Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

ELAPSED TIME PREDICTION WITH STRAVA ACTIVITIES

Borja Hidalgo Toca

Director: Oriol Pujol Vila
Realitzat a: Departament de
Matemàtiques i Informàtica

Barcelona, 27 de Juny de 2019

Index

Abstract

Machine learning is a discipline that allows the automation and extraction of patterns in prediction tasks, among others.
In this project we study the problem of predicting the elapsed time for sport activities taking into consideration the data gathered from the user's profile and their activities and elapsed times. More concretely, in this project we address the following topics: a protocol for gathering the data of the different activities from Strava users is proposed, data cleaning and curation is considered, and finally, the usage of different supervised learning techniques for predicting the elapsed time duration of an activity are compared and appropriate metrics are established.
The project approaches the study of some machine learning methods, such as Elastic Net, Huber Regressor, Regression Trees, and lastly, additional importance is given to the study of deep neural networks (deep learning). Additionally, some metrics are also set about the success of the differents results obtained based on the accepted threshold of the regression values obtained, all of this applied in a case use of the predictors used within a business model.
The results obtained show that neural networks allow us to obtain, for a sparse range of activities within a dataset, successful results in a 60% of the predictions made. But the best performance we have managed to get in this first iteration of the investigation,is yielded by the ElasticNet regressor as it has the lowest percentage of error on average.
The results obtained in this project also leave a door open for potentially commercialize the investigation and being able to apply it on real case scenarios.

Abstract (castellà)

El aprendizaje automático es una disciplina que permite la automatización y extracción de patrones en trabajos de predicción, entre otros.

En este trabajo se estudia el problema de predicción del tiempo de duración de actividades físicas a partir de datos de perfiles de usuarios y de sus recorridos y tiempos.En particular se propone un protocolo de recogida de datos de las actividades de los usuarios de Strava, la limpieza de datos y el uso de diferentes técnicas de aprendizaje supervisado por la predicción del tiempo de duración del trabajo.

Se estudian diversos métodos de aprendizaje automático, tales como Elastic Net, Huber Regressor, Árboles de Regresión, y finalmente, se da un especial énfasis en redes neuronales profundas (deep learning). Adicionalmente, se proponen métricas de logros y éxitos de los resultados basados en la tolerancia aceptada de los valores de regresión obtenidos, en caso de uso de estos predictores en aplicaciones comerciales.

Los resultados obtenidos muestran que las redes neuronales nos permiten obtener por una variedad alta de actividades, resultados válidos en un 60%. Pero aún y así el mejor resultado obtenido en esta primera iteración de la investigación, se obtiene utilizando el ElasticNet regressor, ya que es el que obtiene un menor porcentaje de error medio.
Los resultados obtenidos en este trabajo abren la puerta a su potencial de comercialización y uso en aplicaciones reales.

Abstract (català)

L'aprenentatge automàtic és una disciplina que permet l'automatització i extracció de patrons en tasques de predicció, entre d'altres. En aquest treball s'estudia el problema de predicció del temps de duració d'activitats físiques a partir de dades dels perfils d'usuaris i dels seus recorreguts i temps. En particular es proposa un protocol de recollida de dades de les activitats dels usuaris d'Strava, la neteja de les dades, i l'ús de diferents tècniques d'aprenentatge supervisat per la predicció del temps de duració de la tasca. S'estudien diversos mètodes d'aprenentatge automàtic, tals com Elastic Net, Huber Regressor, Arbres de Regressió, i finalment, es dóna un especial èmfasi en xarxes neuronals profundes (deep learning). Addicionalment, es proposen métriques d'assoliment i èxit dels resultats basades en la tolerancia aceptada dels valors de regressió obtinguts en cas d'ús d'aquests predictors en aplicacions comercials. Els resultats obtinguts mostren que les xarxes neuronals ens permeten obtenir per una varietat alta d'activitats resultats vàlids en un 60% de predicció. Encara i això, els millors resultats obtinguts en aquesta primera iteració de la investigació,són obtinguts pel regressor ElasticNet, ja que es tracta del model que obté un valor mitjà de percentatge d'error més baix.
Els resultats obtinguts en aquest treball obren la porta a la seva potencial comercialització i ús en aplicacions reals.

1. Motivation and introduction

Throughout all my life both computers and sports have been very important to me. Especially when I started my career as a computer engineer, one of them was going to become my lifetime job and the other my hobby for probably most of my free time. Sports also helped me clear my mind during those hard-working days at the university or long study days before exams. As I went through courses at the university, sport became more and more important, even more so when I started working for the first time as an intern.

During all the time I spent practising sports, technology has always been present in my life, e.g. as I listened to music while practising it or recording my sport activities either using my phone or other devices such as smart watches. All the data I was unconsciously collecting was useless at first as I simply threw it in the void and watch it stack while looking at how I was progressing. But, as my time at the university started to come to an end and my knowledge as a computer engineer grew, I realised that I had a lot of data that was just there hanging in the void and waiting to become useful. That was the moment when the idea of this project came to my mind and the eager to research about it started. I also realized that I had a bunch of apps related to my hobby but none of them did fulfill the goal of this project as the purpose of all of them was totally different.

Nowadays we can find at the market two different kind of tools or applications ,whether they are apps or web pages, related to sport, which can be used to register activities or analyze them or even a few that have a mix of both. But most of them are used to register activities. They simply record the movement and a bunch of data about it, such as distance, time or elevation. There is a reduced group of these applications that are used to carefully analyze the recorded activities. However, there's probably only a very small group of people interested in it, mostly professionals who are more interested in analyzing this data for high performance as it can lead to a win or an important performance improvement in certain situations. But these applications just show basic data analysis, not user focused nor user dependant, this means that all kinds of users get the same output for their data even if their train levels are more or less different, as the output usually comes in charts showing user's progress during a specific workout or throughout all the time he has been recording his activities.

The goal of this research is to make all of this data useful to any kind of user, from the very experienced with the use of analytical tools, to the user who just started practising sport and doesn't even know about them. Also, this tool will be different from what a user can expect to

find in the actual market because as we explained a bit before, nowadays there are only two kinds of applications with very strong focused objectives, and the one we are going to focus on is yet to be discovered. The research will be focused in the prediction of the duration of a user's activity given a certain route, being this time calculated specifically with user's data to match his train level and also being able to adapt these time calculations to the route we are going to predict on, as parameters like elevation can make a difference within two routes having the same distance. All of this applied to every user will help his training experience improve in an easier way.

1.1 The big picture and project scope

Predicting elapsed time with data collected from an API is a really broad definition for the scope of the project, so let's try to shed some light on that definition. When it comes down to sport, the time a user needs to complete a certain route depends on a huge amount of factors, both external and internal to the user. What can be a simple thing like practising sport on a day you are feeling a bit sick or if the day is sunny or cloudy can make a big difference on the results the user gets when training on a regular basis. These external factors like the weather can be predicted and thus taken into account for the purpose of predicting elapsed time, but in our case we will not use them as we are going to stick to the data received from the Strava API. As a brief description, Strava gathers data about the user's profile and the terrain features.

The key point of Strava is the user's profile as it contains all the information available about every user on the application. That information is not only about the user but also about every activity recorded alongside its own specific features.

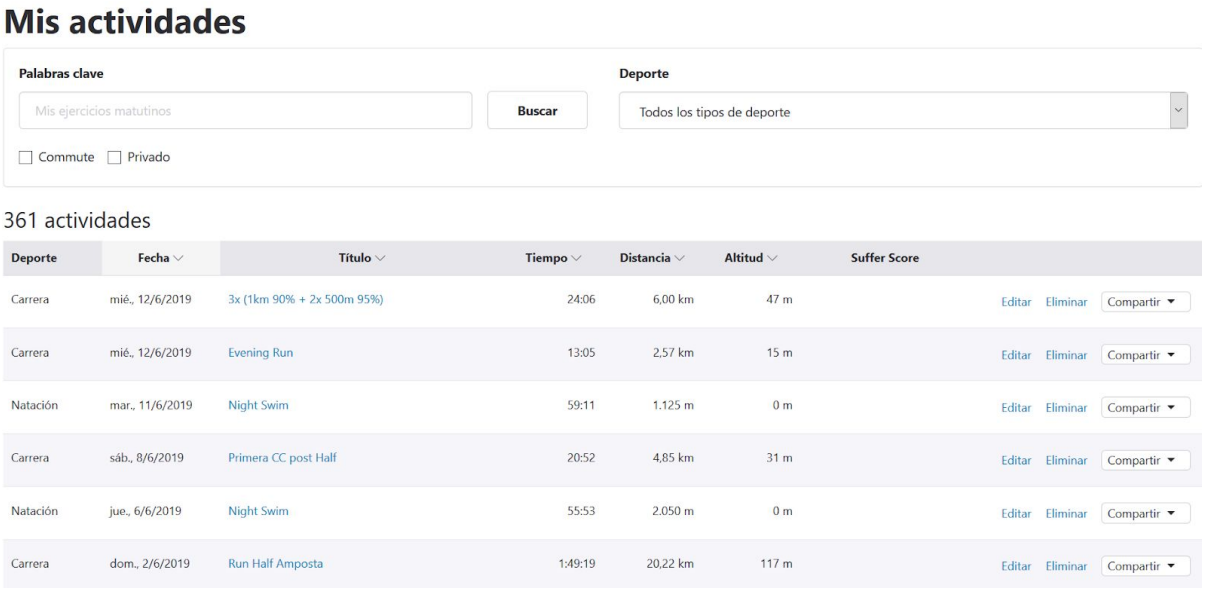

Figure 1. Screenshot of the summary of activities in a user profile

In Figure 1, we can see an example of how the activities of a user would look like. At the profile there's a brief description of the activity which contains the most meaningful information that defines each of them. Here we can see displayed features like which kind of sport was the activity about, the date of creation, the title of the activity, elapsed time, distance and altitude. However, activities are defined by more features than the ones displayed on the user's profile.

When it comes down to features, Strava already collects a lot of them without even the user noticing it and it does so in a fine-grained level. Let's now dig a bit deeper on which are these features for every activity that we are going to retrieve with every call to the API and do a brief explanation for each of them.

Feature	Type	Description
id	Long	The unique identifier of the activity automatically generated by the application.
external_id	String	The identifier provided at upload time
upload_id	Long	The identifier of the upload that resulted in this activity
athlete	Athlete Instance	An instance of MetaAthlete.
name	String	The name of the activity. Either default or provided by the user.

distance	Float	The activity's distance, in meters.
moving_time	Integer	The activity's moving time, expressed in seconds.
elapsed_time	Integer	The activity's elapsed time, expressed in seconds.
total_elevation_gain	Float	The activity's total elevation gain expressed in meters.
elev_high	Float	The activity's highest elevation, expressed in meters
elev_low	Float	The activity's lowest elevation, expressed in meters
type	ActivityType Instance	An instance of ActivityType.
start_date	DateTime	The time at which the activity was started.
start_date_local	DateTime	The time at which the activity was started in the local timezone.
timezone	String	The timezone in which the activity has been recorded.
start_latlng	LatLng Instance	An instance of LatLng. Represents the Latitude and Longitude where the activity started.
end_latlng	LatLng Instance	An instance of LatLng. Represents the Latitude and Longitude where the activity ended.
achievement_count	Integer	The number of achievements gained during this activity by the user.
kudos_count	Integer	The number of kudos given for this activity.
comment_count	Integer	The number of comments for this activity.
athlete_count	Integer	The number of athletes for taking part in a group activity
photo_count	Integer	The number of Instagram photos for this activity
total_photo_count	Integer	The number of Instagram and Strava photos for this activity
map	PolylineMap Instance	An instance of PolylineMap.Represents a map of the activity in a Google Maps style.
trainer	Boolean	Whether this activity was recorded on a training machine or not.
commute	Boolean	Whether this activity is a commute.
manual	Boolean	Whether this activity was created manually.
private	Boolean	Whether this activity is private or public.

flagged	Boolean	Whether this activity is flagged.
workout_type	Integer	The activity's workout type.
average_speed	Float	The activity's average speed, expressed in meters per second.
max_speed	Float	The activity's max speed, expressed in meters per second.
has_kudoed	Boolean	Whether the logged-in athlete has kudoed this activity.
gear_id	String	The id of the gear for the activity.
kilojoules	Float	The total work done in kilojoules during this activity. It applies to only Ride activities.
average_watts	Float	Average power output in watts during this activity. It applies to only Ride activities.
device_watts	Boolean	Whether the watts are from a power meter, value is false if it is estimated by the application.
max_watts	Integer	Maximum watts value generated in the activity, applies only to Ride activities with power meter data.
weighted_average_watts	Integer	Similar to Normalized Power. applies only to Ride activities with power meter data.
description	String	The description of the activity given by the user.
photos	PhotosSummary Instance	An instance of PhotosSummary.
gear	SummaryGear instance	An instance of SummaryGear.
calories	Float	The number of kilocalories consumed during this activity.
segment_efforts	DetailedSegmentEffort Instance	A collection of DetailedSegmentEffort objects.
device_name	String	The name of the device used to record the activity.
embed_token	String	The token used to embed a Strava activity.
splits_metric	Split Instance	The splits of this activity in metric units. It applies only to Run activities.
splits_standard	Split Instance	The splits of this activity in imperial units. It applies only to Run activities.
laps	Lap Instance	A collection of Lap objects.
best_efforts	DetailedSegmentEffort Instance	A collection of DetailedSegmentEffort objects.

Figure 2. Activity Model in-depth description

With all the amount of data available described by that amount of features we can gather a dataset with a really high quality. That quality and level of description favours the usage of machine learning to solve the project main goal as it covers one of the principal issues when it comes to data prediction, data quality. That quality alongside the variety and quantity that we are going to be able to gather, favour the creation of a good dataset that will ease the whole process and speed it up. Also there is the aspect of power,flexibility and adaptability that machine learning models offer compared to other techniques such as linear regressions. These models have the ability to fit better on data that have large amounts of features and that is sparse. All of these features and advantages that machine learning models offer compared to other techniques, lead us to the final decision of using prediction models based on the usage of machine learning and deep learning for the problem of predicting elapsed time for Strava activities.

1.2 Project goals

The main goal of the project and which will be the focus of researching is being able to predict elapsed time for sport activities using data from a free API such as Strava. The goal is not to just predict values without caring about the accuracy or keeping the first results obtained but to do so on an enterprise level like approach. What is pretended to achieve with this kind of approach is to check if the project could be further developed to a real business project or if its usage is limited to investigation only due to the results obtained. That consideration will basically depend on the predictions obtained with the model built in conjunction with the thresholds set up to decide whether the investigation has business potential or not.

Related to the project and not as a goal per se, there is also the challenge of learning about machine learning and deep learning starting from almost no knowledge about anything related to it. That represents a challenge as the process of learning about machine learning and deep learning goes hand in hand with the research and it is totally necessary to achieve the main goal.

1.3 Project layout

In section 2 background on machine learning methods is provided. The topic starts with a brief explanation of what machine learning consists of and then the problem of supervised learning is addressed. These definitions give the background necessary to then explain the different methods studied and used in this work, which are all defined by three components, namely, the prediction model, the loss function and the algorithm used to optimize the loss function. These methods are in fact the Huber Regressor, Elastic Net, Regression Trees and MLP Regressor.

Continuing to section 3, the proposed pipeline for the project is defined, explained and analyzed. Starting from an overview of the pipeline to then break down the different steps that take part of it. For each of them there is an analysis of the actions taken and the justifications behind each of them. The process starts with the whole set of actions needed to gather the data and form the dataset needed for the investigation, continuing with the data curation and preprocessing to end up with the model creation and evaluation breakdown.

For all the data curation and preprocessing it is explained with high detail all the normalization and different transformations that data needed to go through in order to achieve the desired data quality.

Following up, the machine learning model section goes through the process of the creation of the models for each of the prediction techniques used within the project. For each of them there is a breakdown of the different settings needed in order to achieve the best performance in our use case scenario. Alongside we can also find the definition of the evaluation and performance metrics that are needed to check successfulness for each prediction technique.

Lastly, all of the models are in-depth analyzed with the performance metrics set up on the previous pipeline step. Alongside the performance metrics, statistics are also calculated for each of the explored prediction techniques, which will help deciding on which one to pick.

After the whole pipeline explanation there is a follow up summary addressing the final decision about model performances and final analysis and comparison between each of them based on their performances metrics.

That topic leads us to the final discussion and project conclusion. Discussion topic focuses on final decision about which prediction techniques choose as the way to go, based mostly on their average percentage error obtained when used with the training dataset.

Finally the project discussion focuses on personal thoughts and reflections about the investigation and the lessons learned throughout the process. It also leaves a door open for future additional features to be added and further investigation.

2. Background

Machine learning is the study of computing algorithms and statistical models used to infer results, rather than relying on a specific code that solves the problem, looking for patterns. More concretely and according to Wikipedia: *“Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence.”*

Machine learning is usually divided in supervised and unsupervised learning techniques (other branches such as reinforcement learning, and sub-branches such as transductive learning, multitask learning exist but are out of the scope of this document. In this project we focus on supervised learning. Supervised learning is defined as the process of inferring an output based on the labels from training data contained in a training dataset. In supervised learning each value from the training dataset consists in an input-output pair which contains both the value and the expected result from the prediction model. A machine learning algorithm based on supervised learning, maps these input-output pairs to create then a function that infers the output values for new data that it is not contained in the training dataset. For the algorithm prediction to correctly work, the machine learning algorithm needs to generalize from the values contained in the training dataset to then map the new incoming values to a reasonable output. The key aspect of supervised learning is the existence of labels, which are defined by the user and are one of the main aspects to take into consideration when starting to build the training dataset. It is such an important aspect that a change in the labels provided by the user can completely change the results we are going to get and even can make a difference when it comes to model accuracy.

In this research, we will discuss different methods and algorithms based on Machine Learning and Deep Learning in order to solve our problem, which is predicting elapsed time for sport activities. We are going to use four different methods and compare them to search for the one that fits better our expectations and then keep the one that gets closer to the threshold we are going to define. For each of these four models we are going to do a brief introduction to the model, stating its strong points which will be the reason why we have chosen them, alongside a description of the mathematical model they are based on, they

loss function they use for optimization and also the learning algorithm used to minimize the loss function.

As a preface for introducing all the different methods studied and used in this work, it is worth mentioning that almost all methods in supervised machine learning are uniquely defined by three components, namely, the prediction model, the loss function and the algorithm used to optimize the loss function.

The prediction model corresponds to the particular mathematical model used to define the mapping between the input to the target outputs. In general we find two different approaches to these models, namely, linear and non-linear models.

The loss function is also called cost function and defines the rules governing the mismatch between the true label and the predicted one. Usual loss functions are least squares in regression settings, or cross-entropy in classification ones.

Finally, the optimization algorithm defines the process used to optimize the cost function. The most well known method in machine learning for this step is gradient descent.

2.1 Elastic Net

This pre-trained model is useful when there are multiple features correlated with each other, which is our case, and uses a combination of two regularizers L1 and L2. The elastic net regressor does prediction based on the following model:

$$\hat{y} = x^T w$$

That means that the predictions it computes are the result of multiplying the computed weights with the values from the training dataset.

On the training process it uses the least squares error function between the predicted values and the real value from the dataset alongside L1 and L2 regularization:

$$\mathcal{L}(X, y) = \frac{1}{2N}\|y - X^T w\|_2^2 + \alpha\|w\|_1 + \beta\|w\|_2$$

In order to learn, the model needs a way to update the weights and choose the direction in which should keep searching for a minimum local value to minimize the loss function. Then, the function to do all this process is the Gradient descent or

stochastic gradient descent which can be described by the following pseudocode:

For iter = 1 to MAX_ITERS;

(1) Select samples forming the training batch

$$X_{\text{batch}}, y_{\text{batch}} \subseteq X, y$$

(2) Compute an approximation to the gradient using the training batch

$$\nabla_w L(X_{\text{batch}}, y_{\text{batch}}) = \frac{\partial L(X_{\text{batch}}, y_{\text{batch}})}{\partial w}$$

(3) Update weights using standard update rule on the approximate gradient

$$w_{t+1} = w_t - \eta \nabla_w L(X_{\text{batch}}, y_{\text{batch}})$$

Where η is the learning rate value.

2.2 Huber Regressor

Huber regressor is a robust regressor that uses the Huber Loss, which is a method that's less sensitive to outliers. As we are speaking about a linear regressor as well, the mathematical model it uses to predict output is the same used by the ElasticNet:

$$\hat{y} = x^T w$$

Where the predicted values are the result of multiplying the weights learned by the model with the values from the training dataset.

$$\mathcal{L}(X, y) = \begin{cases} \|y - X^T w\|_2 & \text{if } \|y - X^T w\|_1 \leq \epsilon, \\ \|y - X^T w\|_1 & \text{otherwise} \end{cases}$$

In order to learn, the model needs a way to update the weights and choose the direction in which should keep searching for a minimum local value to minimize the loss function. Then, the function to do all this process is the Gradient descent or stochastic gradient descent as in the former case.

2.3 Regression Tree

Uses a decision tree to go from observations (real data) to conclusions about the item (prediction). In this case, given the type of data we are predicting, the tree is called Regression Tree as the data target is numerical and not categorical. Decision trees do not explicitly optimize a loss function, though at each decision node an heuristic is minimized, namely misclassification error, Gini index or Cross-entropy.

This process implicitly reduces the training prediction error at each split

2.4. MLP Regressor

The MLP regressor is a deep learning model based in the simplest predictor which is the perceptron is combined. A perceptron consists in a single unit with a linear model. Compared to our other three linear regression models we are going to compare, this one is not linear as MLP uses multiple linear layers, converting it to a multiple dimension problem solver.

This good feature of the MLP Regressor leads us to also an important flaw, being able to solve problems in multiple dimensions gives us multiple local minimums which can be a problem when deciding which one to choose, also it makes the model sensible to initialization and different initializations may lead to different solutions to the same problem.

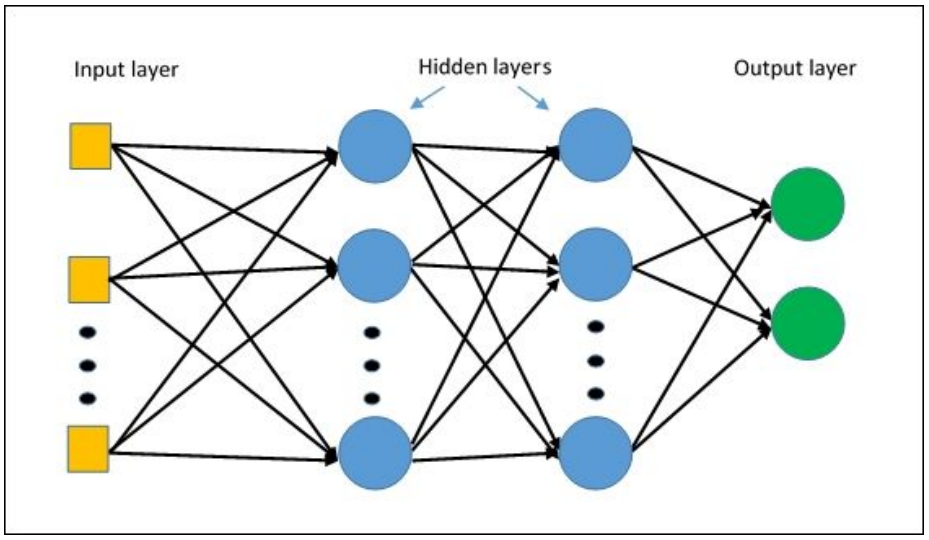

Figure 3. Multi layer perceptron with two hidden layers

After explaining some already defined and models let's now move on and explain another tool who's gonna be used which is focused in neural networks rather than in other kinds of predictions algorithms.

Tensorflow is an open source library used for machine learning developed by Google to create and train neural networks to predict values. It started as an internal project for Google and it ended up being freely available and free to use and mostly used for investigation. It can be run on either CPU or GPUs alongside other extensions for specific purposes like image recognition. TensorFlow its based in the usage of tensors, a kind of multidimensional arrays of data.

After the technical background let's now focus on explaining another very important part of

the whole process which is Strava. Strava is a social network based on GPS location which allow the users to track and register activities with either a mobile phone or other devices such as sport watches or smart watches. All the registered activities get posted on the user's profile and can be publicly available or private, being accessed only by the owner. All these activities are defined by some features like elapsed time, total distance, elevation gain, etc.

As we have a neural network architecture, then the predicted value corresponds to a forward pass of data through that architecture. We can express that as the following mathematical function:

$$\hat{y} = f(x)$$

In which f(x) corresponds to the network output provided by the last layer, named output layer.

Because we are using neural network in a regression setting, we define the loss as a least squares cost function, as follows:

$$\mathcal{L}(X, y) = \|y - f(X)\|_2$$

As we are talking about neural networks set up to perform regression, the algorithm used to learn and update weights is back propagation, which can be understood as an stochastic gradient descent version of the chain rule for updating the network weights.

3. Proposal

3.1 Project pipeline

Solving the problem of predicting elapsed time for sport activities involves following some steps in order to achieve the final goal. It is then needed a workflow or pipeline needed in order to gather, clean and process data before jumping into building a machine learning model that is able to learn from that data and finally predict values from the generalisations it mades.
Under this topic, there is an in-depth analysis of each of the steps involving the pipeline followed in the project. Also the motivation that led to each of the steps required in the pipeline is explained alongside the decisions taken during the execution of each of them with the justification on why they were made.

Figure 4. Project pipeline

Figure 4 exposes the general pipeline used throughout the whole process, starting with the process of collecting data with Strava API from the very beginning to the final point of evaluating the model with the predicted outputs by the model created alongside the process.

3.1.1 Data collection from Strava

Data which we are going to use is accessible through Strava's API and with the approval of the users can be retrieved.
At first, data collection seemed easier than what it turned out to be, as it only involved a simple token exchange between the user and the Strava API which could be solved in two or three steps. However, as the configuration for this token to access data was being built, the API got upgraded from version 2 to version 3 and the authentication process changed due to the GDPR(General Data Protection Regulation Law). The authentication changed to Oauth2

which is an authentication protocol based in three steps, which involves token exchange between the user and the API, and the token needs to be refreshed after a determined amount of time in order to be able to access the data again.

This update meant that the whole data extraction process needed to be rebuilt from scratch. Then the decision was to build a webpage to overcome this situation. A simple webpage was deployed at the cloud, in this case using the software provided by Google Cloud Platform. Google Cloud Platform is a suite of cloud computing services that runs on the same infrastructure that Google uses internally for its end-user products, such as Google Search or YouTube. Alongside a set of management tools, it provides a series of modular cloud services including computing, data storage, data analytics and machine learning.

Using this webpage, the token exchange could be made and also it was possible to store the tokens and the refresh tokens for every user who gave their information to this project's experiment.

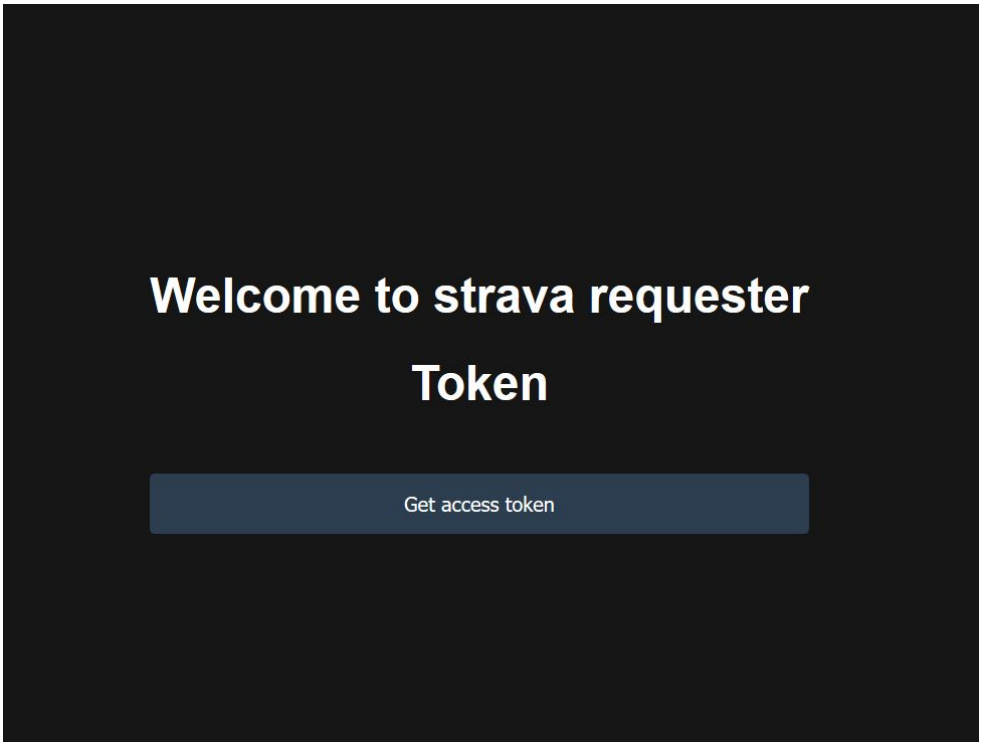

Figure 5. Data extraction pipeline

Figure 6. Authorization prompt to request data access

Figure 7. Web page shown upon token exchange successfully completed

It is worth noting that this data cession is ruled by GDPR, thus a data cession contract is established between the user and the data controller. In appendix 1 we can find the text with all rights and obligations for all parts.

Once we've been able to get both the access token and refresh token from every user, we could access their data (only in read mode to keep their privacy as we only needed to get

their information, not modify it by any means) and store it in a database also provided by the Google Cloud Platform tools. With all this process completed and all the information stored in a database, we can now proceed to the data extraction using the tokens provided by the Strava API. The user's information is extracted through requests to Strava's API exchanging the token to get authorization to get all the user's activities, which are retrieved and stored in JSON format.


```
{
  "athlete_count": 1,
  "trainer": false,
  "start_date_local": "2019-04-16T18:59:25Z",
  "elev_low": 3.5,
  "workout_type": null,
  "private": false,
  "athlete_sex": 0,
  "commute": false,
  "device_watts": false,
  "timezone": "(GMT+01:00) Europe/Madrid",
  "id": 2294790965,
  "location_city": null,
  "heartrate_opt_out": false,
  "location_country": "España",
  "kudos_count": 0,
  "gear_id": null,
  "average_speed": 7.118,
  "has_heartrate": false,
  "elapsed_time": 3419,
  "start_longitude": 2.07,
  "comment_count": 0,
  "type": "Ride",
  "start_date": "2019-04-16T16:59:25Z",
```

```
  "map": {
    "resource_state": 2,
    "id": "a2294790965",
    "summary_polyline": "}pk{FslsKvp@bOvFgvLgxAzIsa@vHsReIzSwJ~e@oMt
  },
  "utc_offset": 7200,
  "photo_count": 0,
  "elev_high": 31.4,
  "location_state": null,
  "moving_time": 2840,
  "visibility": "everyone",
  "start_latlng": [
    41.35,
    2.07
  ],
  "from_accepted_tag": false,
  "total_photo_count": 0,
  "max_speed": 10.6,
  "end_latlng": [
    41.35,
    2.07
  ],
  "has_kudoed": false,
  "distance": 20214,
  "display_hide_heartrate_option": false,


   "name": "Evening Ride",
   "total_elevation_gain": 62.7,
   "athlete": {
     "resource_state": 1,
     "id": 3080082
   },
   "manual": false,
   "flagged": false,
   "pr_count": 3,
   "upload_id": 2442352359,
   "resource_state": 2,
   "achievement_count": 13,
   "external_id": "file.fit",
   "start_latitude": 41.35
}
```


Figure 8. Activity JSON sample

Every JSON contains all the information we've got from every request made, for every user, and with a maximum of 200 registers of activities. As the maximum per request number of activities we can get is 200, pagination is used in order to extract all the registered activities

for every user, thus creating multiple JSON files for every user which will be stored in the following format: [user id] _ [#request].json. Then with all these .json files stored at cloud storage, a process is launched using a Jupyter notebook, which gets all the json files, groups them by user, and adds the data of every of them at the same file which will be saved as a .csv file. Once this process has ended, another one starts that will merge all the information in another file which will be the container of all of our dataset, used later to train our model.

achievement_count	athlete.id	athlete.resource_state	athlete_count	athlete_sex	average_cadence	average_heartrate	average_speed	comment_count	distance
0	10767915	1	4	0	58.9	144.050.015	4.044	6	10009.0
4	10767915	1	2	0	85.1	144.050.015	3.623	2	18014.0
2	10767915	1	5	0	91.0	144.050.015	3.766	0	6281.0
3	10767915	1	2	0	69.3	144.050.015	3.799	0	10341.0
2	10767915	1	1	0	47.1	144.050.015	3.912	0	7097.0

elapsed_time	elev_high	elev_low	id	kudos_count	map.resource_state	max_speed	moving_time	photo_count	pr_count
2639	14.9	10.5	2262868416	16	2	6.8	2475	0	0
5822	16.4	8.7	2258082769	14	2		4972	0	0
1674	15.3	2.7	2251268640	14	2	6.2	1668	0	1
2838	19.7	5.8	2248976973	15	2	6.8	2722	0	1
1919			2246551612	11	2	9.2	1814	0	1

resource_state	total_elevation_gain	total_photo_count	utc_offset	heart_mid	speed_low	speed_mid	speed_high
2	19.7	0	7200.0	157,0790	4,1517	4,6382	6,1797
2		0	7200.0	157,0790	4,1517	4,6382	6,1797
2	13.2	0	3600.0	157,0790	4,1517	4,6382	6,1797
2	30.3	0	3600.0	157,0790	4,1517	4,6382	6,1797
2		0	3600.0	157,0790	4,1517	4,6382	6,1797

Figure 9. Dataset final selected features sample table

At this point, all the information we needed to retrieve is stored in a simple .csv file, information that will be our starting point to begin with the creation of the model and the training and prediction with it. But before starting, data curation needs to be made in order to avoid having erroneous or incomplete data, because even if Strava's data is quite clean and ordered, some errors may appear which can lead to bad predictions or make the process of training harder.

3.1.2 Data preparation and cleaning

Once we've extracted and saved all the data and created the dataset, the need to extract the information has ended, but first we need to make sure that all this data is useful to solve our problem and also that there aren't any errors in the dataset which may lead into future problems due to null values or even corrupted data (most of Strava's data comes from third-party tools people use to record their activities, and even if Strava makes some data cleaning before storing all the information, some may be in a bad state).

Then the process of data curation begins, starting with normalization. First of all, all of the data which is non numerical is discarded with the exception of the user's gender which is converted using a one-hot encoding and converting the values 'M' to 1 and 'F' to 0. The rest of non numerical data consists in different activities attribute which are not relevant to train the model, such as the string which contains the data in a google maps style or the type of device used to record and store the activity. We'll still keep this data for future explorations and other possible uses like data representation. For the purpose of this project we are going to stick with activities which belong to the type 'Run' as it makes more sense to train a model to predict a certain type of activity rather than mixing up different kinds of activities which will probably end up with a model that is not able to generalize anything because of the data being too different.

Feature	Minimum value	Maximum value
achievement_count	0.0	42.0
athlete.id	3,080,082.0	40,901,018.0
athlete.resource_state	1.0	1.0
athlete_count	1.0	581.0
athlete_sex	0.0	1.0
average_cadence	36.3	101.3
average_heartrate	0.0	227.7
average_speed	0.0	7.4
comment_count	0.0	10.0
distance	0.0	58,127.2
elapsed_time	61.0	302,427.0
elev_high	-121.4	2,584.3
elev_low	-485.1	1,748.4
id	193,150,825.0	2,295,246,974.0
kudos_count	0.0	33.0
map.resource_state	2.0	2.0
max_speed	0.0	30.0
moving_time	0.0	36,024.0
photo_count	0.0	0.0
pr_count	0.0	34.0

resource_state	2.0	2.0
total_elevation_gain	0.0	2,285.0
total_photo_count	0.0	5.0
utc_offset	-28,800.0	14,400.0
heart_mid	1,3080	167,10
speed_low	1,0033	5,9960
speed_mid	1,5511	5,4143
speed_high	1.348	7,260500

Figure 10. Table with the final features and their value range

Standardization is often a common requirement on datasets before start using the data for training the model. The idea behind standardization is to distribute data following a Gaussian distribution with zero mean and unit standard deviation. The main reason of why standardization is a key process and hence almost a must on all machine learning problems is that most learning linear regularizers such as L1 and L2 assume that data is centered around zero and then they have variance in the same order. A feature having variance higher than orders may prevent the model from correct generalization and make the estimator bias towards that feature. The method used for standardization in this project is the StandardScaler that is included in the sklearn library. It does the process in two steps, in the first one it computes the mean and the standard deviation for the dataset passed as a parameter and then in the second one it applies the standardization by centering and scaling the whole dataset. With all this process we will also make sure that the optimization algorithm, in this case Adam which is based on stochastic gradient descent, will have a better performance thanks to the data being centered.

3.1.3 Data preprocessing

During the process of normalization and feature extraction there are some values that are missing in some activities and that is due to the wide amount of features an activity can contain and that these are very variable and dependant of the type of activity, the way they are recorded and the user's settings when it comes to registering an activity. Therefore some values need to be imputed based on known values of other registers in the dataset. That is the case for the average values of heart rate and cadence. These two features appeared to be null on some registers and the null values have been replaced by the average values of

the feature that has been calculated beforehand using the rest of values that appear on the dataset. This same process could not be done in the features ‘elevation low’ and ‘elevation high’ which represent the lowest and highest point reached during the activity. And that could not be done because imputing a value here according to the rest of activities within the dataset would probably be quite far from reality. The decision then was to set them to zero to represent them as an activity done in a plain terrain as it will affect less the model as they will deviate less from the reality and it is a good thing to assume that a running activity will happen with close to zero or even zero altitude.

After normalization we’ve also pruned some incorrect data , which was either null or N/A in this case, because after exploring the dataset we could see that most of their attributes seemed to be unused most of the time thus having an empty value for almost all of the registers. Now the dataset is almost free of wrong data with the exception of some outliers. Some registers contain activities with very small elapsed times, with a range of 0 to 60 seconds, which we can not consider a normal activity in order to predict. It is not wrong data but for the purpose of our project and to achieve our goal these are not relevant and they do not add any information we could need, in the worst case these registers may lead our model to wrong predictions. This is also because in a real situation we can not consider an activity to last 60 seconds, which is really a low elapsed time, as in an activity it wouldn’t even represent a distance of one kilometer which we could consider a good starting point to begin with the predictions.

4. Experimental design and baseline experiments.

For the purpose of this project we're going to do two experiments based on two types of models, starting from what we can consider to be easier ones as they are included in a library and already pre-defined to then go all the way to end up with a more complex one configured by hand which will be unique to fit the needs of the project. Also these two experiments will help to discover the best settings for the models and to set up the thresholds of the project with which we will accept or discard based on that.

4.1 Models performance and evaluation

The exploration of every of the different prediction techniques requires the evaluation of the performance for each of them. Doing so, we will be able to accept or reject some of the prediction techniques based on the metrics set up.

Before jumping to the metrics set up to evaluate each of the models, we need a test dataset with which could evaluate them. The process of splitting the actual dataset in two or even three different pieces it is a common procedure with machine learning problems. That is needed as it does not make sense at all to test the model accuracy with the same data values we are feeding it to learn, as it will have already generalised looking at them. Therefore the results we are going to obtain are not going to be realistic. For the purpose of the project and as the size of the whole dataset is not very big, the split is done in an 80:20 ratio. That means we are using 80% of data to train and 20% to test. The decision behind these ratio values is basically the size of the whole dataset, with these percentage we keep a meaningful amount of data for both training and testing.

Coming back to performance metrics, they are going to be used with values coming with predictions using the test dataset. Those values are new to the model so it makes sense using them as the performance metrics will show real values on the predictions.

Lets then proceed to define the metrics used to evaluate the models:

- Mean error: The value reflects the error made by the prediction technique on average expressed in minutes.
- Mean percentage error: The error made by the prediction technique on average expressed as a percentage of deviation from the expected value.
- Minimum error: Lowest error value expressed in minutes that the prediction technique made.

- Maximum error: Highest error value expressed in minutes that the prediction technique made.
- Minimum percentage error: Minimum percentage of deviation error that the prediction technique made.
- Maximum percentage error: Maximum percentage of deviation error made by the prediction technique.

4.1.1 From regression values to business oriented metrics

Given the former predictions we can compare the different methods. However for practical use in the last section we will define the concept of **accuracy of a model** as the percentage of predictions with error smaller than a given threshold.

Now that we are able to predict values given a training dataset, which in a real scenario will come directly from user's data, and validate the values when observing the prediction being output compared to the validation data, how can we know if the data we are getting is good or bad or if we should accept it. We need a metric, a value or threshold to help us decide if a difference of two, three or ten minutes is acceptable and why. Also if we can accept values that are lower than the real time or only values higher, or even if the same range of acceptability works for all the data and the same ratio could be used throughout all range of values.

To address that issue and be able to decide which outputs are acceptable, let's now set up a threshold for these predictions so we will be able to accept or reject predictions. At first, the threshold that we were going to set was the same value for all the dataset, but after taking a look at the predictions and examining the errors produced by this threshold, it did not make sense as some splits were getting too penalized by the threshold being too strict on certain splits or too wide on others. After that, the decision was then to apply a different threshold value for the splits depending on the needs of each of them. Doing so, we will avoid restrictiveness in some cases so the threshold is fair in the sense that it is specifically made for each split.

As for the thresholds the values applied for each split do not come solely from data observation, but also from self knowledge gained when practising sport that helped taking the decision on which values are good for every split. Taking that into consideration, we have decided to split the threshold between two ranges, one same value for all the data that was below a certain value of elapsed time, which could be considered short activities, and other value for longer elapsed time activities.

These thresholds apply in the negative and positive value, this means, our data predicted will be accepted within the range,

$$x - (x * T) <= x <= x + (x * T)$$

where x is the predicted time and T is the value of the threshold, expressed as a percentage of error we are going to accept.

Then, the values to be applied are an accepted error of ±5% when the elapsed time of an activity is less than 60 minutes long and the error accepted for activities with duration higher than 60 minutes is ±10%. These values make sense because if we use, for example, a value of 5 minutes, a threshold of ±5% it is accepting an error of 15 seconds whereas a threshold of ±10% it is accepting 50 seconds of error which is a huge difference in time and can make a difference in such a short span of time. When we get to higher times, the difference we can accept is higher as the values are less relevant, so for an activity that lasts 90 minutes, a prediction that has an error of ±9 minutes is good as 9 minutes are not that relevant with such a large span of time.

4.2 Experiment 1: Using Sklearn machine learning models

For the first experiment we are going to explore a free open source library called Sklearn used alongside Python. As Wikipedia states, “Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.” Even though it is indeed a very powerful library which includes much more functions apart from regression, which is the topic we’re going to be focusing on in order to predict data, we are only going to focus on this one that is the only one that solves the problem than concerns us and all the other go beyond the scope of the project.

As we stated on the background of the project, there are some specific kind of models we’re going to choose to explore and predict with as those have unique features which make them unique and different from the others. Those differences will allow to distinguish a range of outputs each coming from a model and with this information on hand, we will be able to compare and get good information which will then led us into the final conclusions.

Now we are going to start viewing each of the chosen models on action, to see how they behave and what conclusions we can get from each of them. We will end up then, with a comparison of all of them against each other and choosing one as the best pre-built model.

4.2.1 Model definitions and hyperparameters

4.2.1.1. ElasticNet

As we already stated on the background about ElasticNet, this regressor is really useful when data has features which are correlated to each other which is in fact our case. Data about the different activities have a lot of different default features coming from Strava and all of them are related to each other and also they define a unique activity. Two features that would be highly related, for example, could be time elapsed and elevation gain which clearly affects the elapsed time of the activity, as ones with lesser or close to zero elevation gain are less affected by it but the ones with a higher elevation get their elapsed time raised, caused by an irregular terrain where the activity took place.

Even though ElasticNet is a predefined model from sklearn, it offers some hyperparameters that can be chosen in order to tune the model training. For this one, we are going to use the default ones that already come with the model except for the maximum number of iterations, the l1 ratio, the tolerance and one called positive.

- Max iterations: The max iterations parameter is the one that controls the number of times the model will be going through the dataset in order to learn more about it and search for the best possible solution by decreasing the loss. For this model we have set it up to 10.000 iterations which is a really big number, but considering the size of the dataset it is an acceptable value to reach in a decent amount of time. Anyway, the model has a built-in check that will stop the training if within a certain number of iterations the loss does not descend a valuable amount or if the tolerance gets reached within the values predicted and the training ones. So most probably it will never reach those 10.000 but it is a good way to ensure we will train it for the longest time possible to try and reach the best result.
- L1 ratio: Also known as the mixing parameter helps controlling the penalty for the model. When set to 0 it will become an L2 penalty and when set to higher it becomes a combination of both. We've set the ratio to 0.05 to give more weight to L2 as is not linear as L1 and it will fit better to data.

- Positive: Controls the output to be positive or negative and when set to true it forces the outputs to be positive so in this case we set it to true as we don't want negative values for time predictions which will not make sense at all.

4.2.1.2 Huber Regressor

Outliers are one of the most common problems we can find in a dataset, as the dataset we are working with becomes bigger and bigger and the data gets sparser, it is frequent to have outliers which will make the whole process harder. Outliers are values of our dataset that differ a lot from the rest, from a mathematical point of view are those which are very far away from the normal distribution of all the data. Then, this is where Huber Regressor excels, as it computes and optimizes the squared loss of the samples thus making the fitting of the dataset and then the predictions less heavily influenced by these outliers but still having them into account. This is an important point because on sparse datasets there may be data that other loss calculations or other kinds of optimizations may discard these values for being outliers but with Huber, some of them may be considered alright and could be good for the model (we would still need to test it by fitting and predicting to prove that hypothesis correct or wrong).

The Huber model has more limitations compared to ElasticNet when it comes to configuring the parameters of the model itself. In any case the parameters we are going to set are almost the same as here we are going to modify the max number of iterations and the epsilon.

- Max iterations: The max iterations parameter is the one that controls the number of times the model will be going through the dataset in order to learn more about it and search for the best possible solution by decreasing the loss. For this model we have set it up to 10.000 iterations which is a really big number, but considering the size of the dataset it is an acceptable value to reach in a decent amount of time. Anyway, the model has a built-in check that will stop the training if within a certain number of iterations the loss does not descend a valuable amount or if the tolerance gets reached within the values predicted and the training ones. So most probably it will never reach those 10.000 but it is a good way to ensure we will train it for the longest time possible to try and reach the best result.
- Epsilon: The parameter controls the amount of data within the dataset that should be considered as an outlier. The lower the parameter is set then it is more robust to outliers. For the investigation the parameter is set to 1.1 in order to avoid the model

being too strict on the outliers and avoid removing data that it is needed and that would be otherwise deleted in case that the restriction threshold is set too high.

4.2.1.3 Regression Tree

Computational cost is something to keep in mind when trying to fit a model, especially when the dataset is really big. Huge datasets may take many minutes or even hours to iterate over and if we are expecting to do it several times for the model to be trained, it is going to cost a really precious time with the uncertainty of the results which may be different from what we are expecting. Regression Trees are good in that regard as the computational cost they require is lower than other models and much more than what some neural networks may require. And that is because the cost of using a Tree to predict data, is logarithmic in the number of data points used to train it. For example if we have a model which its computational cost is linear, and we have a dataset of 1000 values (1000 data points), the complexity of that model would be O(1000) whereas the cost of the Tree would be O(log (1000)) which is approximately 6,9. That makes a difference of more than 100 times lower in computational time which is huge considering the amount of time this can drown and affect the whole process and that is also considering a very small dataset of a thousand values, if we escalate that to bigger numbers the difference is even more noticeable.

Even though the DecissionTreeRegressor offers a good amount of features to set, the majority of them are related to the behaviour of the tree and its shape, for example the maximum number of leafs and the maximum depth it can have. In this model we are going to set up only two parameters that are more related to the training, hence then those are: criterion and presort.

- Criterion: The function used to measure the quality of a split. We are going to use both the Mean Squared Error(MSE) and Mean Absolute Error(MAE) functions to compare the results between both of them and pick the most performant one. MSE minimizes the L2 loss using the mean on every terminal node and MAE minimizes the L1 loss using the median on every terminal node.
- Presort: That options allows to sort data before training the model to help it find patterns and find the best splits. It is a double edged option as with huge datasets it may slow down the process of training as it has to sort data before anything, but as the dataset is not that big it is going to help and will speed up the process.

4.2.1.4 MLP Regressor

Flexibility is another key aspect when speaking about training a model with a dataset and predicting with it as the final goal of every model independently of which data is going to be trained and predicted on, is to fit the data as good as possible to get the best results (the one that are closer to the user expectations). To go one step further into that flexibility, we will start exploring this MLP Regressor which allows to tune a larger number of hyperparameters thus increasing the flexibility. Also it has a big difference in comparison with the other three models explored, and it is that the other three were all linear regressors whereas the MLP it is a nonlinear regressor which will adapt better to the data. The most important flaw that MLP has is that being non linear means that as the function used to predict is multi dimensional it has more than one local minimum thus making it harder to decide which one is the best and when to stop training the model as we can not longer stop on the first one. This is why also correctly tuning the hyperparameters of the MLP it's very important and can mean a huge improvement.
Among the neural networks, the MLP is the simplest one and in this case as it is already predefined model, we can not modify the structure in the same way we could do if we were building it from scratch. It is then a first approach which will lead us later on into TensorFlow to code and build our model from scratch to give it the features we really want it to have to fit our model as much as possible and get the best predictions.

The MLP sklearn model offers a large list of parameters to be set in order to customize the model creation. That also reflects the fact that neural networks are more complex than the other types of regressors like the ones we've investigated before but also they are more powerful and flexible when it comes to training and prediction. In this case, for the purpose of the investigation we've modified the following parameters.

- Hidden layers sizes: This parameter specifies the number of neurons in the ith hidden layer of the model. It is going to be set to 70 which is a large number but as the model is training on the whole dataset, it is going to help with the training process.
- Learning rate init: Establishes the model learning rate for the weight updates that take place every step. It is set up on 0.01 to avoid the model diverging too much and not reaching a local minimum which could be a possible good point for the model to stop and be used for predictions.
- Learning rate: Modifies the way the rate value will be applied on every step of the training. We are setting it to constant so it means that the value we have set up it is

going to be applied constantly on every step, it will not change or adapt to the loss values the model is computing.

- Max_iter: As comparison from other models the MLP Regressor will iterate the number of value that is set up without stopping automatically when the loss is not changing from one step to another, even though it will when the tolerance that's set up is reached. The value is set to 1000 to make sure that the whole dataset is at least iterated once.
- Tol: The value used by the model to optimize the training. It is also a metric to determine when the convergence is reached and then stop the training, that means that if the score is not improving by at least the value we have set up then it is considered that the model has already reached that point of convergence and the training is stopped even though it did not reach the maximum amount of iterations we set with the max_iter parameter. The value is set to 1e-8 to give the model a bit of margin to train and to avoid early stopping due to the score not improving for several iterations which may change in future ones.

4.2.2.1 General performance and baseline reference

The baseline experiment corresponds to the evaluation of the mean squared error on all data and then evaluating the output with the performance metrics defined above.

Models	Mean Error	Mean Percentage Error	Minimum Error	Maximum Error	Minimum Percentage Error	Maximum Percentage Error
ElasticNet	0,05	16,76	-6,14	1,27	-92,42	43490,56
Huber Regressor	0,05	12,08	-2,01	0,05	-191,25	61,49
Regression Tree MSE	0,06	15,87	-2,03	0,46	-92,94	391,25
Regression Tree MAE	0,07	19,56	-1,59	6,34	-81,83	2364,63
MLP Regressor	0,2	75,18	-1,51	7,31	-479,5	9138,75

Figure 11. Prediction techniques metrics obtained using the whole dataset

Figure 11 shows the performance metrics we have defined before for all the models explored in the first experiment. Observe that the worst performance technique corresponds

to the MLP Regressor which has the worst value in the mean percentage error and also the highest and lowest percentage errors throughout the whole dataset.
Taking a look at the table we can also observe that the best performance metrics are obtained when predicting using the Huber Regressor, as it holds the lowest percentage error on average. Even though the minimum percentage error it is not the best among the others, the maximum percentage error it is the lowest from all the values, fact that compensates for not having the better metric when it comes to the minimum percentage error.

4.2.2.2 Refined experiments: Tier performances

The former results are not good enough as the minimum and maximum percentage errors are not that low yet even though the average percentage of error it is around 10% for all the prediction techniques with the exception of the MLP regressor. For example, the best performance technique achieves an average error of 12,08%. This corresponds to an error of 36 seconds in a 5 minutes course which it is an example of a short activity, but 435 seconds which means almost 8 minutes on an hour course, which is an example of a long run course. This disparity on performance motivates the definition of Tiers and partitioning data in splits to then create a prediction model for each of them using all of the prediction techniques.

Instead of training a unique model we split data according to the length of the run so the activities within each Tier are closer to each other and therefore the dataset is less sparse. This results in the definition of 5 tiers as follows:

- Tier <15 mins: Contains all activities which its elapsed time duration is less than 15 minutes, excluding those that are 15 minutes long.
- Tier 15-30 mins: Contains all activities which its elapsed time duration is between 15 and 30 minutes, including those that are 15 minutes long.
- Tier 30-45 mins: Contains all activities which its elapsed time duration is between 30 and 45 minutes, including those that are 30 minutes long.
- Tier 45-60 mins: Contains all activities which its elapsed time duration is between 45 and 60 minutes, including those that are 45 minutes long.
- Tier 60-90 mins: Contains all activities which its elapsed time duration is between 60 and 90 minutes, including those that are 60 minutes long.
- Tier 90-120 mins: Contains all activities which its elapsed time duration is between 90 and 120 minutes, including those that are 90 minutes long.

- Tier >120 mins: Contains all activities which its elapsed time duration is longer than 120 minutes, including those that are 120 minutes long.

Tables from Figures 12 to 18 show the results obtained when applying the prediction techniques on the different data splits.

Models	Mean Error	Mean Percentage Error	Minimum Error	Maximum Error	Minimum Percentage Error	Maximum Percentage Error
ElasticNet	-0,0017	12,07	-0,05	0,01	-64,52	162,08
Huber Regressor	0,002	6,83	-0,05	0	-100	71,42
Regression Tree MSE	0,0033	12,16	-0,04	0,04	-71,428	181,506
Regression Tree MAE	0,0033	12,296	-0,04	0,04	-46,25	181,506
MLP Regressor	0,0003	67,703	-0,03	0,04	-4025	2342,85

Figure 12. Split less than 15 mins. performance metrics

Models	Mean Error	Mean Percentage Error	Minimum Error	Maximum Error	Minimum Percentage Error	Maximum Percentage Error
ElasticNet	-0,012	1,317	-0,09	0,09	-36,91	50,76
Huber Regressor	0,0115	7,173	-0,11	0,01	-46,27	9,39
Regression Tree MSE	0,0146	8,879	-0,09	0,1	-35,09	68,403
Regression Tree MAE	0,0175	10,654	-0,08	0,12	-32,812	79,098
MLP Regressor	0,0005	9,588	-0,1	0,06	-43,52	38,27

Figure 13. Split between 15-30 mins. performance metrics

Models	Mean Error	Mean Percentage Error	Minimum Error	Maximum Error	Minimum Percentage Error	Maximum Percentage Error

ElasticNet	0,0038	2,36	-0,09	0,06	-23,52	22,71
Huber Regressor	0,022	5,218	-0,14	0,07	-33,197	24,23
Regression Tree MSE	0,02375	7,936	-0,13	0,12	-30,36	41,998
Regression Tree MAE	0,0217	7,336	-0,11	0,12	-24,81	40,806
MLP Regressor	0,0009	6,87	-0,09	0,08	-22,63	27,287

Figure 14. Split between 30-45 mins. performance metrics

Models	Mean Error	Mean Percentage Error	Minimum Error	Maximum Error	Minimum Percentage Error	Maximum Percentage Error
ElasticNet	-0,0009	0,413	-0,08	0,08	-14,98	18,32
Huber Regressor	0,026	6,013	-0,09	0,09	-16,59	19,05
Regression Tree MSE	0,0284	6,41	-0,13	0,11	-22,92	24,96
Regression Tree MAE	0,0335	7,346	-0,13	0,13	-22,92	28,514
MLP Regressor	0,0035	8,013	-0,13	0,17	-25,81	36,83

Figure 15. Split between 45-60 mins. performance metrics

Models	Mean Error	Mean Percentage Error	Minimum Error	Maximum Error	Minimum Percentage Error	Maximum Percentage Error
ElasticNet	-0,005	0,35	-0,27	0,12	-30,82	20,17
Huber Regressor	0,055	8,153	-0,25	0,18	-28,81	28,61
Regression Tree MSE	0,0644	9,86	-0,26	0,28	-30,723	47,478
Regression Tree MAE	0,059	8,8549	-0,23	0,23	-26,885	35,8954
MLP Regressor	0,0026	9,508	-0,35	0,25	-42,089	39,431

Figure 16. Split between 60-90 mins. performance metrics

Models	Mean Error	Mean Percentage Error	Minimum Error	Maximum Error	Minimum Percentage Error	Maximum Percentage Error

ElasticNet	0,0088	1,401	-0,15	0,11	-14,2	12,14
Huber Regressor	0,065	7,072	-0,2	0,15	-18,28	16,24
Regression Tree MSE	0,065	7,165	-0,13	0,25	-12,931	27,6294
Regression Tree MAE	0,0444	4,824	-0,14	0,12	-13,877	11,647
MLP Regressor	-0,0556	16,689	-0,46	0,34	-50,26	33,7

Figure 17. Split between 90-120 mins. performance metrics

Models	Mean Error	Mean Percentage Error	Minimum Error	Maximum Error	Minimum Percentage Error	Maximum Percentage Error
ElasticNet	2,828	144,706	-7,47	9	-88,74	397,85
Huber Regressor	0,463	20,45	-1,34	1,38	-57,68	78,29
Regression Tree MSE	1,0491	29,354	-3,4	5,08	-54,398	80,355
Regression Tree MAE	0,643	16,994	-2,58	0,38	-43,908	21,6271
MLP Regressor	0,61	114,166	-4,39	5,53	-69,01	517,62

Figure 18. Split more than 120 mins. performance metrics

Observe that with the tiers the average error for each tier corresponds to the values in the fable from Figure 19.

	< 15 mins	15-30 mins	30-45 mins	45-60 mins	60-90 mins	90-120 mins	> 120 mins
Average Percentage Error	22,2118	7,5222	5,944	5,639	7,34518	7,4302	69,4366

Figure 19. Average percentage error values for each split

With these calculated percentage errors values in hand, we can now use these calculations from the table and show what these values represent with an elapsed time value for each of the splits.

Values	Predicted Value
0:10:43	0:11:06
0:17:10	0:18:12

0:35:43	0:37:25
0:45:05	0:47:00
1:27:59	1:33:01
1:40:39	1:43:35
2:15:29	3:33:45

Figure 20. Predicted values with average percentage error applied

Figure 20 shows an example of an elapsed time value for each of the splits made and the predicted output once applied the percentage of error that each of the corresponding splits have. We can easily observe that by far, the worst it is the one from the last split, and we were expecting that outcome as the average percentage error was also the highest as we have seen in Figure 19.

4.3 Experiment 2: Exploring deep learning architectures with Tensorflow

In this second experiment the main goal is going to be to experiment with a deep learning model made by hand. Even though it also includes the usage of certain libraries the usage of them is going to be different as for this experiment more configuration needs to be made and also more decisions need to be taken in order to set up certain model settings especially when it comes to build its structure.

Deep learning bases the process of learning in the usage of Artificial Neural Network which resemble, but are not equal, animal brains in the sense that they have artificial neurons connected to each other and they form what is also known as a network. These artificial neurons can receive or send information from or to another artificial neurons during the process of learning. Commonly the information that is shared are numbers that represent the weight of those connections and it's being adapted and modified throughout all the process of learning. Weights represent the strength of the connection between two artificial neurons (connections are often also called edges). Artificial Neural Networks are composed by different layers of artificial neurons, which may contain different amount of them, and the number of layers the network has is defined by the user at the moment of building the model to predict. Size is adapted to the needs of the problem that is going to be worked on.

With all this information on hand,it is easier now to understand how powerful Artificial Neural Networks are and why is a good decision to explore them and compare them against regressors to see the difference when it comes to model accuracies.

When it comes to model configuration it is needed to differentiate between model structure and parameter decision for the model layers.

- **Model Structure**

  Even though model's size is a personal decision, it is very related to the amount odata that is needed to be explored during the training process. For very small datasets, a smaller structure is used as it will learn in a short period of time and we will avoid overfitting that may occur when using a huge network with that amount of data. Model's structure is also a tradeoff between speed and learning capability. Larger models will be capable to digest bigger amounts of data but the processing time is also going to be higher whereas in smaller models both processing time and learning capability.

  For the purpose of this investigation and because the amount of data gathered to generate the dataset we can not consider it to be very large, the model to be created is small and will consist of three layers, one of them being a hidden one. The other two layers are the input and output ones. All of them are dense layers, that means that every artificial neuron is connected to every artificial neuron from the layer before and after the hidden one.

  When it comes to the number of artificial neurons for every layer, the first two ones that are the input layer and the hidden layer, share the same size which is 32 neurons each. For the output, as we are expecting one result per training data input, the layer has set the artificial neurons number to 1 so it will yield only one value.

- **Parameter decision**

  Even though there are parameters that are related only to certain layers within the model, we can set general configuration for the model the same way we could do with Sklearn models. Those parameters include:

  - **Loss**

    In this case the loss metric chosen is going to be the same we had with the other models we have explored, the Mean Squared Error. The reason why this metric is chosen it is because we are seeking for the minimum error throughout all the predictions, this way the prediction error will be distributed equally.

  - **Optimizer**

    Adam Optimizer is the choice to go as an optimization algorithm for the prediction model we are creating. It is an algorithm based on stochastic

gradient descent with the adding of adaptive learning rates with low computation and memory requirements as it only computes first-order gradients.

- **Metrics**

  List of metrics to be evaluated by the model during training and testing, the values set up are both Mean Squared Error and Mean Absolute Error as they are the same as the ones used in the Loss function.

Test time	Predicted time	Deviation hh:mm:ss	Deviation Mins.	Error %
0:57:48	0:44:06	-0:13:42	-0.13	-23,7024
0:44:56	0:50:28	0:05:32	0.05	12,3145
1:07:40	1:33:58	0:26:18	0.26	38,8670
1:14:42	0:45:56	-0:28:46	-0.28	-38,5096
0:31:58	0:40:44	0:08:46	0.08	27,4244
1:10:01	1:13:30	0:03:29	0.03	4,9750
0:11:41	0:10:30	-0:01:11	-0.01	-10,1284

Figure 21. Predicted values sample

Figure 21 shows an extract of the predictions given by the model alongside the calculation of the deviation and the percentage of the error made by the model for every elapsed time value contained in the test dataset.

	Median Error	Mean Error	Minimum Error	Maximum Error
Value (mins.)	< 0	0,0006614173228	-1,59	3,31
Percentage (%)	-0,90	28,19	-3533,58	11878,75

Figure 22. Prediction metrics

The table from Figure 22 shows different statistics calculated after model prediction using the test dataset. As we can observe, even though the median and the mean errors are really low and we could agree that only looking at these numbers the model seems really good at

first sight. Nevertheless, if we look at the percentages, especially at the mean error one, we can observe that the truth is the model on average has around a 30% error when it comes to the final predictions. On certain cases a value of 30% could be acceptable, but if we take a numerical example we will realize that for our use case it is a really high value which needs to be lowered.

With an initial time of 1 hour, the 30% deviation will be of 18 minutes, that means we can have a predicted time of 1h18m or 42mins. That is a huge gap between the real value and the output we are getting from the model.

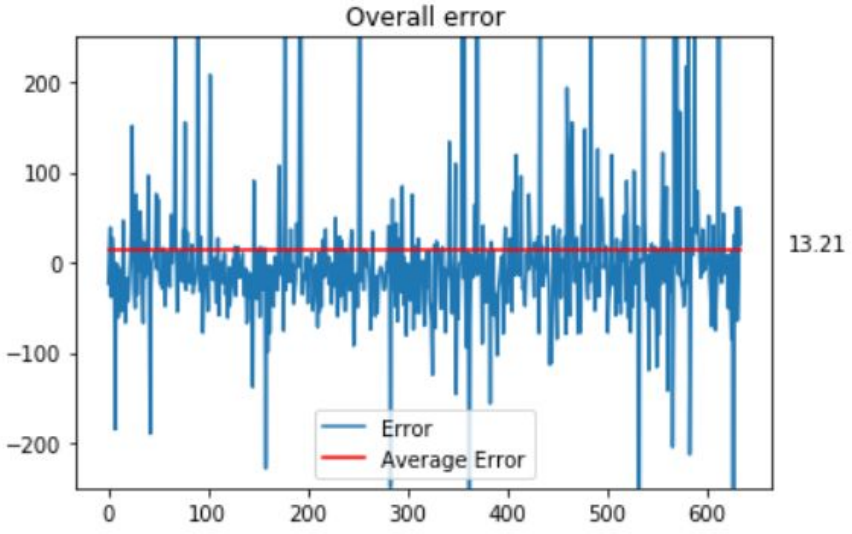

Figure 23. Model's error distribution graph

Figure 23 shows the distribution of the error in the predictions throughout all the dataset. The error shown it is expressed in percentages following the value format we have from the prediction output table. As we can observe there is a huge variation of the error even though the mean it is quite close to zero as its value is 13.21%. That information is a clear indicator that the dataset is too sparse for the model to learn a good generalization for the whole dataset. As the error percentage is not that low, there is still room for improvement. The next step to try to make the error percentage lower is splitting the data in different tiers and make a model for each of them. This way we will solve the problem of data being too sparse and getting so many different values that are avoiding that the model generalises in a correct way.

In order to improve the whole model accuracy, we are going to split the data in seven different datasets and then create a neural network for every of the splits. All of the models

will be trained on the data of each of these splits thus creating specific models that will learn from the specific features of the data contained in the dataset partition is being trained on.
These tiers are made by splitting data depending on the elapsed time, clustering the activities that are similar according to the time value. Then we are creating the seven different splits that consist of activities that are between 0 and 15 minutes, 15 and 30 minutes, 45 and 60 minutes, 60 and 90 minutes, 90 and 120 minutes and more than 120 minutes. After training the different models with the data of each of these splits, we obtained the following results:

0-15 minutes tier

Test time	Predicted time	Deviation hh:mm:ss	Deviation Mins.	Error %
0:11:12	0:09:58	-0:01:14	-0.01	-11,011905
0:07:40	0:07:39	-0:00:01	-0.00	-0,217391
0:10:40	0:10:50	0:00:10	0.00	1,562500
0:10:07	0:09:55	-0:00:12	-0.00	-1,976936
0:09:34	0:08:10	-0:01:24	-0.01	-14,634146
0:09:19	0:04:41	-0:04:38	-0.04	-49,731664
0:04:30	0:05:12	0:00:42	0.00	15,555556

Figure 24. Tier less than 15 mins. predicted values sample

Figure 24 contains a sample of the predictions made by the model trained on the subset of the dataset that contains elapsed times between 0 and 15 minutes. We can observe different test elapsed times used to compare with the model's prediction and the calculated deviation and the percentage of error.

	Median Error	Mean Error	Minimum Error	Maximum Error
Value (mins.)	< 0	0,00317	-0,06	0,01
Percentage (%)	2.3	11,71	-73,027	101,61

Figure 25. Tier less than 15 mins. performance metrics

Figure 25 shows a representation of the model's prediction metrics. Those metrics are an approximation on how well or bad does the model work and we can make assumptions based on the error it does on average and also which is the minimum and maximum error. We can see that the minimum and maximum error are heavily separated from the average

error, and both are close to be 100 which means double the value than what it should be. Looking at the metrics from the table we can conclude that the model behaves well on average but there are some values that differ from the average error and some are huge differences that can be up to double the expected value.

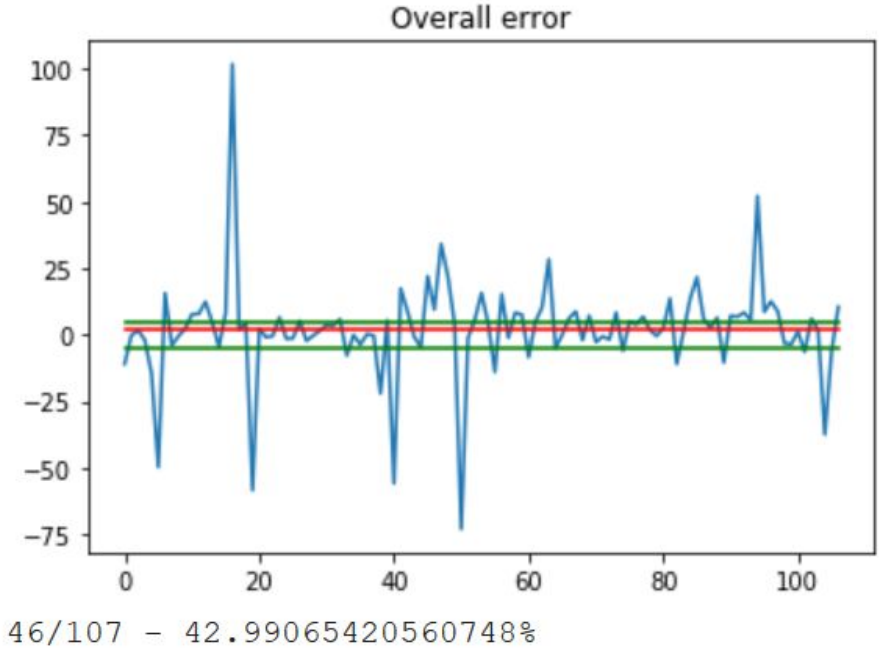

Figure 26. Tier less than 15 mins. error distribution graph

Figure 26 represents the metrics we have been analyzing above in a graphical way. That helps understanding how well is the model working for each of the values predicted. The green horizontal lines represent the threshold that we are accepting, in this case ± 5 percent error value. Represented by the red line we have the mean percentage error made by the model on the predictions that as we can see is inside of the accepted threshold.
Lastly, in the bottom we have the number of correct predictions over the total of data used to predict, which is 43 percent, quite close to half correct values.

15-30 minutes tier

Test time	Predicted time	Deviation hh:mm:ss	Deviation Mins.	Error %

0:17:56	0:17:36	-0:00:20	-0.00	-1,8587
0:21:19	0:21:24	0:00:05	0.00	0,3909
0:26:09	0:23:25	-0:02:44	-0.02	-10,4525
0:22:35	0:23:23	0:00:48	0.00	3,5424
0:15:20	0:20:54	0:05:34	0.05	36,3043
0:22:32	0:23:27	0:00:55	0.00	4,0680
0:16:29	0:18:02	0:01:33	0.01	9,4034

Figure 27. Tier between 15-30 mins. predicted values sample

Figure 27 contains a sample of the predictions made by the model trained on the subset of the dataset that contains elapsed times between 15 and 30 minutes. We can observe different test elapsed times used to compare with the model's prediction and the calculated deviation and the percentage of error.

	Median Error	**Mean Error**	**Minimum Error**	**Maximum Error**
Value (mins.)	< 0	0,0135	-0,09	0,05
Percentage (%)	0,85	8,33	-40,42	36,3

Figure 28. Tier between 15-30 mins. performance metrics

Figure 28 shows a representation of the model's prediction metrics. Those metrics are an approximation on how well or bad does the model work and we can make assumptions based on the error it does on average and also which is the minimum and maximum error. We can see that the minimum and maximum error are less separated from the average than the previous split, hence these values are closer to the real value. Looking at the metrics from the table we can conclude that the model behaves well on average as the error is acceptably low but the maximum and minimum error still are higher than the accepted threshold.

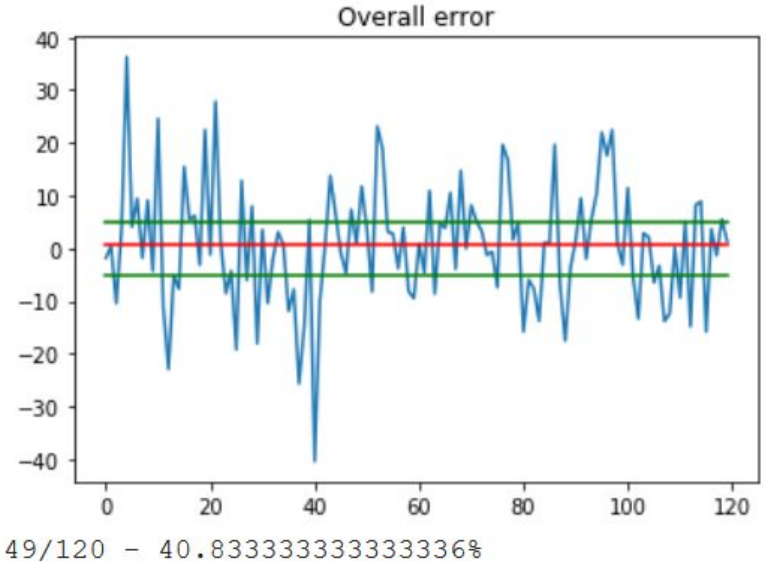

Figure 29. Tier between 15-30 mins. error distribution graph

Figure 29 represents the metrics we have been analyzing above in a graphical way. That helps understanding how well is the model working for each of the values predicted. The green horizontal lines represent the threshold that we are accepting, in this case ± 5 percent error value. Represented by the red line we have the mean percentage error made by the model on the predictions that as we can see is inside of the accepted threshold and also is really close to zero, which in fact is not the reality and we can see it in the graph.

Lastly, in the bottom we have the number of correct predictions over the total of data used to predict, which is roughly 40 percent. This model is a bit worse than the one from the previous split, even though it is using a bit more data, because the errors are also higher on average.

30-45 minutes tier

Test time	Predicted time	Deviation hh:mm:ss	Deviation Mins.	Error %
0:44:23	0:38:32	-0:05:51	-0.05	-13,1806
0:44:29	0:40:58	-0:03:31	-0.03	-7,9056
0:34:48	0:34:09	-0:00:39	-0.00	-1,8678
0:30:38	0:32:44	0:02:06	0.02	6,8553
0:35:36	0:36:44	0:01:08	0.01	3,1835
0:44:53	0:47:32	0:02:39	0.02	5,9042
0:36:43	0:35:53	-0:00:50	-0.00	-2,2696

Figure 30. Tier between 30-45 mins. predicted values sample

Figure 30 contains a sample of the predictions made by the model trained on the subset of the dataset that contains elapsed times between 30 and 45 minutes. We can observe different test elapsed times used to compare with the model's prediction and the calculated deviation and the percentage of error.

	Median Error	Mean Error	Minimum Error	Maximum Error
Value (mins.)	< 0	0,024	-0,1	0,07
Percentage (%)	0,89	8,032	-27,18	23,45

Figure 31. Tier between 30-45 mins. performance metrics

Figure 31 shows a representation of the model's prediction metrics. Those metrics are an approximation on how well or bad does the model work and we can make assumptions based on the error it does on average and also which is the minimum and maximum error. We can see that the minimum and maximum error are less separated from the average than the previous split, hence these values are closer to the real value. Looking at the metrics from the table we can conclude that the model behaves well on average as the error is acceptably low but the maximum and minimum error still are higher than the accepted threshold. In comparison to the last split, both the minimum and the maximum error values are lower. We can see a trend here as the maximum errors are decreasing every split. Therefore despite the average errors for each model is similar, as the maximum values are getting lower, the models are becoming better.

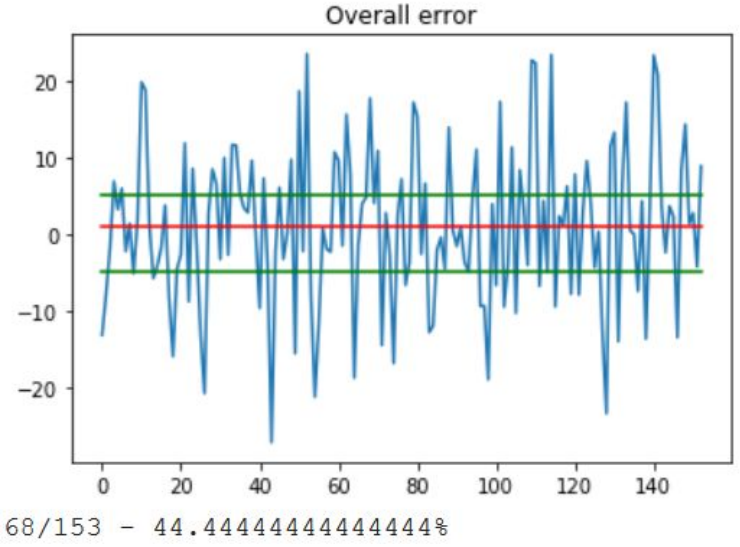

Figure 32. Tier between 30-45 mins. error distribution graph

Figure 32 represents the metrics we have been analyzing above in a graphical way. That helps understanding how well is the model working for each of the values predicted. The green horizontal lines represent the threshold that we are accepting, in this case ± 5 percent error value. Represented by the red line we have the mean percentage error made by the model on the predictions that as we can see is inside of the accepted threshold and also is really close to zero, which in fact is not the reality and we can see it in the graph.
Lastly, in the bottom we have the number of correct predictions over the total of data used to predict, which is around 44 percent. Taking this into account, and also adding the fact that the model has more data to train with, the model is good and it is generalising in a great way and there is room for improvement. This improvement can be done by adding more data to train with.

45-60 minutes tier

Test time	Predicted time	Deviation hh:mm:ss	Deviation Mins.	Error %
0:59:02	0:52:52	-0:06:10	-0.06	-10,4461
0:52:29	0:48:04	-0:04:25	-0.04	-8,4154
0:53:11	0:53:02	-0:00:09	-0.00	-2,8204
0:56:34	0:53:00	-0:03:34	-0.03	-6,3052
0:47:25	0:42:48	-0:04:37	-0.04	-9,7364
0:45:00	0:48:37	0:03:37	0.03	8,0370
0:55:14	0:53:49	-0:01:25	-0.01	-2,5649

Figure 33. Tier between 45-60 mins. predicted values sample

Figure 33 contains a sample of the predictions made by the model trained on the subset of the dataset that contains elapsed times between 45 and 60 minutes. We can observe different test elapsed times used to compare with the model's prediction and the calculated deviation and the percentage of error. This excerpt of predicted data has some good predictions with a maximum of error in absolute value of 10 percent.

	Median Error	Mean Error	Minimum Error	Maximum Error
Value (mins.)	< 0	0,027	-0,07	0,07
Percentage (%)	-1,34	6,18	-12,69	15,61

Figure 34. Tier between 45-60 mins. performance metrics

Figure 34 shows a representation of the model's prediction metrics. Those metrics are an approximation on how well or bad does the model work and we can make assumptions based on the error it does on average and also which is the minimum and maximum error. We can see that the minimum and maximum error are less separated from the average than the previous split, hence these values are closer to the real value. Looking at the metrics from the table we can conclude that the model behaves well on average as the error is acceptably low but the maximum and minimum error still are higher than the accepted threshold. In comparison to the last split, both the minimum and the maximum error values are lower and quite close to the threshold that is 5 percent error both for positive and negative values. The maximum error values being that close to the threshold it is a good sign because the error is spread equally throughout the predictions and the model is generalising well. This also means that there are not spikes of errors on some data that are too far away from the average percentage error, which can mean that there are some specific cases that the model it is not able to generalise and then it will need more examples of those specific values in order to generalise correctly.

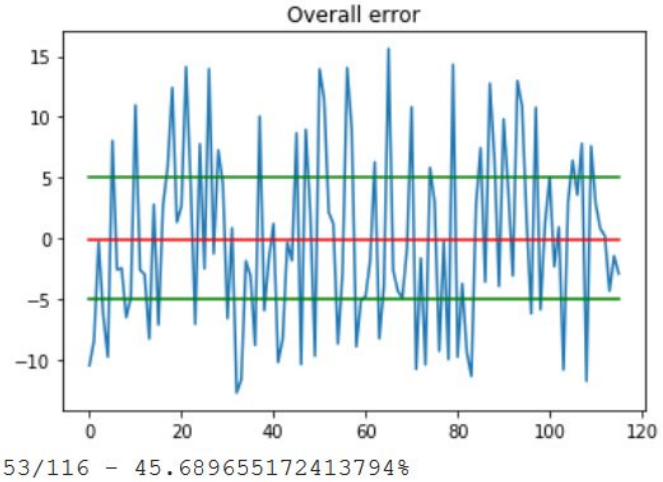

Figure 35. Tier between 45-60 mins. error distribution graph

Figure 35 represents the metrics we have been analyzing above in a graphical way. That helps understanding how well is the model working for each of the values predicted. The

green horizontal lines represent the threshold that we are accepting, in this case ± 5 percent error value. Represented by the red line we have the mean percentage error made by the model on the predictions that as we can see is inside of the accepted threshold and also is really close to zero, which in fact is not the reality and we can see it in the graph.
Lastly, in the bottom we have the number of correct predictions over the total of data used to predict, which is 45 percent, closer to half of correct predicted values. The error percentage keeps on the same range of values as the rest of splits so the only improvement the model has is when it comes to maximum error values that is getting lower compared to the previous ones.

60-90 minutes tier

Test time	Predicted time	Deviation hh:mm:ss	Deviation Mins.	Error %
1:01:34	1:07:14	0:05:40	0.05	9,2041
1:05:30	1:08:00	0:02:30	0.02	3,8168
1:01:46	1:07:09	0:05:23	0.05	8,7156
1:26:28	1:18:16	-0:08:12	-0.08	-9,4834
1:04:12	1:08:18	0:04:06	0.04	6,3863
1:28:04	1:10:34	-0:17:30	-0.17	-1,9871
1:01:11	1:07:33	0:06:22	0.06	10,4059

Figure 36. Tier between 60-90 mins. predicted values sample

Figure 36 contains a sample of the predictions made by the model trained on the subset of the dataset that contains elapsed times between 60 and 90 minutes. We can observe different test elapsed times used to compare with the model's prediction and the calculated deviation and the percentage of error. This excerpt of predicted data has some good predictions with a maximum of error in absolute value of 10 percent.

	Median Error	Mean Error	Minimum Error	Maximum Error
Value (mins.)	0,01	0,065	-0,23	0,15
Percentage (%)	2,19	9,81	-25,99	24,96

Figure 37. Tier between 60-90 mins. performance metrics

Figure 37 shows a representation of the model's prediction metrics. Those metrics are an approximation on how well or bad does the model work and we can make assumptions based on the error it does on average and also which is the minimum and maximum error. We can see that the minimum and maximum error are less separated from the average than the previous split, hence these values are closer to the real value. Looking at the metrics from the table we can conclude that the model behaves well on average as the error is acceptably low but the maximum and minimum error still are higher than the accepted threshold. In comparison to the last split, both the minimum and the maximum error values are higher.

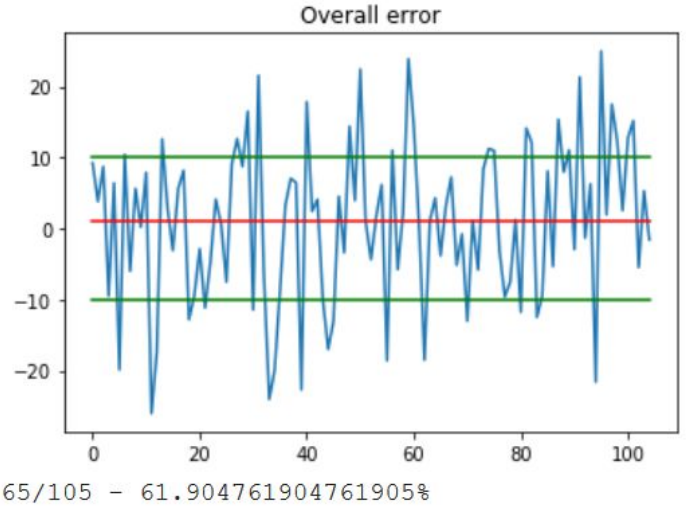

Figure 38. Tier between 60-90 mins. error distribution graph

Figure 38 represents the metrics we have been analyzing above in a graphical way. That helps understanding how well is the model working for each of the values predicted. The green horizontal lines represent the threshold that we are accepting, in this case ± 10 percent error value. The error threshold is higher in this case because the elapsed time values are also higher, so it makes sense to give more room for error as it won't be affecting the predicted time that much. Represented by the red line we have the mean percentage error made by the model on the predictions that as we can see is inside of the accepted threshold and also is really close to zero, which in fact is not the reality and we can see it in the graph.

Lastly, in the bottom we have the number of correct predictions over the total of data used to predict, which is 61 percent. It is a huge improvement in comparison to the rest of splits as it

is predicting over half the dataset right, and also the error percentage it is stable throughout the predictions.

90-120 minutes tier

Test time	Predicted time	Deviation hh:mm:ss	Deviation Mins.	Error %
1:50:28	1:38:05	-0:12:23	-0.12	-11,2100
1:41:13	1:38:48	-0:02:25	-0.02	-2,3876
1:43:36	1:42:33	-0:01:03	-0.01	-1,0135
1:43:51	1:44:24	0:00:33	0.00	5,2961
1:30:59	1:42:50	0:11:51	0.11	13,0244
1:45:41	1:47:31	0:01:50	0.01	1,7347
1:31:56	1:44:02	0:12:06	0.12	13,1617

Figure 39. Tier between 90-120 mins. predicted values sample

Figure 39 contains a sample of the predictions made by the model trained on the subset of the dataset that contains elapsed times between 90 and 120 minutes. We can observe different test elapsed times used to compare with the model's prediction and the calculated deviation and the percentage of error. This excerpt of predicted data has some good predictions, even though the maximum absolute percentage error got higher.

	Median Error	Mean Error	Minimum Error	Maximum Error
Value (mins.)	< 0	0,0448	-0,12	0,19
Percentage (%)	0,529	5,083	-11,21	21,63

Figure 40. Tier between 90-120 mins. performance metrics

Figure 40 shows a representation of the model's prediction metrics. Those metrics are an approximation on how well or bad does the model work and we can make assumptions based on the error it does on average and also which is the minimum and maximum error. We can see that the minimum and maximum error are less separated from the average than the previous split, hence these values are closer to the real value. Looking at the metrics from the table we can conclude that the model behaves well on average as the error is acceptably low but the maximum and minimum error still are higher than the accepted

threshold. The minimum error percentage got lower but as for the maximum error percentage it remains more or less the same value.

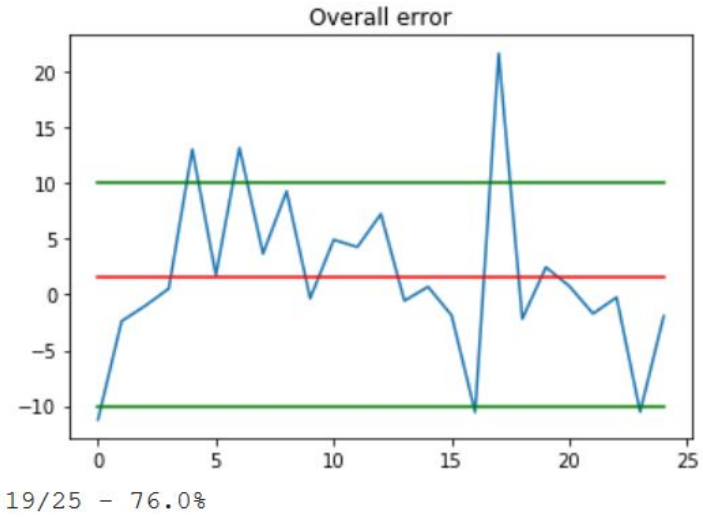

Figure 41. Tier between 90-120 mins. error distribution graph

Figure 41 represents the metrics we have been analyzing above in a graphical way. That helps understanding how well is the model working for each of the values predicted. The green horizontal lines represent the threshold that we are accepting, in this case ± 10 percent error value. The error threshold is higher in this case because the elapsed time values are also higher, so it makes sense to give more room for error as it won't be affecting the predicted time that much. Represented by the red line we have the mean percentage error made by the model on the predictions that as we can see is inside of the accepted threshold and also is really close to zero, which in fact is not the reality and we can see it in the graph.

Lastly, in the bottom we have the number of correct predictions over the total of data used to predict, which is 76 percentage, over the quarter parts of the whole training dataset. This dataset is much smaller to the rest of the splits therefore the reason behind this huge improvement compared to the rest of models is the similarity between the data used to train, that helped to generalise much better.

120 minutes or more tier

Test time	Predicted time	Deviation hh:mm:ss	Deviation Mins.	Error %
3:49:50	2:45:13	-1:04:37	-1.04	-28,1146
6:45:28	-1 days +23:57:55	-6:47:33	-6.47	-100,5138
3:41:07	0:20:30	-3:20:37	-3.20	-90,7289
2:59:15	3:55:12	0:55:57	0.55	31,2134
2:15:52	18:09:57	15:54:05	5.54	702,2203
2:09:38	0:39:46	-1:29:52	-1.29	-69,3237
6:23:29	-1 days +23:58:38	-6:24:51	-6.24	-100,3564

Figure 42. Tier more than 120 mins. predicted values sample

Figure 42 contains a sample of the predictions made by the model trained on the subset of the dataset that contains elapsed times of 120 minutes or more. We can observe different test elapsed times used to compare with the model's prediction and the calculated deviation and the percentage of error. From this excerpt of predicted data we can observe that this split did not work correctly as the data was too sparse and the model has not been able to generalise and learn correctly from the training data.

	Median Error	Mean Error	Minimum Error	Maximum Error
Value (mins.)	-1,165	2,8925	-6,47	5,54
Percentage (%)	-48,71	126,578	-101,05	702,22

Figure 43. Tier more than 120 mins. performance metrics

Figure 43 shows a representation of the model's prediction metrics. Those metrics are an approximation on how well or bad does the model work and we can make assumptions based on the error it does on average and also which is the minimum and maximum error. We can see that the minimum and maximum error are very much higher than any other split. Also the mean error percentage got higher, and in a very high ratio, when compared to any of the other splits.

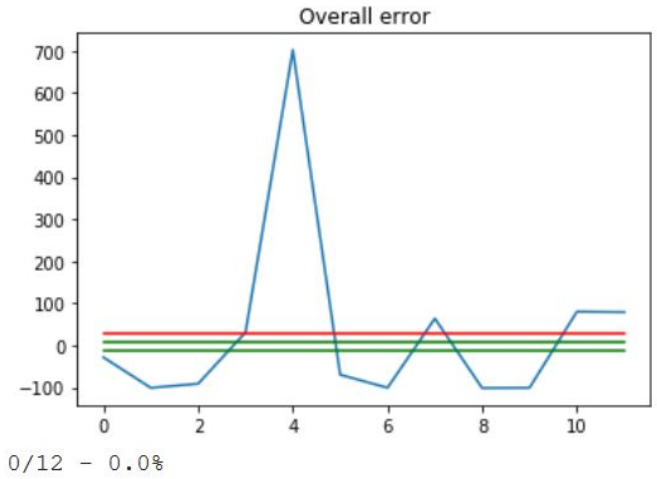

Figure 44. Tier more than 120 mins. error distribution graph

Figure 44 represents the metrics we have been analyzing above in a graphical way. That helps understanding how well is the model working for each of the values predicted. The green horizontal lines represent the threshold that we are accepting, in this case ± 10 percent error value. The error threshold is higher in this case because the elapsed time values are also higher, so it makes sense to give more room for error as it won't be affecting the predicted time that much. Represented by the red line we have the mean percentage error made by the model on the predictions that as we can see is inside of the accepted threshold and also is really close to zero, which in fact is not the reality and we can see it in the graph.

Lastly, in the bottom we have the number of correct predictions over the total of data used to predict, which is 0 percentage, this time the model did not even predict one elapsed time within the threshold range that we have set up. There are two main reasons behind this result being so bad, and that is because there is a really low amount of data for the model to be trained with and also we have to add it up that the data is really sparse as the split is too wide and it includes a high range of values.

	Whole Data	**0-15**	**15-30**	**30-45**	**45-60**	**60-90**	**90-120**	**120+**
Mean Error	0,0006614173228	< 0	-0,0019	-0,00019	-0,0020	-0,0017	0,0119	-0,883
Mean Error Percentage	28,19	2,05	0,64	0,86	-0,0660	0,993	1,55	30,569

Figure 45. Comparison between whole data model vs tiers

Taking a look at Figure 45 we can quickly observe the differences in the mean errors for the different models we have been building. There are important differences especially in the error percentage value, which we can see is much higher in the case where the whole dataset is being used. With those values at hand, we can then confirm that building specific models for each of the tiers is worth the trouble and the amount of accuracy being gained, even if it is on average, it is huge. The reason behind those values being smaller compared to the ones from the whole dataset, is that the data on each of these splits is even more similar to each other and thus the model can generalise better. The data is also less sparse which is helpful in the process of generalisation. These results were expected, not only because the data was similar to each other in each of the partitions made on the dataset, but also because the amount of values on every split is smaller and thus makes all the process simpler. Saying the process is simpler does not mean that the computational process is lower but as the amount of data being processed is lower during the process of learning, the model is able to go through the whole dataset with less cost as if it was doing it with the whole dataset.

Even if the splits are working well, there is an exception though. The last one that contains data for elapsed time values higher than 120 minutes is behaving worse even that the model with all the dataset. The reason of why this is happening is really simple, less data is good but not always, there are some boundaries that upon reached make the model impossible to generalise on data and then behave worse. Here we have an example of one of those boundaries reached, the split consists of only 12 samples of data which is a really low amount and it is impossible then for the algorithm to be able to generalise. Also the data is very sparse as the split it is sparse too. Values of more than 120 minutes can mean activities that lasted 121 minutes or that lasted 300 minutes which is a very big difference.

5. Summary of the results

As a final recap and before heading to the final conclusions for the project, let's do an overview of the performance metrics for all the prediction techniques used throughout the project. Here we are going to focus on the mean error percentage, to compare between each other.

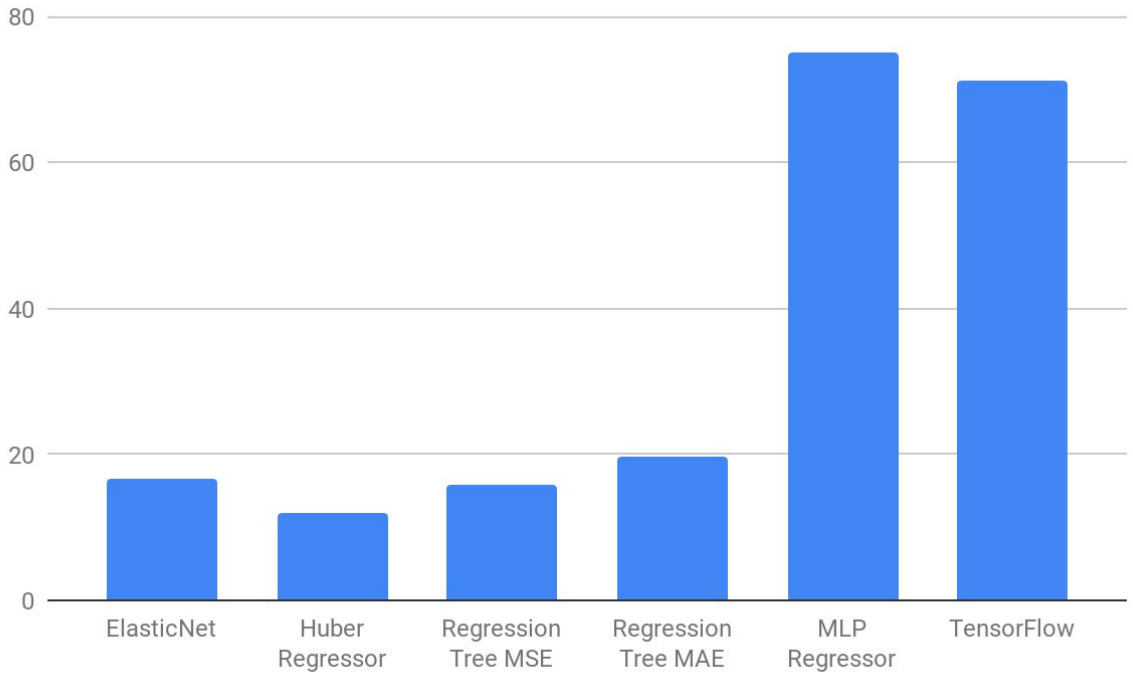

Figure 46. Mean percentage error using the whole dataset

Figure 46 shows the mean percentage error values made by each of the prediction techniques explored within the project. As we can appreciate, the lowest error is yielded when using the Huber Regressor, followed by the ElasticNet and the Regression Tree using Mean Squared Error that are almost even to each other. As the worst performant we have the MLP regressor with an error close to 80% which is huge and thus we will straight discard it. It is worth noting that, our models using Tensorflow barely improve the performance of out-of-the-shelf MLP classifiers. However, they are still not satisfactory for the productivization of the results.

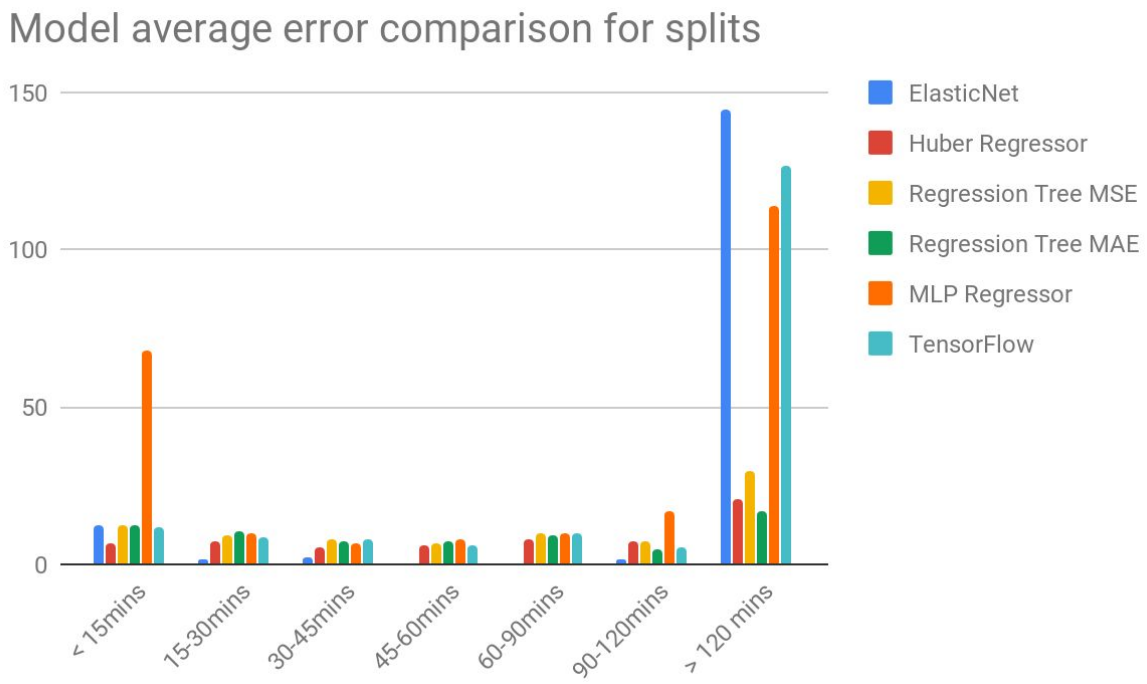

Figure 47. Mean percentage error for every model for each split

Figure 48. Model average percentage weighted error for each split

Figure 47 shows the performance each model is having when taking a look at every specific split. At first sight, we can easily appreciate that the last split, the one corresponding to

activities longer than 120 minutes, it is getting bad performance overall so it is not a good split to do a final choice on which model we should tag as best performant.
Now, if we look at the rest of splits, it is obvious that all of them got lower with the usage of these partitions. We can agree from here that the way to follow is this one, as specific models for partitions containing similar data is behaving much better.
Having to decide which one performs better overall, taking a look at the graph from the figure, we can see that the ElasticNet regression model is the one that yields the lowest error percentage overall. Is it worth noting that the rest of the models are behaving almost equally with the exception of the first and the last splits in which we appreciate some disparities.

Figure 48 shows the performance each model is having according to each split but now with the errors balanced by the weights of every split. But what does that mean, exactly? As we may expect, our data is not equally distributed along the splits, which is normal as some type of activities like the very short ones or the largest ones are rarer in the dataset. That implies that some splits are going to have more data both on training dataset and test dataset. In order to correct that and have a better idea on how the models really perform, we have calculated the weight of each split compared to the total amount of data contained in the dataset. Applying that weight to the performance metrics, it yields the graph from the figure.
Now it is easier to grasp an idea on prediction technique's accuracy. In any case, the results are the same as we have got from analyzing Figure 47, the best overall performing model is the ElasticNet regressor. After applying the weights we can see clearer the gap between this prediction technique and the rest which clearly states that it is the best and therefore the choice to go in this first iteration of our investigation.

6. Discussion

With all the data we have been able to gather and all the statistics retrieved from each of the experiments and also for each of the different prediction techniques applied, we now can discuss over the results and give a final decision on which of them is worth keeping for business purposes due to the results obtained. Let's now do a brief discussion for each experiment before jumping to the final conclusion on which one to choose as the .

Regarding experiment 1, the initial explorations made on the whole dataset were not that good as the average error on the outputs was way too high and looking at it from a business perspective, they did not have any value because of that deviation. With these results on hand, and the afterwards decision on splitting the data in tiers, the percentage of error lowered considerably and these values also reached levels close to the threshold set up that we could also consider as business acceptable metrics. After that change and taking a look at the statistics and performance metrics for each of the prediction techniques, we can conclude that the Elastic Net regressor is the one that has better performance overall. The average percentage error overall it is lower than the others and the maximum and minimum errors it is making are not too distant from the validation values. Being discussed all of this, Elastic Net would then be the choice to go in a business project based only on Sklearn-based prediction techniques.

As for experiment 2 concerns, and taking a look at the statistics and performance metrics, we can also conclude that using tiers when predicting is the best option in order to train an effective model that works well overall, as the mean error throughout the splits is much lower than the one we are getting using the whole dataset. It is a bit surprising though, that the deep neural network model is not overtaking experiment 1 results. But is also a well-known fact that deep learning prediction models need more data in order to be more effective, and in our case the whole dataset it is relatively small and when performing the partitions according to every tier we defined, each of them gets only a small amount of data. The splits containing less data also difficults the model from generalising properly and also the fact that the splits are not distributed equally. That means that not all of them have the same quantity of values in their respective dataset, which in an idyllic situation all of them will have the same size and data distribution will be standardized to have the model easily generalise about the data that is receiving.

Now that we have our best performing prediction technique from Experiment 1, we can go ahead and compare it with Experiment 2. Even though the gap between them on accuracy

for each of the splits it is no really big, we can assure that the ElasticNet regressor has the best performance metrics overall despite the accuracy being good but far from perfect. Taking into consideration that the amount of data is small, we can consider the results to be good enough, and it is also a good sign because the model has room to improve with the addition of more activities which will make all the datasets grow.

After exploring with experiments 1 and 2 and looking at the performance metrics for each of them, it is worth noting that there is one split that is not yielding good results for any of the experiments. That is the case for activities higher or equal to 120 minutes. There are two main issues with the split that are making it fail: the amount of data is low and it is sparse. These two factors combined make the split completely unviable for the project's investigation. Taking that into account, we can state that the split is not useful for the investigation even though it may be in the future with more data added.

7. Project conclusion

What started as a hypothesis about something that could work, and mostly based on a generic idea on how to solve the problem and do so within a business perspective, it ended up being an actual project that has business potential based on results obtained which are really promising.

After the two experiments made with two different approaches on how to predict data with the dataset that was gathered using the Strava API, the viability has been proved true with this first approach. Coming from an almost no knowledge about the topic of machine learning and deep learning, also made me learn a lot of insight and knowledge about the techniques involved and the whole process. Also executing a full pipeline starting from gathering the data, having to clean and prepare it to reach the model to then end up being able to predict valuable outputs with it, has also made me gain experience on what are the requirements that a machine learning or deep learning model needs and the different techniques to use when it comes to data curation and preprocessing. Those specific requirements can vary from model to model, but the general idea is that getting good insight on what data do you have, how do you clean and structure it is one of the key points in order to succeed.
Getting to experiment with different prediction techniques, going to higher to lower level ones, when it comes to coding, and comparing them also made easier the process of learning while trying to build the models.

As future possible extensions to the idea behind this project, we will have mainly the implementation of a web page or application so the investigation and the results could be used by users in a real case scenario. As for the model training and data used to train and validate the predictions, a part of adding lots of new activities, which is obvious that is needed in order to drastically improve the accuracy of the model, could be useful to take into account other external parameters such as the weather the day the activity was done or even recommending resting points within a route along places to eat or sleep.

8. Bibliography

[1] Cristopher Bishop. Pattern Recognition and Machine Learning. Springer Ed. 2006.

[2] Diederik P. Kingma, Jimmy Lei Ba, Adam: A Method For Stochastic Optimization. At Int. Conf. Learning Representations, 2015.

[3] Elastic Net. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.htm. Last visited on 24th June 2019

[4] Huber Regressor. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html. Last visited on 23th June 2019

[5] Keras, Retrieved from https://keras.io. Last visited on 16th March 2019

[6] MLP Regressor. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html. Last visited on 24th June 2019

[7] Regression Tree. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html. Last visited on 23th June 2019

[8] Strava. Retrieved from www.strava.com. Last visited on 22nd December 2018.

[9] Strava Activity Model Detailed, Retrieved from https://developers.strava.com/docs/reference/#api-models-DetailedActivity. Last visited on 18th June 2019.

[10] Strava API documentation. Retrieved from https://developers.strava.com/. Last visited on 22nd December 2018.

[11] Tensorflow. Retrieved from https://www.tensorflow.org. Last visited on 16th March 2019

9. Annex

9.1 Data concession

Participación

Su participación como voluntario, será requerida de manera directa sólamente durante la creación del token para generar acceso a los datos, tiempo que durará como máximo 15 minutos. Después de manera indirecta, se participará durante 2 o 3 días (tiempo que dura el acceso a los datos) para la recogida de datos a través de la aplicación. Su participación es voluntaria y sin ánimo de lucro. No habrá compensación económica o retribución de otro tipo ni para el investigador ni para los voluntarios involucrados.

Uso de los datos

Los datos extraídos serán utilizados únicamente con propósitos científicos, y nunca serán publicados ni divulgados fuera del investigador involucrado en el proyecto. El voluntario/tutor legal podrá tener acceso a sus datos siempre que así lo desee. La información extraída podrá ser eliminada siempre que el voluntario/tutor legal así lo solicite.

Protección de datos

Los datos serán recogidos de acuerdo a las bases de este formulario de consentimiento y la única información que se hará pública será el resultado científico del análisis, el cual será totalmente anónimo respecto a los participantes de los datos recogidos. No habrá ninguna forma de poder identificar a los usuarios participantes en los datos extraídos a partir de los resultados científicos mostrados.

Información de contacto

Para más detalles sobre los datos extraídos(eliminación , consulta..) por favor contacte con el investigador del proyecto , Borja Hidalgo Toca al mail: bhidalto12@alumnes.ub.edu , o a la Secretaría de la Facultat de Matemàtiques de la Universitat de Barcelona, Gran Via de les Corts Catalanes 585, 08007, Barcelona.