



University of Pisa

Department of Computer Science

Data Mining[DM-INF]

Cycling, cycling, and more cycling

Data Mining final assessment report

CANDIDATES

**Mirko Michele D'Angelo
Filippo Morelli
Davide Orsucci**

Report Data Mining 24-25

Filippo Morelli, Mirko D'Angelo, Davide Orsucci

October 2024

Contents

| | |
|--|-----------|
| Task 1: Data Understanding | 1 |
| 1.1 Data quality assessment | 1 |
| 1.2 Races data quality assessment | 1 |
| 1.3 Races imputation | 3 |
| 1.4 Cyclist data quality assessment | 4 |
| 1.5 Feature distribution analysis and relationships between features | 5 |
| Task 2: Data Transformation | 6 |
| 2.1 Feature engineering | 6 |
| 2.2 Engineered features | 6 |
| 2.3 New features analysis | 7 |
| 2.4 New features | 8 |
| 2.5 Revamped data understanding | 8 |
| 2.6 Enhanced Analysis | 8 |
| 2.7 Outlier Detection | 10 |
| Task 3: Clustering | 11 |
| 3.1 DBSCAN | 11 |
| 3.2 OPTICS | 14 |
| 3.3 kmeans | 14 |
| 3.4 Hierarchical Clustering | 15 |
| Task 4: Prediction | 17 |
| 4.1 Base Classifiers | 18 |
| 4.2 Neural networks | 18 |
| 4.3 Bagging models | 19 |
| 4.4 Boosting Methods | 19 |
| 4.5 Model comparisons & conclusions | 20 |
| Task 5: Explanation | 20 |
| 5.1 XGboost | 20 |
| 5.2 Random Forest | 22 |

Task 1: Data Understanding

1.1 Data quality assessment

For this part we are gonna analyze the quality of the data first by analyzing the two dataset separately.

1.2 Races data quality assessment

A first necessary analysis is to see how many values are missing from the dataset itself, to do so we count for each feature the missing values and report percentage w.r.t. the dataset size. The results are depicted in Figure 1.

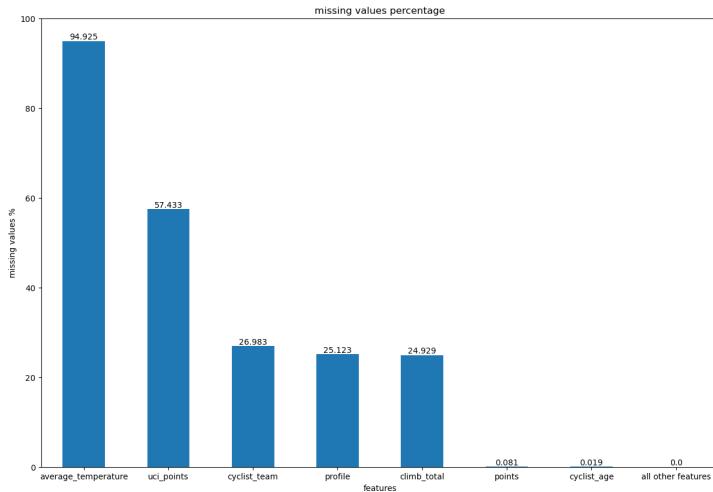


Figure 1: Distribution of the missing values per features in percentage.

Some features miss almost all the data:

- the most missing value is the *average_temperature*, which is almost entirely missing.
- *uci_points* with half the data missing.
- *cyclist_team, profile, climb_total* with ~25% of missing values.
- finally, *points* and *cyclist_age* have less than 1% of missing values so they seem to be very reliable.

1.2.1 Features domain values analysis

So first we did a broad understand of the missing values now we can instead analyse among the present ones if, according to the domain, there are values outside their natural range. To avoid being overly verbose we report a table with the ranges expected on the numerical values and we only do considerations on values actually outside their range (Figure 2).

So this part takes also into consideration the domain and not only the raw data, in this case there some considerations to make:

- the *profile* has 5 values but in the description only 4 are present this issue is handled in the second part.
- as for the other numerical features we don't have much to say aside from *delta* that given the description should be non-negative but it isn't, this indicates noise in the feature.

As for the other types of features we have date-times for the **date** going from 1970-02-28 04:52:00 to 2023-07-29 05:52:14 which is correct.

In the dataset description we have the **is_X** parameters, they are a binary feature symbolizing the kind of terrain for the stage. We assume to not have mixed terrain types and to have at least one terrain type, in this case we can see if such assumption is respected by counting for each instance in the data if there are more than 1 flag, from our analysis we also found out that there are no entries with multiple flags. As we can see in Figure 3 some entries don't have flags at all which is not necessarily an error as it could be a terrain of different type, however this indicates a noisy or poorly informative feature.

For the **name** there are a lot of races that are repeated with spelling errors, slightly different notations or sometimes have their racing category specified and sometimes not. We can extract some information from them using text normalization techniques but a more sound approach at least to perform other kind of analysis is to use the values that can be extracted from the **url** field.

The **url** field always has the same structure, it is a string composed by **race name race-year**, this is reliable as an identifier for races and can also be used to infer other informations.

| Feature name | Type and Natural range | Values outside natural range |
|---------------------------------|--------------------------|------------------------------|
| startlist_quality | continuous $(0, \infty)$ | None |
| average_temperature | continuous $(0, \infty)$ | None |
| delta | continuous $(0, \infty)$ | 86 |
| points | continuous $(0, \infty)$ | None |
| uci_points | continuous $(0, \infty)$ | None |
| length | continuous $(0, \infty)$ | None |
| climb_total | continuous $(0, \infty)$ | None |
| profile | $\{1, 2, 3, 4, 5\}$ | None |
| position | discrete $(0, \infty)$ | None |
| cyclist_age | discrete $[13, 60]$ | None |
| is_tarmac, is_cobble, is_gravel | binary $\{0, 1\}$ | None |

Figure 2: Features of the races analysis with their value type, value ranges, and values outside the natural range.

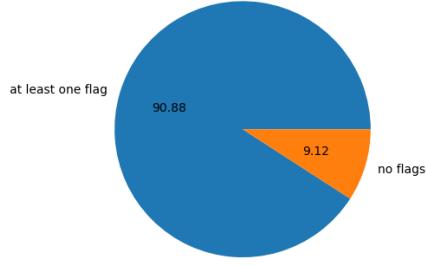


Figure 3: distribution of is_X entries with no flags.

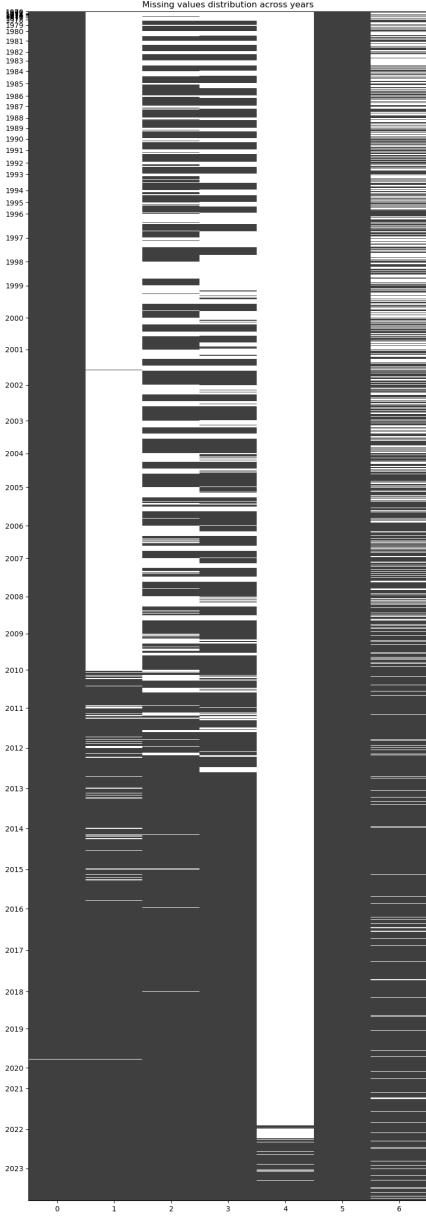


Figure 4: Missing values across years.

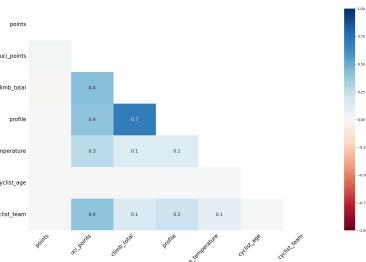


Figure 5: Missing values correlations.

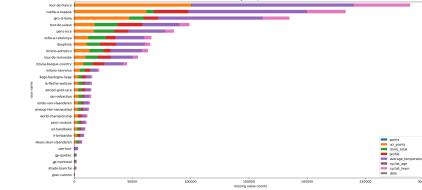


Figure 6: Missing values for the different races.

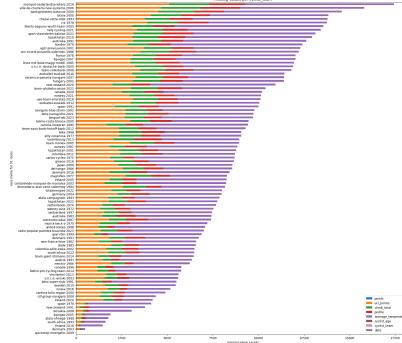


Figure 7: Missing values per cyclist teams.

1.2.2 Missing values across time

Given the time-related nature of the dataset it seems correct to analyze how values are missing across time, since the dataset is very big we observe the missing values across years in a coarse grained manner.

To display such analysis we plot the missing values has black rectangles as shown in Fig. 4. In the plot the black rectangles mean that data in that timespan are present for that feature, the white space represent the absence.

In the graph we can first infer the data size for each year, as years come close to the present we can see that the data size for those years increases.

- The **average_temperature** is entirely missing until 2022 meaning that it is practically useless unless we restrict our analysis there.
- The **uci_points** are also entirely missing until 2010, it could work using imputation but it seems to be heavily correlated to points but with a different kind of point values, probably it follows the rules from the uci federation[4].
- The **cyclist_team** has missing values across years with more or less density in missing values.
- **cyclist_age** and **points** have very few missing values with no particular density in missing values.

We can see from the figure 6 that some features either appear or miss together. To understand better if there is some correlation for this behaviour we plot missing value heatmap (Fig. 5).

In the image we can see a heatmap showing the missing-correlation values for the features, it is closer to 1 when two values are almost always present or missing together, it's close to -1 when one value is present when the other is not and is close to 0 when their presence/absence is not correlated in any case, see [1] to understand how the heatmap is calculated.

1.2.3 Races missing values

Another interesting analysis could be to see, between the various races (Fig. 6, ignoring the year, how the missing values are distributed.

In the image we can see the missing values distribution, the plot is NOT normalized meaning this is just a quantitative plot and is influenced by the number of values per races, hence a race that has occurred more or has a lot of participants will have a longer bar. We can see that we have well defined scale:

- the most famous tours are all over 100,000 missing values
- then we have more local races
- finally other less known races have all more or less the same number of missing value.

We also don't see any surprise in the distribution for the specific type of feature, they all more or less have the same distribution for each race and are very similar to the singular distributions we displayed earlier.

1.2.4 Cyclists teams missing values

It is also reasonable to see if there are teams with too many missing features for their members 7. Here the consideration to do is the same as before we have more or less the same distribution per team.

This distribution is not really informative of teams of cyclists, so to further enhance our view on the data we can restrict to only the cyclists age.

1.3 Races imputation

After analyzing the data quality of the races, we can start with imputing the missing values on the features.

First we can start with feature with a very low number of missing values, in this case we assign the nationality as the team they come from since manually checking data is still unfeasible.

As for the profile given the underlying hidden logic behind the points it sounds reasonable to employ machine learning in this case. Given the length of the process a subsection is due in this case.

For the temperature clearly trying the imputation is not possible in any way, we could try to gather some external data using [3], the idea is to employ web scraping and gather data from the website treating as ground truth, we tried to see if among the values the missing ones are available however we did not manage to find anything useful, other tests included using the augmented geo data gathered from the website and use it in combination with specialized APIs to reconstruct the temperature but it was not possible given the limited precision and capabilities of such APIs.

1.3.1 The profile and climb_total imputation process

The data was split into a development set (where profile is present) and a test set (where profile is missing). The development set was further stratified into training (80%) and validation (20%) sets for a hold-out training regime.

We first tested a linear regression model that resulted in poor performance, achieving an F1-score of 0.48 on both the training and validation sets.

Given the poor performance of the baseline model, we switched to a more sophisticated approach: KNN. This model was tested using a grid search with varying hyperparameters:

- Number of neighbors: 5, 10, 20
- Distance metrics: Euclidean, Lasso, Cosine similarity

KNN achieved an F1-score of 0.97 on both the training and validation sets, demonstrating much better performance compared to the baseline.

Next, we tested a random forest model, known for its ability to handle missing and noisy data well. Hyperparameter tuning was performed using a grid search on:

- Number of estimators: 100, 200, 500
- Maximum depth: 10, 20, 30
- Minimum samples split: 2, 5, 10
- Maximum samples per leaf: 2, 5, 10

The random forest also achieved an F1-score of 0.97, with similar performance to KNN. Given that random forests tend to handle noisy data better, it was selected as the model of choice.

However, upon applying the trained random forest to the test set, we observed that some strong correlations were disrupted, particularly for the profile feature. As a result, we decided to switch to a final imputation method using mean imputation. For climb total, we performed mean imputation, and for the profile, we assigned the profile of the entry with the closest climb total to the calculated mean total. Such method is quite effective and manages to maintain the correlations

1.4 Cyclist data quality assessment

The dataset involved has 6134 elements and 6 features. The key to the dataset is ‘`_url`’, which summarizes the information in the column ‘`name`’. Below we present a summary of the columns and their associated types.

| Feature name | Original Type | Real Type |
|--|--------------------------------|--------------------------------|
| <code>_url</code> , <code>name</code> , <code>nationality</code> | Object (<code>String</code>) | Object (<code>String</code>) |
| <code>birth_year</code> | Float (0, ∞) | Integer (0, ∞) |
| <code>height</code> , <code>weight</code> | Float (0, ∞) | Float (0, ∞) |

Table 1: Features of the cyclists analysis

For the cyclist dataset, we performed a similar cleaning and assessment task. Initially, we checked for wrong data, first by type and then by missing values. The following list contains a summary of the first analysis.

- **url, name:** 0 missing values
- **birth year :** 13 missing values
- **nationality:** 1 missing value
- **height:** 2991 missing values
- **weight:** 3056 missing values

With regard to the validity of the entered countries, we relied on the `pycountry` library and since there are no decimal elements for the height column (in centimetres), we can conclude that it represents whole numbers.

1.4.1 Data Duplicates

Duplicates are searched by name and url. If for the latter we do not have identical rows, we can find homonyms, these are handled for conversion into urls in different ways. In some cases a number is added to the bottom of the url, in others ‘name-surname-alphanumeric code’ in all cases as the url is the key to the dataset, also used to identify the cyclist in the race dataset, we find no relevant problems.

1.4.2 Missing Value Correlation

Regarding the correlation between the missing data, we found that most of the cases in which height is missing are also missing weight (2984 cases in which it is missing in both columns and 79 otherwise) so the analyses can only be performed on one column. The nations with the most missing heights are Italy, Belgium and France, which are however the most represented nations, once the missing ratio is displayed for each nation we can see that Liechtenstein, Yugoslavia and Zimbabwe are the first with a missing ratio of 100 per cent while Italy manages to maintain a high position

The same analysis was performed for the date of birth and again a predictable trend can be seen, the missing_ratio only decreases with the years reaching a new local peak around 2004, but in this case novice cyclists are represented and therefore it can be assumed that the data in question have not yet been collected.

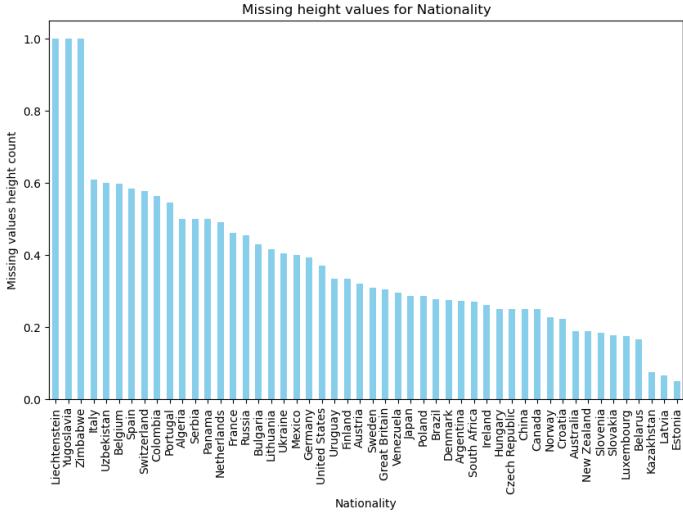


Figure 8: Missing Ratio of Height feature in relation to Nations.

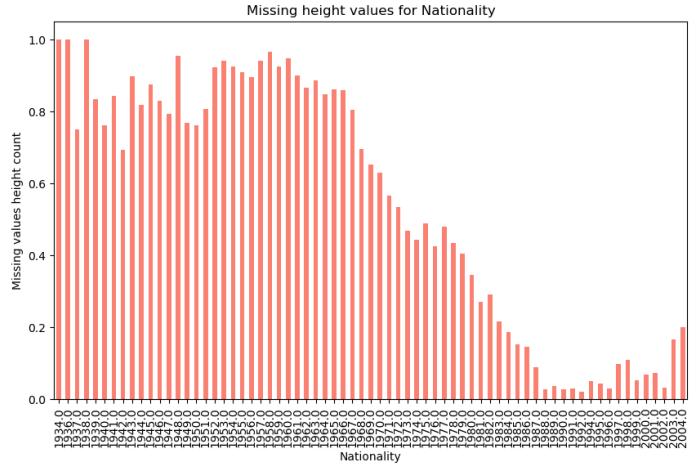


Figure 9: Missing Ratio of Height feature in relation to Birth Year.

1.4.3 Data Fixing

In this case we performed 2 operations, first we removed the outliers for the weight and height columns (19 and 22 respectively) and then we filled in the rows that contained missing values. Specifically, the following list summarises the techniques used:

- **Nationality** (1 missing) : manual replacement thanks to procyclingstats.com
- **Birth Year** (13 missing) : manual substitution thanks to procyclingstats.com and following substitution with more frequent year for cyclists not found (7 missing)
- **Height-Weight** (2993 missing) : here we used random sampling on the distribution of columns with both features present. The sampling is simultaneous to maintain the high correlation between the two features.

1.5 Feature distribution analysis and relationships between features

After assessing the quality and filling the missing values, we looked at the distributions of the features.

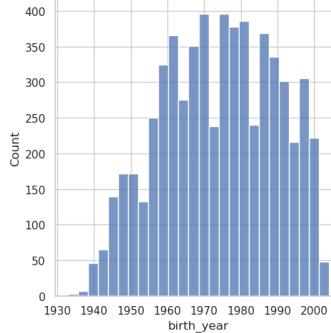


Figure 10: distribution of the birth years

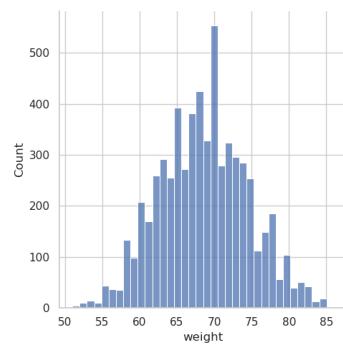


Figure 11: distribution of the weight

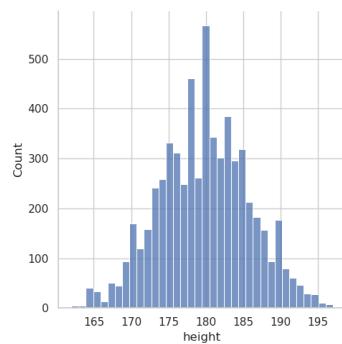


Figure 12: distribution of the height

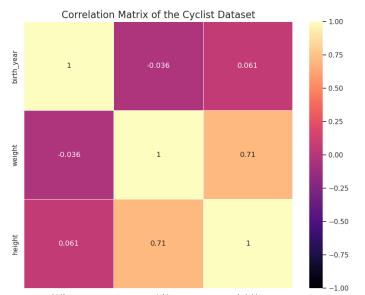


Figure 13: Heatmap of the cyclists feature in detail, calculated using Pearson's correlation.

for the *birth year* it only relevant to say that most of the cyclists are born between the years 1960 and 1980. More interestingly the *weight* and *height* features follow a gaussian distribution and have a similar shape. This is interesting it hints at a correlation which is confirmed the heatmap, on the figure 13, plotted on the features *birth_year*, *weight* and *height*. In particular, we noticed that there is a strong correlation of 0.71 between the *weight* and *height*.

About the races, we used a different approach: first we aggregated the races in the dataset by their *_url*, then we plotted the distributions of the features related to the races. (Fig. 14, 15, 16)

Regarding the profile, we found out that the *profile* feature has the percentages showed in 17. Again, we used a heatmap to discover hidden correlations between the features and we found out that *climb_total* and *profile* have a high correlation, and so we filled the missing values to maintain the same correlation between the two. Also, the heatmap shows a slight positive correlation between *length* and *climb_total* and between *points* and *startlist_quality*. Since *profile* is ordinal feature, we preferred to use Spearman method to calculate the correlation this time.

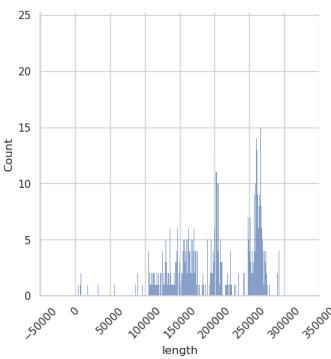


Figure 14: length distribution

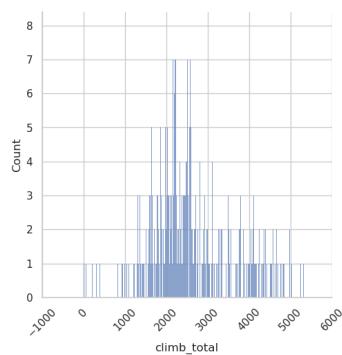


Figure 15: climb_total distribution

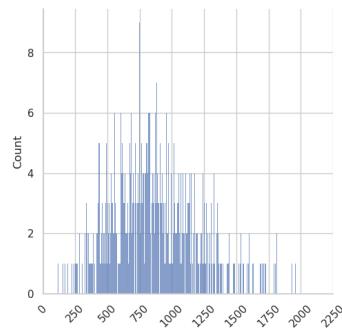


Figure 16: startlist_quality distribution

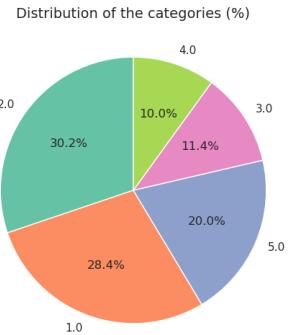


Figure 17: percentage of the profiles across the different races

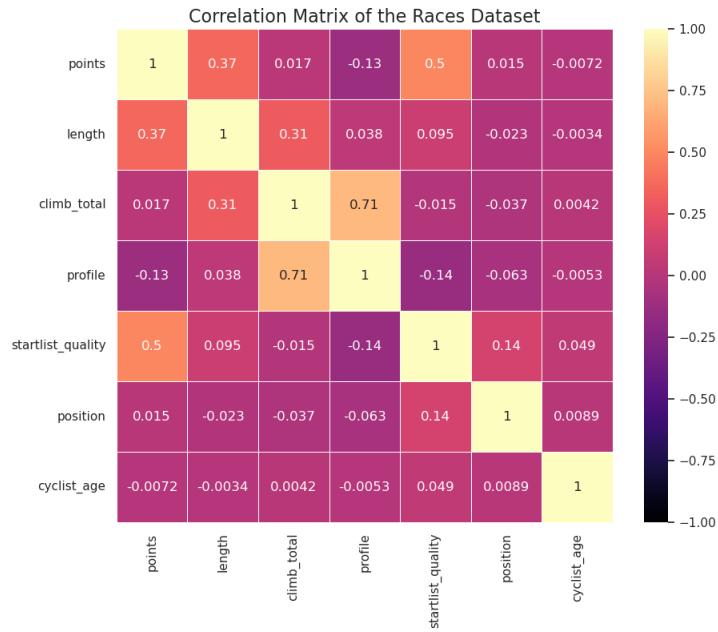


Figure 18: correlations between the races features using Spearman

Task 2: Data Transformation

This section takes care of dataset preparation for clustering using feature engineering, outlier detection, and data cleaning.

2.1 Feature engineering

Before starting the feature engineering we have to do the following consideration.

In particular we have the profile, which takes five values with an unknown mapping to the ones described in the dataset. To understand their meaning, we leverage the correlation between profile values and **climb_total**. Using Sturges' rule to bin climb totals, we infer the meaning of profile values based on their distribution across bins.

Categories 1 and 2 are mapped to "hilly," 3 to "flat," 4 to "mountainous," and 5 to "high mountains," based on their distribution across climb total bins. This mapping will be used to assign difficulty levels in the analysis.

2.2 Engineered features

This section showcases the new fetures we added, for some we also put a brief explanation to understand better their purpose.

2.2.1 Races features

competitive_age = age of the partecipant at the time the race occurred.

$$\text{climbing_efficiency} = \text{climb_total}(m) / (\text{length}(KM) \times 1000)$$

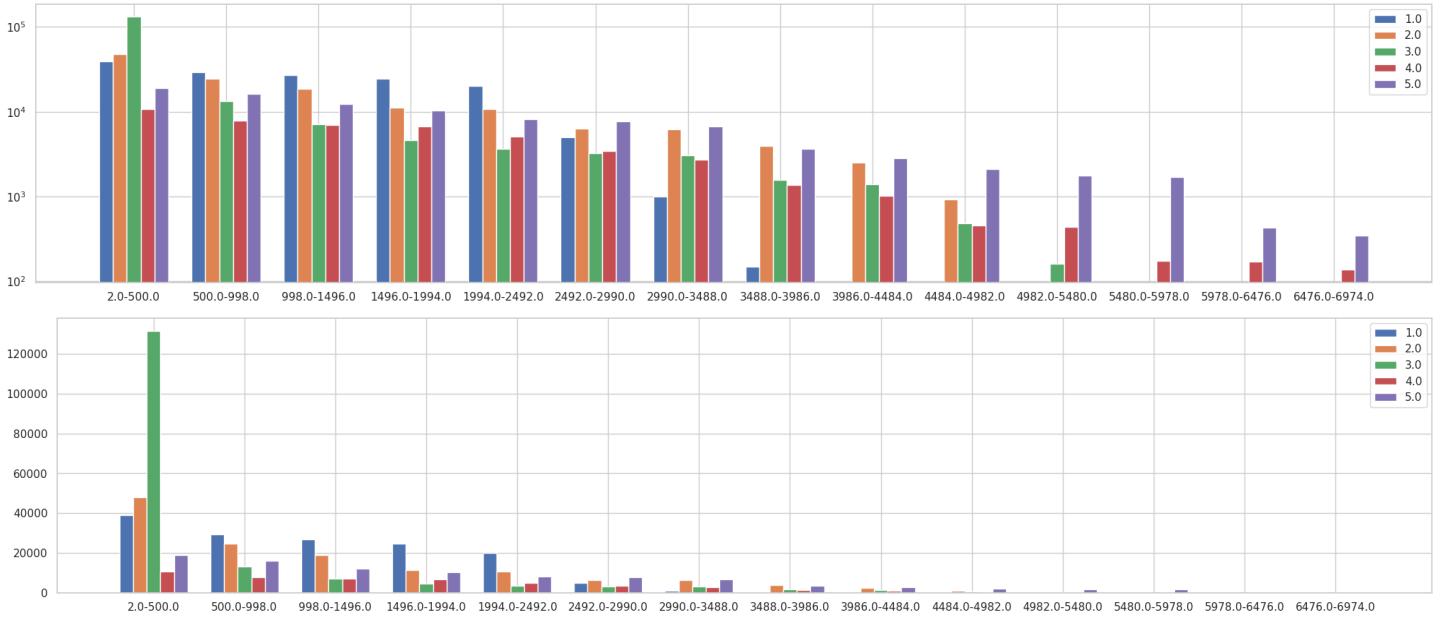


Figure 19: Binnining of the climb_total values. Above we have the normal scale and below a log scale to see better some differences for particularly low values

It captures how much steepness you can encounter for a race, a short race with a high climb_total will be very steep, hence high climbing_efficiency , but with a very long length it is lower with the same climb_total.

$$\text{terrain_score} = \sum_i \frac{(\# \text{of times terrain } i \text{ appears in the race})}{(\# \text{of stages for the race})} * (\text{terrain } i \text{ difficulty})$$

It's an estimate of the difficulty of a race using only the information from the profiles types we inferred in the previous analysis,it is weighted average of the difficulty for each stage based on the profile.

$$\text{difficulty} = \text{climbing_efficiency(race)} \text{terrain_score(race)} \text{BMI(cyclist)} \text{competitive_age(cyclist)}$$

This is an estimate of the difficulty of a race for a single cyclist, for each cyclist we have the climbing efficiency,the terrain score that depend only on the race to add also the dependency on the cyclist characteristics we add BMI and competitive age. The rationale is that a person that is older, overweight and deals with a difficult race has a lot more difficulty than another that is in shape, younger for the same race.

$$\text{convenience_score} = \text{points(cyclist)}/\text{difficulty(cyclist, age)}$$

It is used to estimate the convenience of participating in a race which depends on how much you gain and the estimated difficulty of a certain person.

$$\text{gain_ratio} = \text{points(race)}/\text{difficulty_score(race)}$$

It uses all features coming from a race to estimate the difficulty.

2.2.2 Cyclists features

- debut_year_i = First year of the career of the cyclist.
- career_points_i = all the points obtained by the cyclist through its career.
- $\text{career_duration_days}_i$ = Career duration of the cyclist in days.
- $\text{career_duration_races}_i$ = Number of entries for the cyclist i in the races dataset.
- avg_pos_i = average positioning of the cyclist i at the end of each race.

2.3 New features analysis

Given the introduction of new features we can do a second data understanding focusing on the new features and how they relate between the old and the new ones plus we can see the old features behave after the preprocessing done on the first task.

2.4 New features

In this first section we analyze how the new features interact with each other and the old ones.

Here we can see the heatmaps calculated using old and new features, as shown in the tables on Fig. 20 and 21. Aside from the lightly negative correlation between some features which are to be expected given some formulations of the engineered features we don't see any heavy correlation between new features, as for the comparison between the old and the new ones we see only correlations deriving from the formulations of the features using the old ones without any surprise.

We can also plot the correlation between the old features after applying the imputation and see that it didn't change much and correlations are more or less the same, as shown in figure 22.

So we can see that correlations don't show any relevant surprise after all aside from the expected ones and our imputation method managed to maintain the statistically relevant ones without altering them.

2.5 Revamped data understanding

After applying the imputation we can repeat a revamped data understanding. The distribution of data hasn't changed much for old features, now we can focus on analyzing the new features.

Given the long time span of the dataset we can try to analyze the behaviour of the new features on the races across various time scales, first we start with the most coarse grained one which is yearly, which is displayed on image 24.

If you want more fine grained details you can have a look at the monthly averaged plot in the notebook however we don't gain much more information, notably we have an increase in the years going from 1970 that slowly decrease after for the difficulty, this could also be related to having less data and thus more bias in the average yearly values for the first years as displayed in figure 6.

Given that analyzing graphically for a monthly average would be very difficult and also given the lot of noise we could get in such a signal it would also be very imprecise we can instead try to find any kind of recurring pattern using normalized self-correlation on the monthly average, after our tests we found a strange weak positive correlation of 0.37 on all the new features with a lag of 10 which means that every 10 months we have some kind of repetition, it could be due to training season for cyclists or it could also be determined by recurring weather conditions, please see the notebook to view the calculations.

Finally for each position we can plot the values sorted to observe if there are any interesting distributions of the values for continuous values. It is interesting to notice that we have delta, climbing_efficiency, difficulty, convenience_score and gain_ratio that are long-tailed distributions while climb_total also display a little flat line meaning we have a consistent part of the races having all the same climb_total value in a narrow range.

2.6 Enhanced Analysis

2.6.1 Time relationships

Once the new features have been added, we can perform new analyses concerning the relationship between the features present. We analysed how the year of debut is distributed in the dataset, not noticing any particular behaviour. As far as the age of debut is concerned, we can note a centring on 23 years, which decreases monotonically in both directions. On the other hand, a clear correlation can be found between career duration (in years for visual simplicity) and the number of average points accumulated, this is almost linear and shows a constant growth up to the age of 23, decreasing afterwards for the twenty-fourth. This is confirmation that a long career is synonymous with a good career (Fig.25).

The highest and most obvious correlation is found between the number of races performed and the total point count, where we find an almost totally linear graph (Fig. 26).

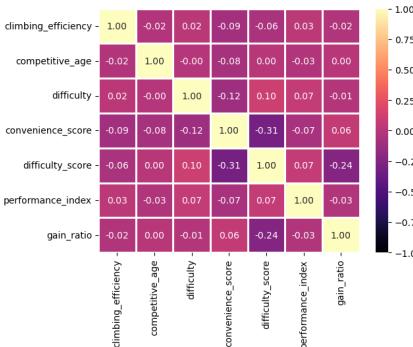


Figure 20: Correlation heatmap of the new features with each other.

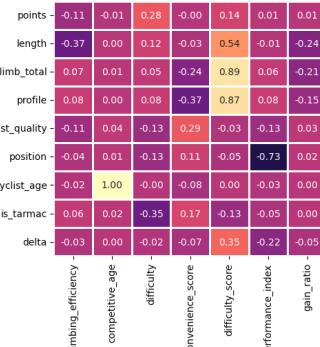


Figure 21: Correlation heatmap of the new features with the old ones.

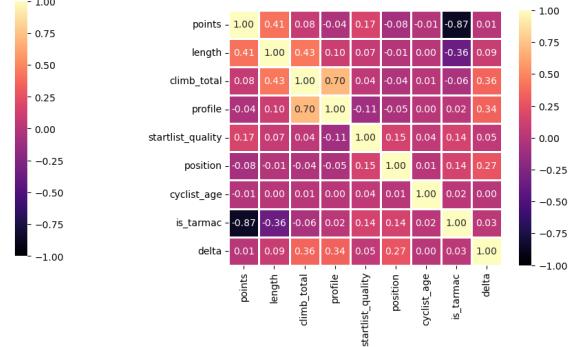


Figure 22: Correlation heatmap of the old features after imputation.

Figure 23: Correlation heatmaps comparing new features among themselves, with old features, and analyzing old features after imputation.

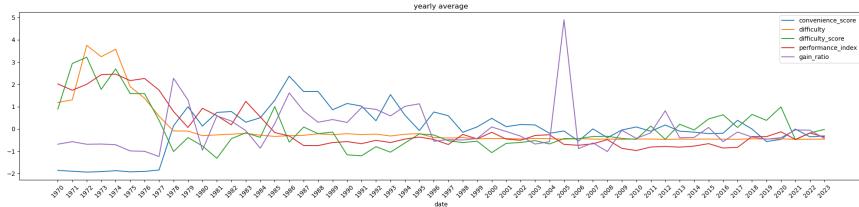


Figure 24: Yearly average of the new features across all dataset.

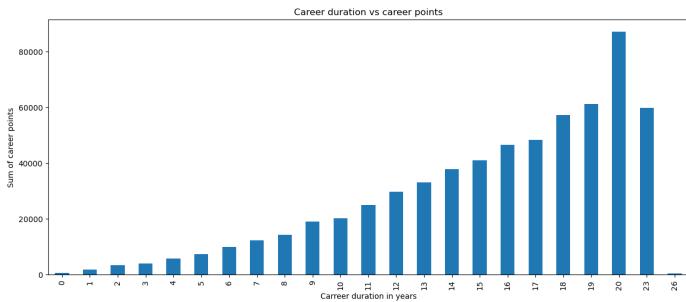
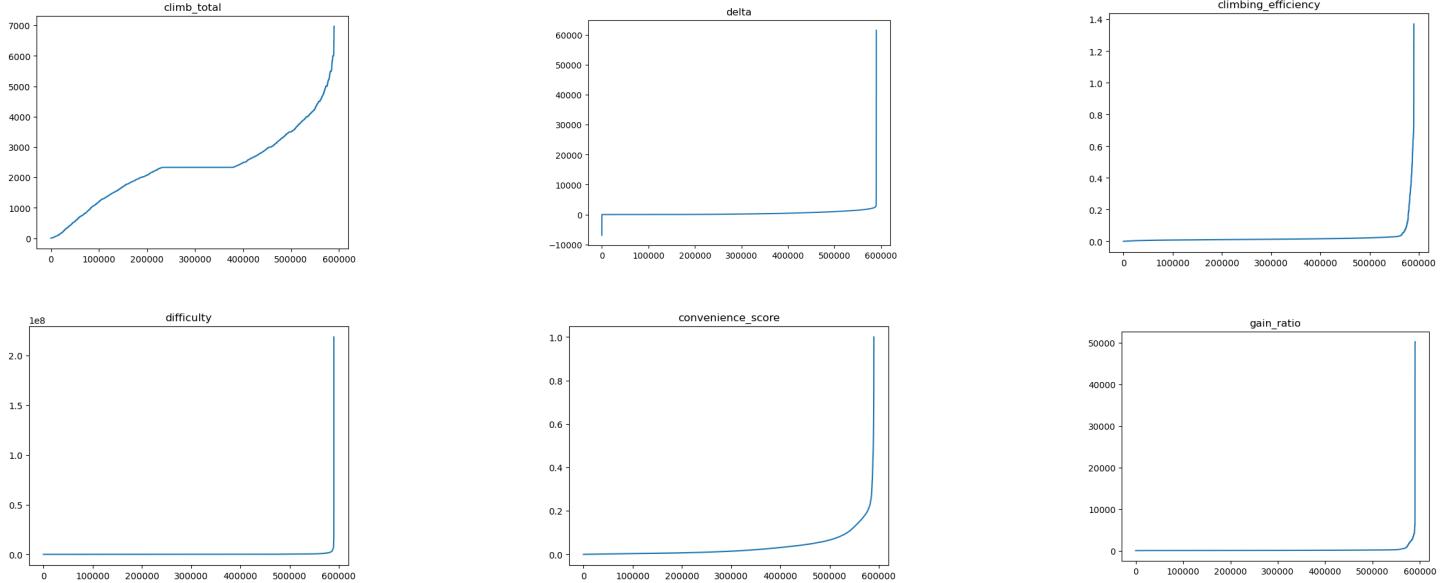


Figure 25: Career duration (days) in relation with Career Points.

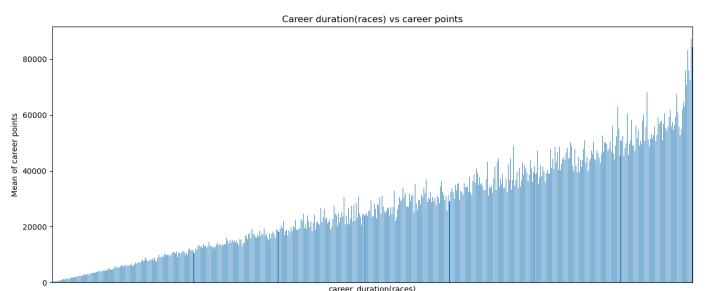


Figure 26: Number of Races and Points relationship.

2.6.2 Performance Considerations

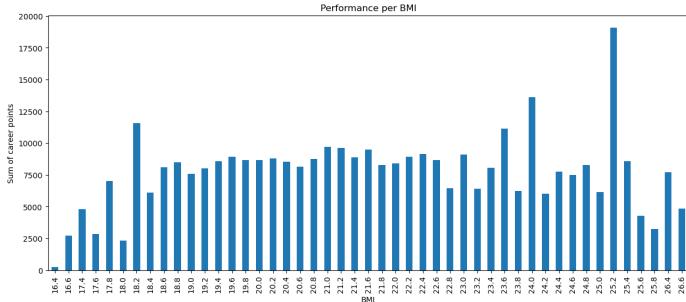


Figure 27: BMI in relation to points.

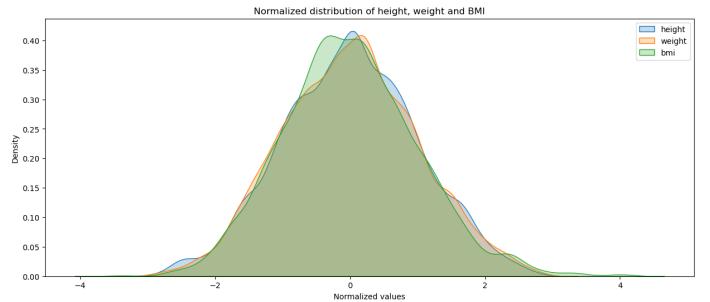


Figure 28: BMI, Weight, Height Distribution.

Analyses have also been carried out in relation to a cyclist's performance (always summarised by the sum of points) in relation to other physical characteristics. If we try to check the relationship between BMI and accumulated points, we do not find any kind of correlation with a monotonic graph throughout as shown in Fig. 27.

Given the high correlation between weight and height, we find no interesting differences to the conclusions made for BMI. A final study we performed is to check the difference in distribution between these three physical parameters that distinguish cyclists (Fig. 28).

As expected, the distributions are almost identical, with a wider centre for BMI indicating how much this parameter varies within the dataset.

2.6.3 General Correlation

To visualise the correlation between the columns of the dataset, we use a heatmap shown in figure 29.

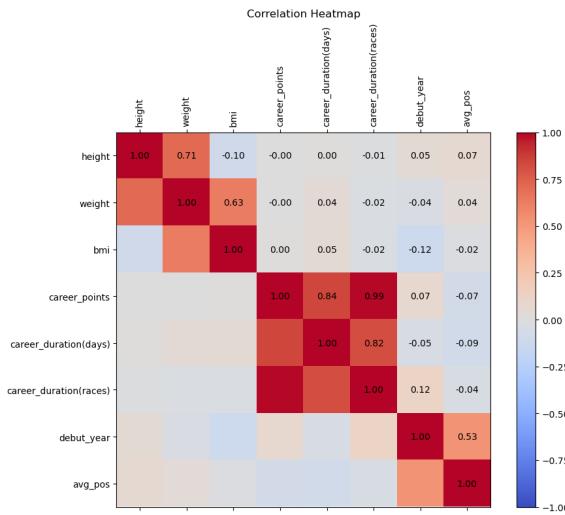


Figure 29: Correlation Heatmap for Cyclist Features

We can therefore see that the results highlighted above can also be found in this analysis. The highest correlation can be found between career length (in number of races) and the number of points, indicating how the number is even more important than career time as already anticipated by the graph 26. For index and height we find a negative one very close to zero. This can be easily explained by the formula itself where height is considered in metres, so it is a small value that has little or no influence on the result of the index, unlike weight, the numerator, and generally variable by the cyclists themselves through targeted training.

2.7 Outlier Detection

The outlier detection has been made one feature at once and by using mainly three algorithms: *Gaussian Mixture Model* (GMM), *Isolation Forest* (IF) and *Local Outlier Factor* (LOF). The decision between this three algorithms has been made exploiting the analysis of the distributions for each feature: for normal distribution we used GMM, otherwise we used LOF if we saw that the feature had some local anomaly in the distribution and IF in case of a large variability across the entire dataset. According to this, we divided the features in this way:

GMM was used for *weight*, *height*, *bmi*, *avg_pos*, *startlist_quality* and *cyclist_age* since they approximatively followed a normal distribution during the feature analysis. The graph of GMM shows on the X-axis the ordered indexes according to their probability of being part of the GMM clusters (showed on the Y-axis).

Here we noticed that while *bmi*, *avg_pos* and *startlist_quality* grow almost regularly with few minor jumps, *weight*30 and *height*31 show a change of pendency indicating the presence of a cluster of outliers at the beginning of the curve because of the low density degree. Also, *cyclist_age*32 presents a low density degree area in the first half of the graph, indicating a high probability of outliers.

Isolation Forest was used for *birth_year*, *debut_year*, *points*, *length*, *climb_total*, *position*, *delta*, *performance_index*, *climbing_efficiency*, *competitive_age*, *career_points*, *difficulty* because they do not follow a normal distribution and present a large variability. In this case the X-axis represent the indeces of the data sorted by their outlier degree (on the Y-axis). For *points* 33 the graph indicates that only a small portion of (the one at the beginning of the graph) is made by outliers. The *climb_total* feature shows a smooth transition from the outliers to the inliers 34, a similar behavior can be found in *delta*35. For all other features we found out that most of the points were inliers with a little presence of outliers.

LOF was used for *career_duration(days)* used **LOF** since it doesn't follow a normal distribution and so cyclists with unusually short or long career can be easily spotted. Here the X-axis represents the indeces of the cyclists sorted by LOF degree

(on the Y-axis). From the graph in 36 we understand that the presence of outliers is very low for this feature, however near the end there is a sharp increase of the curve meaning the presence of extreme outliers.

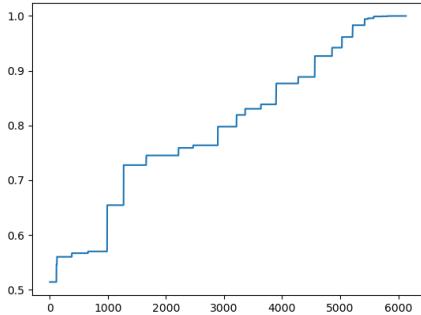


Figure 30: weight

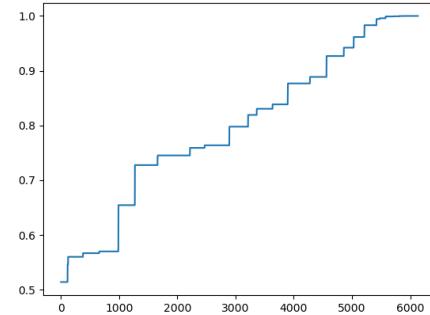


Figure 31: height

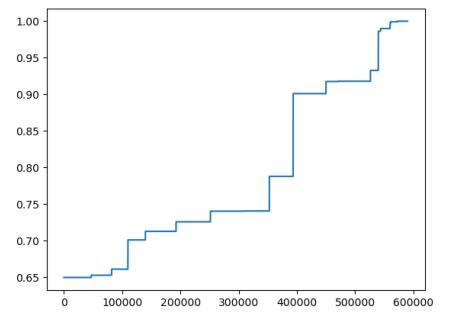


Figure 32: cyclist_age

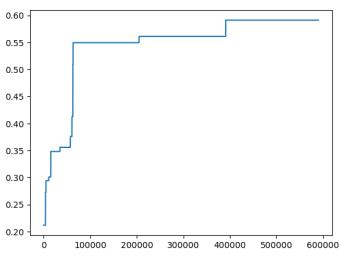


Figure 33: Points

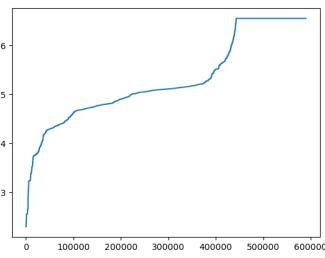


Figure 34: climb_total

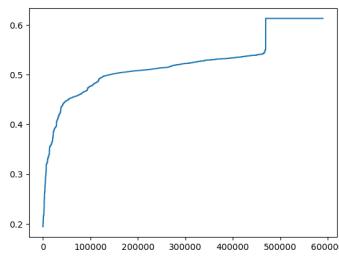


Figure 35: delta

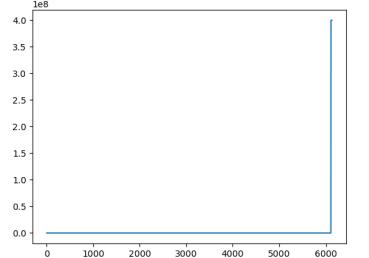


Figure 36: career_duration(days)

Task 3: Clustering

This section takes care of the clustering algorithms we applied and the consequential findings after using them.

3.1 DBSCAN

For the density based approach we performed a grid search, the difficult part of this algorithm is to setup the hyperparameters values used, we decided to employ an automated algorithm called kneed [2],which manages to automatically detect the point where the distance of the k-th nearest neighbor significantly increases to select the Eps value.

Given the the algorithm characteristics a lot of memory is needed, so to make feasible the execution we had to try methods like sampling and segmentation. The clustering was performed in different ways for the races, first a bulk clustering where we try all the dataset at once, then we try to segment using domain knowledge.

3.1.1 Bulk clustering

First we performed an automatic selection of the eps values using kneed, we set up the algorithm taking as starting minpts values various percentages on the size of the dataset.

| Eps | MinPts | % |
|------|--------|----|
| 3.56 | 208 | 1 |
| 3.78 | 415 | 2 |
| 5.44 | 6218 | 3 |
| 4.31 | 829 | 4 |
| 4.46 | 1037 | 5 |
| 4.62 | 1244 | 6 |
| 4.03 | 1451 | 7 |
| 4.85 | 1658 | 8 |
| 4.68 | 1866 | 9 |
| 4.79 | 2073 | 10 |

| Eps | Metric | MinPts | Silhouette |
|------|-----------|--------|------------|
| 3.56 | euclidean | 208 | 0.906246 |
| 3.78 | euclidean | 208 | 0.906246 |
| 4.03 | euclidean | 208 | 0.906246 |
| 4.31 | euclidean | 208 | 0.906246 |
| 4.46 | euclidean | 208 | 0.906246 |

Figure 38: Hyperparameter search results for DBSCAN.

Figure 37: Hyperparameter search results for DBSCAN. The percentage is on a scale from one to 100.

The values where used trying each combination for Eps-MinPts values from table 37 and we report the results on table 38 of the most performing by silhouette score.

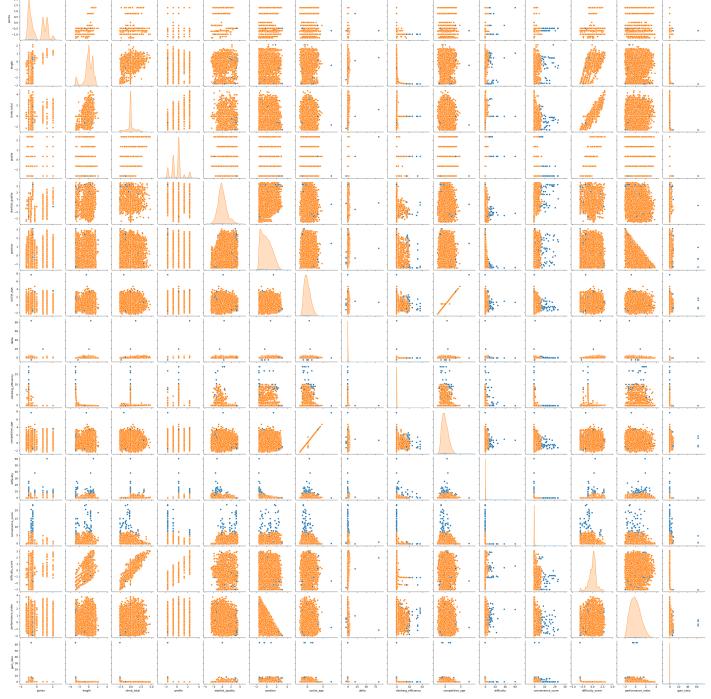


Figure 39: Pair plot of the values obtained from the bulk clustering on the races.

As we can see from table 38 the silhouette score is quite high, however this could be due to the sampling we used, further experiments could be done however since the density can vary a lot trying more could be unfeasible due to time constraints so we can try to use domain knowledge and use the algorithm on segments instead.

In the figure 39 we can see that most features have some level of noise with very few outliers except for difficulty_score and convenience_score which have a lot of noisy points correlated to them outside the main "blobs" for each feature the get compared to. The clusterings display the expected non globular shape. Some interesting informations are that length is higher when points are high while it has a lot of variability when points are lower but it can still have very high or low values also the length tends to be higher as the difficulty_score increases. climb.total displays a similar behavior with difficulty score. the difficulty_score The other features don't display any interesting information.

3.1.2 Segments clustering

The results are not very interesting probably due to high variability in density, we can pick some segments in order to reduce the size and possibility have more uniform density, for our case we grouped by profiles since similar profiles might have similar characteristics for races and performances.

We computed the minpts and eps values using the same logic as before but this time for each separated segment and performed the grid search in the same manner. After applying the method we selected values for each segment, during our tests not all profiles where easy to cluster and for some we did not manage to obtain any kind of cluster aside from the classical all noise or all core with no border thus we show only the most interesting results from the profiles that managed to get good clusters and report their values. From the table 2 we can infer some information, profile 2 and 4 might have a lot of

| Eps | Metric | MinPts | Silhouette | profile |
|----------|-----------|--------|------------|---------|
| 3.132441 | euclidean | 534 | 0.250924 | 1 |
| 2.198667 | euclidean | 6036 | 0.655435 | 3 |
| 2.699926 | euclidean | 930 | 0.947324 | 5 |

Table 2: Results from the grid search for the profile segmentation.

variation in their density thus the inability to find meaningful clusters, as for the present profiles we see that some clusters manage to get quite good results specifically 5 with a decrease for 3 and another one for 1, suggesting that some profiles are much more defined. In figure 40 we can see the pairplots for the clustering done on profile 1.

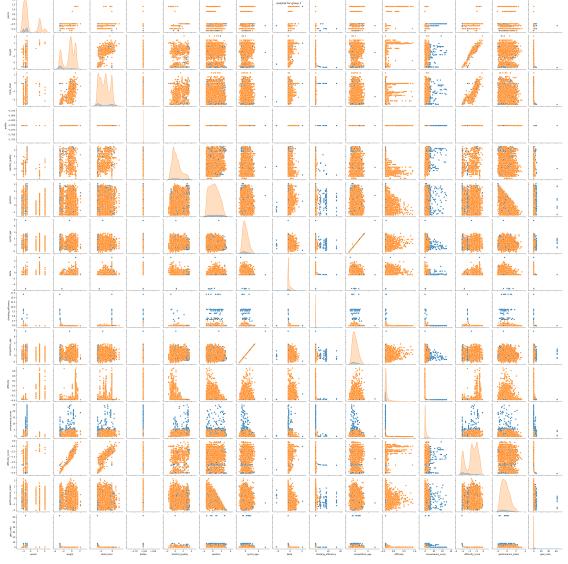


Figure 40: Clustering pairplots for the profile 1 segment.

3.1.3 Cyclists clustering

Now we can take care of the cyclists dataset, in this case we have way less entries so we don't have to use data reduction techniques of any kind.

Here there is no need to have all the method applied for the races we can use kneed with handpicked values according to the dataset, as shown in table 42. In table 42 we have reported the result for the best values, since the dataset is quite small

| MinPts | Eps | kth-neighbor |
|--------|------|--------------|
| 464 | 0.53 | 5 |
| 204 | 0.6 | 10 |
| 304 | 0.69 | 15 |
| 280 | 0.74 | 20 |
| 252 | 0.92 | 50 |
| 212 | 1.07 | 100 |
| 136 | 1.25 | 200 |
| 122 | 1.38 | 300 |
| 123 | 1.59 | 500 |

Figure 41: Eps values and min pts obtained with the kneed algorithm.

| Eps | metric | Minpts | Silhouette |
|------|-----------|--------|------------|
| 4.30 | euclidean | 1339 | 0.564855 |
| 4.30 | cosine | 1339 | 0.564855 |
| 4.30 | euclidean | 1188 | 0.564855 |
| 4.30 | cosine | 1188 | 0.564855 |
| 4.30 | euclidean | 1418 | 0.55774 |

Figure 42: Eps values and min pts obtained with the kneed algorithm along with the corresponding silhouette score.

we also tried to vary other parameters like the metrics used. After we applied the grid search we can visualize through a pairplot, reported in figure 43, the result for the first clustering in table 42.

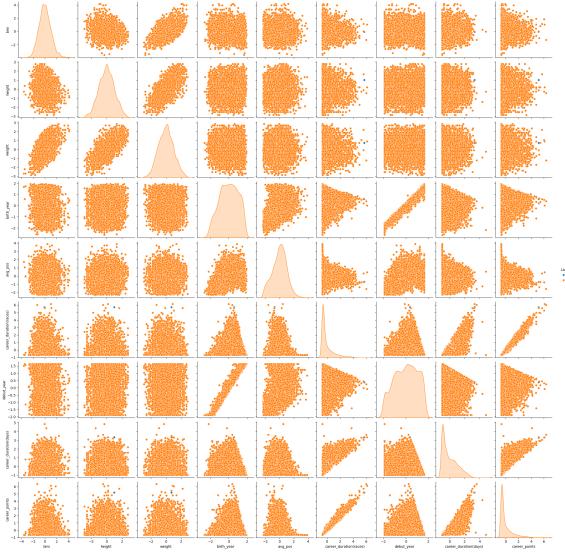


Figure 43: Pairplots used to confront the clustering of the cyclists.

3.1.4 Conclusions

In conclusion this method did appear to be effective, part of this problem could be due to the preprocessing that made for a very diverse density in the dataset and also the nature of the algorithm put a lot of constraints on time and possibilities to test so it did not yield interesting results

3.2 OPTICS

So given the very diverse density of the dataset getting good results is quite difficult, thus we decided to use the OPTICS algorithm which is a density-based clustering algorithm that identifies clusters of varying densities and scales without requiring a predefined neighborhood size. For this algorithm we decided to work with segments and analyze the density on each dataset and tried to analyze the dataset as a whole to understand better if there are any kind of density variations.

3.2.1 Segments clustering

For The clustering segments clustering we left out the data from 1970' to 1980 because of it having a very low number of entries and could hinder clustering then we analyzed the plot going from 1980 to 2000, 2000 to 2010 and 2010 to 2024 and analyze their reachability distance.

The training regime was a bit different: in this case we have a Max eps value that we can control to define the clusters size and limit memory usage so it is limited at 10% of the maximum distance.

We have two different hyperparameters that vary: $MinPts$, which is set like before however this time we decided to weight the percentage we have calculated before with the dimensionality of the dataset. The other one is xi , which determines when the density change is too much and extracts clusters, for this we tried different fixed initial values: 0.05, 0.2, 0.1. The training was done using these hyperparameters values and trying all their combinations like before.

On table ?? you can see the parameters we selected for each segment and in figure ?? you can see also the reachability plot for each segment.

For the cyclists we can just plot the reachability distance on the whole cyclists dataset and we can observe the reachability distance as well.

3.2.2 conclusions

From the theory we understand that sudden drops with low rachability distance in the plot mean that we have a transition from a cluster to another and also displays a lot of variety, in the reachability distance plots we have shown we have a lot of them on the races in the first two segments which have a lot of sudden drops, so parts of our suspicions are confirmed on the results obtained from DBSCAN.

3.3 kmeans

For **kmeans** we performed a hyperparameter search and we came to the conclusion of $k=2$ clusters for the features of the cyclists and $k=3$ clusters for the features of the races. The hyperparameter k has been decided confronting different *silhouette* and *cohesion* scores during the hyperparameter search using different number of clusters.

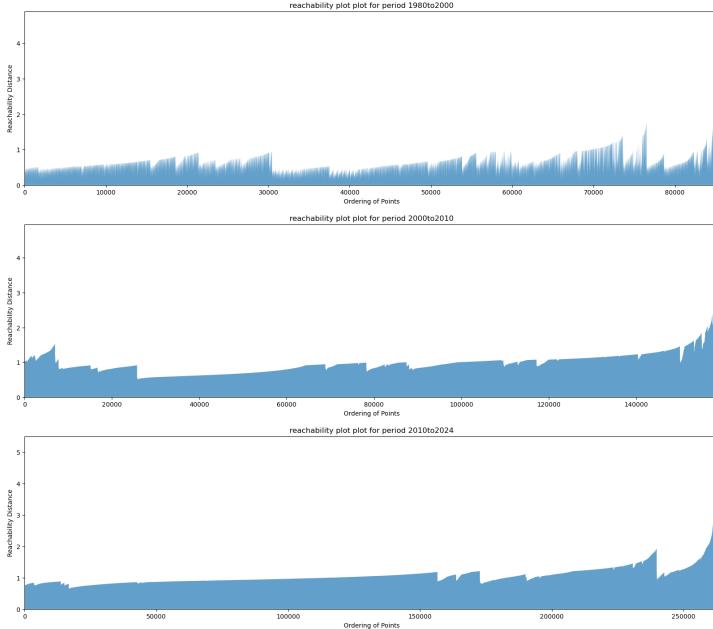


Figure 44: Reachability distance for optics on the period from 1980 to 2000.

For cyclists, the results are resumed in 46. As you can see, there is no clear distinction between the two clusters for what concerns the physical characteristics of the cyclists, the difference between the two clusters is clear for the features *career_duration(days)* and *career_points*.

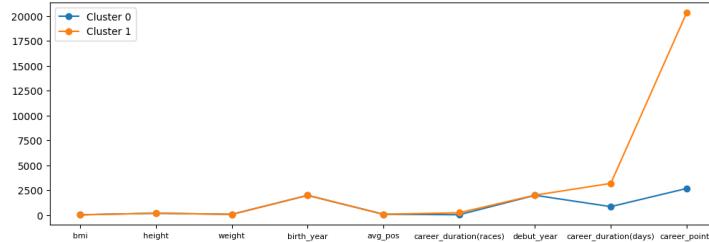


Figure 46: Results of the clustering for the cyclists' features.

In 47 like we've seen during the data understanding there is a strong positive correlation between the career points and the career duration. This is also reflected in the clustering of the cyclists. We can distinguish the following groups: *Cluster 0* represents cyclists with a short career duration and a low number of career points and *Cluster 1* shows cyclists with a longer career duration and higher number of career points. Again, by 48 confronting the average positioning and *career_points* we can distinguish the following groups: *Cluster 1* represents the cyclists with a moderate to high number of career points. We can see that the average positioning of the cyclists in this cluster is less sparse than the other cluster and tend to be more concentrated between the 50 and 100 position with few exception of cyclists that have a higher number of career points and a lower average positioning meaning that they have a better performance; and *Cluster 0* represents the cyclists with a low number of career points. We can see that the average positioning of the cyclists in this cluster is more sparse than the other cluster.

For the features of the races we took a different approach: since the size of the dataframe was large and computationally unfeasible we decided to do a random sampling, large enough to perform the clustering. The results of the clustering are summarized in 49.

Most of the features were not well separate by the clustering, however we can distinguish a bit better the 3 clusters by confronting the *length* and *difficulty* features.

3.4 Hierarchical Clustering

This is an unsupervised clustering technique that is very useful if the final clustering number is not known. The algorithm starts by considering each point a cluster in itself and in each step the closest points are joined, according to a chosen distance, until the selected number of clusters is reached (otherwise only one). Despite the obvious good visibility of the steps taken and the distance at each step, the method is very expensive for large datasets and the empirical nature of the choice of the number of clusters may backfire when one does not have enough time to try all combinations.

| minpts | Silhouette score | xi | period |
|--------|------------------|------|-------------|
| 2 | -0.084049 | 0.05 | 1980 - 2000 |
| 105 | 0.646901 | 0.05 | 2000 - 2010 |
| 175 | 0.589596 | 0.05 | 2010 - 2024 |

Figure 45: Hyperparameters search and results using optics.

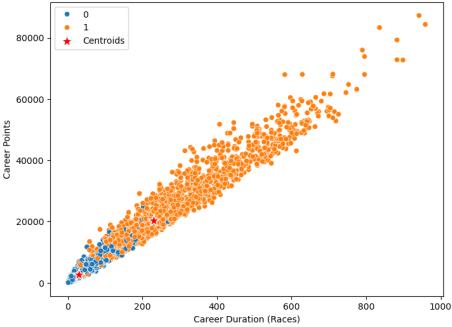


Figure 47: Clustering of career_duration against career_points

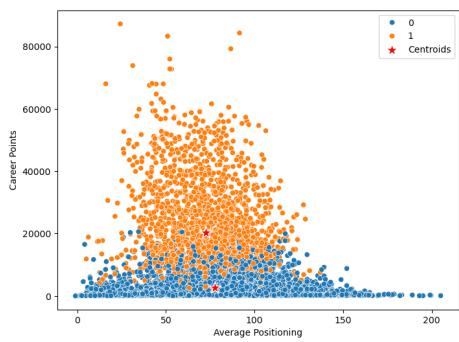


Figure 48: Clustering of career_points and avg_pos

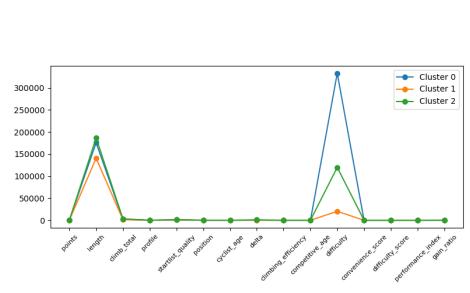


Figure 49: Results of the clusterings of the races

In the study, we tried four different distances, which were inserted into the linkage matrix, the data structure that is updated after each merge:

- **Single:** $d_{\min}(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y)$
- **Complete:** $d_{\max}(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y)$
- **Average:** $d_{\text{avg}}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y)$
- **Ward:** $d_{\text{ward}}(C_i, C_j) = \frac{|C_i| \cdot |C_j|}{|C_i| + |C_j|} \|\mu_i - \mu_j\|^2$

In these formulations, C_i means the cluster i while x and y are two points within the dataset. Distance plays a crucial role in cluster creation as it totally changes how the points are related and each has its pros and cons.

3.4.1 Cyclist Results

The experiments were performed with an increasing number of clusters and the best number was chosen by relating the MSS (Mean Squared Separation) and the SSE (Sum of Squared Errors) as this method favours a high silhouette score with only one cluster.

As for the clustering of the cyclist dataset, we observe the best result with Ward's distance, which is less susceptible to outliers and noise left over from previous cleanings.

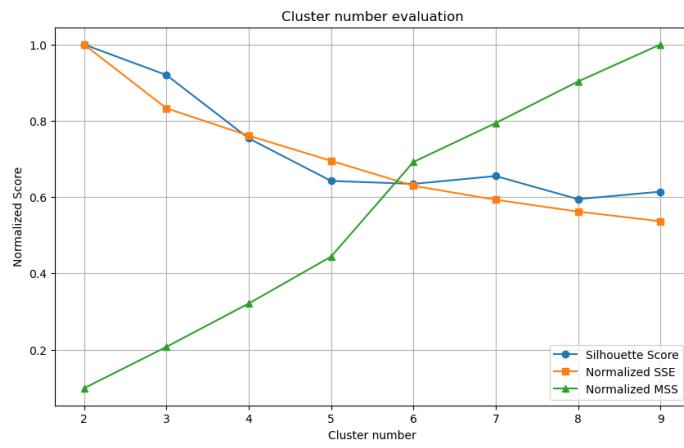


Figure 50: Hierarchical clustering results with Ward's Method

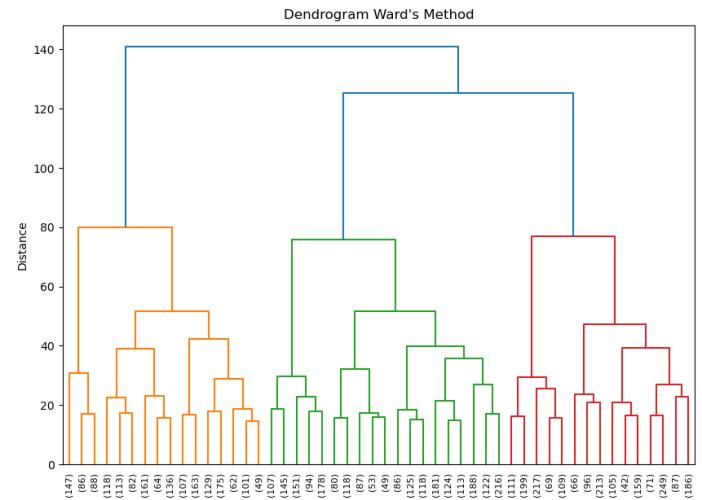


Figure 51: Dendrogram with Ward's Method

As can be seen from the graph, the optimal number of clusters is 6 with a Silhouette score of 0.65, not an optimal result. We can also see from the graph in Fig. 52 how the score decreases as the number of clusters increases, as it takes into account the separation between clusters, and if there are fewer clusters, this can only increase. The positive trend for the MSS and the negative trend for the SSE is in the normal range and indicates that the clustering process is going in the right direction, (these two values should be maximised and minimised in that order).

The dendrogram 51 however (another important factor in the evaluation during this clustering method) suggests 3 main clusters that are very closely spaced, highlighted by the three different colours.

| Cluster_number | Silhouette Score | SSE (Normalized) | MSS (Normalized) |
|----------------|------------------|------------------|------------------|
| 2 | 1.000000 | 1.000000 | 0.098956 |
| 3 | 0.979673 | 0.825192 | 0.209509 |
| 4 | 0.884586 | 0.754166 | 0.402023 |
| 5 | 0.739620 | 0.688524 | 0.532723 |
| 6 | 0.650070 | 0.624576 | 0.678501 |
| 8 | 0.644219 | 0.565107 | 0.889781 |
| 7 | 0.635261 | 0.594718 | 0.794621 |
| 9 | 0.624799 | 0.540221 | 1.000000 |

Table 3: Results Table for Ward's Method.

3.4.2 Races Results

In this case we find another distance in the best case, the Average, this can be easily seen from the graph (Fig. 52).

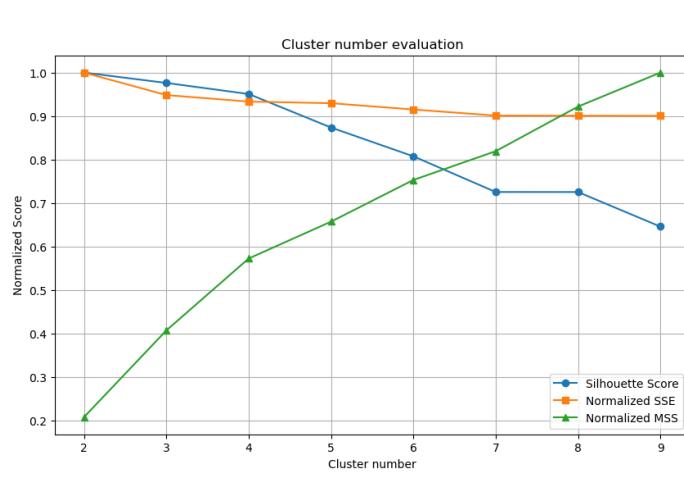


Figure 52: Hierarchical Clustering of races with Average Distance

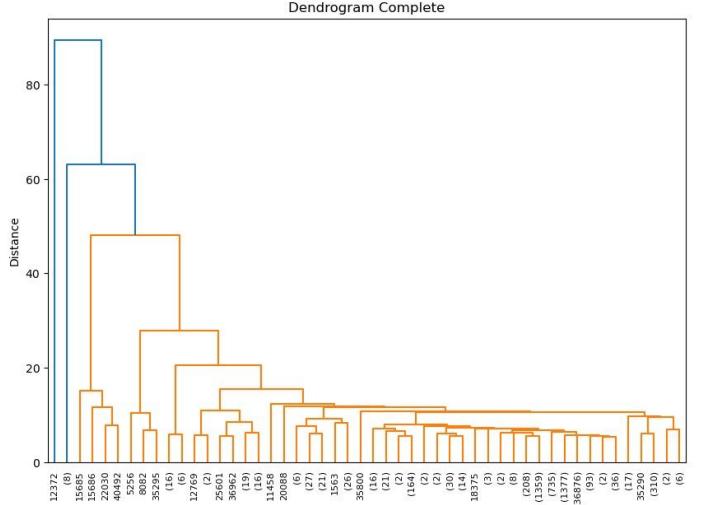


Figure 53: Hierarchical Clustering with Average distance

In this case, we find the best number of clusters as 7 with a silhouette score of 0.72. The trend in this case is more unstable as dealing with a lot more data in each iteration many points are joined together and the division scores undergo sudden changes, the cohesion parameter SSE instead decreases more slowly. The MSS instead increases a lot faster and reach 0.81 in the optimal case (Table 4).

| Cluster_number | Silhouette Score | SSE (Normalized) | MSS (Normalized) |
|----------------|------------------|------------------|------------------|
| 2 | 1.000000 | 1.000000 | 0.208039 |
| 3 | 0.976240 | 0.948313 | 0.407294 |
| 4 | 0.950797 | 0.933373 | 0.572608 |
| 5 | 0.873754 | 0.929642 | 0.657338 |
| 6 | 0.807546 | 0.915177 | 0.753157 |
| 7 | 0.725503 | 0.901173 | 0.819337 |
| 8 | 0.725433 | 0.900907 | 0.921845 |
| 9 | 0.645989 | 0.900676 | 1.000000 |

Table 4: Average Clustering result for Races dataset

Even in this case the dendrogram (Fig. 53) suggests an optimal number of two clusters, with data sparsity it's obvious that the results are discordant between them.

Task 4: Prediction

For the prediction task we used several models that we've seen during the course. We have chosen our classifiers from several machine learning models we have seen during the course. We used base classifiers (decision trees, RIPPER, KNN) and ensemble models (Random Forest, AdaBoost, XGBoost).

Preprocessing was the first stage for the prediction task. We prepared the train and test dataset by merging the dataset of the races and the dataset of the cyclists. Each record of the train and test dataset contain the following features that we have chosen from the initial dataset and from the feature engineering task: bmi, career_points, career_duration(days), debut_year, difficulty_score, competitive_age, is_tarmac, climbing_efficiency, startlist_quality, avg_pos. The selected features reflect characteristics of the cyclists, their level of experience as well as characteristics of the race that may affect the difficulty of the race. During preprocessing, the following three features have been redefined to be more suitable for the task of prediction:

avg_pos is the average positioning of the cyclists *before* until race;

career_points tracks the accumulated points of the cyclists *before* that race; and

career_duration(days) tracks the total duration of the career of the cyclist until that race.

We defined the target *top_20* as 1 if the cyclist ended the race in the first 20 positions, otherwise 0. We divided the train set and the test set by splitting the merged dataset with the date *2022-01-01* as requested by the task.

We noticed after the splitting that the dataset was highly imbalanced there were too many negative labels, so we managed to balance the number of positive and the negative labels so that we did not create a bias towards the majority class.

4.1 Base Classifiers

The three main base classifiers that we chose were *Decision Trees* and *KNN* (using *scikit-learn* library) and *RIPPER* (using *wittgenstein* library). We decided to try this models because of their rich performances in many contexts and their easy explainability (especially for RIPPER and Decision Trees). After the preprocessing pipeline that we described before, a grid search has been performed. The results of the base classifiers with the best hyperparameters found during validation can be found in table 5. Among the three models, Decision Trees seemed to perform a bit better because it was better at predicting the positive class. However, all the three models seem to have better performances on the negative class even after rebalancing the dataset.

| Model Name | Parameters | F1 Score (Averaged) |
|------------------------|--|---------------------|
| KNeighborsClassifier | algorithm='kd_tree', metric='manhattan', n_neighbors=3, weights='distance' | 0.69 |
| RIPPER | k=3, max_rule_conds=3, max_rules=30, prune_size=0.5 | 0.47 |
| DecisionTreeClassifier | class_weight='balanced', criterion='entropy', max_depth=15 | 0.72 |

Table 5: Summary of models, parameters, and averaged F1-scores.

4.2 Neural networks

For the neural networks we wanted try a family of models, in this case we used a feedforward neural network on the dataset using the same split as the others.

| Hyper-parameter name | values tested |
|----------------------|------------------|
| Number of layers | 10, 15, 20, 30 |
| Learning rate | 1e-2, 1e-3, 1e-4 |

Table 6: Hyperparameters used for the feedforward neural network.

| Layers | Learning rate | F1-score val | BCEntropy val |
|--------|---------------|--------------|---------------|
| 30 | 0.00010 | 0.28943 | 0.383070 |
| 20 | 0.00010 | 0.28943 | 0.383234 |
| 15 | 0.00010 | 0.28943 | 0.383785 |
| 10 | 0.00100 | 0.28943 | 0.383857 |
| 15 | 0.00100 | 0.28943 | 0.383898 |

Table 7: Results from using the neural network on the dataset.

In Table 6 we can see the hyperparameters values we used, for the optimizer we have choose Adamax and the BinaryCrossEntropy for the loss functions to minimize with 1024 neurons for each layer. Note: the reason for such restricted number of layers is computational feasability and time constraints.

As expected, on noisy data getting good results with such models is quite difficult. Expending more time is unlikely to yield good results thus given all the time and computational constraints for the experiments we did not go further.

4.3 Bagging models

For the bagging models we decided to try the Random Forests, the training has been done on the afore mentioned traing-test split using and hold-out method and stratification. Given the capacity of Random Forests to handle well noisy datasets, we decided to do an extensive grid search on it with boostraping for the base models and the class weight parameters always balanced, we penalize much more misclassification in the minority class, since the dataset is very imbalanced with very few label 0 entries this should enhance generalization (Table 8).

| Hyper-parameter name | values tested |
|-----------------------------------|------------------|
| Number of estimators | 50, 100, 200,500 |
| Maximum features per estimator | 50, 100, 200,500 |
| Splitting criterion | gini, entropy |
| Maximum trees depth | 10, 20 ,30 ,None |
| Minimum samples required to split | 2, 5, 10,20 |
| Minimum samples per for each leaf | 1, 3, 5,10 |

Table 8: Hyperparameters values used one the grid search for Random Forests.

After doing the grid search we decided to list the results ordering them by f1-score and pick from the top-10 the best one (Table 9). We can see the top performing parameters, as we can see even using countermeasures results, don't reach good results. By analyzing the results we can see that most parameters indicate overfitting. Among all the results we have an "outlier" that has slightly worse training f1-score, less model complexity and manages to get very similar results, so following Occam's Razor principle we pick such a model as the best one from the grid search.

To further evaluate the model performance we also plot ROC curve to observe the model performance on both classes (Fig. 54).

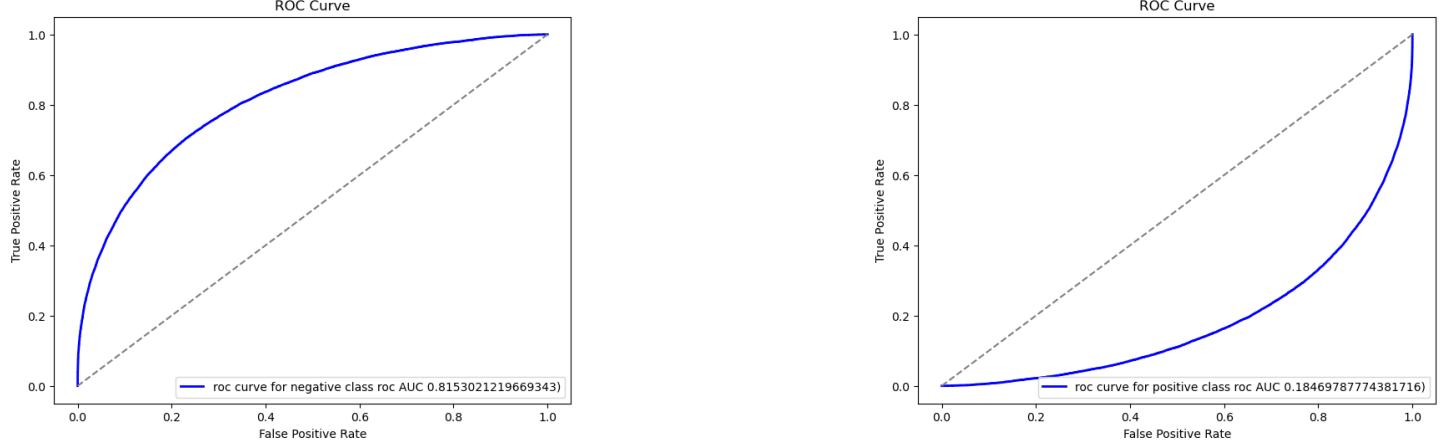


Figure 54: ROC crubes with their AUC value reported for both classes.

4.4 Boosting Methods

Another method we used to predict cyclists in *top20* was booosting. This is generally based on training multiple classifiers that at predict time perform a vote and by majority the predicted label is chosen. In both models we used the decision trees as weak learners, due to their flexibilty and compatibility to the libraries involved.

4.4.1 XGBoost

The XgBoost framework comes from the library xgboost built by Distributed (Deep) Machine Learning Community. Specifically, this method uses gradient descent for training weak learners, decision trees. Once learner i has been towed, the gradient is calculated, the loss function optimised and with the updated version the learner $i + 1$ is towed, at the end of the training all weak learners perform a weighted vote according to the error committed in the training.

In this case we can see that for the first models there is not much difference in the results, range in this case it is the parameter that changes but does not affect the performance of the model. As for lambda, it appears that it does not serve the performance of the model and that 250 estimators with a maximum depth of 3 is the best combination found. In addition, thanks to the initial balancing between the two classes in the dataset, we can find a predictable 1 in the hyperparameter '*scale_pos_weight*', which should in fact weight the least represented class more heavily. The best result with 72% on the validation set with a good result balance between the f1 score of the two classes.

| Criterion | maximum depth | maximum features | minimum samples leaf | minimum samples split | number of estimators | Training f1-score | Validation f1-score |
|-----------|---------------|------------------|----------------------|-----------------------|----------------------|-------------------|---------------------|
| entropy | None | 4 | 1 | 20 | 500 | 0.801598 | 0.560977 |
| entropy | 30 | 4 | 3 | 20 | 200 | 0.778936 | 0.560361 |
| entropy | None | 3 | 1 | 20 | 500 | 0.791579 | 0.560295 |
| entropy | 30 | 3 | 1 | 20 | 500 | 0.785548 | 0.560143 |
| gini | None | 3 | 1 | 20 | 500 | 0.779968 | 0.560074 |

Table 9: Results from the grid search performed on the Random Forests.

| gamma | lambda | learning_rate | max_depth | min_child_weight | n_estimators | train_f1_score | val_f1_score | test_f1_score |
|-------|--------|---------------|-----------|------------------|--------------|----------------|--------------|---------------|
| 0.1 | 2 | 1.0 | 3 | 3 | 250 | 0.724716 | 0.719851 | 0.774840 |
| 0.2 | 2 | 1.0 | 3 | 3 | 250 | 0.724716 | 0.719851 | 0.774840 |
| 0.0 | 2 | 1.0 | 3 | 3 | 250 | 0.724716 | 0.719851 | 0.774840 |
| 0.2 | 0 | 1.0 | 3 | 1 | 250 | 0.724164 | 0.719365 | 0.780020 |
| 0.1 | 0 | 1.0 | 3 | 1 | 250 | 0.724164 | 0.719365 | 0.780020 |

Table 10: XgBoost results with hyperparameters comparison.

4.4.2 AdaBoost

For Adaboost we used sklearn.ensamble libraries. For these experiments we have similar results , even better on the validation (0.72) but really bad generalization capabilities with a f1-score on the test set of 0.57.

| criterion | learning_rate | max_depth | n_estimators | train_f1_score | val_f1_score | test_f1_score |
|-----------|---------------|-----------|--------------|----------------|--------------|---------------|
| entropy | 0.5 | 4 | 200 | 0.729955 | 0.724935 | 0.575789 |
| gini | 0.5 | 4 | 200 | 0.729081 | 0.723360 | 0.617035 |
| gini | 0.5 | 4 | 200 | 0.721850 | 0.718095 | 0.624068 |
| entropy | 0.5 | 4 | 200 | 0.721588 | 0.717031 | 0.617408 |
| entropy | 0.5 | 4 | 100 | 0.716217 | 0.713116 | 0.607488 |

Table 11: Adaboost training results.

The best algorithm is SAMME.R (optimized for decision tree) with a splitting criterion based on entropy. The best result is achieved with 200 estimators with a learning rate of 0.5 (Table 11).

4.5 Model comparisons & conclusions

To compare the different models we evaluated, we take from each one the results for each model trained with the best hyperparameters and we take *f1-score* macro average measure for validation and test (Table 12).

| Model | Validation | test |
|-----------------|------------|------|
| Random Forest | - | 0.64 |
| XGBoost | 0.72 | 0.61 |
| Decision Trees | 0.63 | 0.59 |
| Adaboost | 0.72 | 0.51 |
| Neural Networks | - | 0.50 |
| K-NN | 0.55 | 0.53 |
| Ripper | 0.55 | 0.39 |

Table 12: Comparison between the different models using f1-score

Task 5: Explanation

5.1 XGboost

5.1.1 Feature Importance Analysis

To assess the feature importance ordering we studied the feature importance using both *interventional*(Fig. 55) and *three_path_dependent*(Fig. 56) feature perturbation methods. The graph in Fig. 55 represents the impact of the selected features on model predictions; for this plot SHAP values were imputed preserving the relationships between the features (e.g. if a feature changes, the other features change too, ensuring the relationship preservation between the two).

On the other hand, graph in Fig. 56 uses the *three_path_dependent* method which uses a tree structure that reflects the relationships learned by xgboost directly from the training data and unlike *interventional* it does not rely on an external causal model.

After we calculated the SHAP values in both ways, we calculated the maximum differences between the two methods and saw that most of the instances have a maximum difference of 0.2 in SHAP values indicating that the two methods are not very different in their decisions (Fig. 57).

In Figure 58 the mean and variance of the different features are reported. In particular, we can notice that:

career_points is generally the most impactful instance since it has the highest mean in absolute value;

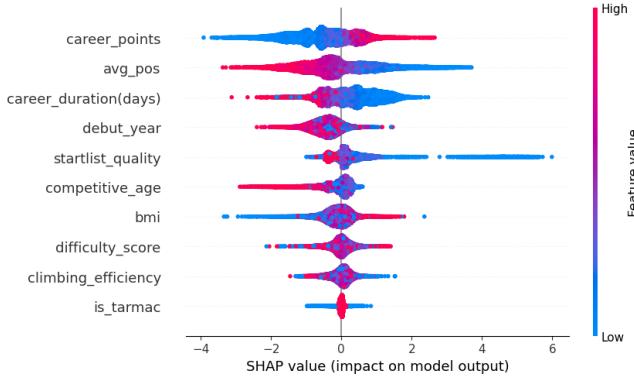


Figure 55: Interventional explanations for feature importance.

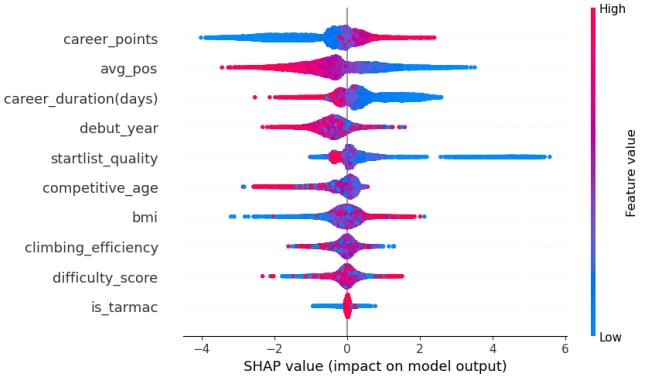


Figure 56: Distributional explanations for feature importance.

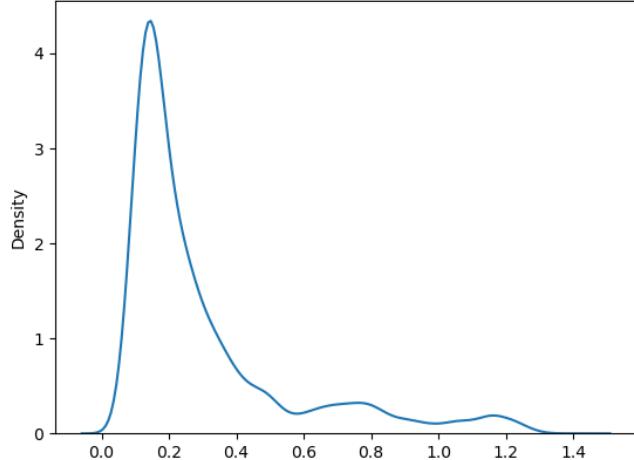


Figure 57: Shap between the two differences.

| Attribute | mean | std |
|-----------------------|-----------|----------|
| career_points | -0.506832 | 0.870802 |
| debut_year | -0.432089 | 0.365037 |
| startlist_quality | 0.235681 | 0.744266 |
| competitive_age | -0.152278 | 0.458998 |
| career_duration(days) | 0.120179 | 0.536170 |
| bmi | -0.116001 | 0.371103 |
| avg_pos | -0.099005 | 0.996696 |
| difficulty_score | -0.015865 | 0.276714 |
| climbing_efficiency | -0.002946 | 0.262612 |
| is_tarmac | -0.005030 | 0.089113 |

Figure 58: Features ordered by their average impact

debut_year, startlist_quality, competitive_age have less impact on the prediction, but still a significant mean and standard variation in SHAP values;

avg_pos has low mean considering SHAP values contribution, but it has an high variance meaning that it is an impacting feature depending on the instance.

is_tarmac is ranked at the bottom of our list.

After studying which are the most impactful features, we divided the instances in two sets: **highly impacted** and **moderately impacted** instances. The highly impacted correspond to the 10% of the most negatively impacted instances by the feature of interest (e.g. for *career_points* are those instances in which *career_points* contributes the most to reduce the probability of success), the moderately impacted are all the others.

We studied the contribution of the five main impactful features described above. As depicted in Fig. 59, starting from the left we can see:

career_points affect most the cyclists with low accumulated points during their career until that race;

debut_year have the most impacted instances in the years between 1970 and 1980 meaning that the cyclists who started back in that decade are the most impacted by their debut year;

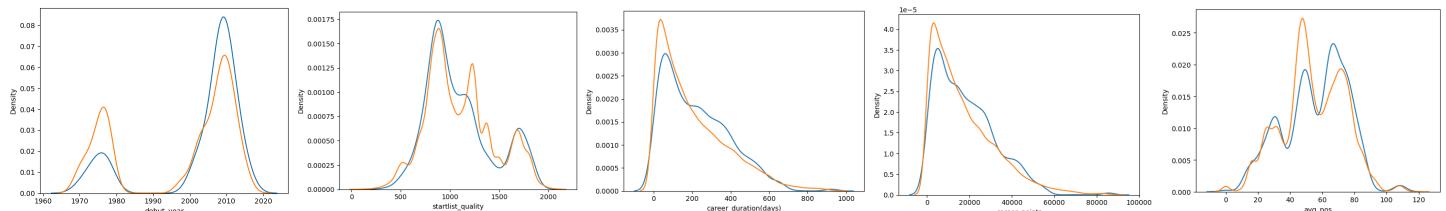


Figure 59: Difference between highly impacted (orange) and moderately impacted ones (blue)

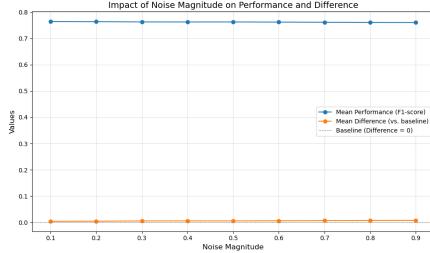


Figure 60: Sensitivity of xgboost at different noise magnitude levels

startlist_quality highest impacted instances range from 1000 to 1500 meaning that cyclists that participate in competitions with high startlist quality are disadvantaged;

career_duration impacts most on instances who have a short number of races they have participated before; and finally **avg_pos** affects most the cyclists whose average positioning is between 40 and 60.

Our analysis went further by studying how the noise affects the predictions on the trained model. To do so, we created different datasets where we introduced different noise magnitude for each feature. After we created those datasets, we validated the model we created on them and we confronted the validation scores using the f1-score metric. We noticed that in the *career_duration(days)*, *career_points*, *competitive_age*, *debut_year*, *is_tarmac* and *startlist_quality* even if noise is introduced they do not affect the performance of the model, while *climbing_efficiency*, *bmi* and *difficulty_score* are more sensitive to noise. The result of how the noise in the test sets affect the prediction are depicted in Fig. 60. As we can see they show that the model proved to have almost constant performances with different noise magnitude, however the difference between the baseline scores and the validation score increases slightly by increasing the noise magnitude resulting in a degradation of the performances.

5.1.2 Rule explanation

| Feature | Value |
|-----------------------|--------|
| bmi | 20.57 |
| career_points | 22335 |
| career_duration(days) | 305 |
| debut_year | 2011 |
| difficulty_score | 1.15 |
| competitive_age | 25 |
| is_tarmac | 1 |
| climbing_efficiency | 0.013 |
| startlist_quality | 985 |
| avg_pos | 53.885 |

Table 13: Feature values in row-wise format.

| Attribute | Operator | Threshold |
|-----------------------|----------|-----------|
| career_points | > | 11172.50 |
| bmi | > | 20.50 |
| avg_pos | ≤ | 54.25 |
| career_duration(days) | ≤ | 500.50 |
| career_duration(days) | > | 19.00 |
| competitive_age | ≤ | 32.50 |
| climbing_efficiency | ≤ | 0.71 |
| climbing_efficiency | > | 0.01 |
| debut_year | ≤ | 2011.50 |
| debut_year | > | 2008.50 |
| difficulty_score | > | 1.09 |

Table 14: Rules for predicting class *top_20 = 1*.

For the rule explanation we used the **LORE** tool, which uses local explanations based on rules to give an explanation to the model prediction. For instance, we've taken this instance from the training set (Table 13).

Using LORE, we retrieved the rules in Table 14. These rules in Table 14 explain why the model classified the instance as *top_20*, but in reality it isn't (false positive).

From the retrieved rules we can know why the instance has been classified in that way. We notice that the cyclist has a high number of *career_points* meaning for the model that he is a expert; The cyclist also has an *avg_pos* under a certain threshold which leads the model to classify it as a positive.

5.1.3 Counterfactual instances

Looking at the table counterfactual instances in table 15 by changing few features we can change the outcome of the prediction. Interestingly we notice in **cf 1** and **cf 2** that by raising too much both *bmi* and *career_duration(days)* the prediction outcome changes.

5.2 Random Forest

5.2.1 Feature Importance Analysis

Due to the computational resources needed, we used a random sampling made up by 100 datapoints. By doing so, we aim to concentrate on representative instances of the original dataset. For the instance depicted in Table 13 we have of a

| Attribute | base | cf 1 | cf 2 | cf 3 | cf 4 | cf 5 | cf 6 | cf 7 | cf 8 | cf 9 | cf 10 |
|-----------------------|--------|-------|-------|-------|-------|-------|------|------|-------|--------|-------|
| bmi | 20.57 | 23.26 | 21.33 | - | - | 19.77 | - | - | 17.21 | - | - |
| career_points | 22335 | - | - | - | - | - | - | - | - | - | - |
| career.duration(days) | 305 | 871.4 | 541.6 | 557.6 | - | - | - | - | - | 415.17 | - |
| debut_year | 2011 | - | - | - | - | - | - | - | - | - | - |
| difficulty_score | 1.15 | - | - | - | - | - | - | - | - | - | - |
| competitive_age | 25 | - | - | - | - | 19 | 45 | 39 | - | 19 | - |
| is_tarmac | 1 | - | - | 0 | - | - | - | - | - | - | - |
| climbing_efficiency | 0.013 | - | - | 0.16 | - | - | - | - | - | - | - |
| startlist_quality | 985 | - | - | - | - | - | - | - | - | - | - |
| avg_pos | 53.885 | - | - | 63.7 | 155.1 | - | - | - | - | - | - |
| top_20 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 15: Counterfactual instances (xgboost)

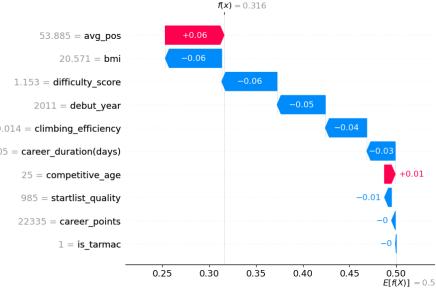


Figure 61: Shap contribution on a instance

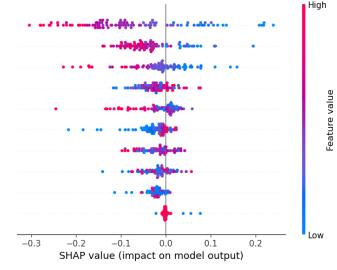


Figure 62: summaryplot random forest

different contribution for each feature, from top to bottom the instances are ordered by their importance. To have a deeper understanding we plotted a summary plot as shown in Fig. 62. This plot allows us to generalize the importance of each feature for the random forest model we trained. In particular, we can notice that *avg_pos* and *debut_year* are ranked among the top features by importance, meaning that the feature we designed have an high impact on the prediction. Other features are placed in the middle (e.g. *difficulty_score* and *career_duration*) while others like *is_tarmac* place themself on the bottom meaning that they have little to no impact on the prediction.

5.2.2 Rule explanation

For what concerns rule explanation, the model correctly classifies the instance as not in *top_20*. From the model we can extract the following rules that explain us why the model gives that outcome. The rules can be found in Table 63.

| Attribute | Operator | Threshold |
|---------------------|------------------|-----------|
| avg_pos | \leq | 54.92 |
| avg_pos | \geq | 33.23 |
| climbing_efficiency | $\leq \vee \geq$ | 0.02 |
| bmi | $\leq \vee \geq$ | 21.43 |
| difficulty_score | $\leq \vee \geq$ | 1.97 |
| startlist_quality | \geq | 516.65 |
| debut_year | \geq | 1990.63 |
| competitive_age | $>$ | 24.10 |

Figure 63: rules used to explain the model outcome on the instance

The rules reflect the feature importance shown in Fig. 62 and show that the instance is correctly classified as 0 because average positioning of the previous races is too high to be classified as top 20, we saw during the clustering that most of the cyclists who have an average career have a average positioning around 50 like this one and very few go into top 20; climbing efficiency, bmi and difficulty score are too low; startlist quality is too high meaning that the participants in the race are too strong for the cyclist; debut year and competitive age are too high, again we saw during data understanding that the debut year has a slight correlation with the average positioning, this may justify why it has been classified in this way.

5.2.3 Counterfactual Instances

For the counterfactual generations for random forest, we used both a random search approach for the counterfactuals by changing the features in input and also by modifying specific the most impactful features, here we report the counterfactual instances that we consider to be the most interesting to explain how the model works (Table 16) From the counterfactual instances in the table we can notice few things: by lowering the *avg_pos* and balancing it with other features (e.g. *startlist_quality* in **cf 2** and *bmi* in **cf 6**) we can change the outcome of the prediction, so this proves that the average positioning we engineered is a feature with a significant impact on the model. In most of the counterfactual instances depicted in the table we can see

| Attribute | base | cf 1 | cf 2 | cf 3 | cf 4 | cf 5 | cf 6 | cf 7 | cf 8 | cf 9 | cf 10 |
|-----------------------|--------|------|------|-------|-------|-------|------|-------|------|------|-------|
| bmi | 20.57 | - | - | 22.17 | 22.59 | 23.21 | - | 21.17 | - | - | - |
| career_points | 22335 | - | - | - | - | - | - | - | - | - | - |
| career_duration(days) | 305 | - | - | - | - | - | - | - | - | - | - |
| debut_year | 2011 | - | - | - | - | - | - | - | - | - | - |
| difficulty_score | 1.15 | - | - | - | - | - | - | - | 2.17 | 2.54 | - |
| competitive_age | 25 | - | - | - | - | - | - | - | - | - | - |
| is_tarmac | 1 | 0 | - | - | - | - | - | - | - | - | - |
| climbing_efficiency | 0.013 | - | - | 0.25 | - | 0.38 | - | 0.06 | 0.08 | - | - |
| startlist_quality | 985 | - | 511 | - | - | - | - | - | - | - | - |
| avg_pos | 53.885 | - | 19.1 | - | - | - | - | 13.4 | - | - | 5.5 |
| top_20 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 16: Counterfactual instances using Random Forest

that *bmi* is the most frequently changed one, in fact it has a major negative impact on the prediction as you can see in Fig. 61, in fact by increasing only it, the prediction on the instance changes (**cf 4 and cf 5**). Even if *is_tarmac* has little impact on the model, for this particular instance by changing it from 1 to 0 it changes the prediction to positive in **cf 1**.

References

- [1] Aleksey Bilogur. *MissingNo*. Accessed: December 24, 2024. 2024. URL: <https://github.com/ResidentMario/missingno>.
- [2] *Kneed*. 2024. URL: <https://github.com/arvkevi/kneed>.
- [3] *ProCyclingStats*. Accessed: December 24, 2024. 2024. URL: <https://www.procyclingstats.com/>.
- [4] *UCI*. 2024. URL: <https://www.uci.org/>.