RESEARCH ARTICLE (preprint)

Machine learning in medicine using JavaScript: building web apps using TensorFlow.js for interpreting biomedical datasets

Jorge Guerra Pires[1]*

Abstract: Introduction: Contributions to medicine may come from different areas; and most areas are full of researchers wanting to support. Physists may help with theory, such as for nuclear medicine. Engineers with machineries, such as dialysis machine. Mathematicians with models, such as pharmacokinetics. And computer scientists with codes such as bioinformatics. Method: We have used TensorFlow.js for modeling using neural networks biomedical datasets from Kaggle. We have modeled three datasets: diabetes detection, surgery complications, and heart failure. We have used Angular coded in TypeScript for the implementation of the models. Using TensorFlow.js, we have built Multilayer Perceptrons (MPLs) for modelling our datasets. We have employed the training and the validation curves to make sure the model learnt, and we have used accuracy as a measure of goodness of each model. Results and discussion: We have built a couple of examples using TensorFlow.js as machine learning platform. Even though python and R are dominant at the moment, JavaScript and derivatives are growing fast, offering basically the same performance, and some extra features associated with JavaScript. Kaggle, the public platform from where we downloaded our datasets, offers a huge amount of datasets for biomedical cases, thus, the reader can easily test what we have discussed, using the same codes, with minor chances, on any case they may be interested in. We were able to find 92% of accuracy for diabetes detection, 100% for surgery complications, and 70% for heart failure. The possibilities are unlimited, and we believe that it is a nice option for researchers aiming at web applications, especially, focused on medicine.
Keywords: bioinformatics — tensorflow — JavaScript — diabetes — medicine — machine learning — Angular

Resumo:
Palavras-Chave:

[1] *Founder at IdeaCoding Lab / JovemPesquisador.com, Brazil*
*__Corresponding author__: jorgeguerrabrazil@gmail.com

1. Introduction

Contributions to medicine may come from different areas; and most areas are full of researchers wanting to support. Physists may help with theory, such as for nuclear medicine. Engineers with machineries, such as dialysis machine. Mathematicians with models, such as pharmacokinetics. And computer scientists with codes such as bioinformatics.

Mathematical models have being applied to medicine for a long time, and eventually they become algorithms, which eventually may become software/packages. Those models can be classified in two big groups: white box models and black box models.

White box models are generally created for specific problems, and they concentrate on details. Their drawbacks is that they are specific, too much energy is expended on a model that cannot be applied elsewhere, except for derivations from the problem solved. They are ideal for simulations and considering scenarios. On the other hand, black box models are generic, and not tied up to specific applications. The strong point is: if one area makes progress, it is automatically transferable to other areas: those models do not consider applications, they are created based on general principles. TensorFlow,js, the technology we are going to use herein, was not designed for medicine, but it uses artificial neural networks as machine learning approach, which has been explored extensively to support medicine. This makes all the achievements elsewhere easily transferable, we shall take advantage of that observation herein.

We shall demonstrate the power of TensorFlowjs, already proven elsewhere, for biomedical problems [1], using a couple of realistic problems, but simple enough to be solved herein.

We shall make the point that JavaScript is also a nice option for machine learning applied to medicine.

One group of researchers that have been changing medicine a lot in the last decades are computer scientists; and those models are a direct example. Computers are everywhere in medicine, and with the rise of artificial intelligence and computer power becoming cheaper and more accessible, it will accelerate this change (e.g., chatGPT in medicine [2]).

The point is: we need to make those apps available, as user-friendly as possible, as easily accessible as possible [3]. An informal survey showed that model parametrization (i.e., getting a model to execute a specific task) is the biggest issues amongst life scientists working with software [4]. Thus, it is something we must consider heavily when desiging our apps: we should strongly think of the final end of the cycle, the user!

Java was the very first computer language built to serve the browser, and it had its moment; the name JavaScript was given to call attention from the Java users. So far, all computer languages (e.g., C++, C, Fortran) were designed for desktop applications, to be used locally, on the user's computers; they needed installation and constant maintenance; it was common to make software available as a command prompt. They were what is now called synchronous programming: the order of your lines of codes will dictate how it is executed, the user had no saying on this paradigm applied to coding. So, it came the idea of asynchronous programming: the user will interact with the program, and dictate how the code will evolve; to be fair, Java also had buttons and more, called listeners. The code will wait for user's interaction, which triggers a set of lines of actions, which by themselves can be asynchronous.

It gave rise to multithreading: the ability of a code to handle more than one user at a time without requiring multiple copies of the program running on the computer, without blocking the main thread. It allowed also multiple tasks running simultaneously, and the app will still respond to the user normally; the actions go to background, and once it is finished, they give back the results.

We had our own experience with Java for creating applications for biomedical cases [5]. Our experience showed a couple of setbacks: i) graphs (e.g., result plotting) and ii) user interfaces (e.g., UI and UX) are hard to design; not to mentions other setbacks such as string manipulation.

We believe that similar setbacks will appear in any desktop computer languages, that includes python. It is true that we can nowadays build a whole app using different languages, and we have done that [6]. The solution found by python programmers, as an example, is using frontend frameworks/libraries (e.g., Django and Angular). The main issue with this workaround is having to use several languages and keeping different servers; for sure, you will need a bigger team, which increases costs, and difficulties to execute the project. Heroku, as an example, provides a possibility to easily build a pipeline based on different apps; thus, they work as one big program from the outside. Notwithstanding, still having to handle several apps in several languages.

In our case when dealing with this issues of having codes in different languages serving the same problem, we have used Galaxy [6]. It does not matter which choice you make, one may be better than other, they still require handling servers, and different codes in different language. The easily seen disadvantage of that is a bigger team; we all know that most research groups operate under zero or no research funding at all, not to say that actually making money from those scientific apps is a challenge. In our case [6], it made sense to choose Galaxy since the codes were already built by previous researchers, and rebuilding would be hard. The case we present herein is interesting mainly if you are starting, or have reasons to migrate to JavaScript. If you are starting a new project that requires machine learning, we recommend starting with JavaScript, and you will be able to work in a startup paradigm [7]: test it fast, as fast as possible. Not sure it is a good strategy to rebuild a complex app from scratch, unless you have reasons to believe it will improve.

Recently, thanks to several libraries, JavaScript became a viable option for designing machine learning. Nowadays, the most well-accepted and showing promising results is TensorFlow.js [8, 9, 10]. Thus, add all the advantages of working with JavaScript and machine learning in one web application.

For our case, the core advantages are: i) the data will never leave the browser, ideal for sensitive data, most likely the case for medical information; ii) the calculations are done on the browser, no need to buy expensive server for scientific computations, ideal for startups and similar experimental endeavours. We believe that startups may support on decreasing the ever-growing costs of medical assistance [11]. They have already done that with text processing using artificial intelligence (e.g., chatGPT).

We have also added another ingredient to this solution: Angular [12]. Angular is a Single Page Application (SPA) creator, and we have already tested it for scientific computation [6]. For our case, the core advantage, on a possible real-scenario: since it is a SPA, the information will never leave the browser. When you have a server (e.g., python), the user data will be sent to the server, and this is well-known to open space for attacks on sensitive data. Talking to servers is also well-know to create delays. We had an experience in [6] with FASTA files that due to their big size, it was hard to keep sending those data around since HTTP calls have limits; we needed programming tricks to actually show the dataset at the frontend.

Our goal here is showing the reader how easy it is to build powerful machine models using TensorFlow.js. Even though we know it is not straightforward to build models in medicine, we can arrive to promising results, with free libraries, and in seconds; of course, using model in real-scenario may require more care, such as making sure the dataset used is not biased against your case. This is just possible due to current state of the art both in web app development and machine learning.

Our approach for achieving this goal is presenting a set of problems using public biomedical datasets. Those datasets

are available freely at Kaggle, a plataform where people make available datasets, and programmers can test their skills on those datasets; competitions also are from time to time created. We also add discussions on the topic of using models in medicine, a topic the authors have being working on for years.

We hope also to call attention to this platform, which collects a sea of free-available datasets. As one example, our dataset for diabetes detection has 10.000 samples, even though we use only 300 samples: go back in time, a couple of years, and such public datasets would not exist so easily for machine learning testing, or any similar data science driven model.

All artificial intelligence nowadays is a prediction machine [13]: their goal is predicting what is next, based on a set of information called features. chatGPT is about prediction of what is the next word on a given text [14], we can predict snakes based on their details from an image [15], and we are going to make some predictions about medical conditions, such as diabetes. based on given features.

According to [13], those prediction machines can be compared to electricity and computer power: as they became cheaper and more accessible, new possibilities became real. Nowadays, those prediction machines are becoming cheaper, most of the time free, and easily used (e.g., pretrained models, APIs and public libraries). We are going to explore it herein, using a public library called TensorFlow.js, making the point that those prediction machines can be widely used, by a broad audience, no cost at all; and with all their capabilities and power to predict. We have already made the point on another article for image classification applied to biology [15], using another platform called Teachable Machine, which is built on top of TensorFlow.js.

The remaining sections are. In Methods, we talk about what is behind our results, the tools, paradigms and methodologies; our hope is that people outside bioinformatics and artificial intelligence could actually read the paper, and get insights for a possible alternative area of research. On this section, we provide also links to GitHub repositories and Google Sheets used on the simulations. Then we shift to results and discussion, where we talk about the simulations we did, and we use the opportunity to add our own work experience with computational biology and computational intelligence; we use graphs and tables to present our results. We finally close the results and discussion section with a summary. One can find at the ending of the article our main references.

2. Methods

On this section, we present our tools, methodologies and paradigms. We have done our best to report all the important details, for a possible replication of the findings or even adaptation to a specific case. If we have forgotten anything important, please, do not hesitate to get in touch.

2.1 Neural networks and machine learning: supervised learning

Artificial Neural Networks, neural networks for short, are a subset of machine learning techniques, focused on numeric algorithms, different from alternatives from artificial intelligence, those algorithms are not focused on reasoning; even though non-experts using those model may think they reason. Moreover, even though they are inspired by the brain, they do not replicate the dynamics of neurons with fidelity. Their only goal is learning from samples, and it does not matter how; in most of the time, it is close superficially from the brain workings.

We are going to use the supervised variety: one presents a set of inputs, and expected outputs (based on human's feedback, a process called annotation), and the algorithm should learn, without human interference, except at annotating the dataset.

On this type of algorithms, they should learn on their own, we do not interfere with the learning process.

The separation we generally do calling them black-box models has to do with the fact that they learn, but we cannot in general explain how it works, except with metaphors how it works.

One well-known warning fact about those models: we cannot ensure they will always behave as we expect: chatGPT is full of examples where its behavior is erratic compared to what we would expect from this model.

2.2 Training the model

The models are trained using the default routines from TensorFlow.js, more details can be found on the provided gists on GitHub on each respective subsection from this section.

This is how we set up the training configurations.

```
model.compile({
  optimizer: tf.train.adam(),
  loss: 'binaryCrossentropy',
  metrics: ['accuracy'],
});
```

On a possible adaption of our reported findings, starting from this part could be a good strategy.

As optimizer for the error: Adam optimizer (or Adaptive Moment Estimation), which is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments, see this blog post. It is a local search, thus, the training may not always converge; alternatives are global optimizations. Except for the heart failure model, the two other models converged well, on all attempts; on the heart failure model, we had to make some attempts before the model would converge accordingly. This is a well-know issue when using local optimization: they are called traps.

For loss function, how we measure our error for guiding the optimizer, we use binaryCrossentropy, which is a binary cross-entropy metric function which uses binary tensors and returns tf.Tensor object. See this blog post. When changing

the model, you may need to consider changing this function. Generally, for the kind of problem we report, this loss function is enough.

For accessing the final results, we are using accuracy, which is a ratio between what the model gets right vs. all the attempts. The ideal is 100%, and we may say that 50% is like chance, randomness. See that 100% of accuracy does not mean the model does not make mistakes, just mean that using the given dataset, it did well; if the dataset is a poor representation from the case, it will fail when applied on real-scenarios.

2.3 Training and validation

It is a common practice in the machine learning community to validate the models splitting the dataset into learning and validation: we are using 20% for validation is 80% for training; these values are widely used.

This is how we set this configuration on TensorFlow.js: $validationSplit$: 0.2. For seeing more details, just access the gists provided. It is set on the training section, namely: $model.fit$.

Below is a sample how it looks like.

```
await model.fit(
features_tensor_raw, target_tensor,
{
  batchSize: 40,
  epochs: numberEpochs,
  validationSplit: 0.2,
```

The validation curve is used to avoid memorization, overfitting. A model that memorizes is useless since it will have to predict outside their training dataset.

2.4 TensorFlow.js

TensorFlow.js is a JavaScript-based library for deep learning, based on the classical TensorFlow, written in Python; you can also do simple learning machine, some simple mathematical operations with tensors and so on. We are going to build multilayer perceptrons (MPLs).

There are several reasons for using TensorFlow.js instead of Python, an imperative reason is using just one language, from the app to machine learning (i.e., JavaScript and derivatives). Another reason is that you do the calculations on the browser, no need to have high-performance servers. If your app starts to gain users, the calculation cost will not grow since each user is responsible for their calculation load. This last point is especially interesting for startups, since you can scale up without also increasing the cost. One reason for medical applications: your data never leaves the browser, it is ideal for sensitive data.

One advantage of using TensorFlow.js on the browser, instead of in the sever, is that it was well-configurated to use local NVIDIA GPUs. Those computation processors are well-known for being powerful on numerical calculations, and they helped on the revolution done by deep learning.

A nice point is that they claim it is possible to transform models in both directions: TensorFlow.js $<->$ TensorFlow [8, 9, 10]. Even the manual transformation is possible since their notations are similar.

TensorFlow.js provides several ways to be used: pretrained models, CDN calls, NPM, and even downloading the models. We are going to train our own model, and use it locally on a App in Angular.

2.5 Multilayer perceptron (MLP)

A Multilayer Perceptron (MLP) is an artificial neural network that has a well-defined structure, architecture.

It has an input layer (no neuron, just inputs), a hidden layer, and an output layer. Those models are widely used on regression. We are going to consider what is called logistic regression. "Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables." IBM . Thus, our model releases a number that can be seen as a sort of probability, even if it does not have the rigour of statistical analysis; it is not a true statistical measure. This number varies from 0 to 1, and generally above 0.5 is considered as 'yes', and below, 'no'; this is called threshold. This strategies is common on machine learning: every model based on classification will give this "probability" measure of a set of classes, or decision options.

We are going to consider binary classification: yes or no. This means that our model has just one output neuron; it has just two options to pick up from. On the models available on Kaggle, they use more than one hidden layers: our experience shows that just one hidden layers is generally enough, and you just need to adjust the number of neurons on the hidden layers. Thus, we always adopt the strategy of just using one hidden layer, and handpicking the hidden neurons configuration.

As one example how it looks like on TensorFlow.js for the diabetes case:

```
    //Input layer & hidden layer
    model.add(
      tf.layers.dense({
       //here we define the number of features
        inputShape: [numOfFeatures],
        //Number of hidden neuros
        units: 50,
        //Activation function
        activation: 'relu',
      })
    );

//Output Layer
    model.add(
      tf.layers.dense(
      //just one neuron
        { units: 1, activation: 'sigmoid' }
      ));
```

Keep in mind that the configuration (mainly number of neuron on the hidden layer may change to the other cases). In case of adaptions, use the number of hidden neurons to try out better models, with higher accuracy.

2.6 How does a machine learning model work?

A machine learning model works in two stages: first we train it, then we use it! The training is done on a dataset similar to the one it supposes to predict on.

There are several issues that may appear. The most important one is bad dataset, as we like to say ”garbage in garbage out”. One type of bad dataset is biased one. It means the dataset is not well-balanced, not well-representative; e.g., for the small dataset of diabetes, we had the care to select an equal number of samples from both classes, and randomly selected, and shuffled the dataset before training. See that biased dataset is not a machine learning problem, it is a statistics problem. If the dataset is well-sampled using statistical principles, it supposes to be good.

For eventual applications on real scenarios using our insights, please, make sure that the datasets used can represent your target population accordingly. As one example, we have no strong evidence, as far as we know, diabetes may change significantly according to population (e.g., genetic factors), but, one should test the final model on local data before using it.

2.7 External resources

2.7.1 Full repository

A full repository, coded in Angular using TypeScript, can be found here on GitHub. One can use those codes for replicating our findings. For specific models, we provide gists on GitHub, as so the parametrizations are also preserved, and possible specific configurations of the code. Gists are files on GitHub, which are generally used to present code snippets.

Our codes are an adaptation from [8], chapter 2; the reader can consult this book for a getting started reading tutorial. One can find here on GitHub the codes for these codes that served as starting point for our own. These details may serve someone wanting to adapt the findings to their own medical cases. One can find a sandbox here on StackBlitz, which can be forked and used for experimenting on this basic model we have used.

2.7.2 Datasets

”Garbage in, Garbage out”

All the datasets were dowloaded from Kaggle. We have done the following editing, as so the datasets would fit our purposes accordingly.

1. We have sampled randomly a small number of samples from the complete dataset. In the case of diabetes detection, the sample was done manually, creating a second spreadsheet from the original. For the other cases, we have used an internal routine in TypeScript, provided inside the code;
2. We have transformed non-numeric features into numerical ones. We have also done either manually or provided a routine on TypeScript;
3. We have selected the features we wanted for each model, accordingly;
4. We have uploaded it to Google Sheets, published it and used their public spreadsheet link for uploading the dataset into our models. When the routine was local for preprocessing, we have used the full spreadsheet from Kaggle. Those details are provided on each case, on the discussions;

See that you can find the detailed preprocessing on the codes, or the final spreadsheet links on the following sections. They are all as CSV (comma-separated values) format, with heading correspondent to the feature; rows are different samples, and columns are the respective values for the feature on the heading.

Disclaimer. Always keep in mind: those are public datasets, we cannot guarantee those datasets are well-curated (e.g., unbiased, well-sampled and more). Nonetheless, the datasets have a score called ’Usability’, and they are opened for comments. One can use those information to search for good datasets.

2.8 Diabetes detection

The complete dataset is here. The following datasets are derivations from this one, adapted accordingly to fit our purposes.

For getting the link when made available, either click on it, which could trigger an automatic download, or right-click on the link and ask to copy the link, and place it on the code; you must replace the variable ’csvUrl’, the code was designed to adapt accordingly. You may need to change the visualization method, deciding which feature you want to plot; as alternative, just comment this method call out; the method for visualization is called ’visualizeDataset’, just comment out the call on the ngOnInit() method, in case you want to train without visualization. In Angular, ngOnInit() is a hook, in this case, make sure something will happen just after the app is started, see [12] for more.

Importat. For the upcoming sections: keep in mind that features are the input to the model, it does not count the output (e.g., diabetes detection).

2.8.1 6-feature model

The code used to train the model can be found here as a gist, the TypesScript file. Since we are concentrating on the logic of the Angular app, we are not changing the remaining component’s files. One can find in case of interest a complete repository here on GitHub, with interface from an Udemy course we have lectured. For the case of diabetes, we have at the moment a working app on Heroku, deployed for presenting the concept.

The spreadsheet link is here.

2.8.2 1-feature model

The spreadsheet link is here.

2.8.3 3-feature model

The spreadsheet link is here. You can find here a gist, which includes the link.

2.9 Predicting In-Hospital Surgery Complication

The spreadsheet used is here. The original dataset is here on Kaggle. This dataset is sadly not very clear on details, we have used a dictionary provided by an user, it can be found here. We used this dictionary to understand what each feature meant, as so we could build our model according to what we wanted to study: complications on surgery based on physiological measures. A gist for the TypeScript file is here on GitHub.

2.10 Heart failure prediction model

A gist on GitHub is here. The full dataset is here on Kaggle. The coded provided, different from the one for diabetes, allows to actually chance the sample size from the full dataset. Just adjust *number_of_samples* in *dataset_to_array* method. It may be useful in case you want to run the model with more samples, or make simulations using another dataset.

3. Results and Discussion

3.1 Models in medicine: mechanical vs. clinical judgement

Mechanical judgments are when algorithms take decisions, whereas clinical judgment is when humans take decisions, concept introduced by Daniel Kahneman and colleagues [16]. We build models, as so they can support us on taking decisions, based on facts, on evidences. Even if you are not aware, humans also consider facts, data, evidences, when deciding. The difference is that humans cannot use massive datasets, integrative approaches and more. Another difference is: we cannot write down clearly how we decided, even though we may think we can. We are contaminated by biases and noise, as highlighted [16]. Aware of or not, we cannot handle multiple sources of information, in our models called features. We create simplified versions of the problem, as so we can handle them.

This concept is valuable for us: we are actually to consider situations where algorithms may actually compete with humans; of course, it is becoming more and more present. However, medicine is an area that has been resistant, even though the models are promising; models have been used on the industry for decades for supporting decision making, called operations research. There are several unsolved questions, and one is about accountability when algorithms make mistakes. Those discussions ought to happen at some point if we want to have more models on medicine in the future.

Daniel Kahneman and colleagues [16] brings to attention that even randomly initiated models may be better than humans, in certain scenarios.

> "In one of the three samples, 77% of the ten thousand randomly weighted linear models did better than the human experts. In the other two samples, 100% of the random models outperformed the humans." [16]

Of course, they are considering specific scenarios, and care is necessary before big generalizations. The important message is: models, even simple ones, can replace humans. The point is: when talking about complex models, big models, integrative models, humans do not have a chance. Nonetheless, even for simple models, humans can be replaced by algorithms; and we should not neglect it based on human's feelings of being left out.

One possible benefit of applying models to medicine is eliminating repetitive tasks, and possibly allowing the doctor to actually concern about the patient. Most of the routine tasks done by medical doctors can be automated, or are on the verge of, such as reading X-ray plates. We have organized a couple here [17]. Another benefit could actually reducing costs: one benefice of AI models is that intelligence starts to be cheap, once the model learns, it can be easily share as API, as an example. Different from human intelligence, machine intelligence can be shared easily, and costless.

Humans, as Daniel Kahneman and colleagues [16] highlight, believe they understand how they decide, and get overconfident. It is not unknown about medical errors in diagnosis, and some areas of medicine may have a strong variation in the same diagnosis, coming from different professionals.

Machine learning, when make mistakes, it is easily traceable, and enhanced. Even though we do not understand precisely how those models learn, we do understand the big picture. This is important to know when we are talking about models being used, and continuously enhanced based on their misdiagnosis. Once they learn, it does not happen again. Machine intelligence is easily repeatable, transferable and reliable once they are properly designed.

3.2 Mathematical models applied to medicine

We have essentially two big groups of models applied to medicine: white box and black models. The former focus on details, whereas the latter on the dataset.

Black box models are closer to how humans think, and decide. A goalkeeper can predict where the ball will most likely be, based on several cues/inputs, but he cannot explain it details. You do not need to know all the physics from the ball to defend your goal. Machine learning is a black box model, and that is why they are so interesting to medicine. When we predict diabetes from a couple of features, it does not mean we understand the dynamics of diabetes, work done by white box models [18].

Most of the models in medicine, the ones called white box models, are complex, but not complex enough for reliable applications. Machine learning models, even though may get it right, are not explainable, they are just numbers that change during the learning process. Transfer learning is a set of

numbers, which is used on another problem: knowledge from a set of images become a set of values on parameters called learning weights. They are just matrix multiplications and tensors manipulations. Different from humans when properly trained can explain their thinking in equations and theories, machine learning just do it right, but does not provide the rules explicitly.

3.3 Diabetes detection

Diabetes is an interesting case for our discussion, it has been widely explored from both perspectives, as machine learning based models ([19, 20, 21]), and white box models [18].

On this section, we shall consider three possible models for diabetes detection using the same dataset, and model. The first model has six features, the second just one, and the last one has three features. The features are chosen from the same set of features.

Always keep in mind that our goal is not comparing results or competing with third-party results presented, they are presented for enriching the discussions. Our goal here is showcasing TensorFlow.js, and supporting on spreading the word regarding this tools for creating advanced machine learning models. Years ago, those models would require either a strong expertise in programming and machine learning, or/and an expensive-paid software (e.g., Matlab).

3.3.1 model 1: predicting with six features

We should always start with the simplest model, nonetheless, we shall start with the almost-full feature model. We are going to use the following features (table 1), six out of seven available features.

The best results on the codes available alongside the dataset, using TensorFlow in python, they have used all the features: seven at total. See here. The nice feature of this dataset is that they have attached codes, created by the users, and publicly available, and editable. TensorFlow in python and TensorFlow.js have similar notations: we have explored those codes for starting our models when necessary.

We have removed the following feature: smoking history. We got essentially the same result, which means that smoking history, at least for this group, does effect diabetes significantly.

The best model on Kaggle arrived to 97% of accuracy, see here, we have arrived to 92% of accuracy (figure 1); sadly, we were unable to see the validation results for making sure they had no overfitting. This extra accuracy presented by the developer at Kaggle was done in addition of using an extra feature by applying Random Forest Regressor (i.e., a different technique), see here. Since the training process can give out different results depending on the starting point and this high accuracy was achieve with an alternative approach, we cannot say for sure smoking accounts for this 5% of lost accuracy. Those third-party results were mentioned for the sake of information, it is not our goal here to compare different approaches on machine learning for logistic regression, as generally data scientists call this problem. We have also not checked carefully the results on those notebooks on Kaggle, for making sure they did not take shortcuts. Here on this notebook they have also found 97% for a model similar to ours, a Multilayer Perceptron (MLP).

Table 1. Features used to train the model with six features for diabetes detection.

Feature	Short Description
HbA1c level	Higher levels indicate a greater risk of developing diabetes.
age	age ranges from 0-80 in our dataset. Diabetes is more commonly diagnosed in older adults.
bmi	BMI (Body Mass Index) is a measure of body fat based on weight and height. Higher BMI values are linked to a higher risk of diabetes.
blood glucose level	Blood glucose level refers to the amount of glucose in the bloodstream at a given time. High blood glucose levels are a key indicator of diabetes. HbA1c level is a long term measure, 2-3 months.
heart disease	Heart disease is another medical condition that is associated with an increased risk of developing diabetes
hypertension	Hypertension is a medical condition in which the blood pressure in the arteries is persistently elevated.

[a] We have kept the information to the essential. See the Diabetes prediction dataset on Kaggle for more information, and updated.

Below is the training curve (figure 2). One can see that both training (in blue) and validation (in orange) curves go downwards, and remain at low values. This is a sign that the model actually learned and was able to generalize. No overfitting happened, based on those graphs.

One point that we should bear in mind: we have not used all the dataset. The complete dataset of diabetes detection has 10.000 samples: we have used just 300 samples (equally balanced between diabetes and no-diabetes, and randomly samples from this pool). It shows how the signals on the dataset, which the model was able to learn, is strong; it also converged without any instability, which is a good sign. One can test a model with the full dataset. Since we are using the browser, uploading the dataset locally for training, it may be slow this uploading process of actually using the complete dataset; we see no reason based on those graphs to keep

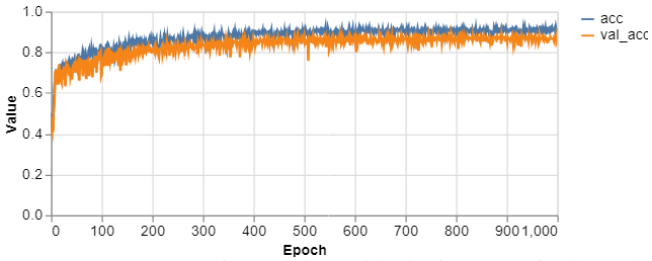

Figure 1. Accuracy for training with six features. final result, about: 92% for training and 85% for validation

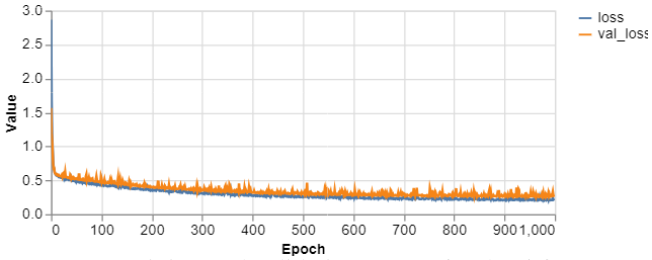

Figure 2. Training and validation curves for the 6-feature diabetes detection model

moving around 10.000 samples of data.

Another point is which features to use, since we have decided not to use them all, it may raise this question. In real scenarios, it may be the case that the person doing the model does not have all the features; or even when applying the model, the user does not have all the features. We are going to consider two extra scenarios in the upcoming sections: choosing the most important one (namely, HbA1c levels), and the three easiest ones to have access to (namely, BMI, gender and age).

From the user's perspective, one can build different models with different features, and this change is made according to how much information the user has. Of course, the prediction accuracy will change accordingly. As we are going to see, the accuracy will drop to 70% in both cases where we have not used all the 6 features.

One good question is why just one, or even three features, can achieve 70%, leaving the remaining with 20%. You can find an explanation on figure 6 (correlation matrix): the features are correlated, which means they do not convey pure information.

From a developer's perspective, one can build the model based on the features available, or measure them accordingly to their power of prediction. We are going to consider a model with just one feature, the one that predicts the most. We are thinking, as a hypothetical scenario, one where one must decides which features to invest on to measure on several patients.

One interesting discussion is how to use this model when we do not have a big dataset; we showed that the model converges even for 300 samples, which may be big if we are using a small town as target for our model, or creating a prototype. A quick search on the internet points out that people are considering transfer learning in regression: it is generally applied to images [15]. We are considering, given we have access to those data in the future, using this model in Brazil; our current setback is actually having access to this data as we have on Kaggle, publicly and easily accessible. Our concern is that this dataset may be biased; biased dataset is a well-known issue on machine learning, they contaminate the learning process, and the machine learning will make mistakes when confronted with categories misrepresented on the dataset used to train. We have a guess that population variations (e.g., genetics, diets) may effect how those features predict diabetes. Our guess is: we cannot trust on this dataset 100% since it is concentrated on USA. We may need, or anyone considering actually using this model locally, make some adjustments to make sure the model is not biased towards the USA population, and their biological, cultural peculiarities.

3.3.2 model 2: predicting with just one feature

We consider just one feature, namely, HbA1c levels, we create three regions when this choice is applied to our dataset: two has diabetes diagnosis well-defined, whereas the third one is a grey area. This regions are based on numbers we can find online regarding how to interpret this coefficient; we also had a big help from chatGPT. As we can see from figure 4, the accuray falls to 72%, the removed features account for about 20% of the accuracy.

The good question is how we decided to use just this feature, which has 70% alone of accuracy: just this feature is better than random guess. Bringing back our previous discussion based on Daniel Kahneman and colleagues: would this model be better than an expert? to which extent? good question! As they have noticed, in some scenarios, a random model was better than human.

One way to look at it is by rationale. Glucose is the number one factor that appears on white-box models, it is not by change; those models aim at explaining datasets by actually understanding the inner dynamics from the biological system, different from machine learning, which is just focused on somehow learning and predicting. However, the fact that your glucose is high for a couple of days does not make you are diabetic (short-term factor); but still something to be attention to. HbA1c is measured in months (long-term factor), thus, it is a measure of high-blood glucose level persistence. It may shed light on why this factor is so powerful on predicting. As we are going to see in the next model, glucose alone account for about 20% of the signal for predicting diabetes, using correlation as indicator.

Another way to look at it is by correlation. Correlation measures the connection between two variables: it can be positive (from 0 to 1) or negative (from -1 to 0). Even though correlation is a linear-relation indicator, if used with attention, it can help us even on nonlinear cases; which is the case for most problems in medicine. However, one should always pay attention to the fact that it measures linear relationship, and can misdirect on nonlinear cases.

As we can see from the diabetes column (figure 6), the two

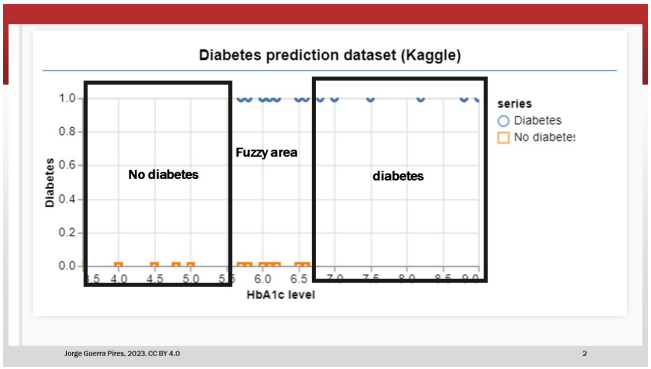

Figure 3. Predicting with just one feature. We have three areas: two are well-limited by HbA1c levels, whereas we have a grey area. This area was created considering what is normally accepted on the medical literature, just an approximation.

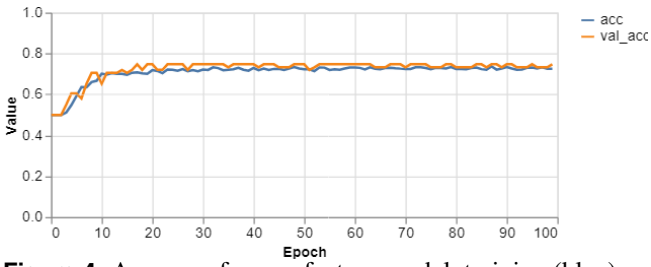

Figure 4. Accuracy for one-feature model, training (blue) and validation (orange). Final: training accuracy, 72%; validation 74%

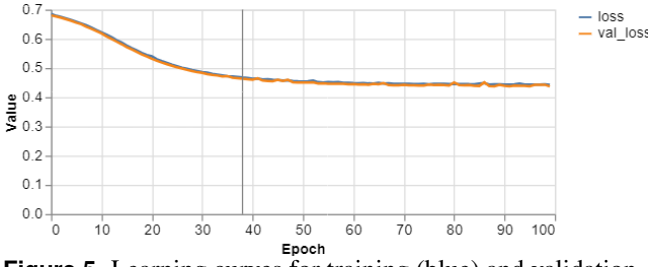

Figure 5. Learning curves for training (blue) and validation (orange) for the one-feature model

highly correlated with diabetes detection are glucose-related. Thus, the best way to diagnosis diabetes is by glucose-related indicators. Diabetes is well-studied, therefore it is no surprise, nonetheless, for less studied cases, when you apply those models, you may want to get to know the inner dynamics before pouring numbers on the model, that is the take home message.

3.3.3 model 3: predicting with age, gender and body mass index

In figure 7, we have a look at how body mass index and age may influence diabetes; keep in mind we have used it on our 6-feature model, we are just narrowing it down to easy-to-use features, namely, body mass index (BMI), gender and age. Those are three features that anyone has access to. As you can see, in blue, diabetes is present at high BMI, and very rare at low: remember that diabetes presents itself mainly as high-blood sugar, known as ”sweet urine”; the body tries to get rid of the extra glucose in the urine, and this can cause a sweet smell. Obesity (high BMI) can cause difficulties on cells’ glucose receptors, which cases what is well-known as diabetes type II, which is generally reversible based on change of habits.

You can see also from figure 6 that both age and BMI are highly correlated to diabetes (about 20% each). An intuitive way to look at correlation is ”the correlation between two variables is their percentage of shared determinants.” [16]

As we can see from figure 8, we have the same accuracy for the 1-feature model. It can be explained on a speculation level by observing that HbA1c levels are correlated with both features we have used, 10% with age. Even though it is not a formal mathematical concept, we can think that the correlation summed up on the 2-feature model: each feature contributes with 20% of correlation.

3.4 Predicting Surgical In-Hospital Complication

We have 25 features, we are going to be even more selective on what to consider on our model. We have a wide guess on what could lead to complications on surgery (table 2).

Surprise at it may seem, the model arrived to 100% of accuracy (figure 10), and almost 0 of loss function (measure of the model’s mistakes, figure 11).

The first caution we would have when seeing those results is overfitting (i.e., the model memorized the training dataset, but did not actually generalized). The validation curve also converged, which is an imperative measure to avoid overfitting, and aiming at generalization.

Disclaimer for using this model on real scenarios, please, make sure the dataset is not biased against your aimed population. We cannot guarantee the dataset used is well-curated, and unbiased.

3.5 Heart Failure Prediction

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for

	age	hypertension	heart_disease	bmi	HbA1c_level	blood_glucose_level	diabetes
age	1.000000	0.251171	0.233354	0.337396	0.101354	0.110672	0.258008
hypertension	0.251171	1.000000	0.121262	0.147666	0.080939	0.084429	0.197823
heart_disease	0.233354	0.121262	1.000000	0.061198	0.067589	0.070066	0.171727
bmi	0.337396	0.147666	0.061198	1.000000	0.082997	0.091261	0.214357
HbA1c_level	0.101354	0.080939	0.067589	0.082997	1.000000	0.166733	0.400660
blood_glucose_level	0.110672	0.084429	0.070066	0.091261	0.166733	1.000000	0.419558
diabetes	0.258008	0.197823	0.171727	0.214357	0.400660	0.419558	1.000000

Figure 6. Correlation between the diabetes detection features. Source: Notebook on Kaggle

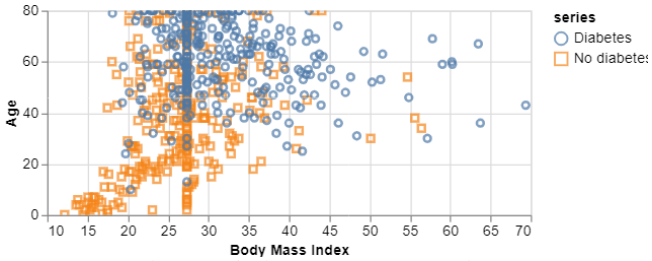

Figure 7. Body mass index vs. age. Normal BMI ranges from 18.5 to 24.9

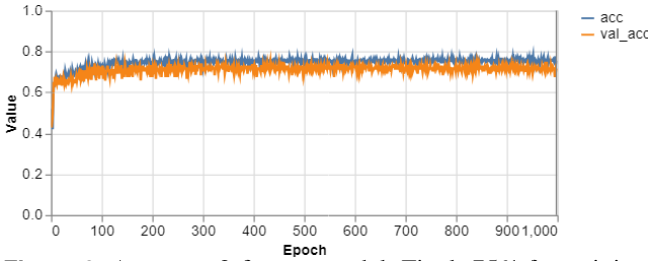

Figure 8. Accuracy 2-feature model. Final: 75% for training (blue curve); 70% for validation (orange curve)

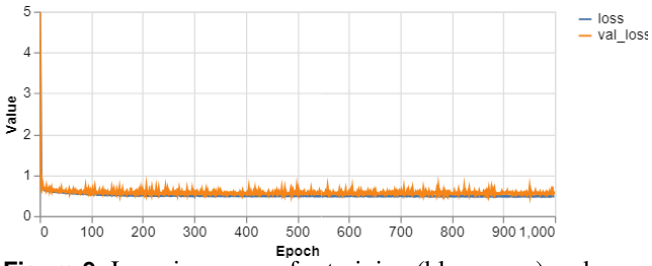

Figure 9. Learning curves for training (blue curve) and validation (orange curve).

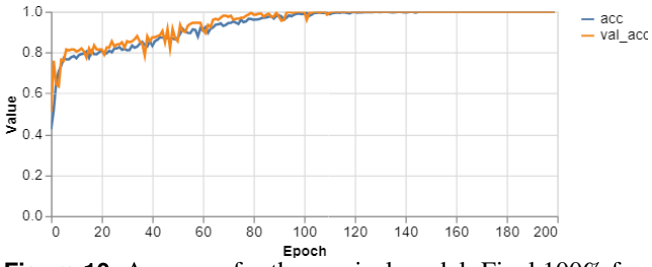

Figure 10. Accuracy for the surgical model. Final 100% for both curves; training in blue and validation in orange.

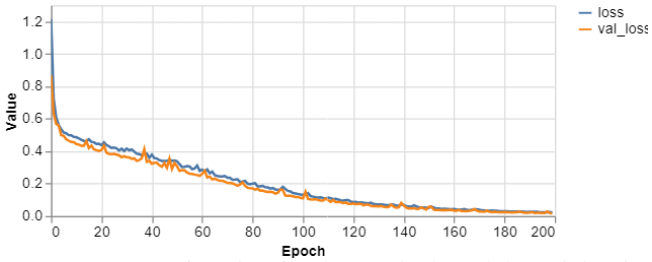

Figure 11. Loss function for the surgical model. Training in blue and validation in orange

Table 2. Features used to train the model for surgical complications.

Feature	Short Description
baseline diabetes	patient has diabetes.
BMI	body mass index. This a measure of body weight related medical conditions, such as obesity and undernutrition
Age	patient age. We generally know that age may influence the final result
Gender	patient gender

[a] We have used an external dictionary to make sense of the features, see here.

31% of all deaths worldwide. Four out of 5 CVD deaths are due to heart attacks and strokes, and one-third of these deaths occur prematurely in people under 70 years of age. Heart failure is a common event caused by CVDs and this dataset contains 11 features that can be used to predict a possible heart disease. People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or already established disease) need early detection and management wherein a machine learning model can be of great help. dataset description

We have selected a couple of features (3), and we got an accuracy of 70% (figure 12). It seems we can do better, one user from Kaggle claims an accuracy of 90%, see here the notebook. Thus, adding all the features we may arrive to this result.

Table 3. Features used to train the heart failure prediction model.

Feature	Short Description
Exercise Angina	It is a common symptom of coronary artery disease (CAD).
Sex	patient's gender
Age	patient's age
Cholesterol	levels of cholesterol of the patient

[a] We have kept the information to the essential. See the Heart Failure Prediction Dataset on Kaggle for more information, and updated.

Alert. it seems there is wrong title for this dataset according to one user. It seems the dataset predicts coronary heart disease. It does not change our results. We assume that when you use those models on a dataset in a real-case, you know the dataset enough.

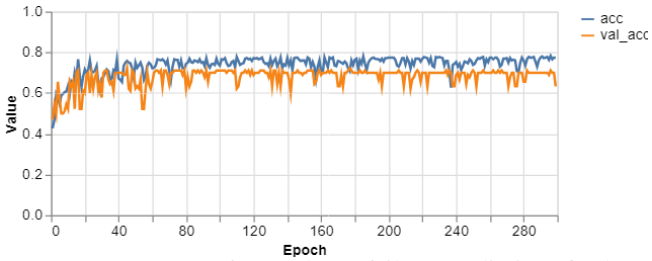

Figure 12. accuracy for our heart failure prediction. final. training 77% and validation 63 %

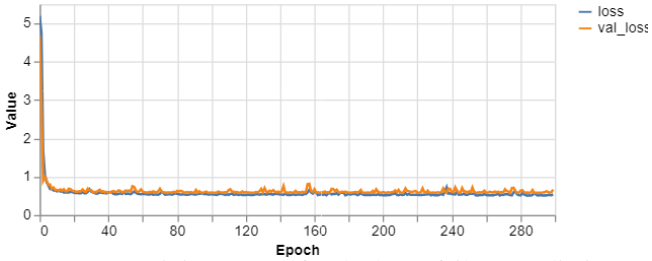

Figure 13. Training curves for the heart failure prediction model

3.6 In summary

We have built a couple of examples using TensorFlow.js as machine learning platform. Even though python and R are dominant on the moment, JavaScript and derivatives are growing fast, offering basically the same performance, and some extra features associated with JavaScript; JavaScript now even offers desktop options for coding, not to mention a full stack language (MEAN Stack): one can build back and frontend without without needing to shift languages. Kaggle, the public platform from where we downloaded our datasets, offers a huge amount of datasets for biomedical cases, thus, the reader can easily test what we have discussed, using the same codes, with minor chances, on any case they may be interested in. After using TensorFlow.js for about three years, and after have used previous options such as Matlab, we are optimistic that this library developed in JavaScript has what it is needed to be a machine learning option for biomedical problems. They offer unique advantages, such as the data never leaves the browser; and the calculation load stay locally on the browser, no need to pay for high-performance servers. Angular, coded in TypeScript, a derivative from JavaScript, is an active community, releasing updates every six month. The possibilities are unlimited, and we believe that it is a nice option for researchers aiming at web applications, especially, focused on medicine.

References

[1] RUNDO, L.; TANGHERLONI, A. *Biomedical Image Segmentation and Analysis using ML and*

CI Techniques. 2017. Disponível em: ⟨https://www.youtube.com/watch?v=1oTWxxe9YJM&t=89s⟩.

[2] PIRES, J. G. *Could chatGPT play a medical doctor?* 2023. Published in Computational Thinking: How computers think, decide and learn. Disponível em: ⟨https://medium.com/computational-thinking-how-computers-think-decide/could-chatgpt-play-a-medical-doctor-8dfcd8c95538⟩.

[3] PIRES, J. G. Innovating with biomathematics: the challenge of building user-friendly interfaces for computational biology. *Academia Letters*, 2022. Disponível em: ⟨https://doi.org/10.20935/AL5792⟩.

[4] PIRES, J. G. *An informal survey presents the gap between computer and medical doctors and biologists*. 2023. Published in Theoretical and Mathematical Biology (blog in Medium publication). Disponível em: ⟨https://bit.ly/3XfhCKH⟩.

[5] PIRES, J. G. *Biomechanics, Computational Intelligence, and Systems Biology with application on Vitreous Dynamics Using Java: an incipient discussion*. 2014. Preprint in Academia Edu. Disponível em: ⟨https://www.academia.edu/8341040/Title_Biomechanics_Computational_Intelligence_and_Systems_Biology_with_application_on_Vitreous_Dynamics_Using_Java_an_incipient_discussion⟩.

[6] PIRES, J. G. et al. Galaxy and mean stack to create a user-friendly workflow for the rational optimization of cancer chemotherapy. 2021. Disponível em: ⟨doi:10.3389/fgene.2021.624259⟩.

[7] BLANK, S.; DORF, B. *The Startup Owner's Manual: The Step-By-Step Guide for Building a Great Company*. Illustrated edition. [S.l.]: Wiley, 2020. 608 p.

[8] RIVERA, J. D. D. S. *Practical TensorFlow.js: Deep Learning in Web App Development*. https://www.amazon.com/Practical-TensorFlow-js-Deep-Learning-Development/dp/1484262727: Apress, 2020. 328 p.

[9] CAI, S. B. S.; NIELSEN, E. D.; CHOLLET, F. *Deep Learning with JavaScript: Neural networks in TensorFlow.js*. https://www.amazon.com/Deep-Learning-JavaScript-networks-TensorFlow-js/dp/1617296171: Manning, 2020. 560 p.

[10] LABORDE, G. *Learning Tensorflow.Js: Powerful Machine Learning in JavaScript*. https://www.amazon.com.br/Learning-Tensorflow-Js-Powerful-Machine-JavaScript/dp/1492090794: O'Reilly Media, 2021. 338 p.

[11] PIRES, J. G. Alguns insights em startups um novo paradigma para a tríplice aliança ciência, tecnologia e inovação: a novel paradigm for understanding the triple alliance of science, technology and innovation. *Revista Gestão amp; Saúde*, v. 11, n. 1, p. 38–54, fev. 2020. Disponível em: ⟨https://periodicos.unb.br/index.php/rgs/article/view/28626⟩.

[12] FAIN, Y.; MOISEEV, A. *Angular Development with TypeScript*. https://www.amazon.com/Angular-Development-Typescript-Yakov-Fain/dp/1617295345: Manning, 2018. 560 p.

[13] GANS, J.; GOLDFARB, A. *Prediction Machines: The Simple Economics of Artificial Intelligence*. [S.l.]: Harvard Business Review Press, 2018.

[14] WOLFRAM, S. *What Is ChatGPT Doing ... and Why Does It Work?* https://www.amazon.com/What-ChatGPT-Doing-Does-Work/dp/1579550819: Wolfram Research, 2023. 112 p.

[15] PIRES, J. G. Snakeface: a transfer learning based app for snake classification. *bioRxiv*, Cold Spring Harbor Laboratory, 2023. Disponível em: ⟨https://www.biorxiv.org/content/early/2023/06/14/2023.06.13.544741⟩.

[16] SUNSTEIN, C. R.; KAHNEMAN, D.; SIBONY, O. *Noise: A Flaw in Human Judgment*. [S.l.]: Little, Brown Spark, 2021).

[17] PIRES, J. G. *Relatório Final de pós-doutorado Programa Nacional de Pós-doutorado PNPD/CAPES*. [S.l.], 2018. Disponível em: ⟨https://www.researchgate.net/publication/329815289_Relatorio_Final_de_pos-doutorado_Programa_Nacional_de_Pos-doutorado_PNPDCAPES⟩.

[18] PALUMBO, P. et al. Mathematical modeling of the glucose–insulin system: A review. *Mathematical biosciences*, v. 244, 05 2013.

[19] CHEN, W. et al. Nonlinear modeling using support vector machine for heart rate response to exercise. In: ____. *Computational Intelligence and Its Applications*. [s.n.]. p. 255–270. Disponível em: ⟨https://www.worldscientific.com/doi/abs/10.1142/9781848166929_0010⟩.

[20] LING, S.; SAN, P.; NGUYEN, H. Hypoglycemia detection for insulin-dependent diabetes mellitus: Evolved fuzzy inference system approach. In: ____. *Computational Intelligence and Its Applications*. [s.n.]. p. 61–85. Disponível em: ⟨https://www.worldscientific.com/doi/abs/10.1142/9781848166929_0004⟩.

[21] ALTY, S.; LAM, H.; PRADA, J. On the applications of heart disease risk classification and hand-written character recognition using support vector machines. In: ____. *Computational Intelligence and Its Applications*. [s.n.]. p. 213–253. Disponível em: ⟨https://www.worldscientific.com/doi/abs/10.1142/9781848166929_0009⟩.