only the rows where `excluido==0` are considered, resulting in 391 samples in total.

2.2.2 Column selection

We have considered only the variables that were selected after the statistical analysis for calculating the TiC-Onco risk score, that is, those that are highly associated with the appearance of VTE (see table 2.1). The columns associated to these variables in the dataset are described in the table 2.3.

Column	Description	Values
bmi	Body Mass Index	Underweight: BMI <18.5 Kg/m2, Normal: BMI ∼18.5-24.9 Kg/m2, Overweight: BMI ∼25-25.9 Kg/m2, Obese: BMI >30 Kg/m2
Family	Family (first degree) history of VTE	0 (no), 1 (yes)
tipusTumor_desc	Primary tumour site	Cáncer colorrectal, Cáncer de pulmón no microcítico, Cáncer de páncreas, Cáncer gástrico o de estómago, Cáncer esófago
estadiGrup	Tumour node metastasis stage	IA, IB, IIA, IIB, IIC, III, IIIA, IIIB, IIIC, IV, IVA, IVB
rs2232698	SERPINA10 gene SNP	CC, CT, NoCall
rs6025	F5 gene SNP	GG, AG
rs5985	F13 gene SNP	GG, GT, TT
rs4524	F5 gene SNP	TT, CT, CC

TABLE 2.3: Columns considered in the dataset.

2.2.3 Target variable

The column `caseAtVisit` takes the values 0, 1, 2 and 3 and indicates whether a patient has suffered a VTE at 0, 6, 12 or 18 months after the monitoring period, respectively. If the patient has not suffered a VTE, then the column contains `NA`.

The objective is to predict VTE within the first 6 months after the diagnosis of cancer, so we will consider values 0 and 1 for the positive class, and the rest for the negative class.

This gives a total of 73 positive cases and 318 negative ones. Thus, we will deal with a highly imbalanced problem, where less than 20% of the data is on the positive class, as can be seen in figure 2.1. We will have to address this issue when considering some models, as will be discussed later in chapter 4.

2.2.4 Data preprocessing

The next step is to preprocess the data of the previous columns to obtain the same variables that were used to compute the TiC-Onco risk score.

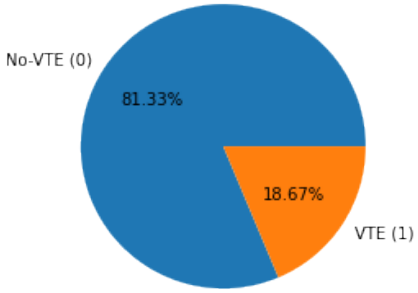

FIGURE 2.1: Distribution of positive and negative classes in the data.

In the genetic variables, each gene can take different pairs of values, representing its alleles (A_1A_2). Some of them are considered as risk factors for thrombosis. Therefore, each allele configuration is replaced by the number of risk alleles it includes, up to a maximum of 2 risk alleles.

Table 2.4 summarizes all the changes introduced. These are the variables that will be used during the rest of the work, unless otherwise specified.

Variable	**Associated column**	**Values**
BMI $>$ 25	bmi	0 ($BMI \leq 25$), 1 ($BMI > 25$)
Family history	Family	0 (no), 1 (yes)
Primary tumour site HR (High Risk)	tipusTumor_HR	0, 1 (cáncer de pulmón no microcítico)
Primary tumour site VHR (Very High Risk)	tipusTumor_VHR	0, 1 (cáncer de páncreas, cáncer gástrico o de estómago)
Tumour stage $>$ 3	estadiGrup	0 (IA, IB, IIA, IIB, IIC, III, IIIA, IIIB, IIIC), 1 (IV, IVA, IVB)
rs2232698 risk alelles	rs2232698	0 (CC), 1 (CT)
rs6025 risk alelles	rs6025	0 (GG), 1 (AG)
rs5985 risk alelles	rs5985	0 (GG), 1 (GT), 2 (TT)
rs4524 risk alelles	rs4524	0 (TT), 1 (CT), 2 (CC)
VTE (target)	caseAtVisit	0 (NA, 2, 3), 1 (0, 1)

TABLE 2.4: Variables considered for the problem.

Thus, we are considering 9 variables derived from the original data to predict the target variable, as done in [4] (see table 2.1).

2.2.5 Data summary

Figure 2.2 shows the characteristics of the population studied. Concretely, it provides the distribution of the different values of the variables within the positive (VTE) and negative (No-VTE) classes. For example, among the patients who have family history of VTE, 6 of them suffered a VTE, representing 8.2% of the patients in the positive class, and 12 did not experience a VTE, which accounts for 3.8% of the patients in the negative class. We can see that some variables are strongly associated with VTEs. For example, patients with tumours located in very high risk sites (stomach and pancreas) represent 50.7% of the patients that suffered VTE.

Variable	VTE (n)	VTE (%)	No-VTE (n)	No-VTE (%)
N	73	100.0	318	100.0
Family	6	8.2	12	3.8
bmi	37	50.7	145	45.6
estadiGrup	49	67.1	132	41.5
rs2232698 - 0 risk alleles	69	94.5	313	98.4
rs2232698 - 1 risk allele	4	5.5	5	1.6
rs4524 - 0 risk alleles	1	1.4	22	6.9
rs4524 - 1 risk allele	23	31.5	114	35.8
rs4524 - 2 risk alleles	49	67.1	182	57.2
rs5985 - 0 risk alleles	38	52.1	182	57.2
rs5985 - 1 risk allele	30	41.1	119	37.4
rs5985 - 2 risk alleles	5	6.8	17	5.3
rs6025 - 0 risk alleles	70	95.9	312	98.1
rs6025 - 1 risk allele	3	4.1	6	1.9
tipusTumor_HR	11	15.1	76	23.9
tipusTumor_VHR	37	50.7	90	28.3

FIGURE 2.2: Data distribution among positive and negative classes.

Figure 2.3 provides a correlation heatmap, that shows the correlation between the different variables considered in the problem. It can be observed that the variables that influence the most on the target varible (VTE) are `tipusTumour_VHR` (tumour located in very high risk places) and `estadiGrup` (tumour in advanced stage). The high negative correlation between the last two variables comes from the fact that these variables are exclusive, as the primary tumour site can only be in one location.

FIGURE 2.3: Correlation between different variables of the problem.

2.2.6 Some remarks

- The column `rs2232698` contains a missing value, which is replaced by `CC` (the mode).
- The data distribution (figure 2.2) is slightly different from the one shown in [4] (Table 1). For example, we end up with 73 samples in the positive class and 318 in the negative one, different from the 71/320 proportion that appears in the paper.

These differences may be due to some error or change in the data that we have not been able to identify, but we will ignore them as they have practically no influence on the results obtained.

2.3 Metrics

We are dealing with a binary classification problem, where the possitive class (1) indicates that a patient is at high risk of suffering a VTE, and the negative class (0) indicates medium or low risk.

This section covers the explanation of the metrics that have been used in the problem. We have considered most of the metrics used in [4], in order to compare the results obtained.

2.3.1 Confusion matrix

The confusion matrix allow us to measure the performance of a model in a machine learning classification problem. It consists of a table with 4 different entries, one for each combination of predicted and actual values, as can be seen in the figure below.

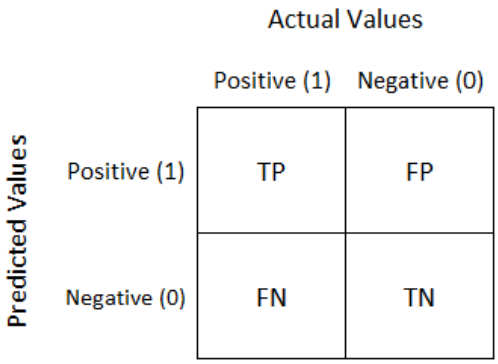

FIGURE 2.4: Confusion matrix (source).

- TP (True Positives) indicates the number of samples correctly predicted as positive.
- TN (True Negatives) indicates the number of samples correctly predicted as negative.
- FN (False Negatives) indicates the number of positive samples incorrectly predicted as negative.
- FP (False Positives) indicates the number of negative samples incorrectly predicted as positive.

These values can be combined to obtain useful metrics. We will use the following:

- **Accuracy**: indicates the proportion of correctly predicted samples.

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

- **Sensitivity** (or TPR, True Positive Rate): indicates the proportion of correctly predicted samples among all the positive ones.

$$\text{sensitivity} = \frac{TP}{TP + FN}$$

- **Specificity** (or TNR, True Negative Rate): indicates the proportion of correctly predicted samples among all the negative ones.

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

- **PPV** (or Positive Predictive Value): indicates the proportion of actual positive samples among all those predicted as positive.

$$\text{PPV} = \frac{TP}{TP + FP}$$

- **NPV** (or Negative Predictive Value): indicates the proportion of actual negative samples among all those predicted as negative.

$$\text{NPV} = \frac{TN}{TN + FN}$$

When dealing with dangerous medical conditions (as in our case), it is particularly interesting to obtain high sensitivity values, since it is crucial to detect all the high-risk patients. The PPV is also important, to avoid false positives, resulting in unnecessary dangerous treatments. On the other hand, the accuracy is not very representative when dealing with unbalanced datasets.

2.3.2 ROC curve and AUC

The ROC (Receiver Operating Characteristic) curve is a graph that shows the performance of a model by plotting two metrics at different values, the TPR (True Positive Rate) and the FPR (False Positive Rate). Each value represents a classification threshold, that is, the probability threshold for classifying a sample as positive.

The TPR is defined above (sensitivity) and the FPR is defined as follows:

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

If the threshold is lower, then more samples will be considered as positives, increasing both False Positives and True Positives. A perfect classification will result in $TPR = 1$ and $FPR = 0$, as can be seen in figure 2.5.

In our experiments, we are working in the point of the ROC curve that gives the same specificity as provided by the Khorana score (around 80%), in order to use the same approach as the authors of [4]. That is, the cut-off to determine whether a patient is at high-risk of developing a VTE is set on this particular point of the curve, and the rest of the metrics are obtained from using the corresponding threshold.

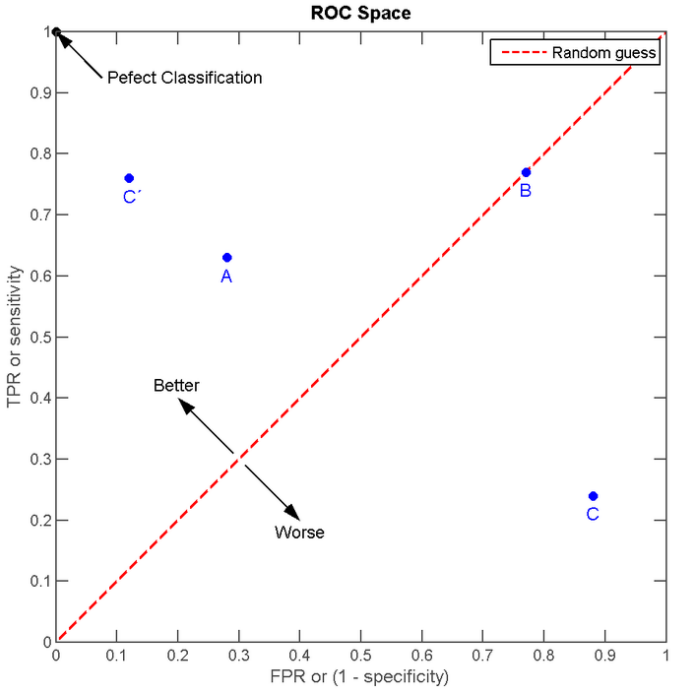

FIGURE 2.5: The ROC space with four prediction examples (source).

The AUC score (Area under the ROC Curve) measures the area underneath the ROC curve, providing an aggregate measure of performance, considering all possible classification thresholds. It can be interpreted as the probability that the model assigns a higher score to a random positive example than to a random negative example.

This metric takes values between 0 and 1 (see fig 2.6), being 1 the most desirable one.

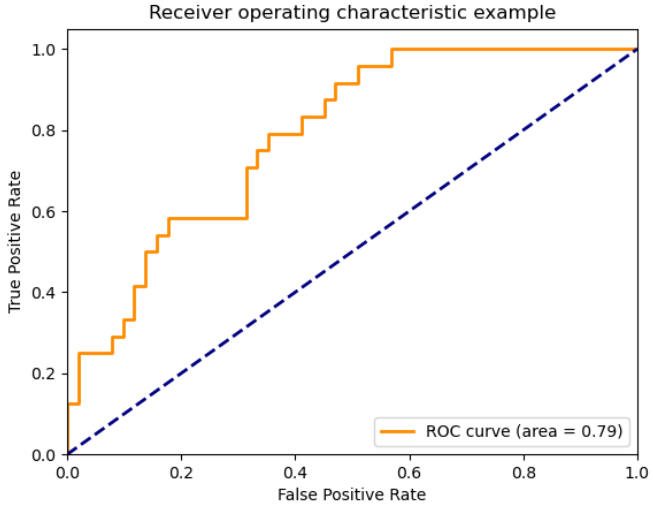

FIGURE 2.6: ROC curve and AUC example (source).

The AUC is interesting because it is scale-invariant, that is, it measures how well predictions are ranked, rather than their absolute values, and it is also classification-threshold-invariant, since it does not takes into account what classification threshold is being used, as opposed to the other metrics defined above.

2.4 Evaluation techniques

Finally, the objective of this section is to explain some techniques that will be used to evaluate the models.

Usually, in a machine learning problem the dataset is split in three subsets:

- A training set, used to fit the model.
- A validation set, that can be used to evaluate a model while tuning its hyper-parameters.
- A test set, used to provide a final evaluation of the model.

In our case, as we are dealing with a small dataset with only 391 samples, we have followed other approaches in order to exploit the data as much as possible, while trying not to bias the results.

2.4.1 Bootstrapping

Bootstrapping is a resampling technique that consists on selecting various subsets of the dataset to train and test the model, aggregating the obtained results.

This is the procedure followed in [4], that can be summarized as follows:

1. Set the number of subsets that will be created, n.
2. Choose the number of samples that will be included in each subset, k.
3. For each bootstrap subset ($1 \dots n$):
   (a) Draw k samples with replacement from the original dataset.
   (b) Fit the model on these samples.
   (c) Calculate the desired metric on these samples.
4. Aggregate the results obtained in each subset, using (for example) the average value.

The main drawback of this procedure is that it uses data that has been previously fed to the model for testing, so the results may be optimistically biased, as we will discuss later.

2.4.2 Cross validation

In order to make more realistic estimations, we decided to use cross validation, a very popular resampling method that uses different splits of the dataset to estimate the performance of a model.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups (folds).
3. For each group:
   (a) Take the group as the test dataset and the remaining groups as the training dataset.

(b) Fit the model on this training set and evaluate on the test set.

(c) Compute the desired metrics.

4. Aggregate the results (usually mean and standard deviation).

The method described above receives the name of *k-fold* cross validation. Figure 2.7 shows an example. We have used $k = 10$ in all the experiments performed using this technique.

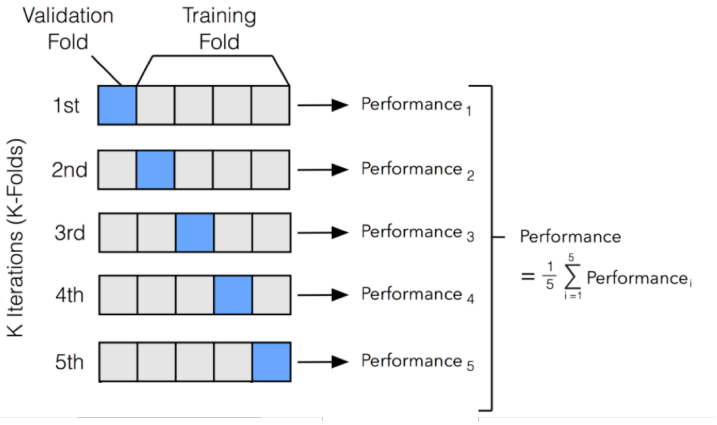

FIGURE 2.7: 5-fold cross validation example (source)

The advantages of this technique are that we are not testing the model using data from the training set, and that different test sets are used to evaluate the model. However, it is slightly biased, since the training set used in cross-validation is smaller than the actual one, specially when considering low values of k (in our case, with $k = 10$, we are training with 352 samples instead of 391). Besides, as we will use this technique for tuning the hyperparameters of the models, the evaluation becomes more biased because skill on the validation set is incorporated into the model configuration. Nonetheless, we will use different partitions for fine-tuning and evaluating a model in order to reduce the bias. In general, this method yields more accurate results than those obtained using only one training set and one test set.

Chapter 3

Learning using statistical invariants

This work explores one of the most recent and promising paradigms in the area of machine learning, known as LUSI (Learning Using Statistical Invariants).

This learning paradigm was introduced in [8], and supposes a new way of addressing the learning problem, not only based on data but also on some statistical invariants (specific for the problem) that act as a teacher during learning.

The purpose of this chapter is to give a brief explanation of how this technique works, explaining the most important technical details. For a complete and more detailed understanding, refer to [7, 8].

3.1 Estimation of conditional probability function

The ultimate goal in any supervised machine learning problem is to obtain an estimate of the conditional probability function $P(y = 1|x)$, i.e., given a data point x, we want to know the probability for this point to belong to a certain class[1].

The classical paradigm of learning relies on the strong mode of convergence in the Hilbert space, trying to find a sequence of functions $\{P_l(y = 1|x)\}$ such that

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

However, there exists a weak mode of convergence (convergence in inner products), given by

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

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

Here is where the new mechanism comes into play, replacing this infinite set of functions with a finite set $\mathcal{P} = \{\psi_1(x), \ldots, \psi_m(x)\}$, whose elements, that receive the name of predicates, are specially designed to describe some interesting properties of the desired conditional probability function. These properties are defined by the following equalities, that receive the name of invariants:

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

[1]Here it is supposed that there are only two classes in the problem, 0 and 1, since any classification problem with more than 2 classes can be reduced to some binary classification problems.

where a_s is, by definition, the expected value of $\psi_s(x)$ with respect to the measure $P(y = 1, x)$.

The exact values of a_s are unknown, but they can be estimated using the training data $\{(x_i, y_i),\ i = 1, \ldots, n\}$. Thus, the above expression is replaced by:

$$\frac{1}{n}\sum_{i=1}^{l}\psi_s(x_i)P_l(y=1|x) \approx a_s \approx \sum_{i=1}^{n} y_i\psi_s(x_i), \quad s = 1, \ldots, m \tag{3.1}$$

In other words, the idea is to look for the approximation $P_l(y = 1|x)$ in the subset of functions that preserve the invariants associated with the predicates in $\mathcal{P}$, reducing the set of candidate functions to those fulfilling 3.1.

The authors of [8] provide a way of sequentially adding different predicates to obtain an approximation of the conditional probability function $P(y = 1|x)$: let's suppose we have constructed an approximation $P_l^k(y = 1|x)$ using k predicates. Before considering the next predicate ψ_{k+1}, we calculate the value

$$\mathcal{T} = \frac{\left|\sum_{i=1}^{n}\psi_{k+1}(x_i)P_l^k(y=1|x_i) - \sum_{i=1}^{n} y_i\psi_{k+1}(x_i)\right|}{\sum_{i=1}^{n} y_i\psi_{k+1}(x_i)} \tag{3.2}$$

If $T \geq \delta$ (δ is a small positive threshold), then the new invariant defined by predicate ψ_{k+1} is considered. Otherwise, expression 3.1 is treated as an equality and the invariant is not considered.

3.2 Closed-form solution

It can be proven [8] that, in a specific type of Hilbert space known as RKHS (Reproducing Kernel Hilbert Space), the estimate for $P(y = 1|x)$ has the form

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

for some vector of coefficients $A \in \mathbb{R}^n$ and some bias $c \in \mathbb{R}$. The vector of functions $\mathcal{K}(x) = (K(x_1, x), \ldots, K(x_n, x))^\top$ is determined by the kernel associated with the RKHS[2], evaluated on the training data.

Let's denote $Y = (y_1, \ldots, y_n)$ the labels of the training set, $K \in \mathbb{R}^{n\times n}$ the matrix with elements $K(x_i, x_j)$, $i, j = 1, \ldots, n$, $\Phi_s = (\psi_s(x_1), \ldots, \phi_s(x_l))^\top$ the vector obtained evaluating the predicate ψ_s on the training data, $1_n = (1, \ldots, 1)^\top$ and I the identity matrix of dimension n.

When considering the constraints defined by 3.1, we can obtain a closed-form solution for A and c, given by

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

where:

$$A_V = (K + \gamma I)^{-1} Y$$

[2] In our experiments we will use the Gaussian (or rbf) kernel, given by

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

$$A_C = (K + \gamma I)^{-1} 1_n$$
$$A_S = (K + \gamma I)^{-1} \Phi_s, \quad s = 1, \ldots, m$$

Coefficients c and μ_s are obtained from the system of linear equations given by

$$c \left[1_n^\top K A_C - n \right] + \sum_{s=1}^{m} \mu_s \left[1_n^\top K A_S - 1_n^\top \Phi_s \right] = \left[1_n^\top A_V - 1_n^\top Y \right]$$

$$c \left[A_C^\top K \Phi_k - 1_n^\top \Phi_k \right] + \sum_{s=1}^{m} \mu_s A_S^\top K \Phi_k = \left[A_V^\top K \Phi_k - Y^\top \Phi_k \right], \quad k = 1, \ldots, m$$

The parameter $\gamma > 0$ controls the amount of regularization applied.

This algorithm receives the name of SVM algorithm with m invariants, or SVM&I_m.

Remark. *The closed-form solution presented in [8] is slightly different from the one described above, as they consider a special matrix (the V-matrix) for some convergence purposes that have not been very useful in our problem, so we have set $V = I$.*

Remark. *The algorithm supposes that the data is scaled in the range $[0, 1]$.*

3.3 A simple LUSI algorithm

Following the example in [8], we have considered the following learning method based on predicates $\psi_k(x)$, $k = 1, \ldots, m$ and training data (x_i, y_i), $i = 1, \ldots, n$:

Step 0: Construct SVM estimate of conditional probability function using the closed-form solution described in 3.2 (without considering predicates).

Step 1: Compute the disagreement values $(\mathcal{T}_1, \ldots, \mathcal{T}_m)$ (see 3.2) for vectors

$$\Phi_k = (\psi_k(x_1), \ldots, \psi_k(x_n))^\top, \quad k = 1, \ldots, m$$

and find the maximum $\mathcal{T}_s = \max(\mathcal{T}_1, \ldots, \mathcal{T}_m)$.

Step 2: If $\mathcal{T}_s > \delta$, add the invariant associated with ψ_s; otherwise stop.

Step 3: Compute the new approximation of the conditional probability function using the SVM&I_n algorithm and if there are more predicates go to step 1; otherwise stop.

3.4 Testing SVM with invariants

Once the SVM&I_n algorithm was implemented, we decided to run several experiments in simple problems (see fig. 3.1) to test its performance.

The first dataset is a linearly separable problem, with 50 points belonging to each class (50 red and 50 blue). The second one presents a non-linearly separable imbalanced problem, with 90 points in the blue class and only 10 in the red one. Lastly, the third one is a non-linear problem, with 50 points of the red class in an external circle, containing 50 points of the blue class in a smaller one. In each case, we are considering 70 samples for training and 30 for testing, that appear lighter in the images.

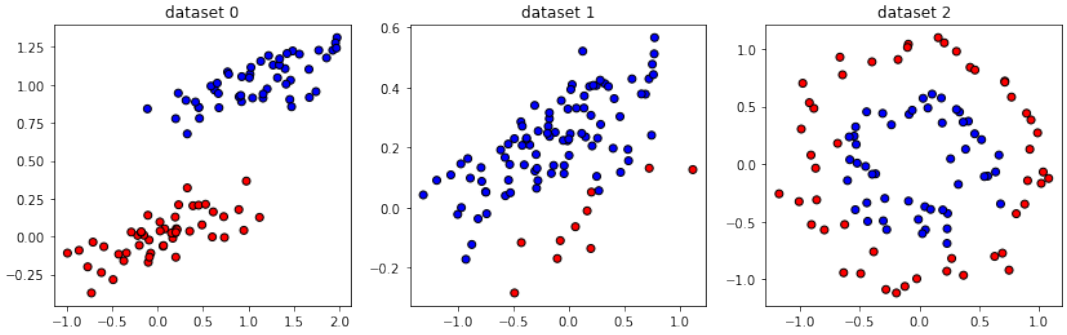

FIGURE 3.1: Datasets used for testing the SVM&I_n implementation.

For each dataset, we will show the accuracy and the classification boundaries obtained with three different versions of SVM:

i) The scikit-learn implementation (`SVC` class).

ii) The SVM algorithm obtained when not considering predicates in the previous closed-form solution.

iii) The SVM&I_n algorithm described above, using some simple predicates.

The goals are to check that our implementation is correct, comparing the first two versions (which should be similar), and to test if we get any improvement with the use of predicates in the third version.

The invariants considered are defined by the predicate functions

$$\psi_0(x) = 1, \quad \psi_1(x) = x_1 \quad \text{and} \quad \psi_2(x) = x_2$$

The first invariant reduce the candidate functions to those preserving the frequency of elements of class 1 observed in the training data, while the last two lead to functions preserving the center of mass (in the x or y axis, respectively) of samples belonging to class 1.

3.4.1 Results in the linearly separable problem

In this case we are dealing with a linearly separable problem, so we used a linear kernel with `C=1` and `gamma=1/2`. All the models obtained a perfect classification on the test set.

The classification boundaries obtained with each version are shown in figure 3.2. The figure on the left contains the results obtained with the scikit-learn implementation, while the other two show the boundaries obtained with our implementation, without using predicates (center figure) and considering the three predicates described above (right figure).

We can see that the decision line (dotted line in the figures) is almost the same in the first two versions. The differences in the colored regions are due to the fact that we are using raw predicted scores in our algorithm, while the scores used in `SVC` are converted to probabilities (in the range $[0, 1]$). Yet, we can see that they follow a similar pattern.

The effect of predicates is really clear in this example. The boundaries in the last figure preserve the alignment of the training data, which is really interesting.

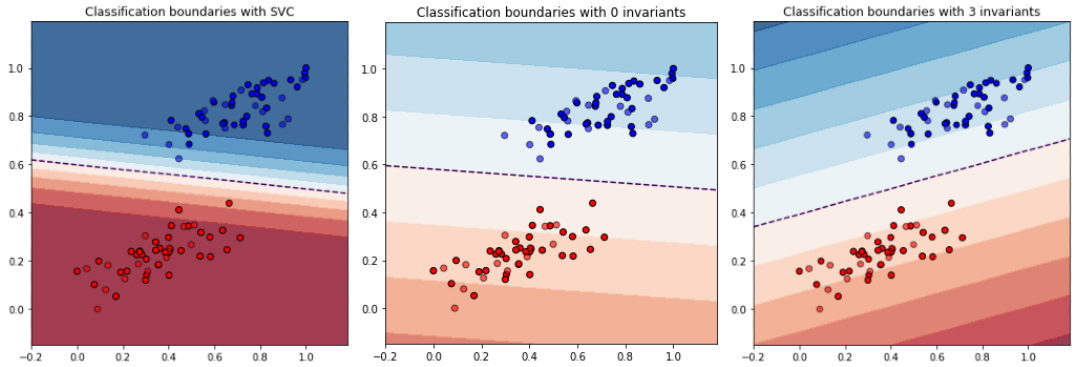

FIGURE 3.2: Classification boundaries obtained with the different versions of SVM in the first problem.

3.4.2 Results in the imbalanced problem

Now we will focus on an imbalanced problem, where 90% of the data belong to the blue class. This is a more difficult and interesting framework, considering that our problem is also unbalanced.

In this case, we are using again a linear kernel with `C=1` and `gamma=1/2`. The classification boundaries obtained with each model are represented in figure 3.3.

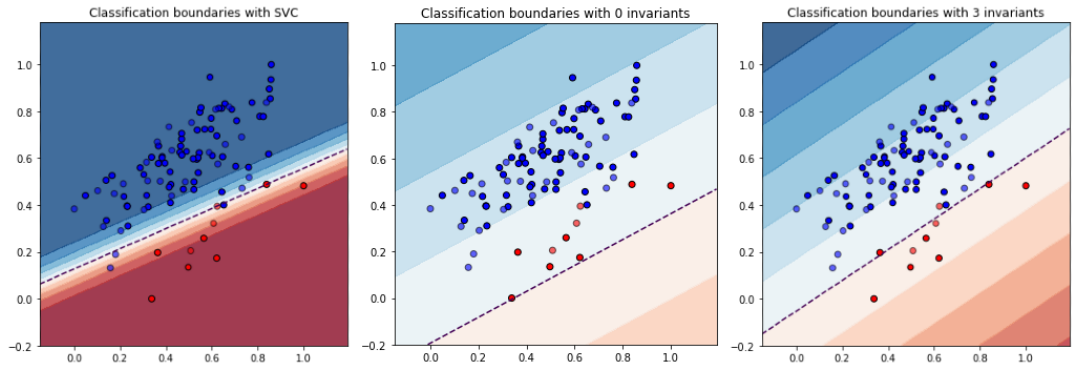

FIGURE 3.3: Classification boundaries obtained with the different versions of SVM in the second problem.

The scikit-learn implementation (left figure) achieves a 90% accuracy on the test set. In our implementation, it can be noted that the solution obtained when not considering predicates (center figure) does not take into account the red points, classifying all points as blue ones. The reason for that behaviour is that, in imbalanced problems, one can obtain high accuracy considering only the majority class, as occurs in this case, where we obtained a 90% accuracy. The use of predicates solves this problem, as can be seen in the right figure. The right picture shows a more appropriate classification boundary, which gives a 97% accuracy, better than the one obtained with `SVC`.

3.4.3 Results in the circle problem

The objective of this last problem is to test non-linear solutions in our implementation. Thus, we are using the `rbf` kernel with `C=0.01` and `gamma=1/2`.

The boundaries obtained are depicted in figure 3.4.

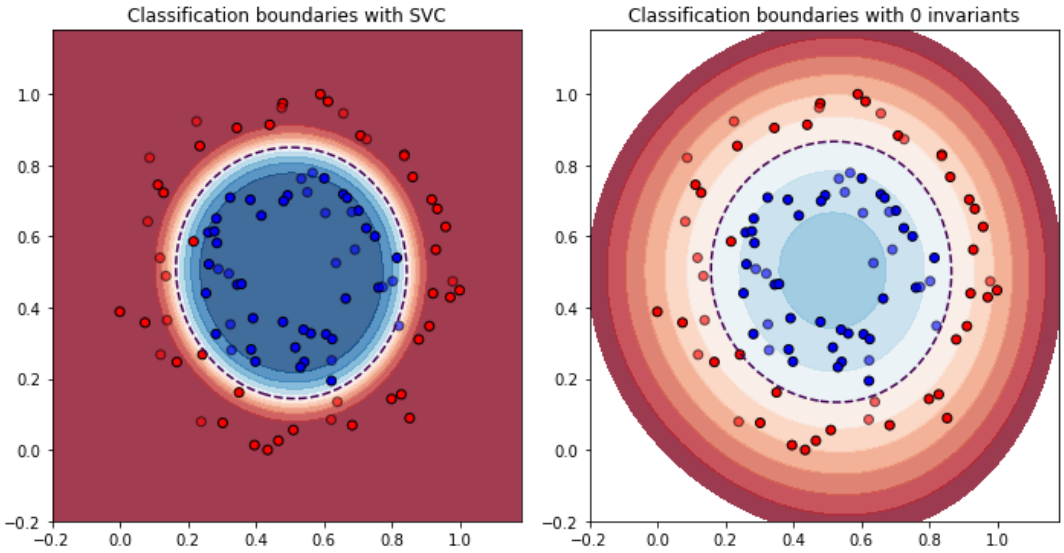

FIGURE 3.4: Classification boundaries obtained with different implementations of SVM in the third problem.

It is interesting that, in this case, our algorithm has not selected any predicate, as the disagreement values (see 3.2) are close to 0. That is, the invariants are satisfied without being included in the algorithm, which achieves a perfect classification on the test set. The scikit-learn implementation also achieves a perfect score, with a similar classification boundary.

Chapter 4

Results

Once the technical part has been explained, we can now proceed to test several machine learning models to reproduce and improve the results of the TiC-Onco risk score.

4.1 Validating TiC-Onco score results

The first step before trying to improve the results is to validate them. This section covers this process, showing the results obtained and comparing them with those reported in [4].

As indicated in [4], *"the genetic risk score and the clinical variables selected were subjected to multivariate logistic regression analysis using an AIC-based backward selection process"*. In this case, we are using a simple logistic regression model to replicate the results. This is approximate, since we do not have the exact details and the code that was used for this process, but it allows us to see that there are no major differences in the results obtained.

The following metrics are used in the paper: AUC, sensitivity, specificity, PPV, NPV, PLR (Positive Likelihood Ratio) and NLR (Negative Likelihood Ratio). Here, we have considered the first five, including the 95% confidence interval, as in the paper.

The authors in [4] use bootstraping to validate the results, considering 100 resamples from the original data. However, we have considered another approach, based on cross validation, as we believe that the previous procedure can lead to optimistic results, as we have explained in section 2.4.

Table 4.1 summarizes the results reported in [4] and our results, using the bootstrap approach (as in the paper) and 10-fold cross validation, which provides a more realistic estimate.

	Original	Bootstrapping	Cross validation
AUC (95% CI)	0.73 (0.67-0.79)	0.75 (0.70-0.81)	0.68 (0.48-0.89)
Sensitivity (95% CI)	0.49 (0.38-0.61)	0.53 (0.39-0.67)	0.34 (0.00-0.71)
Specificity (95% CI)	0.81 (0.77-0.86)	0.80 (0.79-0.82)	0.80 (0.73-0.86)
PPV (95% CI)	0.37 (0.27-0.47)	0.38 (0.30-0.46)	0.27 (0.01-0.52)
NPV (95% CI)	0.88 (0.84-0.92)	0.88 (0.84-0.92)	0.84 (0.76-0.92)

TABLE 4.1: Results comparison for the TiC-Onco risk score.

The results obtained using bootstrapping are close to those reported in [4]. The variations may be due to randomness in the selection of the sets used during for bootstrapping, as well as minor differences in the model considered. However, the results change significantly when cross validation is used. Here we can check that the previous results are indeed optimistic. The AUC, sensitivity and PPV are significantly lower, and provide a more realistic estimate of the accuracy of the TiC-Onco risk score. The specificity and NPV present similar values because there is a trade-off between these metrics and the previous ones. Another aspect to consider is the wide range observed in the confidence intervals when using cross validation, indicating more variability in the results.

These last results obtained with 10-fold cross validation will be used as baseline for the next experiments.

We can validate also the results reported in [4] for the Khorana score, using bootstrapping.

	Original	Validation
AUC (95% CI)	0.58 (0.51-0.65)	0.53 (0.49-0.57)
Sensitivity (95% CI)	0.23 (0.13-0.32)	0.24 (0.16-0.32)
Specificity (95% CI)	0.82 (0.78-0.86)	0.82 (0.78-0.86)
PPV (95% CI)	0.22 (0.12-0.31)	0.24 (0.16-0.31)
NPV (95% CI)	0.83 (0.78-0.87)	0.82 (0.78-0.86)

TABLE 4.2: Results comparison for the Khorana score.

Here the results are almost identical, except for the AUC score, where we obtained a slightly lower value. Nonetheless, we can verify that the TiC-Onco score actually improves significantly the Khorana score.

4.2 Testing some machine learning models

Once we have defined a baseline, the next step is to test several machine learning models, trying to improve the scores obtained.

Finding the right estimator is one of the hardest part when dealing with a machine learning problem, as each estimator may be better suited for different types of data and different problems. In our case, we have followed some of the recommendations of the scikit-learn algorithm cheat-sheet guide.

The approach adopted is as follows: for each estimator under consideration, we performed 10-fold cross validation to find the best hyperparameters, selecting as best model the one with the highest AUC score. Then, we tested the model using 10-fold cross validation again (with different partitions of data), to get reliable results.

Below there is a description of all the models used and the results obtained with each one. At the end of the chapter there is a comparative table containing all the results.

4.2.1 Basic models for classification

Let's start with some classic classifiers, such as logistic regression and support vector machine.

Logistic regression

The first step we took was to tune the hyperparameters of the logistic regression model we used for the baseline. We also considered another version with the data scaled in the range $[0, 1]$.

The best values found for the hyperparameters considered in each case are described in table 4.3.

Estimator	Hyperparameter	Best value
LR	`penalty` (norm used in the penalization)	l2
	`C` (inverse of regularization strength)	0.00625
	`class_weight` (weight associated with each class)	{1:8}

Estimator	Hyperparameter	Best value
LR (normalized data)	`penalty` (norm used in the penalization)	l2
	`C` (inverse of regularization strength)	0.00202
	`class_weight` (weight associated with each class)	{1:10}

TABLE 4.3: Tuned hyperparameters for both versions of logistic regression.

In both cases, the L_2 norm (or euclidean norm) was selected for the penalty on the weights. The low values of `C` indicate a strong regularization, forcing the weights associated with each variable to be small. Regarding the `class_weight` hyperparameter, the positive class receives 8 (10 in the second version) times more importance than the negative class, in order to adjust the imbalance between the two classes.

Figures 4.1 and 4.2 show the results obtained with each version.

score	mean	95% CI
AUC	0.67	(0.44,0.91)
accuracy	0.72	(0.64,0.79)
sensitivity	0.33	(0,0.71)
specificity	0.80	(0.76,0.85)
PPV	0.26	(-0.0,0.52)
NPV	0.84	(0.76,0.92)

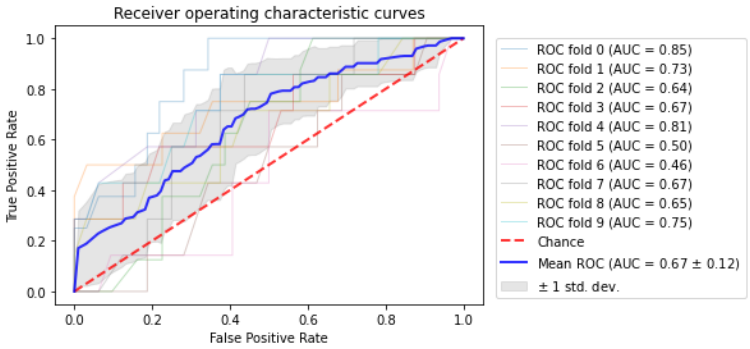

FIGURE 4.1: Results obtained with logistic regression using 10-fold cross validation.

score	mean	95% CI
AUC	0.69	(0.42,0.97)
accuracy	0.71	(0.58,0.84)
sensitivity	0.41	(0,0.9)
specificity	0.78	(0.72,0.84)
PPV	0.28	(0,0.59)
NPV	0.85	(0.74,0.97)

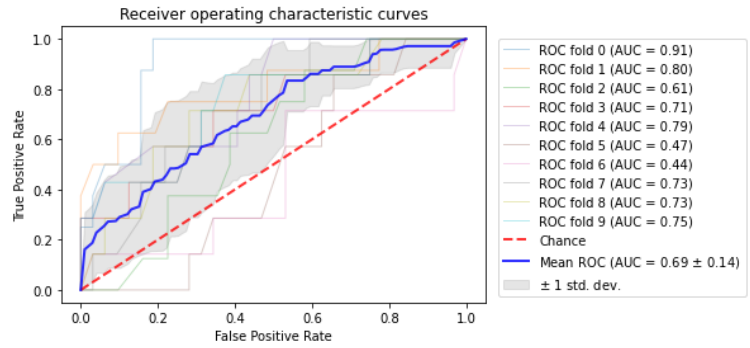

FIGURE 4.2: Results obtained with logistic regression using 10-fold cross validation and scaled data.

Only the second version outperforms baseline results, specially regarding sensitivity, where the difference is significant.

We can appreciate how the ROC curves change significantly with each partition of the data. For example, in the second version, the fold 0 gives an AUC of 0.91 while the fold 6 leads to an AUC of 0.44. This shows that the results are strongly influenced by the subset of the data considered, which translates into wide confidence intervals, as it was observed in the previous section.

Linear SVC

We repeated the process using linear support vector classification. Support vector machines assume that the data is in a standard range, usually $[0, 1]$ or $[-1, 1]$, so we have scaled feature vectors again in the range $[0, 1]$.

The norm selected for penalization is the L_2 norm. The regularization strength and the weight associated with the positive class remain high (see table 4.4).

Estimator	Hyperparameter	Best value
Linear SVC	`penalty` (norm used in the penalization)	l2
	`C` (inverse of regularization strength)	0.00146
	`class_weight` (weight associated with each class)	{1:8}

TABLE 4.4: Tuned hyperparameters for linear SVC.

Results obtained with this estimator are shown in figure 4.3. They are nearly the same as those obtained with the best version of logistic regression (see figure 4.2).

score	mean	95% CI
AUC	0.69	(0.42,0.97)
accuracy	0.71	(0.58,0.84)
sensitivity	0.41	(0,0.9)
specificity	0.78	(0.72,0.84)
PPV	0.28	(0,0.59)
NPV	0.85	(0.74,0.97)

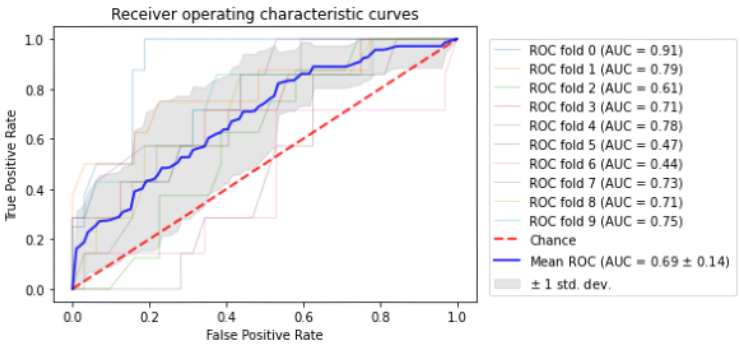

FIGURE 4.3: Results obtained with linearSVC using 10-fold cross validation.

K-Nearest Neighbors classifier

Now we consider a totally different classifier, based on k-nearest neighbors. It relies on majority voting based on the class of the k nearest samples for a given data point.

As we did with logistic regression, we will consider two versions, the second one with the data scaled in the range $[0, 1]$.

The hyperparameters are different in this case, controlling the number of neighbors considered and the way to calculate distances between them. Table 4.5 shows the selected hyperparameters for both versions.

Estimator	Hyperparameter	Best value
KNN classifier	`n-neighbors` (number of neighbors to use)	163
	`p` (power parameter for the Minkowski metric)	1
	`weights` (weight function used in prediction)	uniform

Estimator	Hyperparameter	Best value
KNN classifier (normalized data)	`n-neighbors` (number of neighbors to use)	219
	`p` (power parameter for the Minkowski metric)	2
	`weights` (weight function used in prediction)	uniform

TABLE 4.5: Tuned hyperparameters for both versions of KNN classifier.

It should be appreciated the high number of neighbors considered (more than half of the dataset in the second case), which lead to smoother decision boundaries. This shows that it is difficult to classify an example based only on its immediate environment. The metric used for computing the distances is the Manhattan distance (L_1 metric) in the first version and the Euclidean distance (L_2 metric) in the second one. Both versions use uniform weights, meaning that all points in the neighborhood are weighted equally when voting.

The results obtained with these estimators are shown in figures 4.4 and 4.5.

score	mean	95% CI
AUC	0.65	(0.44,0.85)
accuracy	0.73	(0.65,0.81)
sensitivity	0.35	(0.03,0.68)
specificity	0.82	(0.74,0.9)
PPV	0.30	(0.09,0.5)
NPV	0.85	(0.79,0.91)

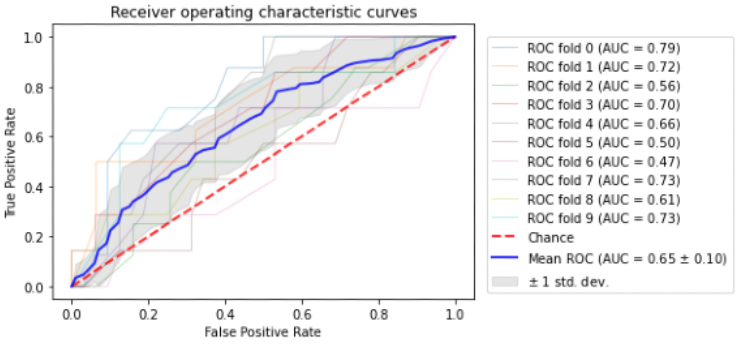

FIGURE 4.4: Results obtained with KNN classifier using 10-fold cross validation.

score	mean	95% CI
AUC	0.66	(0.47,0.86)
accuracy	0.72	(0.64,0.8)
sensitivity	0.38	(0,0.76)
specificity	0.80	(0.73,0.86)
PPV	0.28	(0.03,0.54)
NPV	0.85	(0.78,0.92)

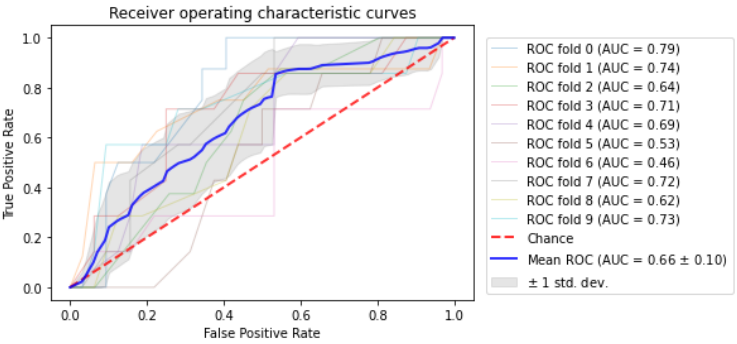

FIGURE 4.5: Results obtained with KNN classifier using 10-fold cross validation and data scaled.

Both versions lead to similar results, with the second one having a slightly better AUC and sensitivity scores, and the first one having a higher PPV. In any case, both

estimators offer a modest improvement on the baseline in all scores considered except the AUC, which is slightly lower.

Support Vector Machine

Support vector machines one of the most robust prediction methods available. The difference with respect to linearSVC is that we can now use kernels to obtain representations of the data in high-dimensional feature spaces, which is a very powerful tool. Again, we are working with the data scaled in the range $[0, 1]$.

Several kernels have been tested, as well as other hyperparameters for the estimator. The selected ones are shown in the table 4.6.

Estimator	Hyperparameter	Best value
SVC	`C` (inverse of the regularization parameter)	0.01
	`kernel` (kernel type to be used in the algorithm)	rbf
	`gamma` (kernel coefficient)	0.1
	`class_weight` (weight of each class)	{1:5}

TABLE 4.6: Tuned hyperparameters for support vector classifier.

Results obtained (figure 4.6) situate this estimator next to the baseline. Only a small improvement in the sensitivity and PPV is observed.

score	mean	95% CI
AUC	0.68	(0.47,0.9)
accuracy	0.72	(0.58,0.85)
sensitivity	0.37	(0,0.78)
specificity	0.80	(0.7,0.89)
PPV	0.29	(0,0.58)
NPV	0.85	(0.75,0.94)

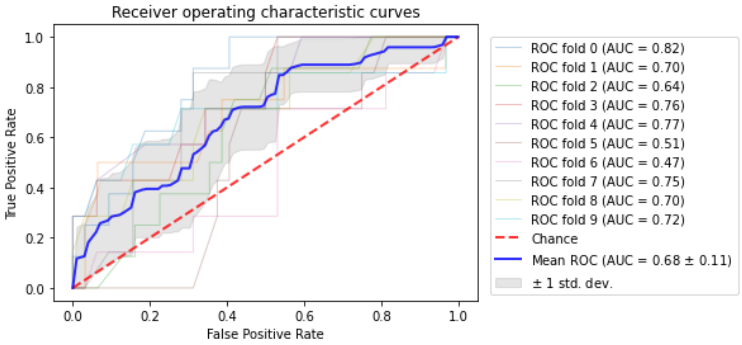

FIGURE 4.6: Results obtained with SVM classifier using 10-fold cross validation.

Until now, the best estimators found have been logistic regression (fig. 4.2) and linear SVC (fig. 4.3), reaching an AUC of 0.69, which underlines the effectiveness of linear models in this particular data set.

4.2.2 Ensemble classifiers

Following the guide mentioned above, we tested some ensemble methods. These kind of classifiers (also known as meta-estimators) allow to combine the predictions of several base estimators in order to improve the generalization capacity and the robustness of a single estimator.

Bagging classifier

This meta-estimator fits several base classifiers on random subsets of the original dataset (using random subsets of features) and then aggregate their individual predictions, either by voting or by averaging, to output the final prediction.

Three different base estimators have been tested: a decision tree classifier, the tuned version of logistic regression (table 4.3 right) and the tuned version of k-neighbors classifier (table 4.5 left).

Various combinations of the number of base estimators and subsets of the dataset and features have been evaluated. Here we only show the results of the best version, obtained when using logistic regression as base estimator.

Table 4.7 indicates the best hyperparameters for this meta-estimator: we are using 40 base estimators, each one trained using 80% of the samples in the dataset (drawn with replacement) and 90% of the features (without replacement).

Estimator	Hyperparameter	Best value
Bagging (LR)	`n_estimators` (number of base estimators)	40
	`max_samples` (proportion of samples used in each base estim.)	0.8
	`max_features` (proportion of features used in each base estim.)	0.9

TABLE 4.7: Tuned hyperparameters for bagging using logistic regression as base classifier.

The results obtained are provided in figure 4.7. We can see that they are slightly better than those obtained with logistic regression, situating this model in the first place so far.

score	mean	95% CI
AUC	0.69	(0.42,0.97)
accuracy	0.72	(0.59,0.84)
sensitivity	0.41	(0,0.9)
specificity	0.79	(0.71,0.86)
PPV	0.29	(0,0.59)
NPV	0.86	(0.74,0.97)

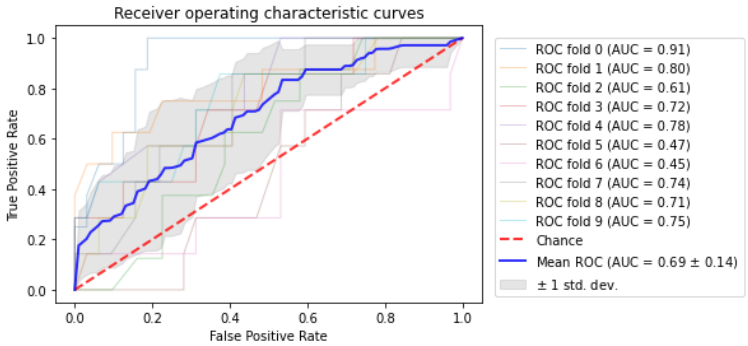

FIGURE 4.7: Results obtained with bagging classifier using 10-fold cross validation.

Random Forest & Extra-Trees classifier

These two algorithms are based on randomized decision trees. In both cases, a diverse set of classifiers is created selecting various sub-samples of the dataset for training each base classifier. The extra-trees classifier also introduces randomness in the classifier construction. The final prediction is given by computing the average of the individual predictions.

After testing both meta-estimators, the extra-trees classifier obtained the best results, which are described below.

Table 4.8 includes the hyperparameter selection. We can see that it uses 5 simple decision trees (with a maximum depth of 2), considering 3 features when looking for the best split. The positive class is given 10 times more importance than the negative class.

Estimator	Hyperparameter	Best value
Extra-trees classifier	`n_estimators` (number of base estimators)	5
	`max_depth` (maximum depth of the tree)	2
	`max_features` (proportion of features used in each base estim.)	sqrt
	`class_weight` (weights of each class)	{1:10}

TABLE 4.8: Tuned hyperparameters for extra-trees classifier.

Figure 4.8 shows the results. The model improves all the scores except the AUC of the baseline. The specificity of this model is particularly high.

score	mean	95% CI
AUC	0.67	(0.43,0.91)
accuracy	0.74	(0.69,0.79)
sensitivity	0.39	(0,0.82)
specificity	0.82	(0.73,0.9)
PPV	0.30	(0.07,0.54)
NPV	0.86	(0.78,0.93)

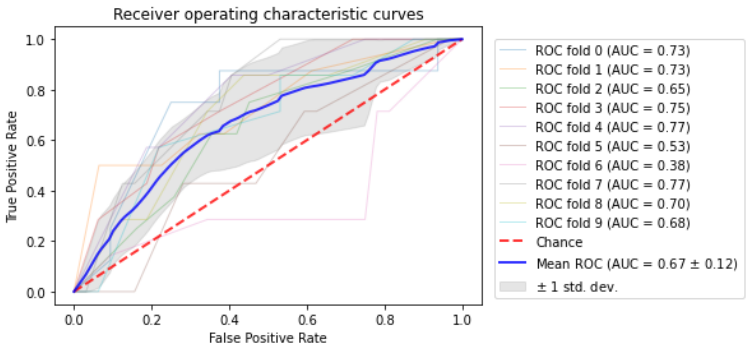

FIGURE 4.8: Results obtained with extra-trees classifier using 10-fold cross validation.

AdaBoost

The AdaBoost classifier is a meta-estimator that fits an initial classifier on the original data and then fits additional copies of the classifier focusing on the difficult cases (incorrectly classified instances).

These are the best hyperparameters found for AdaBoost:

Estimator	Hyperparameter	Best value
AdaBoost	`n_estimators` (number of base estimators)	30
	`learning_rate` (weight applied to each classifier at each boosting iteration)	1.6

TABLE 4.9: Tuned hyperparameters for the AdaBoost classifier.

Results are good in terms of AUC, but not the best ones with respect the other metrics, as can be seen in figure 4.9.

Gradient boosting

Gradient Boosting works using an additive model in a forward stage-wise fashion, (here you can find a detailed explanation).

score	mean	95% CI
AUC	0.69	(0.51,0.87)
accuracy	0.72	(0.65,0.79)
sensitivity	0.37	(0.03,0.71)
specificity	0.80	(0.76,0.84)
PPV	0.29	(0.08,0.49)
NPV	0.85	(0.78,0.92)

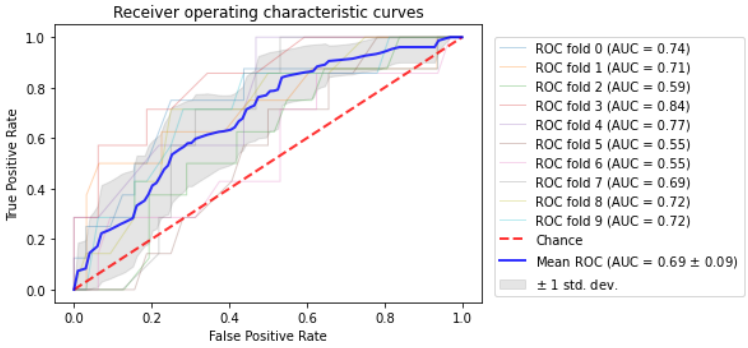

FIGURE 4.9: Results obtained with AdaBoost using 10-fold cross validation.

The hyperparameters selected are explained in table 4.10 and the results obtained with this classifier are detailed in figure 4.10. The AUC score obtained is below the baseline, while the sensivity and PPV are slighlty improved. Thus, this estimator is far away from being the best model.

Estimator	Hyperparameter	Best value
Gradient Boosting	`n_estimators` (number of base estimators)	15
	`learning_rate` (shrinks the contribution of each tree)	0.2
	`max_depth` (maximum depth of each tree)	2
	max_features (features considered for splitting	0.8
	`subsample` (fraction of data used for fitting base classifiers)	0.5

TABLE 4.10: Tuned hyperparameters for Gradient Boosting classifier.

score	mean	95% CI
AUC	0.65	(0.47,0.83)
accuracy	0.71	(0.63,0.78)
sensitivity	0.30	(0,0.69)
specificity	0.80	(0.73,0.87)
PPV	0.23	(0,0.49)
NPV	0.84	(0.76,0.91)

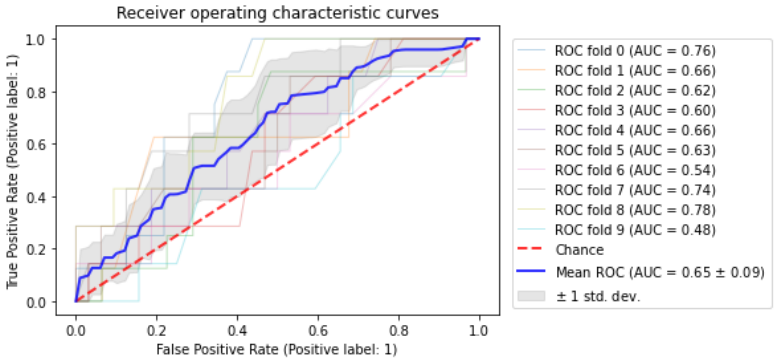

FIGURE 4.10: Results obtained with Gradient Boosting classifier using 10-fold cross validation.

Voting & Stacking classifiers

We have considered these last two ensemble techniques, which allow us to combine different classifiers.

The idea behind the voting classifier is to combine different classifiers using the average predicted probabilities (soft vote) to predict the final class labels.

On the other hand, stacking classifiers use the predictions of each individual estimator, which are stacked together and used as input to a final estimator to compute the prediction.

Among all the combinations of classifiers that we have tested with these methods, none of them improves the results of the best classifiers used individually (although they are very close in some cases), so we will not detail here the results, since in this case no hyperparameter tuning is needed. Table 4.13 at the end of this chapter shows these results for comparison purposes.

4.3 Results obtained with LUSI

This section summarizes the results obtained with the SVM&I_n algorithm explained in chapter 3. The idea is to find some predicates that allow us to outperform the models in the previous section.

We have tested several SVM&I_n classifiers, considering different sets of predicates. The process is summarized below.

4.3.1 Basic predicates

We started by considering two basic sets of predicates:

P1) Predicates based on zeroth and first order moments.

This set of predicates was the one used in the examples in chapter 3. It consists of $d+1$ predicates, where d is the dimension of the data. One predicate is defined by $\psi_0(x) = 1$ and the others d predicates are given by $\psi_k(x) = x_k, \quad k = 1, \ldots, d$. Our dataset has 9 features, so we end up with 10 predicates in total.

The invariant associated with the first predicate defines the set of functions that preserve the frequency of samples belonging to class 1, while the rest of invariants preserve the mean values of each feature among samples of class 1.

P2) Predicates based on local structure.

Here, $\psi_k(x_i)$ is defined as the number of vectors of the positive class among the k nearest neighbors of the point x_i. We are considering $k = 1, \ldots, 11$, that is, 10 predicates in total.

This kind of predicates provide information about the local structure of the conditional probability function $P(y = 1|x)$.

Our implementation of the SVM&I_n algorithm includes an hyperparameter to select whether to use the set of predicates P1 or P2, so we have included it in the hyperparameter tuning process. The other hyperparameters are associated with the kernel and the regularization applied. Table 4.11 lists the values that have been selected.

The high value of `C` indicates that a strong regularization is being applied. The model uses a rbf kernel and the set of predicates based on local structure (P2) is selected. In fact, we can check that only 8 of the total 10 predicates have been used in this case (the invariants associated with two of them are already met).

Figure 4.11 provides the results obtained with this first version of the LUSI paradigm. The model is close to the best ones, but there is still room for improvement.

Estimator	Hyperparameter	Best value
SVM&I_8	C (regularization strength)	942.67
	kernel (kernel type)	rbf
	gamma (parameter for rbf kernel)	0.6
	pred_nn (whether to use set of predicates based on KNN or not)	True

TABLE 4.11: Tuned hyperparameters for SVM&I_8 classifier.

score	mean	95% CI
AUC	0.68	(0.41,0.95)
accuracy	0.71	(0.63,0.79)
sensitivity	0.37	(0.01,0.74)
specificity	0.79	(0.76,0.82)
PPV	0.28	(0.08,0.47)
NPV	0.85	(0.76,0.93)

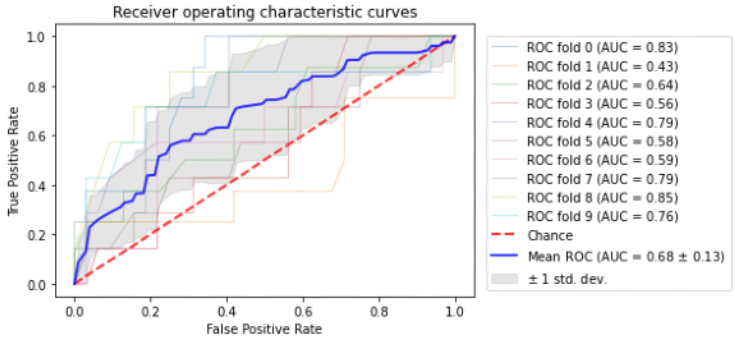

FIGURE 4.11: Results obtained with SVM&I_8 classifier using 10-fold cross validation.

4.3.2 Custom predicates

Let's consider some custom predicates, specifically designed for this problem. This is where the power of this algorithm lies, because it is possible to add some specific information to the algorithm, acting as a guide during the learning process.

Concretely, we have tested the following predicates:

P3) Predicates based on features with low p-value.

The idea behind this kind of predicates is to select those candidate functions that preserve some characteristics of the features with a low p-value (tumour stage, rs4524 and primary tumour site VHR, as reported in [4]), in order to better reflect changes in their values.

P3.1) Preserve risk values: we want to maintain the proportion of patients with risk values in those variables that are predicted to be in the positive class.

$$\psi(x) = \begin{cases} 1 & if \quad x_{stage} > 0,\ x_{rs4524} > 0 \text{ and } x_{VHR} > 0 \\ 0 & otherwise \end{cases}$$

P3.2) Preserve normal values: we keep the proportion of patients with normal values in those variables predicted to be in the positive class.

$$\psi(x) = \begin{cases} 1 & if \quad x_{stage} = 0,\ x_{rs4524} = 0 \text{ and } x_{VHR} = 0 \\ 0 & otherwise \end{cases}$$

P4) Predicates based on the genetic risk score.

Here we focus on some characteristics of the genetic variables: rs2232698, rs4524, rs5985 and rs6025.

P4.1) Preserve total genetic risk score: here we select functions that preserve the total genetic risk score (sum of all the risk alleles) for patients in the positive class.

$$\psi(x) = x_{rs2232698} + x_{rs4524} + x_{rs5985} + x_{rs6025}$$

P4.2) Preserve normal values: select functions that preserve the proportion of patients with normal values in the genetic variables belonging to the positive class.

$$\psi(x) = \begin{cases} 1 & if \quad x_{rs2232698} = 0,\ x_{rs4524} = 0, \\ & \quad x_{rs5985} = 0 \text{ and } x_{rs6025} = 0 \\ 0 & otherwise \end{cases}$$

P5) Combinations of the previous predicates.

Among all these options, the one that provided the best results was using the predicates based on local structure (P2) and the predicate that preserve the number of samples of the class 1 with normal values on the genetic variables (P4.2).

The hyperparameters selected in this case are described in table 4.12. Now, the amount of regularization applied is not so high, and the total number of predicates used has been 10.

Estimator	Hyperparameter	Best value
SVM&I_{10}	C (regularization strength)	55.41
	kernel (kernel type)	rbf
	gamma (parameter for rbf kernel)	0.5
	pred_nn (whether to use set of predicates based on KNN or not)	True

TABLE 4.12: Tuned hyperparameters for SVM&I_{10} classifier.

Figure 4.12 contains the results. We have obtained a model that significantly improves the previous results, specially the sensitivity, which increases up to 0.49.

score	mean	95% CI
AUC	0.71	(0.51,0.91)
accuracy	0.74	(0.66,0.82)
sensitivity	0.49	(0.1,0.88)
specificity	0.80	(0.74,0.85)
PPV	0.34	(0.12,0.56)
NPV	0.87	(0.79,0.95)

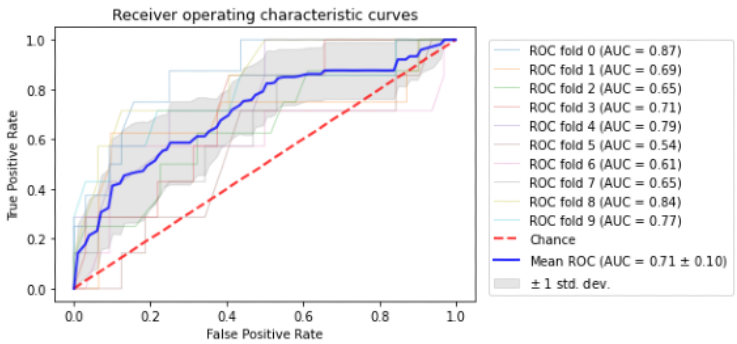

FIGURE 4.12: Results obtained with SVM&I_{10} classifier using 10-fold cross validation.

4.4 Models comparison

Finally, table 4.13 provides a summary of the results obtained with all the models tested, in order to compare them.

Model	AUC	accuracy	sensitivity	specificity	PPV	NPV
Baseline (LR)	0.68 +- 0.10	0.71	0.34	0.80	0.27	0.84
LR	0.67 +- 0.12	0.72	0.33	0.80	0.26	0.84
LR (scaled)	0.69 +- 0.14	0.71	0.41	0.78	0.28	0.85
Linear SVC	0.69 +- 0.14	0.71	0.41	0.78	0.28	0.85
KNN	0.65 +- 0.10	0.73	0.35	**0.82**	0.30	0.85
KNN (scaled)	0.66 +- 0.10	0.72	0.38	0.80	0.28	0.85
SVC	0.68 +- 0.11	0.72	0.37	0.80	0.29	0.85
Bagging	0.68 +- 0.11	0.72	0.41	0.79	0.29	0.85
Bagging (LR)	0.69 +- 0.14	0.72	0.41	0.79	0.29	0.86
Bagging (KNN)	0.67 +- 0.13	0.73	0.37	0.81	0.29	0.85
Random Forest	0.65 +- 0.12	0.72	0.38	0.80	0.29	0.85
Extra Trees	0.67 +- 0.12	**0.74**	0.39	**0.82**	0.30	0.86
AdaBoost	0.69 +- 0.09	0.72	0.37	0.80	0.29	0.85
Grad. Boosting	0.66 +- 0.12	0.73	0.39	0.80	0.29	0.85
Voting (LR+KNN+ET)	0.67 +- 0.13	0.73	0.38	0.81	0.31	0.85
Voting (LR+KNN+bagging_lr)	0.69 +- 0.13	0.73	0.39	0.80	0.30	0.85
Stacking (linSVC+bagging_lr)	0.69 +- 0.14	0.71	0.37	0.79	0.28	0.85
SVM&I_8 (KNN)	0.68 +- 0.13	0.71	0.37	0.79	0.28	0.85
SVM&I_{10} (KNN + gen=0)	**0.71 +- 0.10**	**0.74**	**0.49**	0.80	**0.34**	**0.87**

TABLE 4.13: comparison of all machine learning models tested.

In view of the results, we can conclude that SVM&I_{10} (fig. 4.12) provides the best results for our problem, outperforming the rest of estimators in all the metrics considered (except specificity). Other remarkable estimators are extra-trees, that gives the best results in terms of accuracy and specificity, and those based on linear models such as logistic regression and linear support vector classification.

Chapter 5

Conclusions and future work

This final chapter contains a summary of the conclusions drawn from the entire process, as well as possible improvements for the future.

5.1 Conclusions

The machine learning model developed in section 4.3.2 outperforms all previously tested tools including the Khorana and TiC-Onco risk scores, according to the results obtained (see table 4.13). Nevertheless, it is worth noting that the high variability of the results obtained (wide ranges in the confidence intervals) makes comparison difficult, as there may be certain subsets of data where one of the previous models performs better than ours. Even so, the procedure followed guarantees that in most cases our tool will be the one that obtains the best results, provided that the metrics described in the previous sections are used.

With respect to the initial TiC-Onco score results obtained with bootstrapping (table 4.1), they are still slightly higher than those we have obtained with our methodology, which supports our assumption about the optimistic bias introduced by the evaluation method used in [4].

Concerning the new machine learning paradigm we have tested (LUSI), our experience has been positive because, despite the fact that we have very few data, this approach is capable of extracting a lot of information from it when the appropriate predicates are used.

In conclusion, we believe that our model could be used as as a complementary tool when determining the risk of VTE, contributing to detect cancer patients that are at high risk of VTE. These patients would receive the appropriate treatment, leading to important improvements in their quality of life.

5.2 Further work

Here are some suggestions for possible improvements in the future:

- The SVM&I_n algorithm we have used can be optimized in several ways. We have limited us to implement the closed-form solution described in [8]. Other approaches may consider different flavors for the algorithm, some of them leading to faster and more stable implementations (for example, use stochastic gradient descent instead of the exact solution). In our case, as we have worked

with very few data (less than 400 samples), the execution times have been reasonable, but maybe our implementation is unpractical when considering more data.

- Regarding the LUSI approach, experimentation with more elaborate predicates, based on certain assumptions or expert knowledge, may lead to further improvement of the model.
- The dataset we were given contains lot of information that we have not used for the problem. For example, we have considered only 9 variables out of the total of 90 available, in order to compare our results with those reported in [4]. Although we did some tests considering more data, we did not obtain good results, but perhaps a more detailed study could lead to improve the results obtained.
- Another important point would be to create a larger database for this problem, which would help to obtain more robust results, and also to test different methodologies, such as a deep learning models.

Appendix A

Developed software

Several software tools have been developed to perform the experiments in chapter 4, as well as to generate the dataset. Here we are going to briefly describe the code, which is organized in two python modules and four Jupyter notebooks.

All the code is available in this GitHub repository.

The python modules developed are the following:

- `lusi.py` contains all the code of the SVM&I_n algorithm implemented for the LUSI approach, explained in chapter 3 and used in the last experiments in chapter 4. Concretely, it contains the implementation of the classes `SVM_I` and `SVM`, with this last one being the SVM algorithm obtained when not considering predicates in the learning process.
- `utils.py` defines the rest of functions used during the work. Some of them are used to analyze the data (`print_summary` and `corr_heatmap`), others to validate the results in `paper` (`test_khorana_bootstrap` and `test_model_bootstrap`) and the last one for evaluating the models used in chapter 4 (`test_model`).

The Jupyter notebooks are described below:

- The notebook `data_preprocessing.ipynb` is used to create the dataset that will be used during the project (stored in file `data/data_TiC_Onco.csv`). Concretely, the file with the data provided by the authors of [4] is read and preprocessed, in order to obtain the desired variables, as explained in section 2.2. It also includes a simple analysis of the data.
- The notebook `SVM_I.ipynb` was designed for the experiments performed with the LUSI approach, in order to test the SVM&I_n algorithm (see section 3.4).
- The notebook `validating_results.ipynb` contains the validation of the results reported in [4] for the Khorana and TiC-Onco risk scores, using different evaluation techniques, as detailed in section 4.1.
- Lastly, the notebook `improving_TiC_Onco_score.ipynb` collects all the experiments performed in chapter 4, with all the results obtained to be reviewed if necessary[1].

[1] In all the procedures involving a random component, a certain random seed has been fixed, which allows to reproduce the experiments performed obtaining the same exact results.

Bibliography

[1] Jeanet W. Blom et al. "Malignancies, Prothrombotic Mutations, and the Risk of Venous Thrombosis". In: *JAMA* 293.6 (Feb. 2005), pp. 715–722. ISSN: 0098-7484. DOI: 10.1001/jama.293.6.715. eprint: https://jamanetwork.com/journals/jama/articlepdf/200333/joc42103.pdf. URL: https://doi.org/10.1001/jama.293.6.715.

[2] Alok A. Khorana et al. "Development and validation of a predictive model for chemotherapy-associated thrombosis". In: *Blood* 111.10 (May 2008), pp. 4902–4907. ISSN: 0006-4971. DOI: 10.1182/blood-2007-10-116327. eprint: https://ashpublications.org/blood/article-pdf/111/10/4902/1294353/zh801008004902.pdf. URL: https://doi.org/10.1182/blood-2007-10-116327.

[3] A. J. Muñoz Martín et al. "Incidence of venous thromboembolism (VTE) in ambulatory pancreatic cancer patients receiving chemotherapy and analysis of Khorana's predictive model". In: *Clin Transl Oncol* 16.10 (Oct. 2014), pp. 927–930.

[4] A. J. Muñoz Martín et al. "Multivariable clinical-genetic risk model for predicting venous thromboembolic events in patients with cancer". In: *Br J Cancer* 118.8 (Apr. 2018), pp. 1056–1061.

[5] I. Pabinger et al. "Factor V Leiden mutation increases the risk for venous thromboembolism in cancer patients - results from the Vienna Cancer And Thrombosis Study (CATS)". In: *J Thromb Haemost* 13.1 (Jan. 2015), pp. 17–22.

[6] G. Ugarte Fornell et al. "Predictive Khorana's model in patients with venous thromboembolic disease and cancer". In: *Med Clin (Barc)* 141.11 (Dec. 2013), pp. 479–481.

[7] Vladimir Vapnik and Rauf Izmailov. "Complete statistical theory of learning: learning using statistical invariants". In: *Proceedings of the Ninth Symposium on Conformal and Probabilistic Prediction and Applications*. Ed. by Alexander Gammerman et al. Vol. 128. Proceedings of Machine Learning Research. PMLR, Sept. 2020, pp. 4–40. URL: http://proceedings.mlr.press/v128/vapnik20a.html.

[8] Vladimir Vapnik and Rauf Izmailov. "Rethinking statistical learning theory: learning using statistical invariants". In: *Machine Learning* 108 (Mar. 2019). DOI: 10.1007/s10994-018-5742-0.