Design, implementation and evaluation of data integration methods for biomedical cancer data

Michiel Ruelens

Preface

I would like to thank all of my family and friends that supported me throughout the making of this thesis. I would also like to thank the jury for taking the time to read and evaluate the text.

Next I would like to thank my promotor Roel Wuyts for allowing me to make the thesis on this self-chosen topic and for providing very helpful feedback throughout the year and in the writing of the text.

Last, but not least, I would like to express my deepest gratitude towards my second promotor Olivier Gevaert for giving me the opportunity to explore the world of cancer research and allowing me to visit the Gevaert research lab in San Fransisco during the month of March. On one hand this allowed me to really experience a research environment and meet some amazing people. On the other hand I also got to experience the life and environment in silicon valley, which is obviously a dream for anyone aspiring to be an engineer.

Michiel Ruelens

Contents

Abstract

Technological advancements are causing datasets to become larger and larger. In the biomedical field of cancer research this is reaching the point where we are able to characterize a cancer on a multilevel scale. On the smallest scale we are able to sequence entire genomes of patients and quantify gene expression. On the middle scale we can analyze proteine and other molecule levels in the blood and look at microscopical features of individual cancer cells. And on the large scale we can image the cancer to define surface level features, development stages and indicate spread throughout the body. The challenge we are facing now is to combine all of this knowledge into a single coherent system to provide optimal prognosis prediction, find new insights into the way cancer develops and to come up with novel treatments to fight the disease.

In this thesis we present three different integration strategies that can combine datasets from different sources to build a predictive model. The aim is to study whether different integration strategies have a positive impact on the performance of the resulting models. To do this we performed two case studies on real cancer data. One study focusses on logistic regression to predict a pathological stage outcome and uses ROC analysis to evaluate the models. The second study focusses on cox proportional hazards models to predict survival outcomes and uses statistical significance tests for the evaluation. Both case studies provide evidence that indeed the various integration strategies do have a positive impact on the performance of the resulting models.

List of Figures and Tables

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The need for data integration methods

Current advancements in technology are leading to increasingly larger datasets[22]. This is a trend that is becoming very apparent in many different fields of research. One of these is the biomedical field. By larger datasets we mean that they are increasing both in the number of variables, aswell as in the number of samples that are available. The increase in variable count is mainly due to technology. We are reaching the point where we are able to sequence anyones DNA at a very low cost[40]. Keeping in mind that humans have around 20.000-25.000 proteine-coding genes, it is very common for current genome-datasets to have thousands of variables. Another example is that we have a range of highly advanced imaging instruments in almost every hospital[21][31]. These instruments provide high quality images from which we can extract even more feature variables to analyze. These are just two examples that show the growth of variable count. Next to the explosion of the number of variables, the number of samples that are available in databases grows aswell. This is because we are able to simply store much more data now than we could in the past, and on top of that, there are big efforts going on to make datasets open to the public to help researchers around the world work together[43][39][32][42]. All of these trends lead to huge amounts of data from different sources becoming available for analysis. This raises the question of how we have to combine (or integrate) data from these different sources to get the most information out of them.

1.2 Goals and modus operandi

The aim of this thesis is to develop several integration strategies and show that these strategies have an impact on the performance of predictive models. This aim is encompassed by the bigger study that attempts to find patterns in the huge amounts of cancer data that we have available. By building better predictive models, we can find new insights into the way cancer originates and develops and this will eventually lead to better treatments in the future. Note that there are many kinds of predictive models available in machine learning. There are linear models such as generalized

linear models and perceptrons, and there are non-linear models such as deep neural networks. In this thesis we wil put our focus on two types of models: logistic regression - and cox proportional hazards models. There is no special motivation for choosing these two types of models other than the fact that the data to compute them was available to us. There are many other model types that we could have chosen, and this should indeed be the topic of future research. The first chapter explains the first type of predictive models which are called generalized linear models. More specifically this thesis will focus on the logistic regression model. The next chapter explains the second type of predictive models which are called survival models. In this case we will take a closer look at cox proportional hazards models. Once we have established the notion of predictive modelling, we can define the different integration strategies. This is done in chapter three. In order to investigate whether these integration strategies have an effect on the performance of the predictive models, we will apply them to a real world example in two case studies, this is the content of chapter four. The first case study will use logistic regression to build predictive models using the various integration strategies. The data that is used in this case consists of imaging data taken for patients with colorectal cancer. This data was obtained from the University Hospital in Leuven in order to improve their predictive models for medical decision support. The second case study will create survival models from datasets that are obtained from The Cancer Genome Atlas (TCGA). TCGA is an online, publicly available, database of 2.5 petabytes of data on cancer patients. These datasets mainly focus on anything related to the genome or molecular processes. In both case studies we will construct predictive models for all datasets individually, aswell as constructing predictive models using the various integration strategies. Once we have constructed all these models we will compare them using appropriate metrics. In the case of logistic regression models we will use ROC analysis to evaluate the models. In the case of the survival models we will use the Wald significance test to perform the evaluation.

In order to facilitate all of the previous work we designed an interactive application that streamlines this whole process. The application is capable of comparing the performance of predictive models using the various integration strategies. This allowed us to perform the two case studies, but it also provides a platform for future research.

Chapter 2

Generalized Linear Models

2.1 Introduction

In this chapter we will explain the concepts of generalized linear models [19][2]. This term indicates a generalization of simple linear regression that allows for a wide range of output variables. First we will define exactly what we mean by a predictive model (2.2). Then we will go over the basics of linear models, gradually building up to the definition of generalized linear models (2.3). Next, we will describe what actual data looks like and how a model is trained from it (2.4). After that we will tackle the more recent innovation of regularization that will greatly improve our previous models by exploiting the bias-variance trade-off to reduce overfitting (2.7). Lastly we will outline the validation method that will be used to test the performance of the models (2.8).

2.2 What is a predictive model?

We can define a predictive model in a general way as a mapping from input variables to an output variable. The task of a predictive model is to predict, or estimate, a value for the output variable when given values for the input variables. For example: a predictive model could be asked to estimate the average temperature for a given day. The input in this case is the date, the output is the average temperature. We can imagine that there is a correlation between the two: dates in the summer will tend to result in a higher average temperature, and dates in the winter will have lower average temperatures. We know this because we have years of experience from the past. From this we have learned that winter tends to be cold and summer tends to be warm. This is exactly what we try to capture: tendencies and patterns. Learning from experience is what we try to do when building a predictive model. We call this "training the model". When training a model we show it examples of what we want it to learn. In the example case of average temperature we would show it dates together with the corresponding average temperature for that day. By showing many of these cases to the model, it can build up experience and hopefully it will realize that winter days tend to be cold, and summer days tend to be hot. How the

model remembers these patterns or tendencies (the model representation) depends on the method used to train the model and on the kind of patterns that need to be captured. In the following sections we will show examples of these representations and how they can represent different patterns.

2.3 Classical linear models

In classical linear models we assume that there is a linear relationship between the input variables (also called explanatory variables) and the output variable (also called dependent variable). In this case we can represent the model by a vector of coefficients, one for each input variable. By making a linear combination of the explanatory variables we attempt to estimate a value for the dependent variable. Depending on the type of dependent variable the linear method gets a different name. In the following sections we will outline several of them.

2.3.1 Linear Regression

The simplest version of a linear model is called linear regression [19][8]. In this case the input variables are combined using a linear combination, and the result of this calculation is immediately used as the final estimate. Imagine we have a dataset of size N where each sample has M explanatory variables. Then we can write linear regression as:

$$\hat{y}_n = \sum_{i=1}^{M} w_i x_{in} = \boldsymbol{w^T x_n} \qquad for \ n = 1..N \qquad (2.1)$$

where

- $\hat{y}_n$ is the output for sample n

- w_i is the model parameter (coefficient) for explanatory variable i

- x_{in} is the value for explanatory variable i for sample n

- $\boldsymbol{w^T x_n}$ is the vector notation for the inner product of $\boldsymbol{w}$ and $\boldsymbol{x_n}$

For the other linear methods we will define a function each time that is applied to the result of the linear combination. We could do the same for linear regression and say that the applied function is the identity function. A schema for this computation is shown on figure 2.1.

2.3.2 Linear Classification

The next method is called linear classification [19][7]. The difference with linear regression is that we have a different type of output variable. In a classification task we want to predict a class from a list of potential classes. For instance, we could try
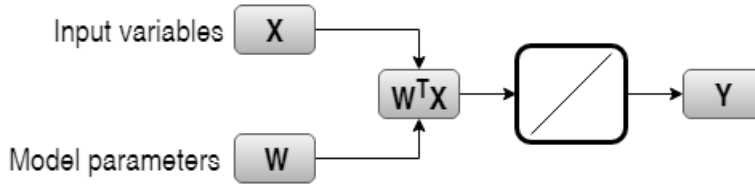
Figure 2.1: Schema for linear regression
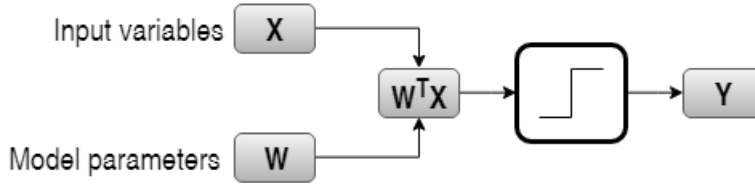
Figure 2.2: Schema for linear classification

to predict whether tomorrow will be a sunny day or not. Notice that there are only two possible outcomes: 'sunny' or 'not sunny' and we could represent these outcomes as -1 and 1 in our model. This form would be called binary classification because we have two possible classes. It is very easy to extend this method to multi-class classification.

The computation in this method starts out exactly the same, combining the input variables using a linear combination. Next, we have to define a threshold to indicate which examples belong to one class or another. In the case of binary classification we would define 1 threshold, and if the result of the linear combination is higher than the threshold we would predict one class. If it is lower, we would predict the other class. The function used here would be called a sign function, which maps real values onto one of 2 possible outcomes. We could represent this computation with the following formula:

$$\hat{y}_n = \begin{cases} -1 & if \ \boldsymbol{w^T x_n} \leq t \\ 1 & if \ \boldsymbol{w^T x_n} > t \end{cases} \qquad for \ n = 1..N \tag{2.2}$$

where

- $\hat{y}_n$ is the output for sample n

- $\boldsymbol{w^T x_n}$ is the vector notation for the inner product of $\boldsymbol{w}$ and $\boldsymbol{x_n}$

- t is the classification threshold

A schema for this computation is shown on figure 2.2.

2.3.3 Logistic Regression

The third method we want to present is called logistic regression [19][9]. In this case, the output variable we want to predict comes from a binomial distribution. This

5
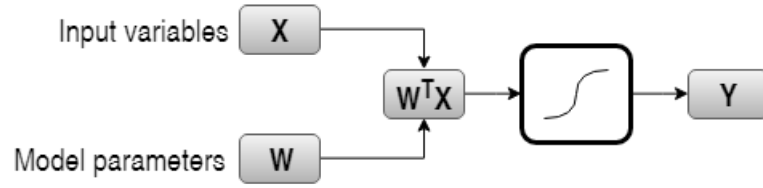
Figure 2.3: Schema for logistic regression

means that they are the result of a probabilistic event. An example would be tossing a coin and checking whether the result is heads or tails. While the outcome is binary (heads or tails) we know that there is an underlying probability for the coin to be heads or tails, and we would like to know this probability.

The idea is still the same. We will make a linear combination of the input variables. However this time we will use a logistic function to produce our estimate. The logistic function is a function that maps real numbers onto the range $[0, 1]$. This result can then be interpreted as an estimate for the probability. We could represent logistic regression mathematically as:

$$\hat{y}_n = \theta(\sum_{i=1}^{M} w_i x_{in}) = \theta(\boldsymbol{w^T x_n}) \qquad for \ n = 1..N \tag{2.3}$$

where

- $\hat{y}_n$ is the output for sample n

- w_i is the model parameter (coefficient) for explanatory variable i

- x_{in} is the value for explanatory variable i for sample n

- $\boldsymbol{w^T x_n}$ is the vector notation for the inner product of $\boldsymbol{w}$ and $\boldsymbol{x_n}$

- $\theta(x)$ is a logistic function, also called a sigmoid function. An example sigmoid function is $\frac{e^x}{1+e^x}$

A schema for this computation is shown on figure 2.3. The logistic regression method is the one that will be most widely used throughout this thesis.

2.4   Training a model

In order to understand the integration strategies that will be explained later on, it is useful to know how exactly the models come to be. Remember from section 2.2 that training a model means: showing it examples of what it has to learn so it can build up experience and capture patterns. This section will explain exactly what this means in the case of linear models: what the data looks like, and how we use it to build up experience.

2.4.1 Training and testing data

When training our model we have to let it build up experience. To do this we have to show it example datapoints generated by the function we want it to learn. An example datapoint consists of 2 parts: a vector of values, one for each of the explanatory variables and a target value for the dependent variable. If we generalize this to many datapoints, we can represent the input data by a matrix for the example datapoints, and a vector of target values. The matrix columns are equal to the explanatory variables and each row represents an example datapoint. The vector of target values contains one target value for each datapoint. It is easy to see that the length of the output vector has to be equal to the amount of rows in the input matrix, indeed there should be one target value for each example datapoint. The amount of datapoints is often called the size of the dataset. The combination of this matrix of datapoints and the vector of target values is what is called the training data. It is the data that we use to train the model, or in other words from which the model can build experience. How this experience is built up is explained in the next section 2.4.2.

Remember that the purpose of a predictive model is to predict (or estimate) a value of the target function when given values for the explanatory values. Once we have trained a model using the training data, it will have built up a representation for the patterns in the data that it has found (from experience). We can then use the trained model to predict an output value by giving it a new datapoint that it has never seen. Based on the experience that it has built up (in its representation) it will be able to estimate a value for the target function. This new datapoint is what we call test data. It is a datapoint that the model has never seen before, and it doesn't contain a value for the target function, because that is exactly what we are asking the model to produce. Another terminology that is often used in this case is labeled data. A datapoint together with the target value is considered a labeled datapoint. The training data we used in this case was thus labeled data. The testing data does not have a value for the target function, and thus is unlabeled data.

2.4.2 Gradient descent

In this section we will explain how we get from the training data to the predictive model [19][4][5]. The idea here is that we have some error measure. The error measure is a sort of rating for our current model as it indicates how big the mistakes are that our current model makes. Once we have a way of computing this error, we can try to minimize the error to obtain our 'best' possible model. Minimizing the error means that we will change the parameter values of our predictive model. Again, what the parameters of the model are depends on the type of model we use. In the case of logistic regression the representation is a vector of weights, learning thus means changing the weights w_i for the explantory variables (see the formula for logistic regression 2.3) such that the estimated outputs $\hat{y}_n$ correspond better to the real outputs y_n.

Error measure

In logistic regression an often used error measure is called the cross-entropy error [19]. The formula for this error is the following:

$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} ln(1 + e^{-y_n w^T x_n})$$

where

- x_n is the vector of values for the explanatory variables for example n.

- y_n is the value of the dependent variable for example n.

- w^T is the transpose of the weights vector. These are the parameters of our model that we can adjust.

- N is the size of our dataset.

- $E_{in}(w)$ is the in-sample error. This is the cross-entropy error that we make on the examples in our dataset. It is a function of the weights w.

Notice that to compute this error measure, we need to have labeled datapoints, the value of y_n is known. We can intuitively see that this is a reasonable error measure. It is an averaged sum over examples (datapoints), where for each example we compute an individual error made on that example. Notice that $w^T x_n$ is the linear combination of the input variables that our current model suggests. This is the prediction $\hat{y}_n$ that our current model would make for example n and is a real valued number. On the other hand y_n is the actual correct prediction for example n and has a value of -1 or 1. If the signs of $w^T x_n$ and y_n agree then our current model actually makes a correct prediction for this example. We can see that in this case the exponential becomes close to 0, making our error for example n very small, as we would expect. If however their signs are opposite, the exponential becomes larger as our incorrect prediction becomes larger. This in turn will increase the error, again as we would expect. Thus we can see that if we were to minimize this error, we are moving towards a model that tries to make correct predictions.

The gradient descent method

When trying to minimize a function, a general approach would be to try and compute the derivative of the function, and find the value where this derivative equals zero. In the case of logistic regression it is not possible to find an analytic solution to this problem [19]. The best we can do is put ourselves somewhere on the error surface and try to move towards the minimum in small steps. This is called an iterative approach. Remember that our error function looks like this:

$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} ln(1 + e^{-y_n w^T x_n})$$

We can now compute its derivative with respect to w:

$$\nabla E_{in}(w) = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n x_n}{1 + e^{y_n w^T x_n}}$$

The problem is to find the set of weights w for which the derivative becomes 0 (or that minimizes the error). We can start out with an initial set of weights $w(0)$ and then iteratively update these weights so we move towards the minimum. Let's call the direction in which we update our weights v. The update we make to w then becomes:

$$w(t+1) = w(t) + \eta v$$

where

- $w(t+1)$ are the updated weights for this iteration.

- $w(t)$ are the current weights before we make a move.

- v is a unit vector pointing in the direction we want to move.

- η is a number that indicates how big the move is that we make, also called the step size.

Remember that the gradient of a function at a certain point always points towards the steepest slope upwards [3][17]. In our case we would like to find the minimum, so it is a good idea to move our weights in the direction of steepest descent. The direction v that we are moving towards then becomes the normalized opposite direction of the gradient:

$$v = -\frac{\nabla E_{in}(w(t))}{\|\nabla E_{in}(w(t))\|}$$

We can now summarize the gradient descent method as follows:

Data: x, y
initialize weights w(0)
while *Stopcondition is not met* **do**
    Compute gradient $\nabla E_{in}(w(t))$
    Compute update direction v
    Update weights $w(t+1) = w(t) + \eta v$
end

Algorithm 1: Gradient Descent algorithm

There are two non-trivial issues in this computation: the initialization of the weights and the stopcondition.

Weight initialization is sometimes a very tricky thing to do, in the case of logistic regression however it is acceptable to set $w(0)$ equal to the zero-vector as this corresponds to no correlation between any of the input variables and the output variable, and the result of the sigmoid function would be 0.5 or 50% meaning the

model has no preference for either outcome.

The stopcondition however is a bigger issue and usually the way to go here is to make a combination of several stop criteria. One criteria would be to simply limit the amount of iterations to a fixed number. This could avoid endlessly overfitting. Another criteria is to set up a target error we want to achieve (a small number), and stop when we have reached this target. This however raises the question of picking the target error, and this is mostly an application dependent choice.

In the version of logistic regression explained here, it can however be shown that the error surface we are dealing with is a convex surface [10]. This makes it very easy to find its minimum and we don't need very complex initialization and stopping criteria to get good results. In other machine learning methods however these surfaces aren't always as nice, and the issue of local minima versus global minima becomes a big deal. There has been much research on this topic however and many sophisticated methods have been developped to deal with this issue [34].

2.5 Summary of training a model

We now have seen everything that we need to train a model. The first thing we have to decide on is the method. If we believe there is a linear relationship between the input variables and the output, we could choose a linear method. Suppose the output variable comes from a binomial distribution then we know we can use logistic regression and our model representation will be a set of weights. Training then starts by initializing the weights of our model to some value $w(0)$. Next, we have seen in the gradient descent method (2.4.2) that we can change our weights in such a way that our new weights will do a better job at predicting the outcomes than our old weights. Notice that we need a labeled dataset to compute this update. If we repeat this process for a large amount of labeled datapoints (until we meet a certain stopcondition) then we end up with a trained model that attempts to predict the outcome correctly for most of the examples that we have presented it.

2.6 Overfitting

Now that we have established a method of training our models, it is time to deal with an issue known as overfitting [19][20]. Overfitting points to the fact that there are several mechanisms at work when we are building a model that prevent us from reaching the perfect model (a model that predicts correctly at all times). These mechanisms essentially originate from noise and uncertainty in many aspects of the learning process (the input data, choice of model, choice of algorithm, ...). We can however try to decompose this noise into several components and then attempt to influence them by making changes to our model computation. We will present two ways in which overfitting can be tackled: regularization and validation.

2.6.1 The problem of overfitting

Let's introduce some notation. From now on we will refer to the notion of 'in-sample error' or in symbolic notation E_{in} as the error that a model makes on the examples in our training set. The training set consists of the examples that were used to train (compute) the model in the first place.

Similarly we will define 'out-of-sample error' or E_{out} as the error we make on examples that were not used for training the model. Notice that E_{in} is something we could compute because we have access to the training data, but E_{out} is a quantity we cannot exactly compute but we could try to estimate it if we have some examples left that we did not use for training. Notice also that it is E_{in} that we minimize during our model computation, but it is E_{out} that we actually want to minimize! Indeed, E_{out} corresponds to the error that we get when we are going to deploy our model in practice and use it on examples we have never seen before. We can do this because we believe that E_{in} tracks E_{out} to a certain degree. And thus if we manage to minimize E_{in} we also minimize E_{out} to some extent.

We can only speak of overfitting when we are comparing two models. We say that one model, call it model A, is overfitting with respect to another model, model B, when model A managed to get a lower E_{in} than model B, but model B has a lower E_{out}.

Another way of looking at it is during the learning process. Let's have model A be the model that we computed when we started from model B and performed one more iteration of the training algorithm. Thus model A is 'more trained' than model B. Now let's suppose model A is overfitting:

$$E_{in}^{modelA} < E_{in}^{modelB} \tag{2.4}$$

$$E_{out}^{modelA} > E_{out}^{modelB} \tag{2.5}$$

The additional iteration has decreased the in-sample error, and thus we are able to fit our training data better, but the out-of-sample error has increased, meaning that our model doesn't generalize as well to other examples outside the training set. This means that we are actually fitting our training data too well, while we are not really getting a better grasp of the underlying pattern that we wish to learn. We are overfitting the training data.

2.6.2 The bias and variance trade-off

There are several ways of looking at overfitting and pointing out its origins. We will introduce the notions of bias and variance and how they can describe the noise in our system.

Average hypothesis

Before we can introduce the notions of bias and variance we first note that the data we have is produced by a target function f that we wish to learn (or model). In

order to do this we have to decide on the type of functions that we will use to model. The type of functions could be very simple functions such as a line or a polynomial, or they could be very complex function sets like a deep neural network with 500 parameters. In machine learning this is called the hypothesis set. It is the set of all functions that we consider possible candidates to fit our target f. Training a model then boils down to using the examples x in our dataset D to decide which hypothesis to pick from the hypothesis set.

Next, let's introduce the notion of average hypothesis $\bar{h}$. First let us note that this is a purely hypothetical notion, we will never be able to compute it in a real scenario. Imagine we have an infinite number of datapoints available. We could group these datapoints together and pretend we have access to an infinite number of finite datasets. For each of these datasets we apply the learning process and we will pick a certain hypothesis h from our hypothesis set. The average hypothesis is then equal to the average of all the hypothesis' just learned. Or in a formula:

$$\bar{h} = \mathbf{E}_D[h^{(D)}]$$

where

- $\bar{h}$ is the average hypothesis.

- $\mathbf{E}_D$ is the expected value over an infinite number of datasets

- $h^{(D)}$ is the hypothesis that was learned for a specific dataset D

We can also look at this average hypothesis as sort of the best we can do with the given hypothesis set. Indeed, when we imagine having an infinite number of datasets we would end up cancelling out much of the variation in the learned hypothesis' and end up with a very good one.

Bias

We can now define the bias [19] as the distance between the average hypothesis $\bar{h}$ and our target function f.

$$bias = (\bar{h} - f)^2$$

We can see the bias as an error we make due to our own choices. For instance our choice of hypothesis set. If we choose a very simple hypothesis set (meaning a hypothesis set with simple function), we cannot expect to be able to find a fit for a very complex function. The target function simply isn't contained in our hypothesis set.

Therefore we introduced the notion of average hypothesis. We can view this as the best we can do given our current hypothesis set, and the distance to the target is what we call the bias.

Variance

In a real learning situation we generally never find this average hypothesis, because it requires a large amount (or even infinite amount) of datapoints. In reality we always have a finite dataset and that is all we can use to navigate through the hypothesis set. This is where the notion of variance comes in. We can define variance [19] as the error we get from not having the best hypothesis possible in our hypothesis set. Or in other words as the distance between the hypothesis that we actually found by learning from our finite dataset and the average hypothesis.

$$variance = (h^{(D)} - \bar{h})^2$$

Error due to variance mainly comes from two sources: the first is our finite dataset. In reality we are given a dataset of N examples and usually this dataset is not sufficient to find the best hypothesis and thus there will be a variance error made. The second is the complexity of the hypothesis set. As we increase the complexity, it becomes increasingly difficult to navigate through this set. There are simply many more hypothesis' to choose from. Again this makes it harder for our learning process to find the optimal hypothesis and as such will introduce a variance error.

The tradeoff

Having both bias and variance defined we can see that they are not disconnected, there is a tradeoff. If we look purely at bias we could think that simply choosing a super complex hypothesis set is always optimal. Taking this to the extreme we can imagine choosing a hypothesis set that contains every possible function in the universe. Indeed, our bias will be zero since the target function will always be inside our hypothesis set.
However, an increasingly complex hypothesis set makes it harder to actually find the optimal hypothesis. We know that the optimal hypothesis is there, but we just cannot find it. An increasingly more complex hypothesis set will require more and more parameters to train because we have to be able to represent it's complex functions. For a finite dataset it thus becomes increasingly more difficult to find a good value for the increasing number of parameters. Let's look at a more concrete example: imagine we have a dataset of just 3 points in the 2D-plane. If we choose the hypothesis set of lines, we can imagine there are several possible hypothesis' (lines) that we can compute with our data. We could then take the more complex hypothesis set of parabolas. In this case we really have only 1 possible hypothesis left, the parabola that fits through our data. As we increase to higher order polynomials however, it becomes increasingly more difficult to find a sensible fit, because there are more parameters than we can fit with our data. The take-away message here is that we have to choose a hypothesis set complexity based on the resources that we have. In this case the resource is our dataset. The larger the dataset that we can learn from, the more complex hypothesis sets we can afford, and the better our results will be. But there is no gain in choosing overly complex hypothesis sets when you don't have the resources to afford them. This will simply cause you to find

13

hypothesis' that fit your training data very well (imagine fitting the 3 datapoints with a 7th order polynomial, you would get an exact fit) but this model will not generalize to anything in the real world, it is a complete overfit.

2.7 Regularization

Now that we have the concepts of bias and variance, let's use this information to try and improve our models. The first method is called regularization [19][46][26]. In very simple terms this method will add a small amount of bias in order to greatly decrease the amount of variance, reducing the overall error we make.

2.7.1 Adding bias

Remember that bias is defined as the distance between the average (or best) hypothesis $\bar{h}$ and our target function f. Adding bias effectively means we are going to make another choice, which will impact the average hypothesis. The choice we are about to introduce is based on the following observation: when confronted with a set of similarly performing models, the simplest model is usually the best. Or in other words we should try to prefer simple models over very complex ones.

This observation does not have a mathematical proof, it is rather an observation from experience and reason. One good argument is the fact that noise is usually of high frequency [19]. Meaning that distortions of our dataset (for instance measurement errors) will often be very scattered and random, while the underlying pattern that really makes up the data will be rather smooth. A similar argument can be made for the error due to bias, when we choose a hypothesis set that does not contain the target function, the error due to bias will be mostly random and of high frequency [19]. Therefore if we want to reduce the impact of this noise in our final model, we should prefer models that are not able to fit these high frequencies exactly. Lastly we can remark that if we look at our current understanding of nature (let's say at a larger scale), systems almost always have smooth transitions. The most important laws of nature that we find are all written down in small, simple formulas. Nature doesn't work with instantaneous changes (high frequency), but rather it has smooth functions that govern the basic principle, and then it adds random noise and fluctuations on top of it. This principle is what we try to extrapolate here to machine learning.

Thus, the choice we will make is that we will prefer simple models over complex ones by adding a constraint to the weights in our model.

2.7.2 Regularization types

Adding a constraint to the weights is called regularization. There are many kinds of constraints that we could add to the weights and, depending on the constraint we choose, the regularization gets a different name and it will have a different effect. One of the most famous regularizers is called ridge or weight-decay [26]. The constraint

for this regularizer is the following:

$$\sum_{i=1}^{N} w_i^2 \leq C$$

where

- w_i is the weight (model parameter) for the i'th explanatory variable.

- C is the constraint value

Using this regularizer will result in a preference for models with smaller weights. This keeps certain weights from getting out of control. This form of regularization is also often called the L_2 penalty.

Another popular regularizer is called the lasso penalty (or L_1 penalty) [26]. The constraint in this case is:

$$\sum_{i=1}^{N} |w_i| \leq C$$

In addition to keeping the weights small, this form of regularization also performs variable selection [44]. This means that instead of just keeping the weights small it will also prefer to make weights actually zero for variables that don't contribute much to the model. This will cause the resulting model to have fewer parameters, but the parameters that do survive the penalty are sure to be important. This regularizer is often used when there is a huge number of explanatory variables, and we wish to find only those that are really descriptive. Later in the thesis we will use datasets that contain gene expression information about cancer patients, these datasets often have thousands of explanatory variables and will provide a good example for using the lasso regularization (see section 5.5).

The last form of regularizer we wish to demonstrate is called the elastic net penalty. This regularizer is simply a linear combination of the ridge and lasso penalties and provides a way of balancing the two. It has the following constraint:

$$\alpha \sum_{i=1}^{N} w_i^2 + (1 - \alpha) \sum_{i=1}^{N} |w_i| \leq C$$

To demonstrate the difference between the types of regularization, it is useful to look at a figure that plots the evolution of the weights as a function of the amount of regularization used. As explained in the next section (2.7.3), the amount of regularization used is called λ. Figure 2.4 shows the coefficient paths respectively for ridge, lasso and elastic net regularization. These are the values of the coefficients as a function of λ. The sample dataset in this case contained 18 explanatory variables. The ridge penalty selects all 18 variables in the model, while the lasso and elastic net penalties respectively select between 10-13 and 11-18 variables in their models, depending on the value of λ. This shows the variable selection of the lasso. Also notice how the y-axis dimension is different for the ridge regularization, showing the shrinkage of the weights.

15

(a) Ridge ($\alpha = 0$)   (b) Lasso ($\alpha = 1$)   (c) Elastic net ($\alpha = 0.5$)
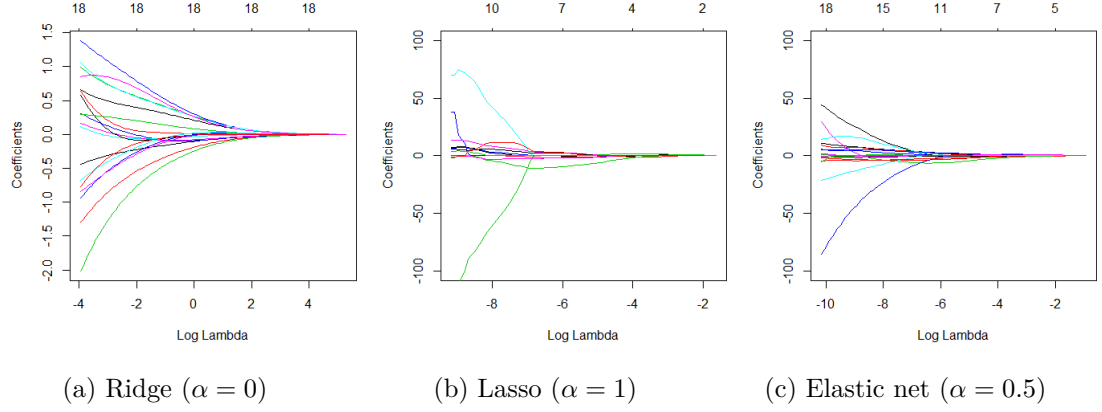
Figure 2.4: Coefficient paths as a function of λ for different regularization types.

2.7.3   Lambda

Using a regularizer introduces a constraint, this means that we now have to deal with a constrained optimization problem which is much harder to solve than an unconstrained problem. Fortunately, through some clever mathematics it is possible to convert the constrained minimization problem to an unconstrained one by incorporating the regularization constraint in the formula for the error itself [19]. The formula for the error with regularization then becomes:

$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} e(x_n, y_n, w) + \lambda R(w)$$

where

- $E_{in}(w)$ is the in-sample error.

- $e(x_n, y_n, w)$ is the individual error made on example n. In the case of logistic regression this would be a cross-entropy error term $ln(1 + e^{-y_n w^T x_n})$.

- x_n is the n'th sample in the dataset.

- y_n is the outcome for the n'th sample in the dataset.

- w is the vector of weights, our model parameters that we are trying to find.

- λ is the regularization parameter that will be explained below.

- $R(w)$ is the regularization term dependent on the type of regularization used. For instance: $\sum_{i=1}^{N} |w_i|$ for lasso, $\sum_{i=1}^{N} w_i^2$ for ridge, ...

Notice that we now again have an error measure that we want to minimize, and it is an unconstrained optimization problem. The important new part is λ. This is the amount of regularization we want to use. It is a new form for the constraint

constant C that was introduced earlier in the types of regularization (section 2.7.2). The higher λ the tighter the constraint (lower C) and vice versa. The value of λ will prove to be critical in getting good models. The way to calculate it is through validation, which is the topic of the next section.

2.8 Validation

In this section we will explain the method of validation which is used to estimate the out-of-sample error, remember that this is the error we make on samples that were not used for training. We will first explain the issue of sample size and then present a method to work around this limitation.

2.8.1 The sample size dilemma

Remember that when we are training a model we use the in-sample error to navigate the hypothesis space (2.4). We do this because we believe that the in-sample error is a valid surrogate for the out-of-sample error (which is the error we actually want to minimize). A valid question to ask is: why don't we just estimate the out-of-sample error and minimize it directly? Consider the following situation:
We are given a dataset of N samples. We will use K samples of the dataset to train my model, this leaves us with $N-K$ samples that are not used for training. Since K samples were used for training, if we would compute the error the model makes on these samples we would be computing the in-sample error. If we want to estimate the out-of-sample error we have to use the $N-K$ samples that were not used for training. This sample set of $N-K$ samples if often called the validation set.
We can now make the following observations:

- The larger we choose K, the more samples are available for training and thus the better our model can be (due to lower variance!)

- The smaller we choose K, the more accurate our out-of-sample error estimate is, because we have more datapoints for the estimation

So now it is clear that we have a tradeoff to make. We would like K to be as large as possible so that we have a large amount of samples to train a model from. On the other hand we would like K to be as small as possible so that we have enough samples to estimate the out-of-sample error. Or in other words, if we choose K too large we will be able to compute a good model but our estimation for the out-of-sample error will be very bad and thus uninformative. If we choose K too small then we will get a very precise estimate for the out-of-sample error. But the estimation will be for a very bad model, which is totally not representative of our real performance. The solution to this apparent contradiction is cross-validation.

2.8.2   Cross-validation

Cross-validation [19][35] is a technique that allows us to have plenty of samples left for training, while still getting a pretty good estimate for the out-of-sample error. The method is as follows: divide the dataset of N samples into F equal parts (often called folds). Each fold now has N/F samples. Train a model on $F-1$ folds and use the remaining fold to estimate the out-of-sample error. Repeat this process F times, one time for each of the F folds, each time leaving out a different fold. In the end we have F estimates of the out-of-sample error and we can average them to get a final result.

Notice that we are really using all samples for validation and training, but never both at the same time. Validation is often used to determine parameters of the learning process, for instance the λ parameter for regularization. We simply try several values for λ, compute the out-of-sample error using validation, and pick the λ that gives the lowest error. Once we have decided this λ we can then train a model on the full dataset of N points and use the λ we have just calculated to be optimal.

We have to remark however that when we use validation to calculate a value for λ as described above, we are really using the validation to help train the model. In this case we can no longer make the statement that the validation samples are not used for training. This is called data pollution. We are using the same data to train training parameters aswell as training the model itself and it is obvious that this will give rise to additional correlations in the data. However in practice it is generally accepted that if you use this technique to decide on just a few learning parameters (often just λ), and you have a big enough dataset, the data pollution is minimal and the results and estimates you get are still reliable [19].

Cross-validation is not only used to estimate learning parameters. It can also be used to test the performance of the model. In this case the fold that is not used for training is usually not called a validation set, but rather a test set. Because samples in this set will be used to test the models performance.

2.9   Conclusion

We have now covered the basis of Generalized Linear Models. We have described the gradient descent method that we can use to navigate the hypothesis space by minimizing an error function. We have seen that overfitting is a serious issue that is caused by noise at different levels of the learning process. We have broken down this noise into bias and variance and shown that we can have an impact on this process. We can use regularization to add a slight bias in order to greatly decrease the error due to variance and we have based this on the principle that we should prefer simple and smooth models. Lastly we have covered the method of validation. A method

that we can use to estimate the out-of-sample error and which gives us the ability to choose learning parameters like λ and also test the performance of our model.

Chapter 3

Cox proportional hazards models

3.1 Introduction

Since the second case study will use survival models, it is essential that we know what these models are. This chapter explains the cox proportional hazards model [41][44][13] which is the type of survival model that we will be using. The first section explains what survival analysis means and how it is different from the generalized linear models (3.2). Next we will define what the proportional hazards assumption is (3.3) and lastly we will show kaplan-meier survival curves and how we can compute them from hazard functions (3.3.3).

3.2 Survival analysis

Survival analysis points to the fact that the outcome variables are time-to-event datapoints. A common example in biomedical research would be the time between the start of a patients treatment and the time of death. Note however that not all patients have to die in order to be useful in a survival analysis. A second outcome variable is used to indicate whether the patient survived until the end of the study, or not. This (often binary) variable is called the censoring variable.

A typical outcome in survival analysis is thus represented by two vectors: a time to event vector and a censoring vector. An example is shown in table 3.1. The idea of survival analysis is now to find a pattern between a set of explanatory variables and the time-to-event. A common term used in survival analysis is hazard, meaning 'risk of the event occurring'. A resulting model from survival analysis usually contains two components. The first component is a $\lambda_0(t)$ baseline hazard function. This function describes how the risk of the event occurring changes with time, assuming there is no influence from any of the explanatory variables. The second component describes the influence of each explanatory variable on the hazard. This brings us to the notion of proportional hazards which is the topic of the next section.

Time (days)	Censor
249	1
345	1
152	1
452	0
120	1
...	...

Table 3.1: Example outcomes for survival data

3.3 Cox proportional hazards model

3.3.1 Proportional hazards condition

The proportional hazards condition means that we assume that hazard ratios are independent of time. A hazard ratio is the relative risk between two entities (patients). Furthermore the condition states that changes in the explanatory variables have an exponential effect on the hazard. The following paragraphs explain these statements in more detail.

If we assume the proportional hazards condition to be true then this gives us the opportunity to estimate the effect of explanatory variables without dealing with the underlying baseline hazard function. This was an observation made by Sir David Cox [23], hence the name of the model. We can represent the cox model as follows:

$$\lambda_i(t) = \lambda_0(t)e^{X_{i1}\beta_1 + ... + X_{iN}\beta_N} \tag{3.1}$$

where

- $\lambda_i(t)$ is the hazard for patient i at time t.

- $\lambda_0(t)$ is the baseline hazard at time t.

- $X_{i1}...X_{iN}$ are the values of the explanatory variables for patient i.

- $\beta_1...\beta_N$ are the values of the coefficients for each explanatory variable.

Now, let's look at relative risks between two patients:

$$\frac{\lambda_i(t)}{\lambda_j(t)} = \frac{\lambda_0(t)e^{X_{i1}\beta_1 + ... + X_{iN}\beta_N}}{\lambda_0(t)e^{X_{j1}\beta_1 + ... + X_{jN}\beta_N}} = e^{(X_{i1} - X_{j1})\beta_1 + ... + (X_{iN} - X_{jN})\beta_N} \tag{3.2}$$

This quantity is called the hazard ratio. Notice that this ratio does not depend on time. This means that if at the start of the study a patient has twice the risk of the event occurring compared to another patient, it will have twice the risk at any other time aswell. It's risk remains proportional, independent of time.
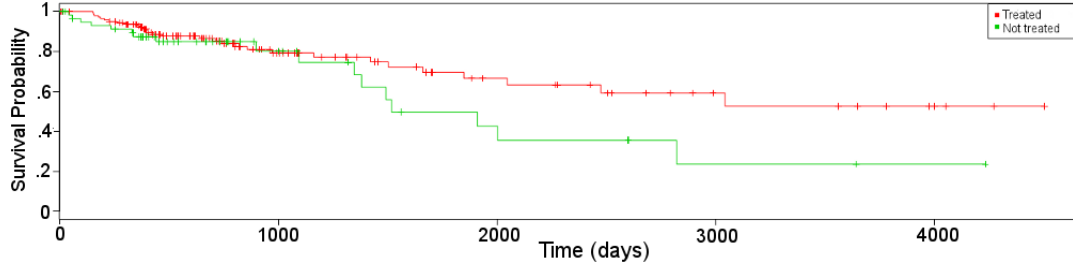
Figure 3.1: Example Kaplan-Meier survival curve. Red and green lines represent survival probabilities for treated and untreated patient-groups respectively.

We can also compute the effect on the hazard for a unit increase in an explanatory variable X_j. In the following formula the subscripts i that were used to indicate the patient have been omitted since this is patient independent. The hazard ratio then becomes:

$$\frac{\lambda(t|X_j + 1)}{\lambda(t|X_j)} = e^{(X_j+1-X_j)\beta_j} = e^{\beta_j} \tag{3.3}$$

It is directly given by the exponential of the coefficient for the explanatory variable. This reflects the proportional hazards assumption that hazard ratios do not depend on time and that explanatory variables have an exponential effect on the hazard.

3.3.2 Using the hazard ratio

The proportional hazards assumption allows us to compute relative risks or hazard ratios without knowing the actual baseline hazard function. Knowing just relative ratios can be very useful however. Imagine the following scenario: we want to test if a certain drug increases the survivability of patients with a specific disease. We randomly give half of the patients the actual drug, the other half receives a placebo (or nothing at all). Performing a survival analysis using a cox model then allows us to compute the ratio of the hazards for the two groups. The coefficient for the explanatory variable that indicates which patients received the drug and which did not then tells us the effect of the drug on the hazard. We could then make a statement such as: "Taking the drug tends to half the risk of the event occurring (at any time)." This is obviously a useful result to support further research or development for the drug.

3.3.3 Kaplan-Meier survival curves

Kaplan-Meier survival curves [27] are often used in survival analysis. These curves show the probability of surviving up to any point t. It is thus a declining curve starting from 1 at $t = 0$. An example curve is shown on figure 3.1. In formula notation we could write this as:

$$S(t) = P(T > t) \tag{3.4}$$

where

- $S(t)$ is the survival function

- $P(x)$ means 'the probability of x'

- T is the observed event time (e.g. death)

- t is the time variable

Linking hazard to survival

Creating a kaplan-meier survival curve based on the outcome vectors (time and censoring) is very straightforward to do, it is simply a visual representation of the event occurring at certain times. However when we create a model that tries to explain these outcomes from explanatory variables we end up with a hazard function. In order to create survival curves we have to link this hazard function to the survival function [37]. Recall the hazard function $\lambda(t)$ from section 3.3. We could write this function in a similar probabilistic notation as we did for the survival function:

$$\lambda(t) = \lim_{\delta \to 0} \frac{P(t < T \leq t + \delta | T > t)}{\delta} \tag{3.5}$$

this is really just the definition of hazard: the risk of the event happening at time t is defined as the event happening during an infinitesimal timeframe after time t, given that the event has not happened yet up to time t. As we take the limit $\delta \to 0$ this becomes an instantaneous hazard. We can now apply the conditional probability rule $P(A|B) = \frac{P(A \cap B)}{P(B)}$ to get:

$$
\begin{aligned}
\lambda(t) &= \lim_{\delta \to 0} \frac{P(t < T \leq t + \delta \cap t < T)}{\delta} \frac{1}{P(T > t)} \\
&= \lim_{\delta \to 0} \frac{P(t < T \leq t + \delta)}{\delta} \frac{1}{S(t)} \\
&= \lim_{\delta \to 0} \frac{P(t < T) - P(t + \delta \leq T)}{\delta} \frac{1}{S(t)} \\
&= -\lim_{\delta \to 0} \frac{P(t + \delta \leq T) - P(t < T)}{\delta} \frac{1}{S(t)} \\
&= -\frac{\partial S(t)}{\partial t} \frac{1}{S(t)} \\
&= -\frac{S'(t)}{S(t)}
\end{aligned}
\tag{3.6}
$$

Let's look at this result in an intuitive way. Assume the event we are talking about is dying. The hazard function then tells me what the risk is of dying at time t. Equation 3.6 tells us that this is equal to $-\frac{S'(t)}{S(t)}$. There are three parts to explain here:

- $\frac{1}{S(t)}$: The first thing we have to do is survive up to time t. Otherwise it makes no sense to ask what the risk of dying at time t is. The survival function tells us how probable it is to survive up to time t so dividing by this quantity exactly provides what we need. If $S(t)$ is large, then this means many people survive up to time t and so the risk is low. If $S(t)$ is small then few people actually survive up to time t and thus the risk of having to survive up to this point gets bigger.

- $S'(t)$: Now that we have survived up to time t, we need to calculate the risk of dying at this point in time. Remember that the survival curve declines whenever people die at certain points in time. The slope of this function therefore tells us how many people die at that time t. If $S'(t)$ is very large, it means that many people die here. The risk of dying thus becomes large.

- The hazard is supposed to be a positive quantity. Since the survival function is a strictly declining curve (people usually don't resurrect from the dead), $S'(t)$ is negative. Therefore we need the minus sign to end up with a positive result.

Next, using the chain rule, we can observe that

$$\frac{\partial log(S(t))}{\partial t} = \frac{S'(t)}{S(t)} \tag{3.7}$$

and thus combining equations 3.6 and 3.7:

$$\lambda(t) = -\frac{\partial log(S(t))}{\partial t} \tag{3.8}$$

$$\int_0^t \lambda(t)dt = \int_0^t -\frac{\partial log(S(t))}{\partial t}dt \tag{3.9}$$

The left hand side of equation 3.9 is called the cumulative hazard fuction and often denoted $\Lambda(t)$. We can then write:

$$\Lambda(t) = -log(S(t)) \tag{3.10}$$

$$S(t) = e^{-\Lambda(t)} \tag{3.11}$$

Equation 3.11 allows us to relate the survival function to the hazard function. This allows us to create survival curves for any form of hazard function. The only thing that is left to do is to define the baseline hazard function, because otherwise we could not compute the integral in $\Lambda(t)$. Usually an estimator based on likelihood is used to estimate the baseline hazard function. There are many forms available for this estimator [38], but we will not go into any more detail here.

3.4   Conclusion

In this chapter we have explained the cox proportional hazards method. We have shown that it is used for survival analysis, where the targets are time and censoring variables. We have explained the proportional hazards assumption and shown its implications. Lastly we have shown how we can compute survival curves using the Kaplan-Meier method and we have shown a relationship between the hazard function and the survival function.

Chapter 4

Integration Strategies

4.1 Introduction

Now that we have established a firm understanding of the methods we will use to create predictive models, it is time to turn our attention to the integration step. In this chapter we will present three different integration strategies. We will start by providing a description of the datasets that we will use to build our models (4.2), followed by the integration strategies that show how we can combine these datasets to create predictive models. The first strategy is called early integration (4.3) and is based on concatenation of datasets. The second strategy is called late integration (4.4) which is based on ensemble learning. The last strategy is intermediate integration (4.5) which makes extensive use of the variable selection present in the lasso regularization (see section 2.7).

4.2 Description of the data

In order to integrate the datasets, they have to adhere to a number of requirements. Suppose we have a set of D datasets, each dataset can be represented by a matrix where the rows are the samples and the columns are the explanatory variables. Each dataset can have different explanatory variables, but they all need to have rows that can be linked across datasets, for instance using a unique patient identifier. Let me give a concrete example: imagine we have a set of 200 cancer patients. Each of these patients has had images taken of the cancer by different machines: an MRI scan, a PET scan, ... and they also had a bloodtest done measuring different proteïne levels in the blood (an ELISA test). Each of these methods will yield a dataset: one for each imaging scan and one for the blood test. The explanatory variables are obviously different for each dataset: the imaging datasets will have image features like blobs and pixel intensities, while the bloodtest will have values that represent the proteïne levels in the blood. Each dataset will however have a single row per patient, uniquely identified by that patients ID. This is not true in the case of missing data (e.g. a patient missed his MRI scan appointment). But there are ways of dealing with this: we can try to estimate the missing values using cluster-

ing techniques, or we can simply leave the patients with missing data out of the study.

Now that we have our input data, we need a target function (dependent variable). In the example case this could be whether the patient fully recovers from the treatment or not. This variable is supposed to have a binomial distribution and thus we could use logistic regression to try and estimate this variable. The task of integration is now the following: how do we combine the samples across the different datasets to come up with a predictive model that uses the information in all the datasets? In the following sections we will present three ways to do this: the early integration method which is concatenation, the late integration method that is based on an ensemble of models, and the intermediate integration method that takes advantage of variable selection present in the lasso regularization.

4.3   Early integration

The first integration method is called early integration. In this method we are simply going to concatenate the data for each sample. Figure 4.1 shows the schema for early integration of data sources. A new large dataset is constructed by concatenating the individual datasets by matching sample. The number of explanatory variables is now equal to the sum of explanatory variables of all individual datasets. This integrated dataset can then be used to train a predictive model using techniques seen in previous chapters (2,3).

4.4   Late integration

In late integration, as the name suggests, we will combine the different datasets at the end of the learning process. The schema for late integration is shown on figure 4.2. First, a model is learned for each dataset individually. This gives rise to D different models. To reuse the example from section 4.2, there will be a model for MRI data, PET data, etc. When we want to predict an outcome for a new sample (patient) we will present each model with the corresponding input from the patient. Each model will then compute an output and these are combined using a linear combination. If we have no preference for any model we could simply compute the average output. This average would then be our final output for the integrated model. In this case we cannot represent the final model by some model parameters (weights) but rather we have to view the full set of D models as well as the linear combination we chose as the full integrated model.

There is an analogy between late integration and the ensemble averaging technique in machine learning. Ensemble learning [25][24][1] means that instead of just building one model for a dataset, we build several models for the same dataset and then average their outcomes. Late integration takes this idea but applies it to a set of data sources instead of just one.
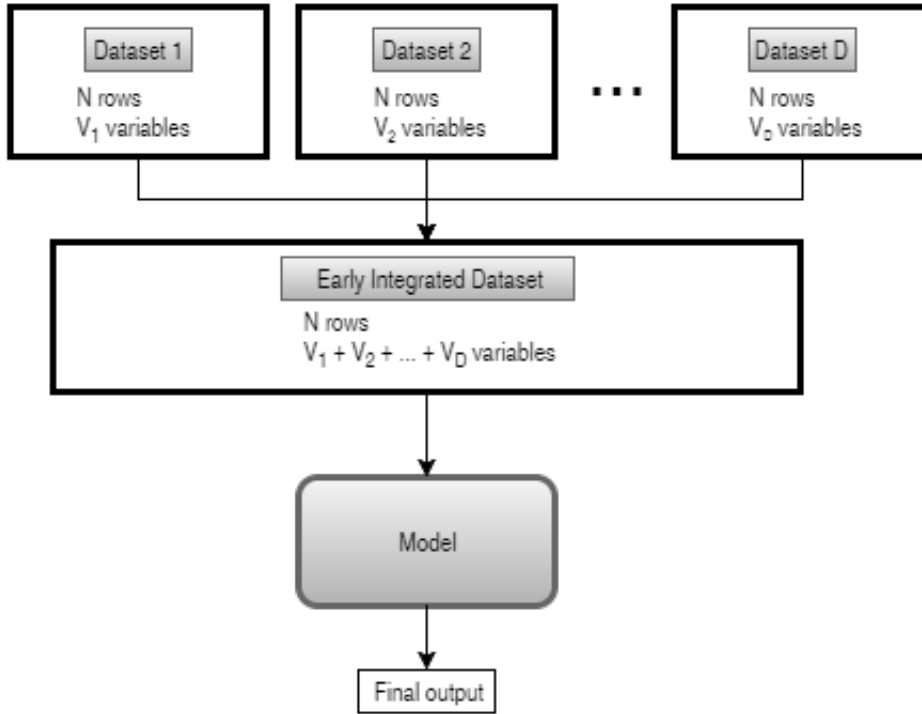
Figure 4.1: Scheme for early integration

The thought behind this model is that we try to learn as much as possible from each dataset individually, before we combine them. An example of this is that the variable selection by the lasso regularization (section 2.7.2) now has the opportunity to select the best variables for each dataset individually. While in the case of early integration, all the variables from the datasets are competing against each other at the same time to make it into the final model.

4.5 Intermediate integration

Intermediate integration is a novel integration technique that tries to take advantage of the variable selection when using a lasso penalty. The schema for intermediate integration is shown on figure 4.3. It starts out the same way as late integration, for each dataset we compute an individual model. However, instead of using the output of these models we will simply look at the variables that were chosen by each model to be informative. We will then go back to the original datasets and extract only those explanatory variables and concatenate them together into a new integrated dataset. Lastly, we use this integrated dataset to learn a final model.

The intermediate integration can be seen as a two-step learning process. First we preprocess the datasets by computing individual models and we extract only the informative variables. Then we will train our final model on the reduced dataset.
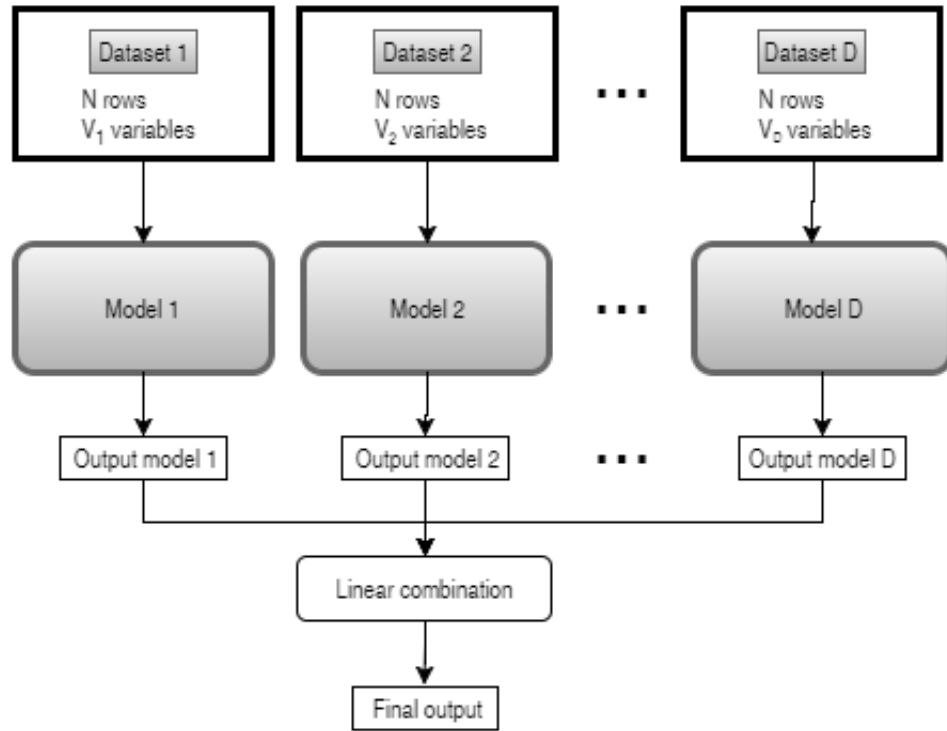
Figure 4.2: Scheme for late integration

This is very advantageous if the initial datasets have a huge amount of explanatory variables, as the preprocessing step will reduce this amount drastically while still keeping as much useful information as possible.

4.6   Conclusion

In this chapter we have shown the issue of large and high-dimensional datasets that we are currently facing. We have presented three different strategies to deal with this issue. The early integration method is the naïve method that simply concatenates the datasets together. The late integration method uses an analogous technique to ensemble learning in order to extract as much information as possible. Lastly, the intermediate integration technique introduces a two-step learning method that takes advantage of variable selection to reduce the dimensionality of the datasets.
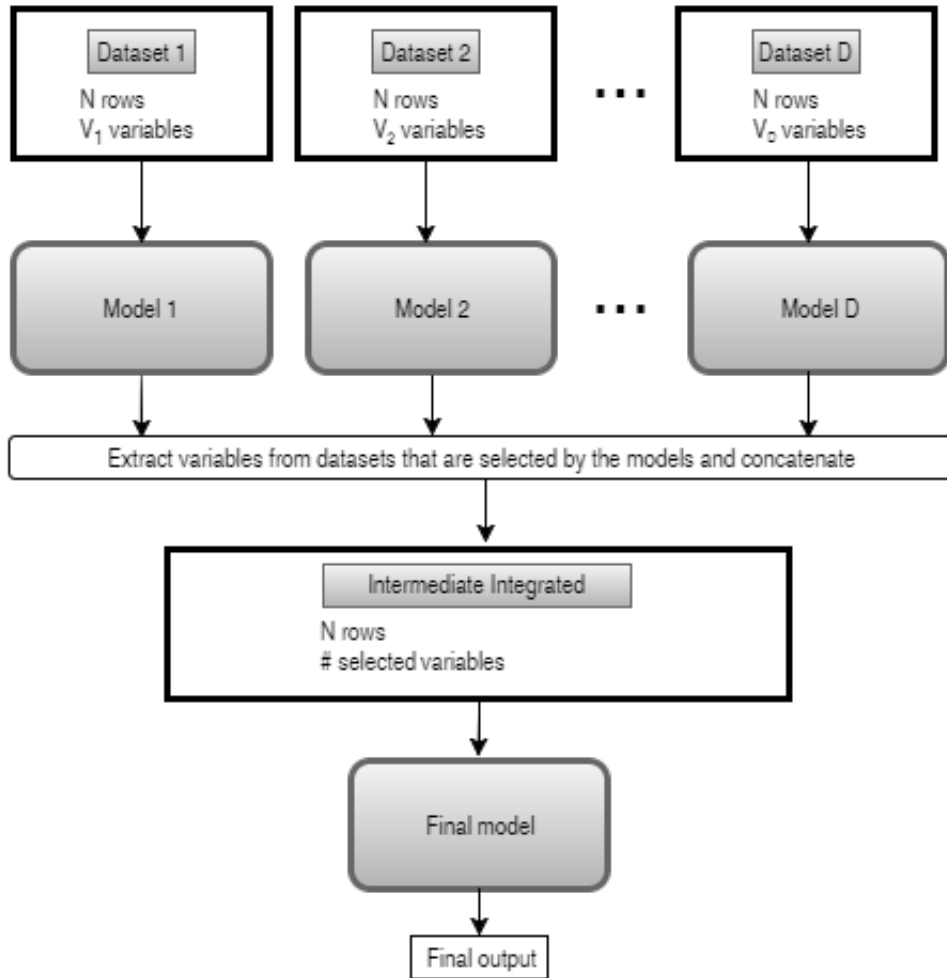
Figure 4.3: Scheme for intermediate integration

Chapter 5

Evaluation of integration strategies

5.1 Introduction

In this chapter we will investigate whether different integration strategies have an impact on the performance of the resulting models. We will start of by explaining ROC analysis which is the method we use to evaluate the logistic regression models in the first case study (5.2). The next section will provide details on the first case study and shows the results (5.3). After that we will explain the Wald test and how we will use it to evaluate the survival models that are trained in the second case study (5.4). The last section shows the details of the second case study and presents the results (5.5).

5.2 Evaluating a logistic regression model

The first evaluation will be made using a logistic regression model. This is a model computed for a dependent variable that has a binomial distribution. To evaluate such model, a common technique used is called Receiver-Operating-Characteristic analysis [47][28][14] or ROC analysis for short. Remember that when we are using a logistic regression model to predict outcomes, we have to define a threshold to separate positive from negative cases. An ROC curve shows the performance of the model for any possible threshold, using metrics such as sensitivity and specificity. This gives a very good general impression of the performance of the model.

5.2.1 Predictions using validation

What we really want to do is estimate the out-of-sample error for our model. To do this we have seen that we can use the technique of cross-validation. In this case we are not going to compute any learning parameter but we are going to use the validation set as a test set. We will split our dataset into K folds. We will train a model on $K-1$ folds and we will use this model to predict the outcomes of the

remaining samples in the test fold. By repeating this process for each folds we obtain a predicted outcome for each sample in the dataset. Notice however that we also have the real outcome of the samples in our dataset, and thus we can compare our predictions with the real values to estimate our out-of-sample error.

5.2.2 Metrics

When we compare our models predictions with the real outcomes, there are four possible scenario's:

- True Positive (TP): the model predicted positive and this is correct

- True Negative (TN): the model predicted negative and this is correct

- False Positive(FP): the model predicted positive and this is wrong (type 1 error)

- False Negative (FN): the model predicted negative and this is wrong (type 2 error)

Notice that this gives us more information than just registering whether we are right or wrong. It also tells us the type of mistake we made. This is a very important distinction because depending on the application there is usually a different cost attached to making type 1 or type 2 errors. And as we will see later on, there is a tradeoff between the two and we can tune our system to be more resistant towards making one type of error.

Now that we have established the notion of typed errors, there are several ways in which these numbers are combined to form evaluation metrics. Depending on your field of research these often get different names, we will use the ones applicable to the biomedical field.

Sensitivity

Sensitivity [15] measures the proportion of the real positive samples that our model correctly predicted as positive. It can be calculated with the following formula:

$$Sensitivity = \frac{\sum TP}{\sum P}$$

where

- $\sum TP$ is the number of true positives

- $\sum P$ is the number of real positive cases in the dataset

Sensitivity can be thought of, as its name suggests, as how sensitive the model is to detecting positive cases. If the model gets a real positive sample's input, what is the probability that it will detect it as such. A sensitivity of 1 means that our model is capable of correctly identifying all positive cases in the dataset. This means that the higher the sensitivity, the lower the type 2 error rate.

Specificity

Specificity [15] is the analogous metric to sensitivity, but for negative cases. It measures the proportion of the real negative samples that our model correctly predicts as negative.

$$Specificity = \frac{\sum TN}{\sum N}$$

where

- $\sum TN$ is the number of true negatives

- $\sum N$ is the number of real negative cases in the dataset

A specificity of 1 means that our model is capable of correctly identifying all negative cases in the dataset. This means that the higher the specificity, the lower the type 1 error rate.

The tradeoff

At this point it is obvious that we would like to maximize both sensitivity and specificity. Indeed, if both metrics are 1 then we make no errors and we have a perfect model. In practice however this is nearly impossible to achieve. It is easy to see that there is a tradeoff between the two metrics: when we try to increase the sensitivity, we will lose out on specificity and vice versa.

Consider the totally useless model that has a threshold lower than the smallest possible output. Meaning that this model will always predict a positive outcome regardless of the input. This model will have a sensitivity of 1, as we will get all positive cases correct. But it will also have a specificity of 0 because we get none of the negative cases correct. As we gradually increase the threshold we will cross critical values where some sample inputs will now produce an output below the threshold and thus be classified as negatives. If this happens to be a correct prediction our specificity will go up and sensitivity will remain unchanged. If it happens to be an error, specificity is unchanged and sensitivity will go down. The gradual increase of the threshold will thus cause sensitivity to drop and specificity to increase, up to the point where we reach the other extreme. The threshold is now bigger than the biggest possible output and the model always predicts negative regardless of the input. This threshold gives us a sensitivity of 0 and a specificity of 1.

5.2.3 Receiver Operating Characteristic curve

The ROC curve is a way of showing exactly this gradual increase of the threshold. The typical way of constructing a ROC curve is to put $(1 - specificity)$ on the X-axis and *sensitivity* on the Y axis. The values for these metrics are then plotted for values of the threshold ranging from the all-positive model to the all-negative model. An example ROC curve is shown on figure 5.1. The metric we will use to evaluate the performance of different models is the area under the ROC curve (AUC). This metric

Threshold: -0.89
Sensitivity: 0.75
Specificity: 0.76
Area under curve: 0.79
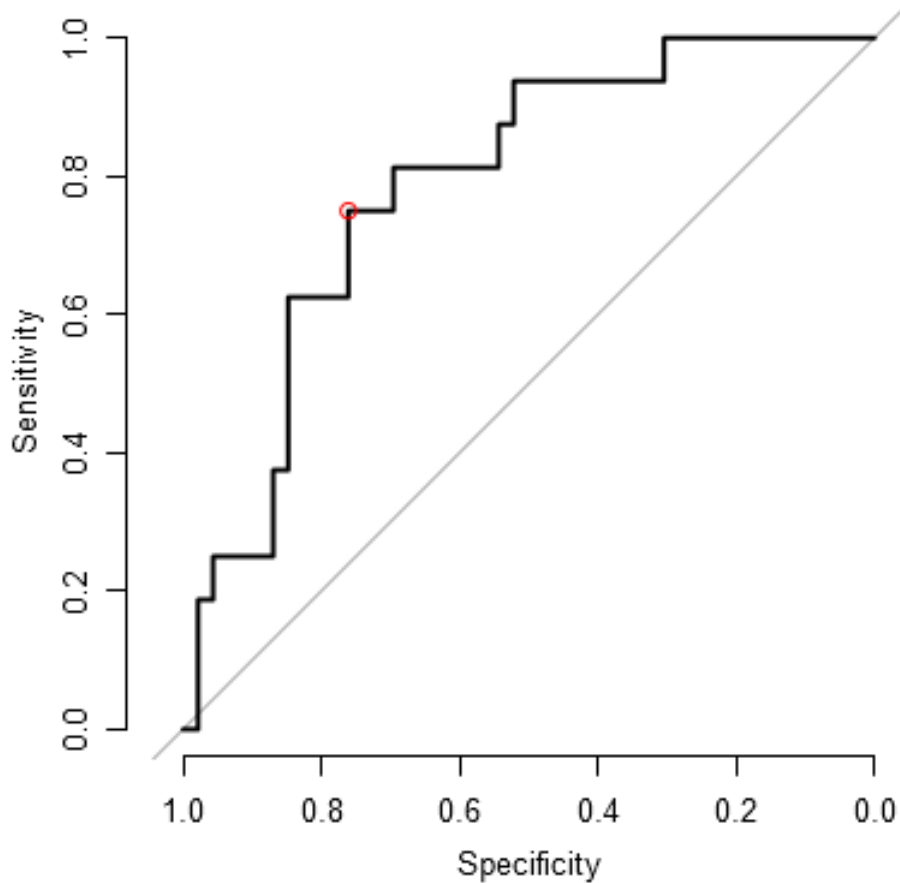
Figure 5.1: Example ROC curve and metrics

works because it indicates how far we can optimize both sensitivity and specificity for different thresholds. It shows how close we can get to the upper left corner of the plot, which indicates the perfect model. The AUC metric ranges from 0 to 1, but in fact its value is only informative in between 0.5 and 1. Imagine a model that makes completely random predictions, this model will get a correct prediction 50% of the time. If we would create a ROC curve for this model we would on average get a straight diagonal line from the bottom left to the top right. This line is therefore considered a lower bound for the performance of a model.

5.3 Case study 1: predicting stage outcome

5.3.1 Case details

The case we are showing here uses datasets created by the University Hospital of Leuven in a study on colorectal cancer patients. There are three source datasets that all contain imaging data: a dataset from MRI (Magnetic Resonance Imaging) scans, a dataset from DWI (Diffusion Weighted Magnetic Resonance Imaging) scans, and a dataset from PET (Positron Emission Tomography) scans. It is important to note that these scans were taken three times per patient at different periods in their treatment: one in the beginning, one roughly in the middle, and one near the end. This means that there is some temporal information present in the datasets, but this has been encoded in the explanatory variables. For instance: there is a variable in the MRI dataset that shows the size change of the tumour between the first and second scan. In this way we can treat this as a regular explanatory variable and we don't have to worry about the temporal aspect.

The dependent variable is a binary outcome based on whether the patient got a pathologically complete response. For a more in-depth explanation on what this means see appendix A. For now, we can view a positive outcome as a patient who has recovered from the cancer, and a negative outcome as a patient that has not. The aim is now to build a model that receives the explanatory variables from the scans as input, and predicts the outcome for that patient.

5.3.2 The results

The resulting AUC's for the individual and integrated datasets are shown respectively in tables 5.1 and 5.2. We can clearly see that the model for the individual MRI dataset contains the most predictive information out of the three datasets. This however doesn't mean that the other datasets are useless. Indeed, when we integrate the other dataset with the MRI dataset we get even better performances. From table 5.2 we can see that in this case the intermediate integration method outperforms the other strategies regardless of which datasets are used.

We have also included an overview of the model parameters that were selected (on average) by each strategy. The values in this table show the weights given to each corresponding variable in the model. A positive weight indicates that a higher value for this variable increases the chance of a positive outcome. A negative weight correlates the variable with a decreased probability of positive outcome. The overview is shown in table 5.3. The first column shows the name of the selected variable, these are imaging features. The next three columns show the model parameters for the individual models. The last two columns show the parameters that were selected by the integrated models. It is interesting to see how all models tend to select the same kind of variables and give them similar weights. This shows us that these variables indeed do contain some predictive information. Notice that the intermediate

Dataset	AUC
DWI	0.67
MRI	0.75
SUV	0.65

Table 5.1: Area under the ROC curve for models of individual datasets

	All	DWI+MRI	DWI+SUV	MRI+SUV
Early integration	0.76	0.82	0.69	0.74
Intermediate integration	0.79	0.83	0.78	0.75
Late integration	0.73	0.78	0.66	0.70

Table 5.2: Area under the ROC curve for models of various combinations of integrated datasets

	Individual			Integrated	
Variable	DWI	MRI	SUV	Early	Intermediate
ADCratioavg_TP1TP3	0.69			1.09	1.19
ADCratiohigh_TP1TP3	0.08			0.50	0.62
DeltaSphere_TP2TP3perc		1.67		1.35	1.44
DeltaSphere_TP1TP3perc		0.48		1.78	1.64
Volume_TP2		-0.99			
Volume_TP3		-0.24		-0.173	-0.94
RIdiameter_TP2TP3			0.16		
Diameter_TP3			-0.88	-0.01	
SUVmax_TP2			-0.88	-3.30	-4.41
deltadiameter_TP1TP2			-0.32		
RISUVpeak_TP1TP2				0.32	
RITLG_TP2TP3				-0.08	

Table 5.3: Overview of the model parameters (weights) for all of the computed models

model selects fewer variables than the other models, while still outperforming them. It removes those variables that have very small weights in other models and thus acts as a very strict variable selector. This is no surprise if we remember how the intermediate integration method uses two steps of variable selection to perform the integration (chapter 4, section 4.5).

5.4 Evaluating a cox proportional hazards model

The evaluation of a cox proportional hazards model is different compared to logistic regression [36][29]. This is because, as we have seen in chapter 3, we normally don't directly compute survival times from a cox proportional hazards model. This means we cannot compare them to the real survival times. However we can still evaluate the models based on statistics that indicate significance.

5.4.1 Predictions for survival models

Recall the hazard ratio from section 3.3. If we have a model computed ($\beta's$ defined) and we have a test sample (with values X_i for the explanatory variables), we could compute the following prediction quantity:

$$prediction = exp(\sum_{i=0}^{N} X_i \beta_i) \tag{5.1}$$

where

- $exp(x)$ is the exponential of x

- X_i are the values for the explanatory variables

- β_i are the model coefficients

We can view this as the hazard ratio between the test sample and an imaginary patient that has the value 0 for all explanatory variables which we will call the baseline patient (subscript b):

$$\frac{\lambda_i(t)}{\lambda_b(t)} = \frac{\lambda_0(t)e^{X_{i1}\beta_1+...+X_{iN}\beta_N}}{\lambda_0(t)e^{0\beta_1+...+0\beta_N}} = e^{(X_{i1}-0)\beta_1+...+(X_{iN}-0)\beta_N} = exp(\sum_{j=0}^{N} X_{ij}\beta_j) \tag{5.2}$$

What this quantity tells us is whether the risk of our test patient is higher or lower than the risk of the baseline patient. This does not tell us anything in the absolute sense about survival times, but we can compare this prediction value in a relative way between patients.

We could, as we did in section 5.2, compute this prediction value for all patients in our dataset using cross-validation. We now want to test the predictive value of these predictions. To do this we construct a new survival model, but this time there is only one explanatory variable and that is our vector of predictions. If we can show that this variable is significant in predicting the survival, then our original model is also significant. The significance test we will use is called the Wald Test.

5.4.2   The Wald test

The Wald test [18] is a statistical test that can be used to test the significance of an estimated parameter. The Wald test can be used on a single parameter or on a vector of parameters at once. In the case of one parameter β the formula for the Wald test is:

$$\frac{(\hat{\beta} - \beta_0)^2}{var(\hat{\beta})} \sim \tilde{\chi}_1^2 \tag{5.3}$$

where

- $\hat{\beta}$ is the maximum likelihood estimate we calculated and want to test

- β_0 is the value of β if the null-hypothesis is true

- $var(\beta)$ is the variance of β

- $\tilde{\chi}_1^2$ is the chi-squared distribution with one degree of freedom

When we are testing the significance of a coefficient in our survival model we have to compare it against a so called null-hypothesis. In this case the null-hypothesis is the model where all coefficients are equal to zero. The values for $\hat{\beta}$ and $var(\beta)$ are defined by the model. Intuitively the Wald test computes the distance between the obtained value for the parameter and the parameter value under the null-hypothesis, relative to the variance of the parameter. If the variance is large, then we are unsure about the value for the parameter that we calculated and it is harder to reject the null-hypothesis. An example is demonstrated on figure 5.2.

The Wald test can tell us something about the significance of the predictions vector and therefore about the predictive ability of our initial model. We can go one step further and compute a p-value [6][12] for the wald-test statistic. This p-value contains the exact same information as the wald-test statistic, but it is a very well known quantity statistics so we will add it for completeness.

5.5   Case study 2: predicting survival outcomes

5.5.1   Case details

The case we are showing here uses datasets from The Cancer Genome Atlas [32]. TCGA is a 2.5 petabyte database of cancer data for 33 different types of cancer made publicly available for research. We will only use a subset of this database for this case study. The cancer type we will use is called Colon adenocarcinoma, or COAD for short. From this cancer we will use 3 datasets: the first contains copy number variation (CNV) data. CNV's indicate the deletions or duplications of parts of DNA causing there to be more (or less) copies of this piece of DNA. The second dataset contains information about the messenger RNA (mRNA) which directly influences the expression of genes. The last dataset contains information
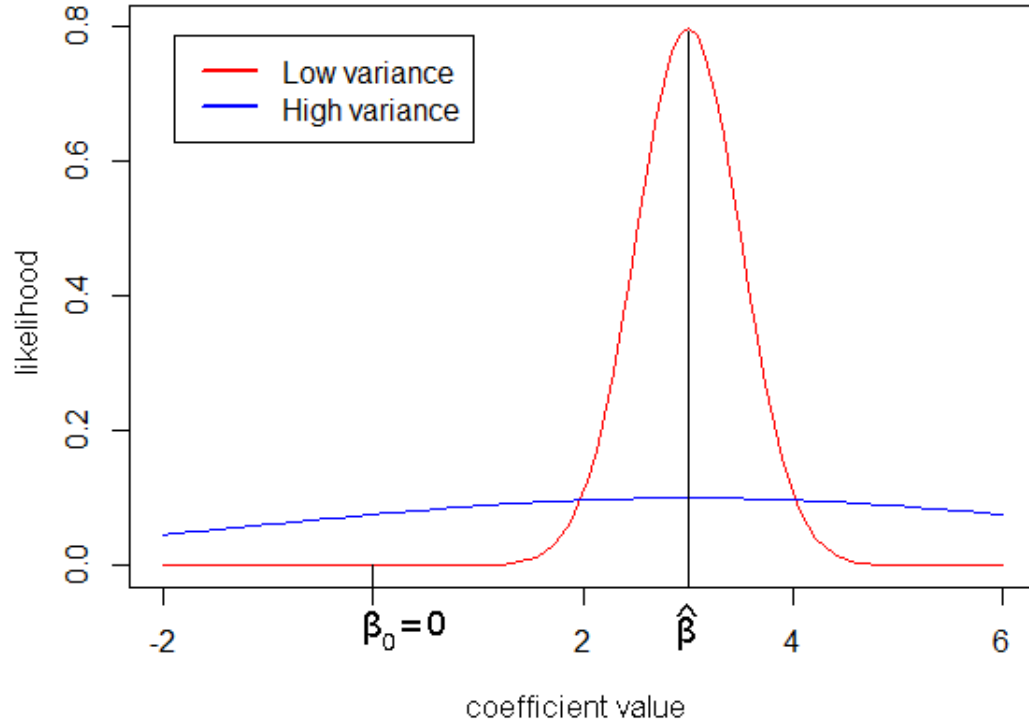
Figure 5.2: Example Wald test variance visualisation. Both curves give the same maximum likelihood estimate $\hat{\beta}$. The red curve however has low variance, providing good evidence to reject the null-hypothesis with value β_0. The blue curve has high variance, in this case $\hat{\beta}$ does not significantly differ from β_0 and we cannot reject the null-hypothesis.

about micro RNA (miRNA) which is known to have an RNA silencing function and also regulates gene expression. This last dataset is extra interesting because there is already evidence that abnormal expression of miRNA is correlated with cancer development[33][45][11]. The dependent variables in this case are survival data, and as we have seen in chapter 3 this involves a time vector and a censoring vector.

Table 5.4 presents the dimensions of the used datasets. It is worth mentioning that these datasets are all the result of a preprocessing step to make them ready for analysis. However the preprocessing is not the main focus of this thesis so we will not go into any more detail.

Dataset name	Number of variables	Number of patients
Copy Number Variations	16	219
messenger RNA	3869	219
micro RNA	414	219

Table 5.4: Dimensions for the datasets used in the survival analysis.

5.5.2 The results

The resulting performance statistics for all the models are shown in tables 5.5 and 5.6. The first table shows the statistics for the models of individual datasets. The second table shows the statistics for the integrated datasets.

5.5.3 Interpretation of the statistics

The first statistic is the hazard ratio. This number shows the multiplicative effect on the hazard for a unit increase of the variable. Remember from section 5.5 that the variable in this case is the vector of predictions. The predictions are the result of our originally trained model. A hazard ratio of 1.39 thus means that a unit increase in the outcome of the model has the effect of increasing the hazard by 39%.

The second statistic is the hazard ratio confidence interval. This shows the lower and upper bounds of the 95% confidence interval for the hazard ratio. If this interval is very wide then this means the value for the hazard ratio is uncertain. The narrower the interval, the more certain we are about the predicted value for the hazard ratio.

The third statistic is the wald test, which has been explained in section 5.4.2. The larger this number, the more significant the results of that model are. This is immediately related to the fourth, and last, statistic which is the p-value for the wald test. This value also indicates the significance of the model that we have obtained. A usual practice in statistics is to say that an observation (or model in this case) is significant if the p-value is below the 0.05 threshold.

5.5.4 Interpretation of the results

It is immediately clear that the majority of the models are not significant. The p-values are way too high, the wald test scores are too low, and the confidence intervals are too wide to say anything meaningful about the obtained hazard ratios. However it is interesting to note that all the integrated models have a significantly lower p-value when compared to the individual models. In fact, the intermediate integrated model actually reaches a p-value that is lower than the threshold of 0.05 which makes this model significant. More research would have to be performed if one really wanted to use this model to make predictions, but for our purposes this analysis is sufficient to conclude that the integrated models have outperformed

	CNV	mRNA	miRNA
Hazard ratio (HR)	1.39	0.74	0.84
HR Confidence interval	[0.29 - 6.72]	[0.14 - 3.87]	[0.64 - 1.11]
Wald test	0.17	0.13	1.46
P-value	0.6799	0.7209	0.2276

Table 5.5: Performance statistics for survival models of individual datasets

	Early	Intermediate	Late
Hazard ratio (HR)	0.96	0.90	0.13
HR Confidence interval	[0.03 - 1.47]	[0.82 - 1.00]	[0.01 - 2.24]
Wald test	2.43	3.89	1.98
P-value	0.1192	0.0485	0.1595

Table 5.6: Performance statistics for integrated survival models

the individual models and they clearly have an impact on the performance of the resulting model.

5.6 Conclusion

In this chapter we have shown the results of the two case studies that we performed. The first section (5.2) explains how we use ROC analysis to evaluate the logistic regression models that resulted from the first case study. The results of this case study (section (5.3)) show that indeed the intermediate integration strategy outperforms all other strategies aswell as the individual models, which provides evidence for the thesis aim that integration strategies have an impact on the performance of the resulting models. The third section ((5.4)) explains how we come up with predictive values in a survival analysis and how we can use the wald test to evaluate the significance of the obtained models. The last section ((5.5)) shows the results of the second case study on survival data obtained from TCGA. The results of this study are less evident than those of the first case study due to insignificance of most of the models. But we can still conclude that the integration strategies had an impact on the performance of the models.

Chapter 6

Tool for automated evaluation

6.1 Introduction

In the previous chapter we have shown that different integration strategies do make a difference for the performance of the resulting model. In order to facilitate future research on this topic, we have developed a tool that allows anyone to very easily compute all the integrated models and compare them. In this chapter we will give an overview of the used technologies (6.2) and a quick demonstration of the application (6.3).

6.2 Technologies

To develop this tool we needed a programming language that provided a lot of support for statistical algorithms, and we also wanted to create a very user-friendly interface to make an abstraction of the underlying machine learning methods.

6.2.1 The R language

The language we used to create the tool is R. R is a very popular language for statistical research as it provides many state-of-the-art algorithms in statistics. R has a huge repository with packages for all kinds of purposes. The package that we used to compute models is glmnet [30]. It is developed by researchers at the Stanford University and provides very fast implementations to compute the generalized linear models explained in chapter 2.

6.2.2 Web interface

We wanted to offer a very easy-to-use interface to the application. Therefore we chose to work with shiny [16]. Shiny is a framework for R that allows you to create a web-based front-end (using html, css, javascript, ...) that can also run R code in the back-end. This allows the creating of very interactive web applications that can run the powerful statistical analyses available in R.
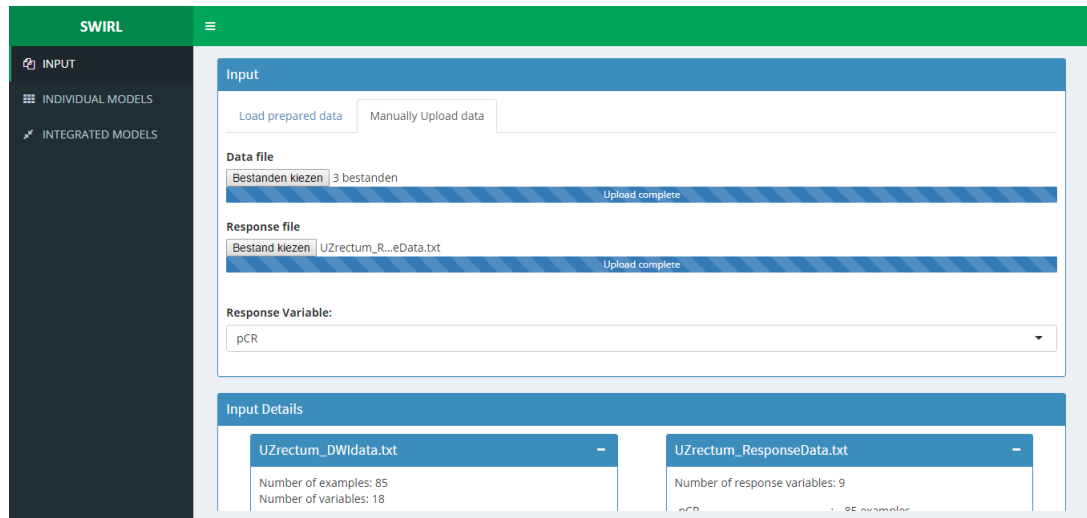
Figure 6.1: The input tab of the tool

6.3 Demonstration

The tool we developed is divided into three sections: an input section where the user can upload their datasets, a section that computes and evaluates models for each dataset individually, and an integration section where all of the integrated models, explained in chapter 4, can be computed and evaluated.

The input tab

The input tab allows the user to either use a preloaded dataset or upload their own datasets to the application. It is possible to define multiple dependent variables in one file, the tool then gives you the option to select which variable has to be modeled.

When datasets are uploaded, the tool gives a small overview of the dimensions of each dataset so the user can verify everything works correctly. Figures 6.1 and 6.2 show the input page where a user has uploaded some datasets.

The individual models tab

Once a user has uploaded their datasets they can compute a model for each dataset individually. The tool will give a concise overview of the model parameters for each model, and show an evaluation of the model. In case of logistic regression, the evaluation would be a ROC curve (explained in section 5.2). Notice that the evaluation is interactive: the user can use the slider to see the performance metrics for different thresholds. Figure 6.3 shows an example model description. Figure 6.4 shows an example ROC curve.

Figure 6.2: Example input overview. The boxes on the left show explanatory variable dataset dimensions, the box on the right shows the possible dependent variables.

Figure 6.3: An example model description showing the coefficients for the variables in the model separated by their sign.

The integration tab

The integration tab is the most important part of the application, as it allows the user to compute and evaluate integrated models for their datasets. The integration tab is further split up into early-, late- and intermediate integration tabs. Each tab allows the user to compute and evaluate its respective integration method. The results will look exactly the same as shown for the individual models tab, as can be seen on figures 6.3 and 6.4.

6.4 Conclusion

To make further research easier we have made an interactive web-based tool. The tool allows users to compute and evaluate all integration strategies explained earlier. It currently supports logistic regression models and cox proportional hazard models. The back-end uses the statistical language R and the glmnet package to compute the models. The front-end is purely web-based and this is made possible by the Shiny framework.
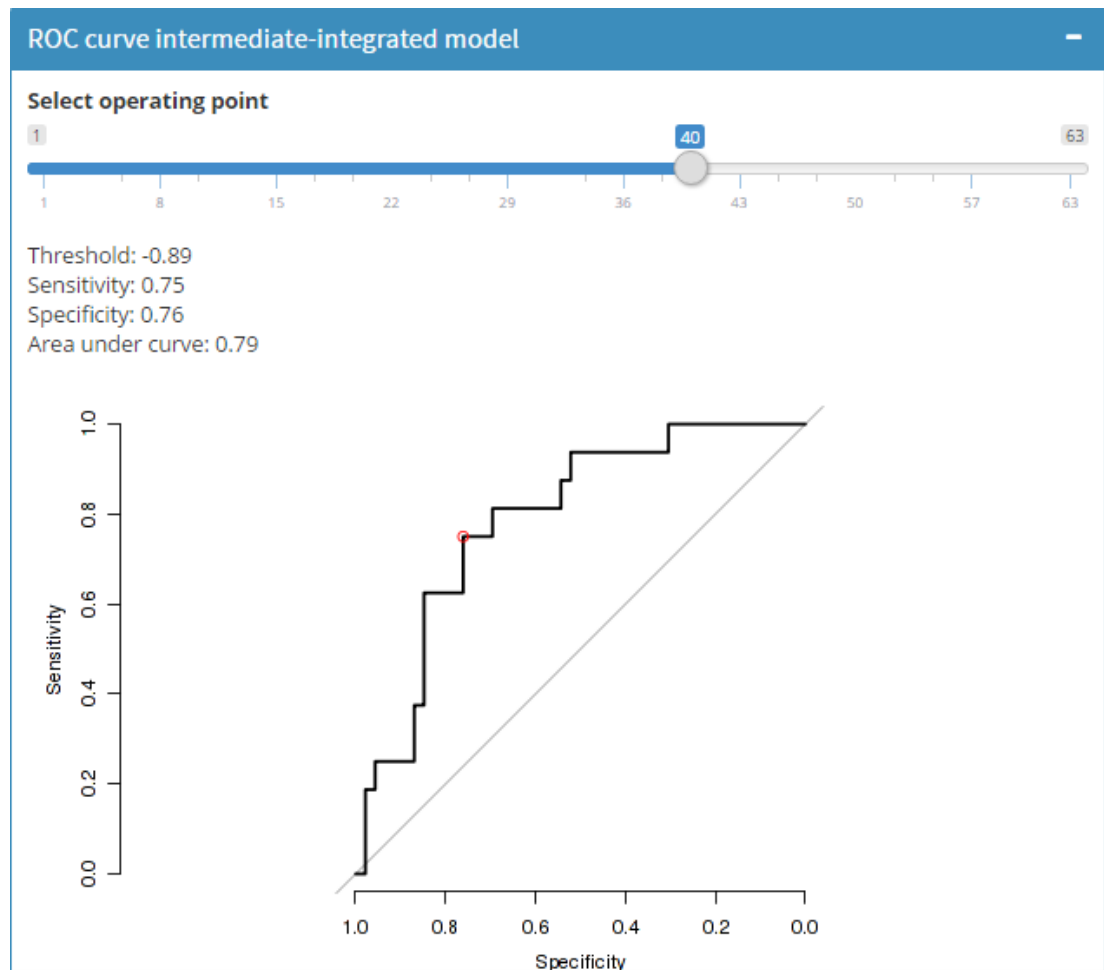
Figure 6.4: An example interactive ROC curve. The slider on top allows the user to vary the threshold. The plot interactively adjusts itself to the chosen threshold.

Chapter 7

Conclusion

The first chapter in this thesis shows that due to increasingly larger datasets there is a need for data integration methods that can join together datasets from different sources to create more powerful predictive models than the ones we can build from individual datasets. The aim of this thesis is to show that indeed different integration strategies have a varying impact on the performance of the resulting predictive models.

Chapters two and three respectively give an extensive overview of two machine learning methods: logistic regression- (chapter 2) and cox proportional hazard analysis (chapter 3). By providing a solid foundation for the concepts of training, overfitting, regularization, validation and survival analysis, the reader should be able to fully grasp the integration strategies presented in the next chapter.

The fourth chapter (4) explains the three integration strategies that we developed. The first is called early integration which is based on concatenation of datasets. The second is called late integration which builds on the ensemble learning concept. And the third is called intermediate integration which makes extensive use of the variable selection present in the lasso regularization.

In order to show that these different integration strategies have a varying impact on the performance, we applied these strategies in two case studies on real cancer data (chapter 5). The first study used logistic regression to build the predictive models. The data for this study came from the University Hospital of Leuven and concerned imaging data for patients with colorectal cancer. The variable that we tried to predict in this case was a binarized version of the pathologic stage (see appendix A). The results of the study showed an improvement in performance when using integration strategies, especially the intermediate integration strategy. The second study used cox proportional hazards analysis to build the predictive models. The data for this study came from TCGA which is a publicly available database for cancer data. More specifically we used the Copy Number Variation (CNV), messenger RNA and micro RNA datasets for colon cancer. The results in this study were a bit less evident due to insignificance of many of the resulting models. However we can still conclude that

also in this study the intermediate integration strategy proved to provide the best models. Both studies therefore provided evidence to support the aim of the thesis that integration strategies have an effect on the performance of predictive models.

In order to facilitate the computation and evaluation of the predictive models in this thesis we have developed an interactive application in the programming language R. The application allows us to easily train and evaluate the predictive models presented in this thesis. The application is built keeping in mind that this thesis is only part of a greater study. We have showed in two concrete studies that the integration methods have an impact, but this only creates more possibilities for future research. We have evaluated three integration strategies, but it should be evident that there are many more ways of integrating data and all of these should be explored in the future. Furthermore, we have used logistic regression- and cox proportional hazard models to support our aim, but there are many other machine learning methods for which similar research needs to be done.

And with that in mind we have to remember that this study is encompassed by an even larger study aim which is to find patterns in the large amounts of cancer data that is being produced. By building better predictive models, we can develop novel insights into the way cancer originates and develops. Paving the road for better prognosis and treatment of cancer in the future.

Appendices

Appendix A

Cancer staging

Cancer staging is a system that provides a means to indicate to what extent a cancer has developed. This allows for easier communication and the aim is to have a standardized way of classifying cancers. The simplest form of staging assigns a number from 1 to 4 to the cancer. Where stage 1 indicates an isolated cancer of small size, and 4 indicates a large tumour that has spread throughout the body. Another very popular, more specific staging system is the TNM system.

A.1 TNM staging system

In the TNM staging system a separation is made based on clinical stage and pathological stage. Clinical stage is based on information that can be obtained while the tumour is still present in the patient (scans, bloodtests, biopsy, ...). Pathological stage is based on information gained by microscopically inspecting the tumour after it has been removed from the patient. This process is performed by a pathologist, hence the name.

The TNM staging system uses three combinations of a letter and a number. The three letters each point to a specific type of development for the tumour, and the number indicates what the extent is of this development. The three letters are, unsurprisingly, TNM.

- T stands for the reach/extent of the primary tumour

- N stands for the spread of the tumour to regional lymph nodes

- M stands for distant spread to other parts of the body (also called metastasis)

Each letter is then joined by a number indicating the extent of the development. However, unusual situations are indicated by multiple letters. The possible combinations for each letter and their meaning is shown in table A.1. An example pathologic stage for a cancer could then be $pT1N0M0$ indicating a small primary tumour but no spread to lymph nodes or metastasis.

Code	Meaning
Tx	Tumour cannot be evaluated
Tis	Carcinoma in situ (pre-cancer)
T0	No signs of a tumour
T1,T2,T3,T4	Tumour present, with increasing size from 1 to 4
Nx	Lymph spread cannot be evaluated
N0	No spread to lymph nodes
N1	Spread to small number of nearby lymph nodes
N2	Spread to small number of distant lymph nodes
N3	Spread to multiple distant lymph nodes
M0	No metastasis
M1	Metastasis to distant organs

Table A.1: TNM staging system, meaning of codes

A.2   Pathologic stage in case study 1

In case study 1 (5.3) we use logistic regression to predict the pathologic stage outcome. To do this, the pathologic stage variable for the patients has to be binary. A value of 1 (positive case) was given to patients with a TNM pathologic stage of $pT0N0M0$ or $pT0N1M0$. These are basically the two best stage outcomes a patient can get, all other patients were labeled as 0 (negative case).

Bijlage B

Samenvatting

B.1 Introductie

B.1.1 De nood aan data integratie methoden

Recente technologische vooruitgang zorgt ervoor dat datasets groter en groter worden. Dit is een trend die in vele gebieden merkbaar is, ook in het biomedische veld. Met grotere datasets bedoelen we dat de datasets zowel in aantal variabelen toenemen, als in het aantal datapunten. Het toenemende aantal variabelen is vooral te wijten aan de technologie. We zijn op het punt gekomen dat we relatief goedkoop ieders DNA kunnen uitlezen. Als we dan weten dat een gemiddelde persoon zo'n 20.000 tot 25.000 genen heeft die coderen voor proteïnen, dan is het niet moeilijk om je in te beelden dat datasets duizenden variabelen bevatten. Een ander voorbeeld vinden we in de vooruitgang van beeldvormingsinstrumenten. Elk ziekenhuis beschikt tegenwoordig over talloze gigantisch geavanceerde scanners (MRI, PET, ...) die uiterst nauwkeurige afbeeldingen kunnen maken. Uit deze afbeeldingen kunnen heel wat variabelen worden afgeleid. Dit zijn slechts twee voorbeelden die aantonen dat het aantal variabelen enorm groeit. Naast deze groei zien we ook dat het aantal datapunten in de datasets stijgt. Ook dit heeft meerdere oorzaken. Een eerste oorzaak is simpelweg het feit dat we nu meer gegevens kunnen opslaan dan vroeger. Het is niet langer ongewoon om gigabytes of zelfs petabytes aan gegevens bij te houden. Een tweede oorzaak is dat onderzoekers over heel de wereld ijveren tot openstelling van gegevens. Als alle onderzoekers hun gegevens publiek delen met de hele wereld, dan heeft iedereen gewoonweg meer datapunten om te analyzeren, wat uiteindelijk de algemene vooruitgang zou stimuleren.

De groei van datasets brengt echter ook problemen met zich mee, we moeten namelijk zoeken naar technieken om al deze gegevens van verschillende bronnen op de beste manier met elkaar te combineren tot een coherent systeem.

B.1.2   Doel en werkwijze

Het doel van dit thesis is om enkele integratie strategieën te ontwikkelen en aan te tonen dat deze strategieën een impact hebben op de performantie van predictieve modellen. In plaats van predictieve modellen te bouwen voor individuele datasets zullen we dus proberen om meerdere datasets te combineren en 1 predictief model te bouwen dat alle gegevens gebruikt en dat een betere performantie biedt dan alle andere individuele modellen. Om dit te doen is het belangrijk dat we eerst goed begrijpen wat een predictief model is en hoe we er een kunnen opbouwen. Het eerste hoofdstuk legt uit wat we bedoelen met predictieve modellen en stelt het eerste type voor: het logistieke regressie model. Het volgende hoofdstuk geeft een tweede type van predictief model dat we overlevingsmodellen noemen. Specifiek zullen we ons focussen op het cox model. Vervolgens zullen we de verschillende integratie strategieën voorstellen. Om aan te tonen dat deze strategieën effectief een impact hebben op de performantie van de modellen zullen we twee concrete case-studies tonen. De eerste case-studie zal logistieke regressie gebruiken om predictieve modellen te bouwen, gebruik makend van alle integratie strategieën. De tweede case-studie zal hetzelfde doen voor de survival modellen. In beide case-studies zullen alle modellen geëvalueerd worden met geschikte technieken.

Om het hele proces te ondersteunen hebben we een interactieve applicatie ontwikkeld die in staat is om de performantie van predictieve modellen te evalueren. Dit liet ons toe om de twee case-studies op te bouwen, maar het vormt ook een platform voor toekomstig onderzoek.

B.2   Predictieve modellen

B.2.1   Introductie

In dit hoofdstuk leggen we uit wat een predictief model inhoudt en hoe deze worden opgebouwd. We zullen hiervoor twee concrete voorbeelden gebruiken: logistieke regressie en cox survival modellen. Verder leggen we ook het concept van regularisatie uit, dit zal belangrijk blijken in het volgende hoofdstuk over integratie strategieën. Tenslotte zullen we ook kort uitleggen wat validatie inhoudt.

B.2.2   Wat is een predictief model?

Een predictief model is een relatie tussen input variabelen en een doelfunctie (doel variabele). De taak van een predictief model is om een waarde te voorspellen voor de doelfunctie, gegeven een set van waarden voor de input variabelen. Bijvoorbeeld: we kunnen een predictief model bouwen dat de relatie probeert voor te stellen tussen een datum en de gemiddelde temperatuur op die dag. De input is hier de datum, de doelfunctie (of output) is de gemiddelde temperatuur voor die dag. We kunnen ons inbeelden dat er tussen deze twee variabelen een verband bestaat. Een datum ergens in de zomer zal namelijk een relatief hoge gemiddelde temperatuur opleveren, en een

dag in de winter een relatief lage temperatuur. We weten dit omdat we jarenlang ervaring hebben opgebouwd en ons hebben gerealiseerd dat het in de zomer normaal warm is en in de winter koud. Dit is precies wat we proberen te vatten met een predictief model, het onderliggende patroon. En de manier om daartoe te komen is door ervaring op te bouwen. Op eenzelfde manier gaan we proberen het model ervaring te laten opbouwen, we noemen dit dan 'het model trainen'. We gaan het model voorbeelden tonen van wat we willen dat het model leert, en het model zal hieruit leren en intern een representatie opbouwen om zijn kennis voor te stellen. In het eerder gegeven voorbeeld zouden we, om het model te trainen, een hele lijst met data en de corresponderende gemiddelde temperatuur voor die dag tonen aan het model, dit noemen we de training set. Het model zal dan intern een representatie opbouwen van zijn kennis en als alles goed verloopt zal deze representatie inderdaad weergeven dat het in de zomer normaal warmer is en in de winter normaal kouder. Welke representatie het model intern gebruikt is een keuze die we zelf maken, en hangt af van het type van patronen dat we willen representeren en dus van de relatie die we denken dat er bestaat tussen de input en de doelfunctie. Er zijn talloze representaties mogelijk, in dit thesis focussen we ons op lineaire representaties. Dit wil zeggen dat we veronderstellen dat we een lineaire combinatie kunnen maken van de input variabelen, en daarmee accuraat de doelfunctie kunnen benaderen. Afhankelijk van het type van de doelfunctie die we willen benaderen zijn hier echter nog verschillende methoden voor, in de volgende secties bekijken we twee concrete voorbeelden: logistieke regressie en cox modellen.

B.2.3 Logistieke regressie

In logistieke regressie maken we twee veronderstellingen. De eerste is dat er een lineair verband bestaat tussen de input variabelen en de doelfunctie. De tweede is dat, in de training set, de waarden van de doelfunctie het resultaat zijn van een binomiaalverdeling. Denk hierbij bijvoorbeeld aan het genezen of niet-genezen van een kankerpatiënt, in onze training set zullen we enkel de waarden 'genezen' en 'niet-genezen' aantreffen, maar we weten dat onderliggend deze waarden gegenereerd zijn met een bepaalde probabiliteit van overleving. De taak in logistieke regressie (en dus van ons predictief model) is om deze onderliggende probabiliteit te schatten. Stel dat we N voorbeeld datapunten hebben waarbij elk datapunt M input variabelen bevat, dan kunnen we logistieke regressie als volgt neerschrijven:

$$\hat{y}_n = \theta(\sum_{i=1}^{M} w_i x_{in}) = \theta(\boldsymbol{w^T x_n}) \qquad for \ n = 1..N \qquad \text{(B.1)}$$

hierin is

- $\hat{y}_n$ is de geschatte waarde voor de probabiliteit voor datapunt n

- w_i is de model parameter (coëfficiënt) voor input variabele i

- x_{in} is de waarde voor input variabele i voor datapunt n
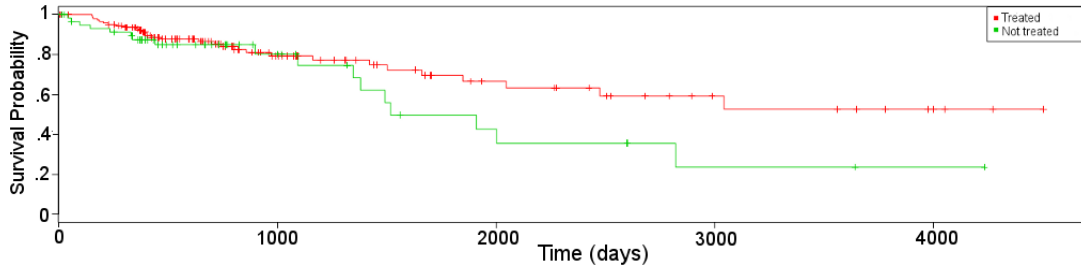
Figuur B.1: Voorbeeld kaplan-meier survival curves. De X-as geeft de tijd weer, de Y-as geeft de kans weer op overleving.

- $\boldsymbol{w^T x}$ is de vector notatie voor het inwendig product van $\boldsymbol{w}$ en $\boldsymbol{x}$

- $\theta(x)$ is de logistieke functie. Een voorbeeld voor deze function is $\frac{e^x}{1+e^x}$

Deze formule geeft weer dat, gegeven een input vector $\boldsymbol{x_n}$, ons model in staat is om zijn interne kennis (met representatie $\boldsymbol{w}$) te gebruiken om een waarde te voorspellen $(\hat{y}_n)$ voor de probabiliteit. De functie θ is de logistieke functie, deze functie is een mapping van de reële getallen naar het bereik [0..1] en zorgt ervoor dat we het resultaat kunnen interpreteren als een probabiliteit.

B.2.4 Cox proportionele risico modellen

Het volgende type modellen dat we bespreken valt onder de noemer van survival (overlevings-) modellen. Bij deze modellen proberen we de de tijd te schatten tot het voorkomen van een bepaald event op basis van de input variabelen. In ons geval is het event dat we bestuderen het overlijden van patienten aan kanker, vanaf nu zullen we dit ook zo behandelen. De doelfunctie wordt daarom ook wel de overlevingsfunctie genoemd. Om overlevingsfuncties te plotten worden vaak kaplan-meier curves gebruikt, op figuur B.1 worden twee voorbeeld overlevingsfuncties getoond. Bij overlevingsanalyze wordt echter ook vaak gebruik gemaakt van de zogenaamde hazard (of risico) functie. Deze functie geeft weer wat het risico is dat op een gegeven tijdstip het event zich voordoet. In het cox proportionele risico model dat we zullen gebruiken wordt deze risico functie als volgt genoteerd:

$$\lambda_i(t) = \lambda_0(t)e^{X_{i1}\beta_1 + ... + X_{iN}\beta_N} \tag{B.2}$$

where

- $\lambda_i(t)$ is het risico op overlijden voor patiënt i op tijdstip t.

- $\lambda_0(t)$ is de baseline risico op tijdstip t, dit is het risico op overlijden zonder enige invloed van de input variabelen.

- $X_{i1}...X_{iN}$ zijn de waarden voor de input variabelen voor patiënt i.

- $\beta_1...\beta_N$ zijn de waarden voor de parameters van het model (de kennis representatie).

Een tweede interessant concept is wat met noemt de risico-verhouding (hazard ratio). Dit is de verhouding van twee risico functies:

$$\frac{\lambda_i(t)}{\lambda_j(t)} = \frac{\lambda_0(t)e^{X_{i1}\beta_1+...+X_{iN}\beta_N}}{\lambda_0(t)e^{X_{j1}\beta_1+...+X_{jN}\beta_N}} = e^{(X_{i1}-X_{j1})\beta_1+...+(X_{iN}-X_{jN})\beta_N} \tag{B.3}$$

Merk op dat de vorm van de risico functie een keuze is die we zelf hebben gemaakt, door te kiezen voor cox proportionele risico modellen. Deze keuze heeft twee heel belangrijke gevolgen die duidelijk worden in de formule van de risico-verhouding (B.3). Het eerste gevolg is dat risico-verhoudingen onafhankelijk zijn van de tijd. Dit wil zeggen dat de risico-verhouding tussen twee patiïten dus niet verandert doorheen de tijd. Vandaar ook de naam 'proportionele risico modellen'. Het tweede gevolg is dat veranderingen in de input variabelen een multiplicatief effect hebben op het risico. Dit kunnen we ook zien in formule B.3, maar is nog duidelijker als we de risico-verhouding neerschrijven met in de noemer het risico voor een patiënt met bepaalde waarden voor de input variabelen, en in de teller het risico voor een patiënt met exact dezelfde input waarden, behalve voor 1 variabele X_j een verhoging met 1:

$$\frac{\lambda(t|X_j+1)}{\lambda(t|X_j)} = e^{(X_j+1-X_j)\beta_j} = e^{\beta_j} \tag{B.4}$$

We zien dus dat een verhoging met 1 voor een input variabele, het risico doet veranderen met e^{β_j}, de exponentiële van de bijhorende coëfficiënt.

Het predictief overlevingsmodel

Net zoals bij logistieke regressie hebben we hier dus te maken met een set gewichten die onze kennis representatie voorstellen. We zullen ook heel analoog een iteratief algoritme gebruiken om deze gewichten te bepalen. We zullen het model een hele reeks voorbeeld datapunten tonen (input variabelen met bijhorende overlevingskans) om zo het model ervaring te laten opbouwen en op die manier zo goed mogelijk de overlevingskansen van patiënten te laten schatten.

B.2.5 Regularizatie

Een belangrijk concept dat we moeten toevoegen aan onze modellen is regularizatie. Herinner je uit sectie B.2.2 dat we een predictief model opbouwen door het een reeks voorbeeld datapunten te geven om het op die manier zijn interne representatie van de kennis te laten opbouwen. In beide voorbeeld modellen die we hebben gezien is deze representatie en set van gewichten. Het concept van regularizatie houdt in dat we een of meerdere beperkingen gaan opleggen aan deze gewichten. Het doel hiervan is om bepaalde modellen (bepaalde combinaties van gewichten) te prefereren boven anderen. De onderliggende gedachte is dat we gaan trachten van simpele modellen te bekomen, die een vloeiende functie voorstellen. We doen dit om te voorkomen dat het model patronen gaat vinden die er eigenlijk niet zijn of die enkel het resultaat zijn van ruis is de gegevens. Er zijn verschillende beperkingen mogelijk die we kunnen opleggen, en elk krijgen ze hun eigen naam: de ridge penalty, de lasso penalty en het elastisch-net pentalty.

Ridge penalty

Bij de ridge penalty (ook wel L_2 panalty genoemd) is de beperking die we opleggen de volgende:

$$\sum_{i=1}^{N} w_i^2 \leq C$$

Deze beperking zal ervoor zorgen dat gewichten klein worden gehouden, en het dus onmogelijk wordt dat een gewicht extreem groot wordt relatief ten opzichte van de andere gewichten.

Lasso penalty

Bij de lasso penalty (ook wel L_1 penalty genoemd) is de beperking:

$$\sum_{i=1}^{N} |w_i| \leq C$$

Deze beperking heeft ook als gevolg dat de gewichten klein worden gehouden, maar heeft als extra dat gewichten ook effectief 0 worden indien ze niet belangrijk genoeg zijn in het model. We noemen dit daarom ook 'variabele selectie'. Gebruik van de lasso penalty heeft als gevolg dat het resulterende model vaak minder parameters gebruikt, omdat het enkel deze parameters gebruikt die echt belangrijk zijn. Deze techniek zal zeer belangrijk blijken bij een van de integratie technieken die later worden besproken.

Elastisch-net penalty

Bij het elastisch-net penalty is de beperking een combinatie van de ridge en lasso penalties:

$$\alpha \sum_{i=1}^{N} w_i^2 + (1 - \alpha) \sum_{i=1}^{N} |w_i| \leq C$$

Deze methode heeft daarom ook eigenschappen van zowel ridge als lasso, en met de parameter α kan men kiezen hoezeer men wil aanleunen bij een van de twee technieken. Dit vormt als het ware een gulden middenweg.

De parameter λ

Wanneer met regularizatie echter toepast op een concreet model, zal men altijd het optimalisatie probleem met beperking proberen omvormen naar een equivalent optimalisatie probleem zonder beperking. Bij deze omvorming introduceert men dan meestal een parameter λ die als vervanging dient voor de parameter C in bovenstaande beperkingen. λ kan men zien als de hoeveelheid regularizatie die men wil gebruiken. Hoe groter λ hoe sterker de beperking en dus hoe kleiner C. Hoe kleiner λ, hoe minder regularizatie, hoe groter C.

B.2.6 Validatie

Het laatste concept dat we moeten behandelen is validatie. Dit zal ons toelaten om een getraind model te evalueren. Bij validatie delen we onze dataset op in twee delen: een training set en een validatie set. De training set kennen we reeds, deze datapunten zullen we gebruiken om een model te trainen. Merk op dat we nu echter nog datapunten over hebben die niet gebruikt werden voor de training, dit is de validatie set. We zullen het getrainde model vragen om voor deze datapunten een predictie te berekenen voor de output variabelen. We kennen van de datapunten in de validatie set echter de correcte output waarde. We kunnen vervolgens dus de output van ons predictief model vergelijken met de juiste waarde, en op die manier een idee krijgen hoe goed ons predictief model presteert.

Merk echter op dat de training set die we gebruiken om het model te trainen kleiner is dan de volledige dataset waarover we beschikken. We houden inderdaad de datapunten in de validatie set opzij. We willen echter zoveel mogelijk datapunten gebruiken om te trainen, want dan bekomen we een beter model. Langs de andere kant willen we ook genoeg datapunten in de validatie set om een goed beeld te krijgen van de performantie van het bekomen model. Een oplossing voor deze contradictie is cross-validatie. Bij cross-validatie delen we onze dataset op in K (vaak 10) gelijke stukken. We gebruiken 1 van de K stukken als validatie set, en de andere K-1 stukken als training set. We herhalen dit proces K keren, waarbij we iedere keer een ander stuk nemen als validatie set. Op die manier hebben we op het einde voor elk datapunt in onze dataset een predictie. Deze predicties kunnen we dan vergelijken met de echte waarden en dit zal ons een redelijk goed beeld geven van de performantie van het model dat we zouden bekomen indien we op de volledige dataset zouden trainen.

B.3 Integratie technieken

B.3.1 Inleiding

In dit hoofdstuk stellen we drie verschillende integratie technieken voor. Eerst zullen we kort overlopen hoe de data eruit ziet waar we van vertrekken, en vervolgens zullen we de drie technieken overlopen. De eerste strategie noemen we vroege-integratie en is gebaseerd op het aan elkaar voegen van datasets. De tweede strategie heet late-integratie en is gebaseerd op het concept van 'ensemble-learning'. De laatste techniek heet partiële integratie en maakt gebruik van de variabele selectie in de lasso regularizatie.

B.3.2 Beschrijving van de data

Integratie duidt op het combineren van verschillende elementen. In dit geval zijn de elementen waarmee we te maken hebben datasets. Herinner je dat we, om een model te trainen, een lijst van gelabelde datapunten nodig hadden. Zo een lijst noemen we een dataset. Nu hebben we te maken met meerdere datasets, en moeten we een
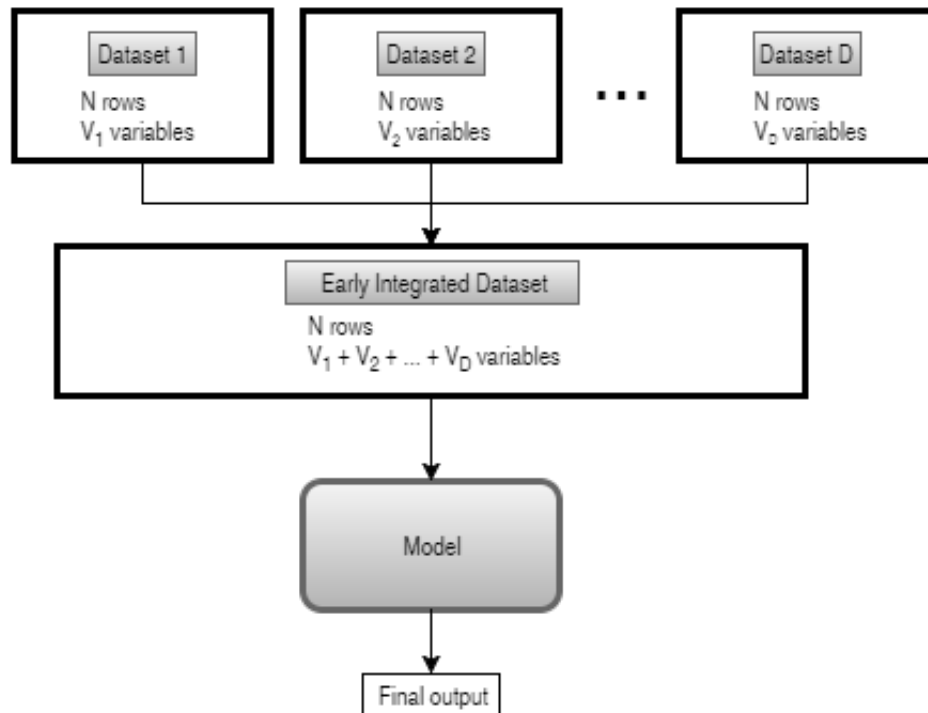
Figuur B.2: Schema voor vroege integratie

strategie bedenken om deze verschillende datasets te gebruiken om een model te trainen. Merk op dat het noodzakelijk is om datapunten over verschillende datasets aan elkaar te koppelen. Met andere woorden, alle datapunten in de datasets moeten een vorm van unieke identificatie krijgen waardoor we gegevens van verschillende datasets, die spreken over eenzelfde datapunt, aan elkaar kunnen koppelen.

B.3.3 Vroege integratie

Bij vroege integratie gaan we op voorhand (voor het trainen) alle gegevens over eenzelfde datapunt aan elkaar koppelen. Dit is mogelijk omdat elk datapunt een unieke identificatie krijgt. We kunnen dit bezien als het aan elkaar plakken van alle datasets, waardoor we uiteindelijk overblijven met slechts *één* grotere dataset. Een schematisch overzicht van vroege integratie wordt gegeven op figuur B.2.

B.3.4 Late integratie

Bij late integratie zullen we eerst voor elke dataset apart een model trainen. Als het aantal datasets gelijk is aan D dan zullen we dus ook D predictieve modellen bekomen. Wanneer we dan voor een nieuw (ongezien) datapunt een predictie moeten maken, zullen we dit datapunt aan elk van de D predictieve modellen geven. Zij zullen elk individueel een predictie maken. Vervolgens combineren we de D voorspelde waarden in een lineaire combinatie om tot een finale predictie waarde te komen. Indien we
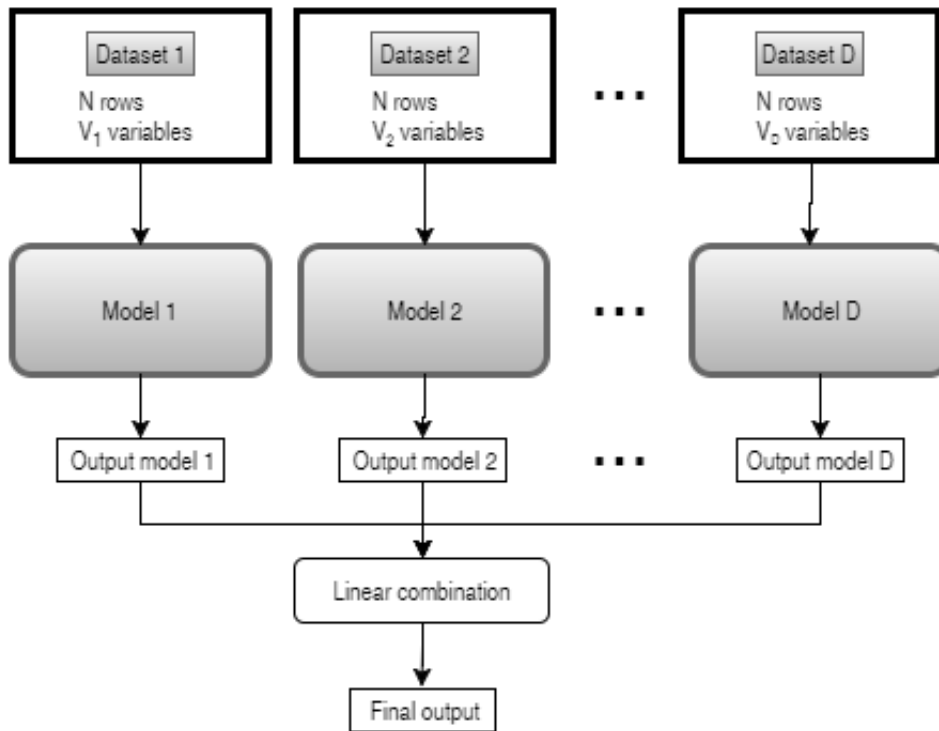
Figuur B.3: Schema voor late integratie

geen extra waarde willen hechten aan een van de D modellen kunnen we voor deze lineaire combinatie gewoon het gemiddelde nemen van alle voorspelde waarden.

In het veld van machine-leren is het concept van ensemble-leren reeds bekend. Dit houdt in dat voor een bepaald probleem meerdere verschillende modellen worden getraind op dezelfde dataset. Het combineren van de output van de verschillende modellen kan dan een beter resultaat geven vergeleken met een enkel model. Dit komt doordat de verschillende modellen mogelijks fouten maken op verschillende vlakken, en door uitmiddeling van de outputs worden deze fouten relatief verkleint. De late integratie techniek neemt deze gedachtengang over. We gebruiken nu niet meerdere modellen voor dezelfde dataset, maar we gebruiken meerdere modellen voor meerdere datasets. Door deze uitmiddeling hopen we echter eenzelfde foutenreductie te bekomen. Een schematisch overzicht van de late integratie techniek wordt getoond op figuur B.3.

B.3.5 Partiële integratie

De laatste methode die we voorstellen is de partiële integratie. Deze methode vertrekt analoog aan de late integratie door voor elke dataset individueel een predictief model op te stellen. Belangrijk hierbij is op te merken dat bij het trainen van deze modellen de lasso regularizatie wordt gebruikt. Dit doen we om gebruik te maken van de
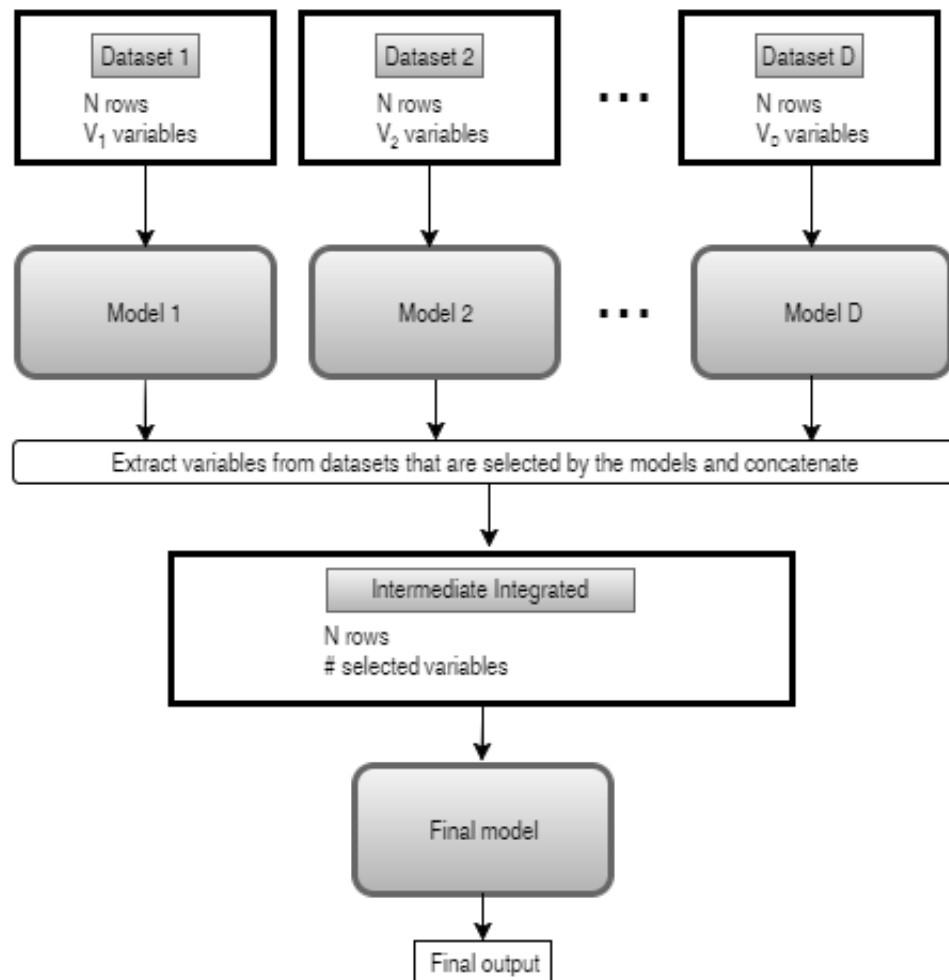
Figuur B.4: Schema voor partiële integratie

variabele selectie eigenschap. Door deze variabele selectie zullen de resulterende modellen ons namelijk vertellen welke variabelen zij belangrijk achten in de datasets. De volgende stap is dan om uit elke dataset enkel die variabelen te extraheren die door de modellen geselecteerd werden. Deze geselecteerde gegevens worden dan aan elkaar geplakt zoals in de vroege integratie, op basis van unieke identificatie van de datapunten. Dit geeft ons een nieuwe, gereduceerde, dataset. Deze dataset gebruiken we vervolgens om opnieuw een predictief model op te stellen, dit is het finale geïntegreerde model dat we zullen gebruiken voor predictie. Merk op dat deze techniek dus in twee stappen werkt, er zijn twee training fases. Een schematische voorstelling van de partiële integratie wordt gegeven op figuur B.4.

B.4 Evaluatie van integratie methoden

B.4.1 Introductie

In dit hoofdstuk zullen we onderzoeken of de verschillende integratie strategieën en impact hebben op de performantie van de resulterende modellen. We doen dit door in twee concrete case studies, op echte kankergegevens, een model uit te werken gebruikmakend van elke strategie. En vervolgens de bekomen modellen te evalueren met een passende metriek.

B.4.2 Case studie 1: logistieke regressie

Case overzicht

In de eerste case studie zullen we logistieke regressie modellen gebruiken. De gebruikte datasets komen van het Universitair Ziekenhuis te Leuven en betreffen gegevens van patiënten met darmkanker. Van deze patiënten zijn tijdens de behandeling scans genomen met een MRI scanner, PET scanner en DWI scanner. We hebben in dit geval dus te maken met drie datasets. De output variabele die we zullen trachten te voorspellen is een binaire versie van het pathologisch stadium van elke patiënt (zie appendix A). We kunnen stellen dat patiënten met een label 1 van de kanker zijn genezen, en patiënten met een label 0 niet. Deze output die het predictief model ons zal geven is dus een kans op genezing.

Evaluatie methode

We zullen de logistieke modellen evalueren met een Receiver-Operator Characteristic analyse. Herinner je dat we via cross-validatie voor elk datapunt in onze dataset een predictie konden bekomen. In het geval van logistieke regressie is deze predictie een probabiliteit. De output voor de datapunten in onze dataset zijn echter binaire waarden (de realisaties van de binomiaalverdeling). Om de twee te kunnen vergelijken moeten we een drempelwaarde definiëren. Als de voorspelde probabiliteit hoger is dan deze drempelwaarde dan voorspellen we een positieve output, anders een negatieve output. Voor elk datapunt zijn er hierbij dan vier scenario's mogelijk:

- True Positive (TP): het model voorspelt een positieve output, en dat is correct

- True Negative (TN): het model voorspelt een negatieve output, en dat is correct

- False Positive(FP): het model voorspelt een positieve output, en dat is fout (type 1 fout)

- False Negative (FN): het model voorspelt een negatieve output, en dat is fout (type 2 fout)

Merk op dat we in dit geval niet enkel zeggen of het model juist of fout was, maar ook de type fout registreren. Dit laat ons toe om volgende begrippen van sensitiviteit

65

Dataset	AUC
DWI	0.67
MRI	0.75
SUV	0.65

Tabel B.1: AUC voor modellen getraind op individuele datasets

en specificiteit te definiëren:

$$sensitiviteit = \frac{\sum TP}{\sum P}$$

$$specificiteit = \frac{\sum TN}{\sum N}$$

hierin zijn

- $\sum TP$ het totale aantal true positives

- $\sum P$ is het totale aantal datapunten met positieve output in de dataset

- $\sum TN$ het totale aantal true negatives

- $\sum N$ is het totale aantal datapunten met negatieve output in de dataset

Het perfecte model, dat altijd juist voorspelt, heeft een sensitiviteit en specificiteit gelijk aan 1. In een realistische setting is dit echter zelden haalbaar. We willen beide metrieken echter zo dicht mogelijk bij 1. We kunnen nu een plot maken waarbij we op de sensitiviteit plotten in functie van de specificiteit voor verschillende waarden van de drempelwaarde. Merk op dat inderdaad verschillende drempelwaarden ervoor zullen zorgen dat het model verschillende voorspellingen maakt en dus sensitiviteit en specificiteit zullen beïnvloeden. De curve die we bekomen op deze plot noemt men een Receiver Operating Characteristic curve. Een voorbeeld curve wordt getoont op figuur B.5. De metriek die we zullen gebruiken is de oppervlakte onder deze ROC curve, ook wel afgekort AUC(Area Under the ROC Curve). Hoe hoger deze waarde, hoe groter de waarden voor sensitiviteit en specificiteit bij verschillende drempelwaarden, en dus hoe beter ons predictief model.

B.4.3 Resultaten

De resulterende AUC waarden worden getoond in tabellen B.1 en B.2, respectievelijk voor modellen op individuele datasets, en geïntegreerde modellen. Uit de eerste tabel is het duidelijk dat de MRI dataset de meeste predictieve informatie bevat. Maar door gebruik te maken van integratie technieken kunnen we duidelijk de performantie van de modellen verbeteren. Tevens kunnen we opmerken dat, onafhankelijk van de gebruikte datasets, de partiële integratie de beste methode blijkt in dit geval.

Threshold: -0.89
Sensitivity: 0.75
Specificity: 0.76
Area under curve: 0.79
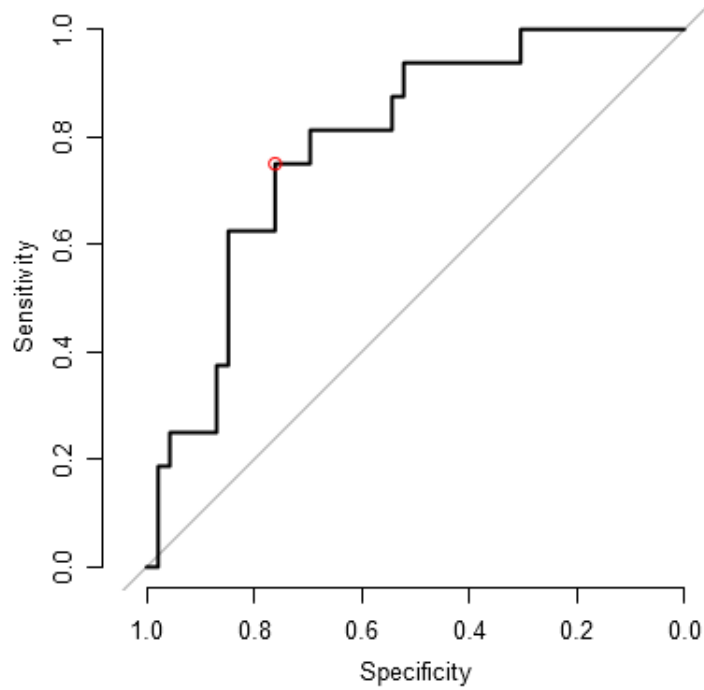
Figuur B.5: Voorbeeld ROC curve en metrieken

	Alle datasets	DWI+MRI	DWI+SUV	MRI+SUV
Vroege integratie	0.76	0.82	0.69	0.74
Partiële integratie	0.79	0.83	0.78	0.75
Late integratie	0.73	0.78	0.66	0.70

Tabel B.2: AUC voor geïntegreerde modellen voor verschillende combinaties van de datasets

B.4.4 Case studie 2: cox proportionele risico modellen

Case overzicht

In de tweede case studie zullen we cox modellen gebruiken. De gebruikte datasets komen in dit geval van The Cancer Genome Atlas. Dit is een project van het National Cancer Institute waarbij 2.5 petabytes aan gegevens werden publiek gemaakt over patiënten met 33 types van kanker. Hiervan zullen we gebruik maken van enkele datasets voor keelkanker patiënten. Meerbepaald de datasets met Copy Number Variaties, messenger RNA data en micro RNA data. Dit zijn drie datasets die aanwijzingen geven over defecten in het DNA. We zullen deze gegevens proberen te correleren met de overlevingskans van patiënten gebruikmakend van het cox model.

Evaluatie methode

Om de overlevingsmodellen te evalueren zullen we gebruik maken van een significantie test genaamd de Wald Test. Een significantie test vergelijkt het bekomen model met een ander hypothetisch model genaamd de null-hypothese. In dit geval is de null-hypothese het overlevingsmodel waarbij alle gewichten (de parameters van ons model) gelijk zijn aan 0. Dit zou willen zeggen dat geen enkele van de variabelen in de datasets een impact heeft op de overleving (of op het risico tot overlijden) van de patiënten. Een significantie-test is bedoeld om aan te tonen dat deze null-hypothese te verwerpen is, namelijk dat de gegevens die geobserveerd worden heel erg onwaarschijnlijk zijn, in het geval dat de null-hypothese waar is. Als we die onwaarschijnlijkheid kunnen aantonen dan zeggen we dat we de null-hypothese verwerpen, en dat geeft aan dat het model dat we gevonden hebben significant is. De formule voor de Wald test waarbij 1 enkele variabele wordt getest is:

$$\frac{(\hat{\beta} - \beta_0)^2}{var(\hat{\beta})} \sim \tilde{\chi}_1^2 \tag{B.5}$$

where

- $\hat{\beta}$ is de waarde voor de variabele die we zijn bekomen in ons model

- β_0 is de waarde van de variabele β indien de null-hypothese waar is, in ons geval is dat de waarde 0.

- $var(\beta)$ is de variantie van de variabele β

- $\tilde{\chi}_1^2$ is de chi-kwadraat verdeling met 1 vrijheidsgraad

Wat de Wald test dus berekent is de afstand tussen de gevonden waarde, en de waarde onder de null-hypothese, relatief ten opzichte van de onzekerheid over de variabele (de variantie). Hoe groter de waarde van de Wald test, hoe significanter het model. Een schematische voorstelling wordt gegeven op figuur B.6. In de statistiek wordt deze waarde echter vaak omgevormd tot een p-waarde. De p-waarde is een waarde tussen 0 en 1 (het is een probabiliteit), en is berekenbaar uit de waarde van
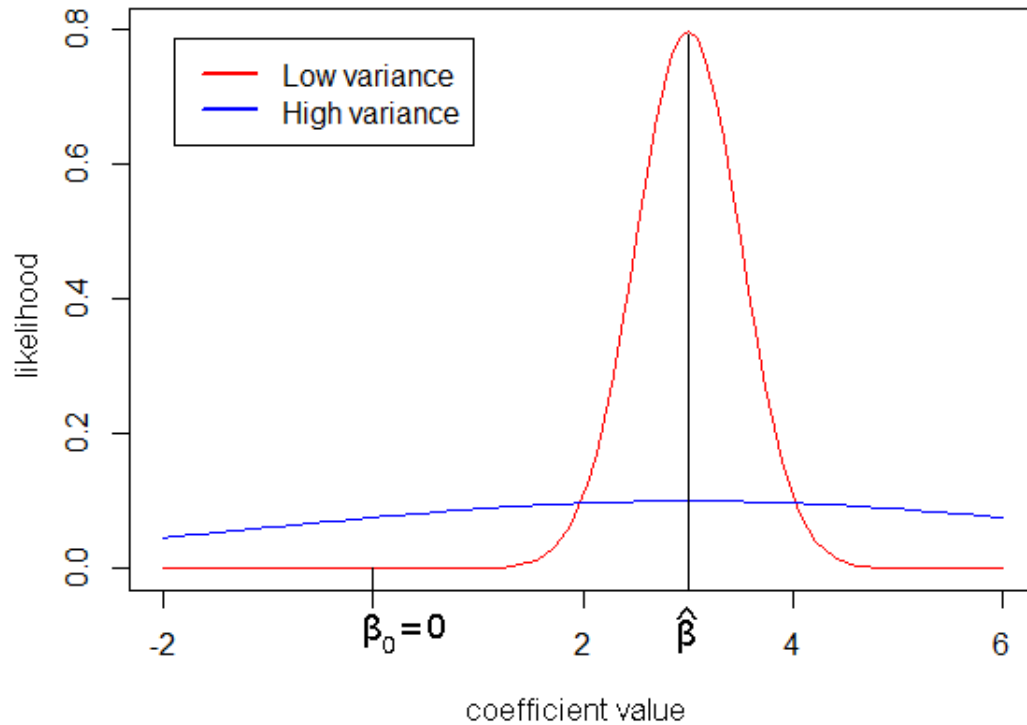
Figuur B.6: Voorbeeld Wald test variantie visualisatie. Beide curves geven dezelfde maximum likelihood schatting voor $\hat{\beta}$. De rode curve heeft echter een lage variantie, wat veel evidentie biedt om de null-hypothese te verwerpen. De blauwe curve heeft een hoge variantie, in dit geval is het niet meteen duidelijk dat $\hat{\beta}$ echt verschilt van β_0 en dus kunnen we de null-hypothese niet verwerpen.

de Wald test. Het geeft dus geen enkele extra toegevoegde waarde, maar we tonen deze waarde omwille van zijn populariteit. Daarnaast is het ook veelvoorkomend om een drempelwaarde in te stellen voor de p-waarde. Indien de p-waarde lager ligt dan deze waarde beschouwen we het model als significant. Een veelgebruikte drempelwaarde is 0.05.

B.4.5 Resultaten

Tabel B.3 geeft een overzicht van de dimensies van de gebruikte datasets. De resultaten van de significantietests voor alle modellen worden getoond in de tabellen B.4 en B.5. Het is meteen duidelijk dat bijna alle modellen niet significant zijn. De berekende p-waarden zijn duidelijk ver boven de vooropgestelde grens van 0.05. Dat

Dataset naam	Aantal variabelen	Aantal patiënten
Copy Number Variations	16	219
messenger RNA	3869	219
micro RNA	414	219

Tabel B.3: Dimensies voor de datasets die gebruikt werden in de overlevingsanalyse.

	CNV	mRNA	miRNA
Wald test	0.17	0.13	1.46
P-waarde	0.6799	0.7209	0.2276

Tabel B.4: Significantie waarden voor de overlevingsmodellen voor individuele datasets

	Early	Intermediate	Late
Wald test	2.43	3.89	1.98
P-waarde	0.1192	0.0485	0.1595

Tabel B.5: Significantie waarden voor de geïntegreerde overlevingsmodellen

terzijde is het interessant om op te merken dat alle geïntegreerde modellen duidelijk meer significant zijn dan de modellen voor individuele datasets. Meer nog, het partieel geïntegreerde model is het enige model dat het significantie niveau van 0.05 bereikt. Er zou echter meer onderzoek moeten gebeuren om de werkelijke predictieve waarde van dit model te onderzoeken, maar voor deze studie is het duidelijk dat dit model het beste model is. En dus kunnen we concluderen dat de geïntegreerde modellen ook hier aantonen dat ze een positieve impact hebben op de performantie van de bekomen modellen.

B.5 Conclusie

Het eerste hoofdstuk toont aan dat er een nood is aan data-integratie methoden door de steeds groeiende datasets waarmee we moeten werken. Door op een slimme manier verschillende datasets te integreren kunnen we predictieve modellen bouwen die beter presteren.

Vervolgens beschreven we wat predictieve modellen inhouden en we toonden twee concrete voorbeelden: logistieke regressie - en cox modellen.

Met deze informatie konden we vervolgens de verschillende integratie strategiën voorstellen. De eerste strategie, vroege integratie, voegt alle datasets aan elkaar door overeenkomende datapunten te linken. De tweede strategie, late integratie, bouwt

individuele modellen voor elke dataset, geïnspireerd op het concept van ensemble-leren. De laatste strategie, partiële integratie, bouwt het predictief model in twee stappen en maakt gebruik van de variabele selectie eigenschap in de lasso regularisatie.

Om aan te tonen dat de verschillende integratie strategieën een impact hebben op de performantie van de predictieve modellen, passen we de strategieën toe in twee case studies met echte gegevens over kankerpatiënten. De eerste studie gebruikt logistieke regressie modellen. De data voor deze studie komt van het Universitair Ziekenhuis te Leuven en betreft scan-gegevens voor patiënten met darmkanker. De resultaten van deze studie tonen een duidelijke verbetering van de performantie door het gebruik van integratie. De tweede case studie gebruikt cox proportionele risico modellen. De data van deze studie kwam van de online TCGA database. De resultaten in deze studie waren minder evident, maar we kunnen toch besluiten dat ook in dit geval de geïntegreerde modellen beter presteren dan de individuele modellen. En specifiek het partieel geïntegreerde model bleek het beste te zijn. Beide studies leveren daarom evidentie aan dat integratie van datasets inderdaad de performantie ten goede komt, en dat verschillende strategieën een verschillende impact hebben.

Het moet echter duidelijk zijn dat dit niet het einde van de studie is. In dit thesis toonden we drie verschillende integratie strategieën, maar er zijn duidelijk nog veel andere manier mogelijk om datasets te integreren. Verder spitste de case studies in dit geval zich toe op twee lineaire modellen: logistieke regressie- en cox modellen. Een interessante uitbreiding is om te kijken of deze claims ook gelden voor andere (niet-lineaire) methoden.

Daarnaast moeten we in ons achterhoofd onthouden dat deze studie kadert in een grotere studie wiens doel het is om patronen en nieuwe inzichten te vinden in de grote hoeveelheden kankergegevens waarover we beschikken. Door betere predictieve modellen te bouwen kunnen we in de toekomst nieuwe inzichten verkrijgen in de manier waarop kankers ontstaan en zich ontwikkelen. Dit zal op zijn beurt in de toekomst een weg bieden naar nieuwe behandelingen voor kanker.

Appendix C

Scientific paper

An interactive application for Regularized Linear Models

Michiel Ruelens

Promoters Olivier Gevaert Roel Wuyts

Abstract— **This document explains the development of an interactive application for constructing and evaluating regularized linear models. The models themselves are computed using the glmnet package [1] for the R scripting language. The application is built using the shiny framework for R which provides a link between R-scripts and a front-end website.**

I. INTRODUCTION

Generalized linear models are known for a long time in the fields of machine learning and data analysis. But the addition of regularization over the last few years has really made these models much more powerful. Combine this with a really fast implementation such as provided by glmnet [2] and these models really become state-of-the-art. In this paper I will explain why the addition of regularization is such a big step and how an interactive application can allow non-experts to use and interpret these regularized linear models.

II. BACKGROUND

A. Generalized Linear Models

The basis for generalized linear models are standard linear models. These models simply make a linear combination (called weights) of the input variables to produce an output. Mathematically this can be written as:

$$s = \sum_{i=1}^{d} w_i x_i$$

Where s is the output variable, x_i is the i'th input variable, and w_i is the weight corresponding to x_i. One of the simplest methods one could use is called binary linear classification. In this case we would pick a threshold. If the output s is greater than the threshold we predict one class, otherwise we predict the other class. This however is one of many possible things one could do. We could directly use the numeric value of the output as prediction, in this case the method is called linear regression. If the output is sent to a sigmoid function, we could treat it as a probability and we would call the method logistic regression.
The only difference in all these methods is the kind of output variable we are trying to predict: classes, numeric values, probabilities, ... the routine for finding the output variable however is the same for all. Therefore all these methods have been grouped under a common name: generalized linear models (GLM).

B. Computing a GLM

The way a generalized linear model is computed is by using a very well known technique called gradient descent. In this method, we start out with an initial set of weights and every iteration we improve these weights by a small amount. Repeat this process a sufficient amount of times and the weights will converge to a solution. This is possible because the weights are updated by using a so called error-surface. We define an error between our current solution and the target. We then update our weights in order to minimize this error, and therefore bring our solution closer to our target. One can view that process also as finding the minimum of the error-surface. Since the error-surface in the case of linear models is a convex surface, one can prove this method will always converge to a minimum [3].

C. Problems with GLM

A generalized linear model, computed using gradient descent, will provide a solution that can be used to predict outcomes for a set of input variables. However there is a problem with these kind of methods. The problem is often called overfitting [5]. Overfitting means that the model we have found is very good at predicting the outcomes for cases that appeared in the training set (the set of inputs that were used to find the model in the first place using gradient descent). However the model is not very good at predicting outcomes for cases outside of the training set. But this is exactly what we want the model to do. Overfitting often happens when the model we choose is too complex. Complexity, in the case of linear models, can be very roughly measured by the amount of input variables, because we associate one weight with each input variable. In current real life problems however we have a huge amount of input variables, and thus a huge amount of weights making the model very complex. These very complex models are able to fit the training data so well that overfitting makes the model completely useless outside of the training set.

D. Bias-variance tradeoff

The last things to go over, before talking about regularization, are the concepts of bias and variance [5]. In the general machine learning case we are trying to find a model h to fit a target function f. We choose a certain set of models (e.g. linear models) from which we are going to pick h. We call this set the hypothesis set. It is very important to realize that this is a choice we make, and this is a restriction we are putting on ourselves. We are confident that there is a hypothesis h in the set that fits the target function f well.

However because we are limiting ourselves to this specific hypothesis set, we are possibly making an error. We can never be sure that there even exists a hypothesis that fits the target perfectly. Call $\overline{h}$ the best possible hypothesis in our hypothesis set. This means that $\overline{h}$ is the hypothesis that best fits our target f. However, we know that $\overline{h}$ might not be perfectly able to fit f. We call this error the bias. It is an error we make due to the fact that we chose to use this specific hypothesis set.

On the other hand, when we actually use a machine learning technique to find a model h in our hypothesis set, we won't always find the best one ($\overline{h}$). If we call the model that we actually find h. Then we can define the error between $\overline{h}$ and h as the variance. This is an error we make due to the fact that our machine learning method is not perfect, and as such doesn't always find the optimal solution. Notice that the variance is linked to the size of the hypothesis set we choose. A larger hypothesis set can have very large variances, a very small hypothesis set is bound to have a small variance. This brings us directly to the concept of regularization.

III. REGULARIZATION

The ending of the last section immediately raises the question: is there a way we can influence bias and variance in order to force our machine learning method to find better models (reduce the overall error). This is exactly what regularization does. It will introduce additional bias (which raises the error), in order to reduce the variance significantly. More precisely it will add additional constraints to our hypothesis set. There are three main constraints that are used nowadays and they are called ridge-, lasso- and elastic net regularization.

A. Regularization types

In ridge regularization the added constraint is a maximum on the sum of squared weights.

$$\sum_{i=1}^{d} w_i^2 \leq C$$

Where C is the constraint constant that defines how strong the constraint is. The result of this constraint is that the model tends to go for smaller weights, and only if the weight is really important will they get larger. This form of regularization is also called the L_2 penalty.

In lasso regularization the added constraint is a maximum on the sum of the absolute values of the weights.

$$\sum_{i=1}^{d} |w_i| \leq C$$

Again, C is the constraint constant that defines how strong the constraint is. In this case the model doesn't only tend to prefer smaller weights, but this form of regularization also performs variable selection. This means that it will tend to make certain weights zero if they are not considered important. This is especially useful nowadays because we often have very large amounts of input variables, and we want to use only these variables that are actually useful. It also reduces the size of the model we end up with. This form of regularization is also called the L_1 penalty.

The last form of regularization I will discuss is the elastic net. This type is really just a combination of the two previously mentioned L_1 and L_2 penalties.

$$\alpha \sum_{i=1}^{d} w_i^2 + (1-\alpha) \sum_{i=1}^{d} |w_i| \leq C$$

Where α indicates the proportion of L_1 and L_2 penalty. This type of regularization has the added benefit that it performs well in the case of group selection. Sometimes there are sets of input variables that are very correlated. In this case we would like to have the corresponding weights correlated too. The elastic net penalty tends to do this properly, while the other two penalties do not [4].

There are a number of reasons why regularization helps us to find better models. It tends to prefer smaller weights, making it impossible for a small number of weights to get out of control. It performs automatic variable selection reducing the model size and helping eliminate useless input variables. It also allows for group selection which can be very useful in certain scenarios. And lastly but most importantly it strongly reduces the possibility of overfitting by the introduction of the constraints. There will be a slight bias error introduced, but the amount we gain by lowering the variance error is far more important.

IV. EVALUATING A MODEL

A. Cross validation

To evaluate a model we would like to use the model on some examples that it hasn't seen before (that haven't been used for training). However what we usually have is one set of examples and we also would like to use all of them to train our model. The solution here is cross-validation. In this technique we split up the initial dataset into chunks (usually ten). We then proceed to use one of the chunks as test set and use the other nine to train a model. We repeat this process ten times, using each chunk once as the test set. If we then take the average performance of the ten models we just computed we can get an estimate for the performance of the model that we would get if we used the full set for training.

B. ROC analysis

The way in which we test the performance of a single model against a test set is called ROC analysis. In this method, we distinguish four different cases when a model predicts an outcome for a test example.

- True positive: the model predicts positive, and the true condition is positive
- True negative: the model predicts negative, and the true condition is negative
- False positive: the model predicts positive, and the true condition is negative

- False negative: the model predicts negative, and the true condition is positive

These four cases reflect the performance of the model. In practice however we usually don't use these numbers by themselves, but rather combine them to make new characteristics for the model. The five main characteristics used are accuracy, sensitivity, specificity, positive predictive value (PPV) and negative predictive value (NPV).

- $Accuracy = (TP + TN)/(TP + TN + FP + FN)$
- $Sensitivity = TP/(TP + FN)$
- $Specificity = TN/(TN + FP)$
- $PPV = TP/(TP + FP)$
- $NPV = TN/(TN + FN)$

A last metric that is often very indicative of the model performance is the Area Under the Curve (AUC). This is the area under the ROC curve that is obtained by plotting sensitivity against (1-specificity) for all possible thresholds, or also called operating points. The area under the curve is an indication of how well the model is able to distinguish between positive and negative cases. A model that randomly predicts positive and negative would have an AUC of 0.5. The perfect model would have an AUC of 1. In practice models that have an AUC below 0.7 are considered bad. Between 0.7 and 0.9 is acceptable. Higher than that would be exceptional.

In all of this there is really only one degree of freedom. It is the threshold that determines whether a certain output is classified as a positive or a negative case. Changing this threshold will change the predicted outcome for the examples and thus influence all the aforementioned metrics.

V. Interactive Application

If someone wants to use regularized linear models and evaluate their performance, the details of the algorithms can get very tedious. It is for this very reason that I will develop an interactive application using the shiny framework for R. The application will make abstraction of all the details of regularized linear methods and simply provide the end-user with sliders to control the variables that really matter. One of these variables is the threshold or operating point. This point directly influences the evaluation metrics discussed earlier and is very application dependent. Some applications will require a very high sensitivity, while others really don't care about this but need a very high specificity or PPV. Changing the operating point can allow end-users to search for the optimal point for their specific application.

In current applications their are often multiple data sources. It is not always clear in advance which data sources should be used and how they should be combined. For instance the data sources could all be combined at the very start to create 1 big data source and use that for learning (called early integration). Another approach could be to treat each source individually and construct a model for each data source. Then when predicting the outcome for a test example, one could take the outcomes of all the models and combine them to create a final answer (called late integration). The application could perform different kind of integration methods at the same time and provide the user only with the best method, or allow the user to specify the kind of integration it wants to use.

The application would thus deal with the tedious bits so the end-user can focus on interpreting the results and building the optimal model for their purpose.

References

[1] https://cran.r-project.org/web/packages/glmnet/index.html
[2] Jerome Friedman, Trevor Hastie, Rob Tibshirani, Regularization Paths for Generalized Linear Models via Coordinate Descent, Journal of Statistical Software January 2010, Volume 33, Issue 1
[3] Geoffrey Hinton; Nitish Srivastava; Kevin Swersky. Part of a lecture series for the Coursera online course Neural Networks for Machine Learning
[4] Hui Zou and Trevor Hastie, Regularization and variable selection via the elastic net. Stanford University, USA, 2004
[5] Yaser Abu-Mostafa, Machine Learning online course, http://work.caltech.edu/telecourse.html

Appendix D

Poster

Design, implementation and evaluation of data integration methods for biomedical cancer data

KATHOLIEKE UNIVERSITEIT LEUVEN

FACULTEIT
INGENIEURSWETENSCHAPPEN

Master
Computer Science

Master thesis
Michiel Ruelens

Promotors
Roel Wuyts
Olivier Gevaert

Academy year
2015-2016

Background

Ever increasing datasets due to:

- Cheaper DNA sequencing
- High-resolution imaging (MRI, PET,...)
- Globalisation of research

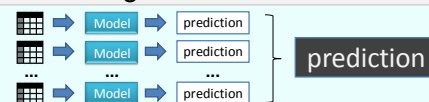## Challenge

Integrating datasets into a predictive model

Machine learning
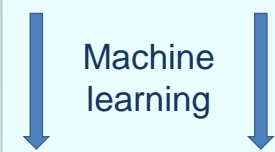
Predictive Model → prediction

Integration strategies

Early integration

Model → prediction

Late integration

Model → prediction
Model → prediction
...   ...
Model → prediction

→ prediction

Intermediate integration

Model →
Model →
...   ...
Model →

→ Model → prediction

Results

Case study 1		Case study 2	
Integration	AUC	Integration	p-value
None	0.75	None	0.22
Early	0.82	Early	0.12
Late	0.78	Late	0.16
Intermediate	**0.83**	**Intermediate**	**0.05**

Bibliography

[1] Ensemble learning. https://en.wikipedia.org/wiki/Ensemble_learning.

[2] Generalized linear model. https://en.wikipedia.org/wiki/Generalized_linear_model.

[3] Gradient and directional derivative. http://math.oregonstate.edu/home/programs/undergrad/CalculusQuestStudyGuides/vcalc/grad/grad.html.

[4] Gradient descent. https://en.wikipedia.org/wiki/Gradient_descent.

[5] Gradient descent variants. http://sebastianruder.com/optimizing-gradient-descent/.

[6] Hypothesis testing (p-value approach). https://onlinecourses.science.psu.edu/statprogram/node/138.

[7] Linear classification. https://en.wikipedia.org/wiki/Linear_classifier.

[8] Linear regression. https://en.wikipedia.org/wiki/Linear_regression.

[9] Logistic regression. https://en.wikipedia.org/wiki/Logistic_regression.

[10] Logistic regression error surface proof of convexity. http://mathgotchas.blogspot.be/2011/10/why-is-error-function-minimized-in.html.

[11] Micro rna. https://en.wikipedia.org/wiki/MicroRNA.

[12] P-value. https://en.wikipedia.org/wiki/P-value.

[13] Proportional hazards model. https://en.wikipedia.org/wiki/Proportional_hazards_model.

[14] Receiver operating characteristic. https://en.wikipedia.org/wiki/Receiver_operating_characteristic.

[15] Sensitivity and specificity. https://en.wikipedia.org/wiki/Sensitivity_and_specificity.

[16] Shiny framework for r. http://shiny.rstudio.com/.

[17] Understanding the gradient. https://betterexplained.com/articles/vector-calculus-understanding-the-gradient/.

[18] Wald test. https://en.wikipedia.org/wiki/Wald_test/.

[19] P. Y. Abu-Mostafa. Machine learning online course. https://work.caltech.edu/telecourse.html, 2012.

[20] M. A. Babyak. What you see may not be what you get: a brief, nontechnical introduction to overfitting in regression-type models. *Psychosomatic medicine*, 66(3):411–421, 2004.

[21] W. C. Black and H. G. Welch. Advances in diagnostic imaging and overestimations of disease prevalence and the benefits of therapy. *New England Journal of Medicine*, 328(17):1237–1243, 1993.

[22] F. S. Collins, E. D. Green, A. E. Guttmacher, and M. S. Guyer. A vision for the future of genomics research. *Nature*, 422(6934):835–847, 2003.

[23] P. R. Cox. *Life Tables*. Wiley Online Library, 1972.

[24] T. G. Dietterich. Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer, 2000.

[25] T. G. Dietterich. Ensemble learning. *The handbook of brain theory and neural networks*, 2:110–125, 2002.

[26] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[27] M. Goel, P. Khanna, and J. Kishore. Understanding survival analysis: Kaplan-meier estimate. *International journal of Ayurveda research*, 1(4):274, 2010.

[28] K. Hajian-Tilaki. Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation. *Caspian journal of internal medicine*, 4(2):627, 2013.

[29] F. E. Harrell, K. L. Lee, and D. B. Mark. Tutorial in biostatistics multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in medicine*, 15:361–387, 1996.

[30] P. T. Hastie and J. Qian. Glmnet vignette. http://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html, 2014.

[31] S. Healthcare. Medical imaging types. http://www.medicalradiation.com/types-of-medical-imaging/.

[32] N. C. Institute. The cancer genome atlas. http://cancergenome.nih.gov/.

[33] M. V. Iorio, M. Ferracin, C.-G. Liu, A. Veronese, R. Spizzo, S. Sabbioni, E. Magri, M. Pedriali, M. Fabbri, M. Campiglio, et al. Microrna gene expression deregulation in human breast cancer. *Cancer research*, 65(16):7065–7070, 2005.

[34] D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.

[35] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145, 1995.

[36] R. B. Newson et al. Comparing the predictive powers of survival models using harrell's c or somers' d. *Stata Journal*, 10(3):339, 2010.

[37] G. Rodriguez. Survival models. http://data.princeton.edu/wws509/notes/c7.pdf.

[38] P. Royston and A. House. Estimating a smooth baseline hazard function for the cox model. *London: Department of Statistical Science, University College London*, 2011.

[39] A. Sandelin, W. Alkema, P. Engström, W. W. Wasserman, and B. Lenhard. Jaspar: an open-access database for eukaryotic transcription factor binding profiles. *Nucleic acids research*, 32(suppl 1):D91–D94, 2004.

[40] J. Shendure and H. Ji. Next-generation dna sequencing. *Nature biotechnology*, 26(10):1135–1145, 2008.

[41] N. Simon, J. Friedman, T. Hastie, R. Tibshirani, et al. Regularization paths for coxs proportional hazards model via coordinate descent. *Journal of statistical software*, 39(5):1–13, 2011.

[42] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L. J. Goldberg, K. Eilbeck, A. Ireland, C. J. Mungall, et al. The obo foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology*, 25(11):1251–1255, 2007.

[43] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. Biogrid: a general repository for interaction datasets. *Nucleic acids research*, 34(suppl 1):D535–D539, 2006.

[44] R. Tibshirani et al. The lasso method for variable selection in the cox model. *Statistics in medicine*, 16(4):385–395, 1997.

[45] N. Yanaihara, N. Caplen, E. Bowman, M. Seike, K. Kumamoto, M. Yi, R. M. Stephens, A. Okamoto, J. Yokota, T. Tanaka, et al. Unique microrna molecular profiles in lung cancer diagnosis and prognosis. *Cancer cell*, 9(3):189–198, 2006.

[46] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

[47] M. H. Zweig and G. Campbell. Receiver-operating characteristic (roc) plots: a fundamental evaluation tool in clinical medicine. *Clinical chemistry*, 39(4):561–577, 1993.

Fiche masterproef

Student: Michiel Ruelens

Titel: Design, implementation and evaluation of data integration methods for biomedical cancer data

Nederlandse titel: Design, implementatie en evaluatie van data integratie methoden voor biomedische kankergegevens

UDC: 621.3

Korte inhoud:

Meer en meer worden we geconfronteerd met grootschalige problemen. Datasets met duizenden variabelen zijn eerder de norm dan de uitzondering en dit is niet anders in de biomedische wereld. We zijn tegenwoordig in staat om het volledige genoom van een patient te sequentiëren, expressie-niveau's van genen te bepalen en proteïnewaarden in het bloed te meten. En door hoogtechnologische instrumenten zoals MRI en PET scanners krijgen we zeer duidelijke beelden van de problemen waar we voor staan. De taak is nu om al deze bronnen van informatie te gebruiken om tot nieuwe inzichten en behandelingen te komen. In dit thesis gebruiken we de technieken van artificiéle intelligentie om predictieve modellen te bouwen. In essentie zoeken deze technieken naar patronen in de gegevens om voorspellingen te kunnen maken over interessante varaiabelen. Om de verschillende grote datasets te gebruiken als input voor deze technieken is het echter nodig een methodiek te definiëren waarop de datasets worden samengevoegd, we noemen dit data integratie en dit vormt de essentie van de thesis. We presenteren drie verschillende integratie methoden die elk de data op een ander niveau integreren. Vervolgens evalueren we de integratie technieken op twee concrete case-studies. Beide studies gebruiken gegevens omtrent darmkanker, bij de eerste studie gebruiken we een logistisch regressiemodel om pathologische respons te voorspellen, bij de andere stellen we een overlevingsmodel op met behulp van cox regressie. Beide studies tonen aan dat de performantie van de modellen afhankelijk is van de gebruikte integratie techniek. Dit geeft aan dat onderzoekers in de toekomst meer aandacht moeten besteden aan de manier waarop gegevens worden geïntegreerd. En dat er onderzoek mogelijk is naar nieuwe integratie technieken.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie
Mens-machine communicatie

Promotoren: Prof. dr. ir. Roel Wuyts
                      Prof. dr. ing. Olivier Gevaert

Assessoren: Mario Cruz Torres
                     Prof. dr. ir. H. Blockeel

Begeleiders: Ir. A. Assistent
                      D. Vriend