UNIVERSITÀ DI PISA

Data Mining
A.Y. 2021/2022

A Data Mining Process on Tennis Matches Data

Francesco Fattori, Giacomo Frigo, Monica Amico

Contents

1 Data Understanding **2**
- 1.1 Data Semantics . . . 2
- 1.2 Data quality . . . 2
  - 1.2.1 Missing values . . . 2
  - 1.2.2 Duplicated records . . . 3
  - 1.2.3 Wrong values . . . 3
  - 1.2.4 Outliers . . . 4
- 1.3 Distributions and Statistics . . . 6
- 1.4 Correlations . . . 8

2 Data Preparation **9**
- 2.1 Duplicates . . . 9
- 2.2 Wrong and Missing Values . . . 9
- 2.3 Outliers . . . 10
- 2.4 New Features . . . 11
- 2.5 Profiles . . . 12

3 Clustering **13**
- 3.1 Pre-Processing . . . 13
- 3.2 K-means . . . 13
  - 3.2.1 Find the best value K . . . 13
  - 3.2.2 Results . . . 14
- 3.3 Density based Clustering (DBScan) . . . 15
  - 3.3.1 Parameters evaluation . . . 15
  - 3.3.2 Results . . . 16
- 3.4 Hierarchical Clustering . . . 17
  - 3.4.1 Implementation . . . 18
  - 3.4.2 Results . . . 18

4 Predictive Analysis **20**
- 4.1 Pre-processing . . . 20
- 4.2 Models . . . 20
- 4.3 Models evaluation . . . 21
  - 4.3.1 Area Under The Curve - AUC . . . 22
  - 4.3.2 Receiver Operating Characteristic - ROC . . . 22

5 Time Series Analysis **23**

1 Data Understanding

1.1 Data Semantics

The dataset contains some features that were not described in the project assignment, with the following semantic meaning:

- `tourney_spectators`: total number of spectators of the tourney
- `tourney_revenue`: total tourney's revenue
- `score`: match's final score
- `round`: match round. Could have these values: QF (quarter of final), SF (semifinal), F(final), R16(Round 16) and some other possible values

1.2 Data quality

Poor data quality negatively affects many data processing efforts, so we need to detect problems and then try to fix them without altering data statistical properties. Examples of data quality problems are: wrong or missing values, data duplication, noise and outliers.

1.2.1 Missing values

tourney_id	55	round	30
tourney_name	25	minutes	104461
surface	188	w_ace	103811
draw_size	29	w_df	103809
tourney_level	29	w_svpt	103811
tourney_date	28	w_1stIn	103811
match_num	27	w_1stWon	103809
winner_id	55	w_2ndWon	103812
winner_entry	160008	w_SvGms	103810
winner_name	27	w_bpSaved	103806
winner_hand	46	w_bpFaced	103809
winner_ht	136516	l_ace	103808
winner_ioc	29	l_df	103802
winner_age	2853	l_svpt	103806
loser_id	28	l_1stIn	103817
loser_entry	141731	l_1stWon	103810
loser_name	31	l_2ndWon	103809
loser_hand	98	l_SvGms	103803
loser_ht	147489	l_bpSaved	103810
loser_ioc	26	l_bpFaced	103815
loser_age	6537	winner_rank	19402
score	193	winner_rank_points	19420

We have detected missing values by counting for each attribute the number of records with NaN value.

As can be immediately noticed many missing values are present in the dataset, in particular in the game statistics like the number of winner's aces and the number of winner's double faults. Some more important features also have many null values, like `tourney_id` and `winner_name`.

We will try to replace these values in the data preparation phase.

We also checked for null values into male ad female datasets, and we have obtained:

- Female dataset: 1667 Names with null values, 0 Surnames with null values
- Male dataset: 177 Names with null values, 42 Surnames with null values

1.2.2 Duplicated records

We looked for duplicate records within the dataset and we found 309 records, that will be subsequently eliminated to remove noise and mis-understanding.

1.2.3 Wrong values

The dataset could have data with syntactic and semantic problem. We checked the **semantic accuracy** and the **syntactic accuracy** examining these attributes of the matches dataset:

- `tourney_date`, checking the date format (by looking the number of digits) and we discovered that there are 185791 strings with length 10, and 28 with length 3 (`NaN` values), so the tourney's dates are formatted correctly;
- `winner_entry` and `loser_entry`: which value should be in WC, Q, LL, PR, ITF domain and some other occasionally used. We noticed that probably 'Alt' and 'ALT' refer to the same value (we will fix this value later during preparation);
- `winner_hand` and `loser_hand`: which value should be L, R, U, all values are syntactic correct;
- `winner_ioc` and `loser_ioc`: checked that the length of all the values is 3, so all values are well formatted.
- `winner_age` and `loser_age`: we searched some wrong values (i.e. age = 0) and no wrong values were found. In this phase we are doing a very basic check and we will analyze better outliers in the dedicated section.
- `minutes`: the longest tennis match ever played lasted 11 hours. We checked that the values are smaller than 12 hours (12 * 60) and greater than zero minutes and we found some suspicious data (i.e. a value equal to 4756 minutes that are around 80 hours). These are probably wrong values and they should not be considered when we will create data statistics. We proceeded by analyzing the score of the matches to understand if zero minutes played is actually a wrong value. We found that in case the score is equal to "W/O" (walk-around) or "0-0" the zero value is correct. There is also the case in which on of the players retires and in this specific case it might be correct.

  Matches which have a valid score (some sets were played) and played minutes equals to 0 must be fixed.

- `winner_rank` and `loser_rank`: we checked that the rank are greater than zero. There were not negatives or zeros rank values;

We also examined attributes of the Male and Female dataset. We sorted player's name based on surname (and name) length and saw the first records.

Typically wrong values have just one letter or a sigle wrong character. We found `(X,X)` as value that maybe could be considered a wrong value, probably a default value used in some data source and we find a person called `(Name, Surname)`. We found also some surname specified only by using initial letter. We will try to fix these in the data preparation phase in which we try to find real name/surname by introducing data integration or by looking the matches dataset.

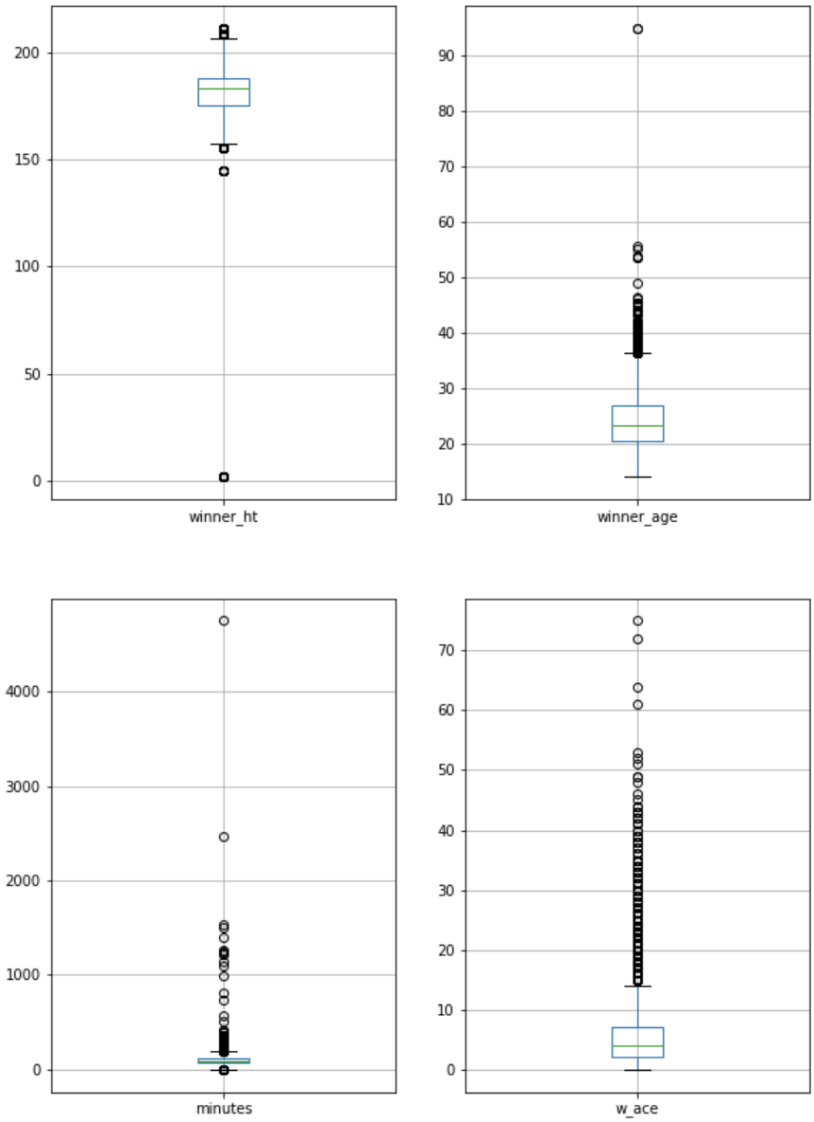
By analyzing the two dataset joined together we discover that about 60 full names are present both in the male dataset and in the female. These data (named *non-binary people*) will be fixed in data preparation phase by looking the matches dataset and check the gender of the opponent: a tennis match is played between people of the same gender.

1.2.4 Outliers

During the data understanding process is also important to find outliers. Outliers can lead to wrong computation of statistical metrics on the dataset. To detect them we examined numeric single dimension attributes, using the Interquartile range method.

For each of these attributes, we have defined a range with lower bound and upper bound, and we have considered as candidate outliers all the attributes that are outside of this range. We examine outliers both by applying filters on the dataset and by exploting barplots in order to have also a graphical representation.

A lot of outliers were found for every attribute, but we want to point out that some of them, although are numeric, they are actually treated as categorical attributes (like `draw_size` or `best_of`). We removed from our outliers analysis attributes which have not the concept of outlier like `match_num`, `winner_id` or `loser_rank`. Furthurmore we want to clarify that values which reside outside the interquantile range are not automatically identified as outliers. These data are identified as candidate outliers that need for further investigations.



In some attributes is simpler to understand outliers than others.

For example analyzing winners heights we can observe some values outside the boxplot's branches but in our opinion some of them can be interpreted as true values (like the ones just above the upper bound). Surely we can not interpret in the same way the value near 0 (it should be 2 cm), because this is clearly a wrong value and we have to fix them in the data preparation phase, where we will try to derive the heigth of that winner/loser by other matches played by him/her.

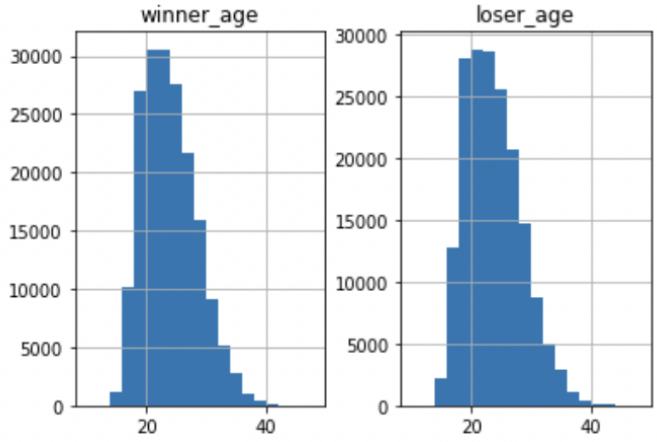
Winner/Loser age also contains a really high values. In our opinion data around

50 years old can be considered as true values, so we do not want to replace them. The really high values (95 for winner and about 75 for loser) seem to be wrong values and as for the heights we will investigate and, if neccessary, fix them in data preparation phase.

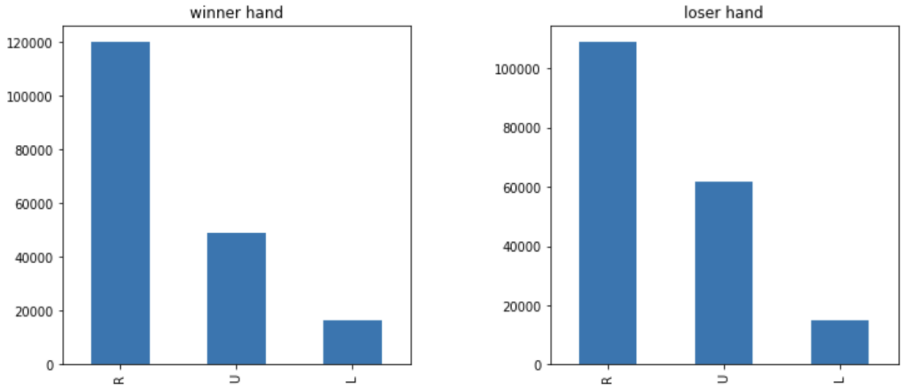
Minutes is a feature which contains many outliers (total of 1136), we think that matches below 300 minutes per match could still be real values. We decided to check for data integration in matches which last more than 300 minutes.

1.3 Distributions and Statistics

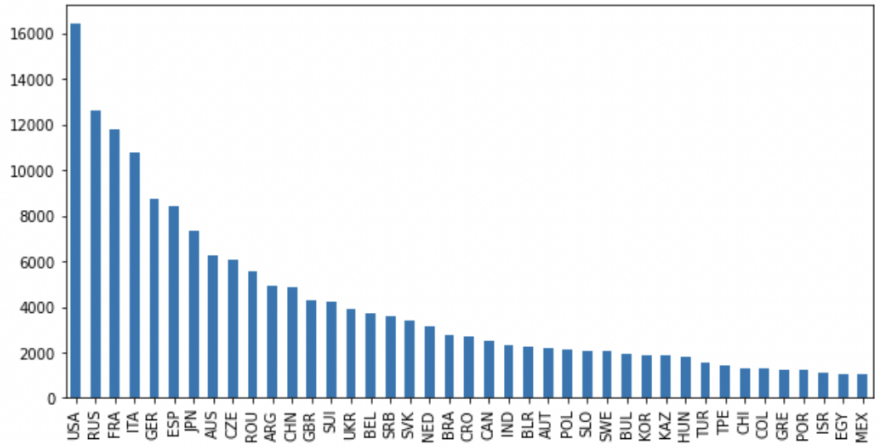
During the data understanding process is important to explore the data distributions of our attributes. This allows to check if our data follows an expected behavior. For example, we report the players' age distribution that follows a skewed distribution centered around 20-30 years old. This data distribution follows the expected behavior.



Another simple and useful example of data distribution that we want to show is the winner player's hand.



The data distribution follows our expected behavior: the preferred players' hand is the right one. A lot of matches does not report the player's hand, these cases fall in the U category. This is a signal of bad quality data. In our opinion data labelled as *"U"* can not be fixed by analyzing hand used by the same player in other matches, because, even if this is a very rare case, we can not state that a player uses always the same hand.



As last example about data distribution we report the winners' nationality distribution. This is a positive skewed distribution: bigger and more tennis-active countries have a lot of won matches.

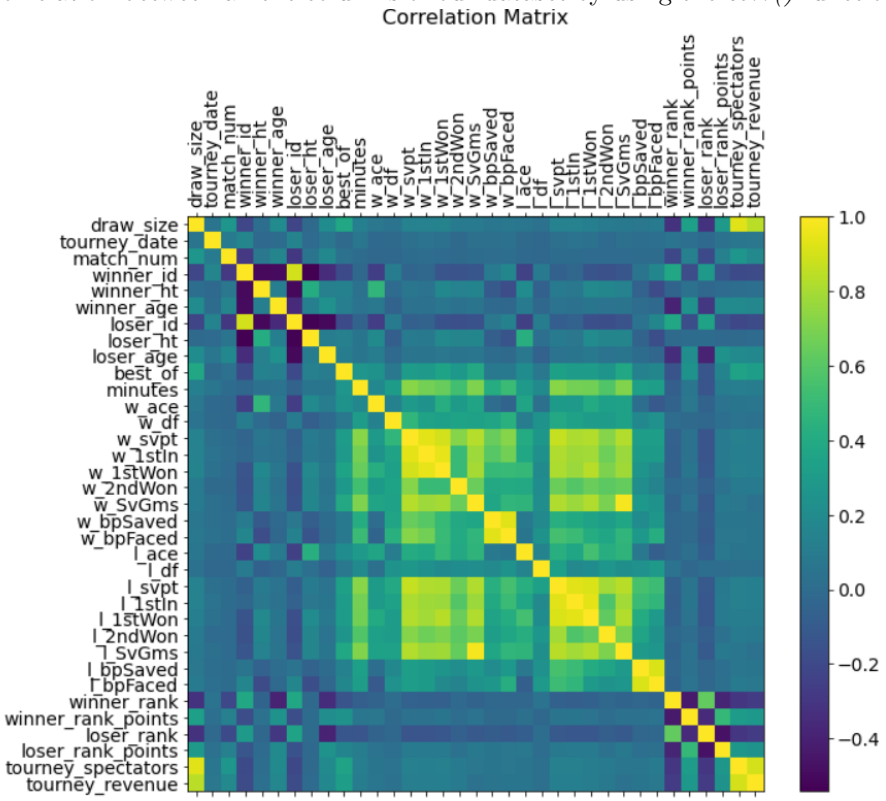
An interesting statistical tool is provided by pandas' describe method. By analyzing the function output we were able to detect some wrong values. In the following table we reported the most interesting output columns.

	winner_ht	winner_age	loser_ht	loser_age	minutes	tourney_spectators	tourney_revenue
count	49341.000000	183275.000000	38348.00000	179590.000000	81660.000000	186101.000000	1.861020e+05
mean	181.407106	23.963517	181.56308	23.765932	97.675753	4108.569153	8.226442e+05
std	11.630899	4.462318	10.81565	4.629857	41.492701	2707.042984	6.008570e+05
min	2.000000	14.042437	2.00000	14.006845	0.000000	91.000000	1.786574e+04
25%	175.000000	20.492813	175.00000	20.131417	72.000000	2836.000000	5.473662e+05
50%	183.000000	23.457906	183.00000	23.227926	91.000000	3340.000000	6.633297e+05
75%	188.000000	26.869268	188.00000	26.767967	119.000000	4008.000000	8.340290e+05
max	211.000000	95.000000	211.00000	74.485969	4756.000000	18086.000000	5.002794e+06

We can immediately notice some already discovered wrong values like the minimum value of 2 in winner and loser height or the maximum value of 95 in winner age.

1.4 Correlations

Correlation is used to identify linear relationships between attributes. If two attributes are highly correlated one of the two can be dropped, since it can be easily derived by using the other one. In this case we compute the pearson correlation between all the columns of our dataset by using the *corr()* function.



We noticed high correlation levels focused on the match stats attributes like the number of serve points, the number of aces etc..
There are high correlation values between `tourney_spectators`, `tourney_revenue` and `draw_size`. This is reasonable since many players typically means bigger tourney and therefore more spectators and revenue. In our opinion, even if high correlation values are reported for these attributes, their semantic is too far away from each other to linearly combine them in a unique attribute. Furthermore there are few rows in which they are all specified, resulting in creating many NaN values in case of a linear combination.
The less correlated attributes seems to be `winner_id` and `loser_id` which uniquely identify players.

2 Data Preparation

During this phase we used the information obtained during the data understanding phase to treat missing values, outliers, wrong values and improve the data quality. We tried to integrate data coming from different sources when possible in order to fix wrong values that cannot be inferred by looking only our dataset. In this section we will describe new features and the profiles dataset in which we collected the information for each player. These new data can be used for deeper analysis like clustering or classification.

2.1 Duplicates

The first thing we did was removing the duplicates from the dataset, which brought noise. We applied the function `drop_duplicates` to all the three dataset.

2.2 Wrong and Missing Values

We fixed wrong and missing values by examining one by one all the attributes.

In particular, the wrong and missing values were adjusted by applying data integration or, when it was possible, by using other rows of the dataset in order to obtain or to fix the values.

In the following list we report some of the main attributes analyzed and how the wrong and missing values have been treated:

- `tourney_id`: We found 55 missing values, so we first tried to fix them using other rows with the same *tourney_name* and *tourney_date*, in this way we took the missing values from 55 to 38. Then we tried to fix the remaining ones using the *tourney_revenue* and the *tourney_date* attributes. At the end we got only 27 lines with id equals to NaN.
- `tourney_name`: we tried to fix the NaN values by using the *tourney_id* and *tourney_date* attributes, but without success.
- `tourney_date`: we tried to fix the NaN values by using the *tourney_id*, *tourney_name* and *tourney_revenue* attributes, but we were not able to fix any of the null values.
- `surface`: we checked if there was some rows with *surface* that is NaN and with *tourney_id* defined. By looking to rows with same *tourney_id* and not-null *surface* we managed to fix 9 lines.
- `draw_size`: we checked if there was some rows with *draw_size* that is NaN and with *tourney_id* defined. By looking to rows with same *tourney_id* and not-null *draw_size* we managed to fix 10 lines.
- `tourney_spectators`: we checked if there was some rows with *tourney_spectators* that is NaN and with *tourney_id* defined. By looking to rows with same *tourney_id* and not-null *tourney_spectators* we managed to fix 8 lines.

- `tourney_revenue`: we checked if there was some rows with *tourney_revenue* that is NaN and with *tourney_id* defined. By looking to rows with same *tourney_id* and not-null *tourney_revenue* we managed to fix 7 lines.

  Then we analyzed all players related attributes. We immediately discovered that different players name as the same *player_id*, so we need to pay attention if we want to replace some player related attribute by searching for other rows with same *player_id* in which that attribute is specified.

- `winner_name/loser_name`: as we mentioned before, there are different players who have the same player_id, so we decided to consider also the name as unique identifier for the players. Unfortunately, there weren't enough information to replace the missing values in *winner_name*. Instead we managed to fix all missing values in *loser_name*. Furthermore we implemented a data cleaning process for *winner_name* and *loser_name* attributes since many values as some alphanumeric codes appended to the end of the string value.

- `winner_ioc / loser_ioc`: All values were fixed by using *winner_name / loser_name* and *winner_id / loser_id* attributes.

- `winner_ht / loser_ht`: we checked if there are some players who do not have height in all their matches rows, and there were, of 10104 unique names, only 541 with a values not null. They are very few and we do not think they are enough to create meaningful statistical features (i.e. average height for player of the same gender and nationality) in order to replace the missing values.
  We integrate data from different data sources for wrong data found in data understanding section.

- `winner_age / loser_age`: We integrate data from different data sources for wrong data found in data understanding section.

- `name / surname`: we fixed all the wrong values also in the male and female dataset, deleting alphanumeric codes and removing Mr/Mrs. We tried to recover real full name for rows in which name or surname is specified with the initial letter by looking other rows in the two dataset but no full name was found.

Concluding our analysis about wrong and missing values we decided to remove all the rows in which neither *winner_name* and *loser_name* are defined.

2.3 Outliers

In this section we described the operation that we applied in order to investigate more in deep data analyzed in outliers detection during the data preparation.

We have already analyzed (and in some case fixed) outliers detected in the following attributes: *winner_ht*, *loser_ht*, *winner_age*, *loser_age*, so we started

investigating **minutes outliers** (values greater than 300 and values equal to 0).

- (minutes $>$ 300): by applying data integration we were able to find real minutes played in these matches and we decided to replace them.
- (minutes == 0): in case the played minutes is equal to 0 we checked the match score. If the match lasts 0 minutes and it has the *score* == W/O (walk-over) or the *score* == "0-0 RET" it is true that the match lasts 0 minutes and we do not need to replace these values. Instead, matches which have a valid score (some sets were played) and played minutes equals to 0 must be fixed. We decided to replace their minutes value with the mean of the matches' played minutes.

Analyzing match statistics outliers is more complicated since we do not know in which range these values typically reside and it is not possible to derive the real values from other matches. We analyzed each attribute and by using data integration we tried to correct as many wrong values as possible.

We started analyzing for each attribute the values that are very far away from the mean value by integrating data from different data sources. We quickly discover that some matches have almost all attributes altered. We decided to remove these matches by dropping the rows.

2.4 New Features

Once we finished the data cleaning process we introduced new features in the dataset. In particular, in order to create players profiles, we created following features:

- `gender_w / gender_l`: a string $\in$ ['M', 'F'] which indicates the winner/loser gender. Computed joining matches datset with male and female dataset.
- `season`: a string $\in$ ['Winter', 'Summer'] which indicates the season in which the game has been played.

Joining male and female dataset with matches lead to some names in matches that do not find a match in the male and female dataset (since our player profile dataset, that we are going to create in the next section, is built from matches dataset we were not interested in names defined in male and female dataset which are not present in the matches dataset).

We found that these values are typically mis-typed version of the same name so we fixed all the names and we added the real gender.

2.5 Profiles

In this section we will describe the new dataset that we have built from matches dataset. The aim of all these new features is to extract and summarize information from all the matches and grouping them by player.

Once the set is built we can apply a set of different nature algorithms with the goal of extracting extra knowledge from our data.

The attributes we have created for the new dataset are the following:

- `fullname`: fullname
- `gender`: gender
- `height`: last known height of the player
- `hand`: mode of the used hand in matches
- `birth`: player's year of birth
- `won_matches`: number of won matches
- `lost_matches`: number of lost matches
- `matches`: number of matches
- `percent_won`: percent of won matches $\frac{won\_matches}{matches}$
- `spectators`: average number of spectators
- `country`: nationality of the player
- `minutes`: average played minutes per match
- `rank`: average rank position
- `best_season`: season (Summer/Winter) in which the player won more matches
- `best_year`: year in which the player won more matches
- `best_surface`: surface on which the player won more matches
- `bp_save_ratio`: average ratio of saved breakpoints $\frac{bp\_saved}{bp\_faced}$
- `svpt_won_perc`: avarage number of served points won $\frac{1stWon+2stWon}{svpt}$
- `aces`: average aces scored per match
- `df_perc`: average double faults made

3 Clustering

First of all we decided to filter the attributes and do the clustering on a limited number of attributes, the most significant ones. In our opinion, they are: `rank, matches, svpt_won_perc, df_perc`

3.1 Pre-Processing

Before executing a Clustering process the data must be pre-processed by eliminating outliers and normalizing them. To prepare our data, since the profiles dataset already has no outliers, we only dropped all the rows with NaN values. Regarding normalization, we applied two different type of it: **min-max normalization**, which normalizes the data in the range [0,1] and **z-score normalization** which normalizes based on the mean and the standard deviation.

3.2 K-means

K-means is a *Partitional Clustering* approach, where the number of clusters need to be specified. All clusters have a center point, and each point is assigned to one of them, the closest one, by Euclidean distance. Typically, the centroid is the mean of the points in the cluster.

3.2.1 Find the best value K

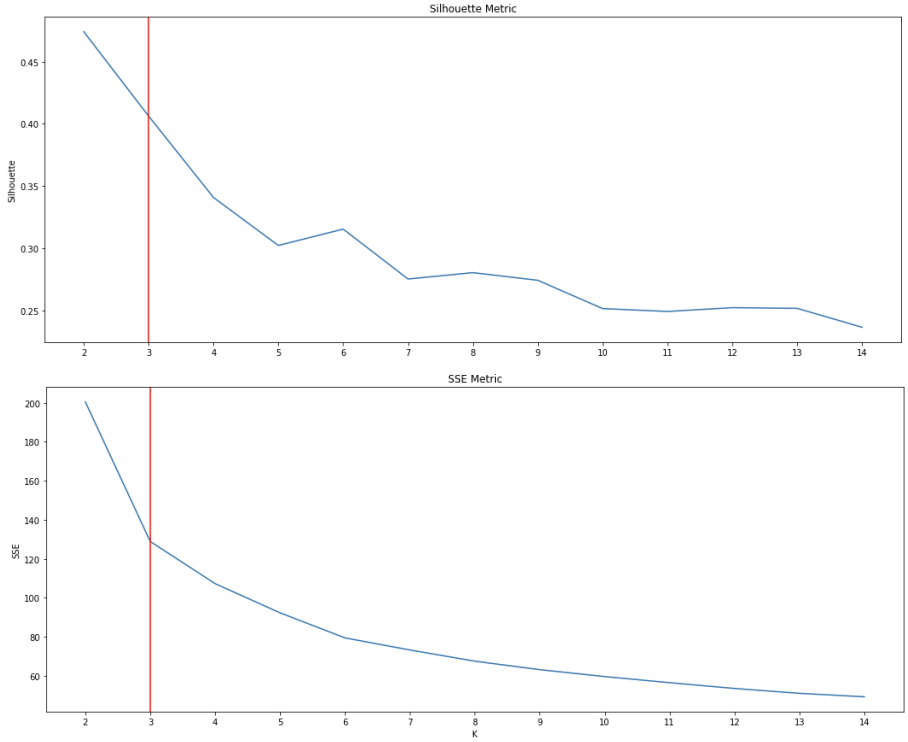
Find the right number of clusters is the most important aspect of K-means. The best way to find the right parameter K is using complementary evaluation techniques rather than one, so we have chosen these two techniques:

1. Silhouette coefficient: It quantifies how well a value fits into its assigned cluster based on two factors: how close the data point is to other points in the cluster and how far away the data point is from points in other clusters. The values of the coefficient goes from -1 to 1.

2. Sum of Squared Error: it is the sum of the squared distances between each point and the cluster centroid. Given two sets of clusters, we prefer the one with the smallest SSE. As the value K increases, the value SSE decreases, but we need to find a trade-off between error and number of clusters.

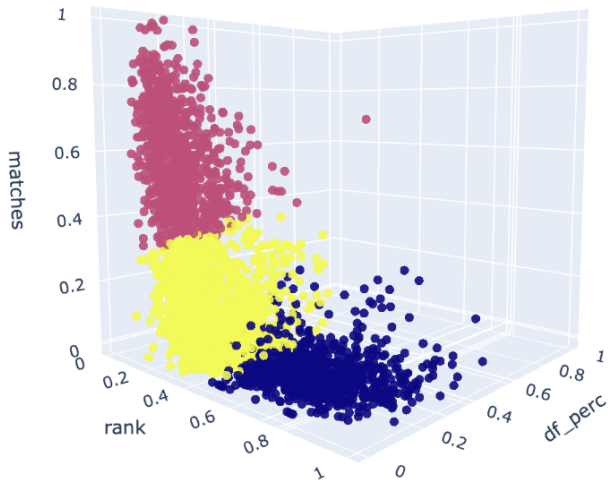
To perform the both method, we have run K-means 15 times by varying K, and at each iteration we have stored the values needed: the Silhouette coefficient and the SSE.

With the SSE and Silhouette values obtained, we created these plots, one containing the Silhouette Metric and the other containing the SSE Metric. By observing the plots, it was possible to determine the best value for K which in our case is equal to 3.



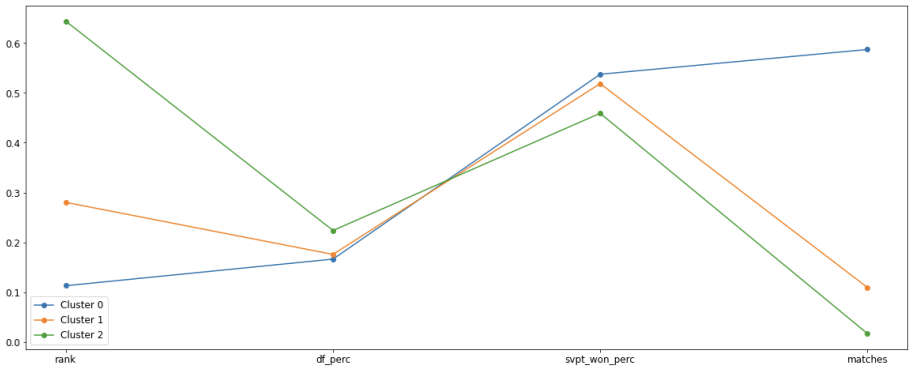
3.2.2 Results

For the implementation of the algorithm we used the `sklearn` library, and in order to report the result we plotted the dataset data points in a 3D scatter plot by reporting only three attributes.



By observing the cardinality of each cluster we can state that the result is well balanced:: `Cluster 0: 891, Cluster 1: 1387, Cluster 2: 878`

Then we analyzed the centroids of each cluster, exploiting the use of a linear plot. In this plot each line represents a cluster and the dots represent the centroids for each attribute.



The plot above shows how the three clusters represent three different types of players. The blue cluster represents the best players according to our dataset: they have a low rank, a low percent of double faults, a high number of service points won and many played matches. The yellow cluster represents the mid-ranked players with all the value between the other two clusters. Finally the green cluster represents the worst players according to our dataset. We can observe that it has a high rank and an high percentage of double faults while the number of played matches and the service points won are very low.

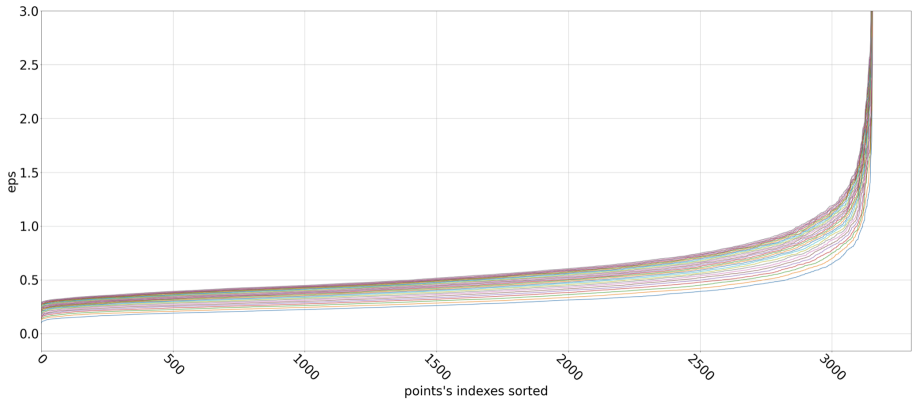
3.3 Density based Clustering (DBScan)

In our work density based clustering is implemented through DBScan algorithm. DBscan requires two inputs:

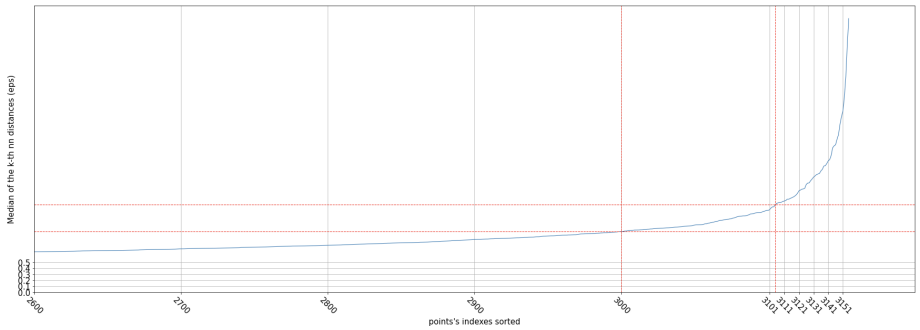
- `minSample`: the minimum number of sample that are necessary to form a cluster
- `eps`: the distance radius in which we search for at least minSample points

3.3.1 Parameters evaluation

To find the best parameters for the DBScan algorithm we started computing a matrix with the pairwise distances of all the points by using Euclidean distance as distance metric. Then we plotted a graph with the distance from kth distance neighbor of every points' indexes sorted with k from 3 to 30.



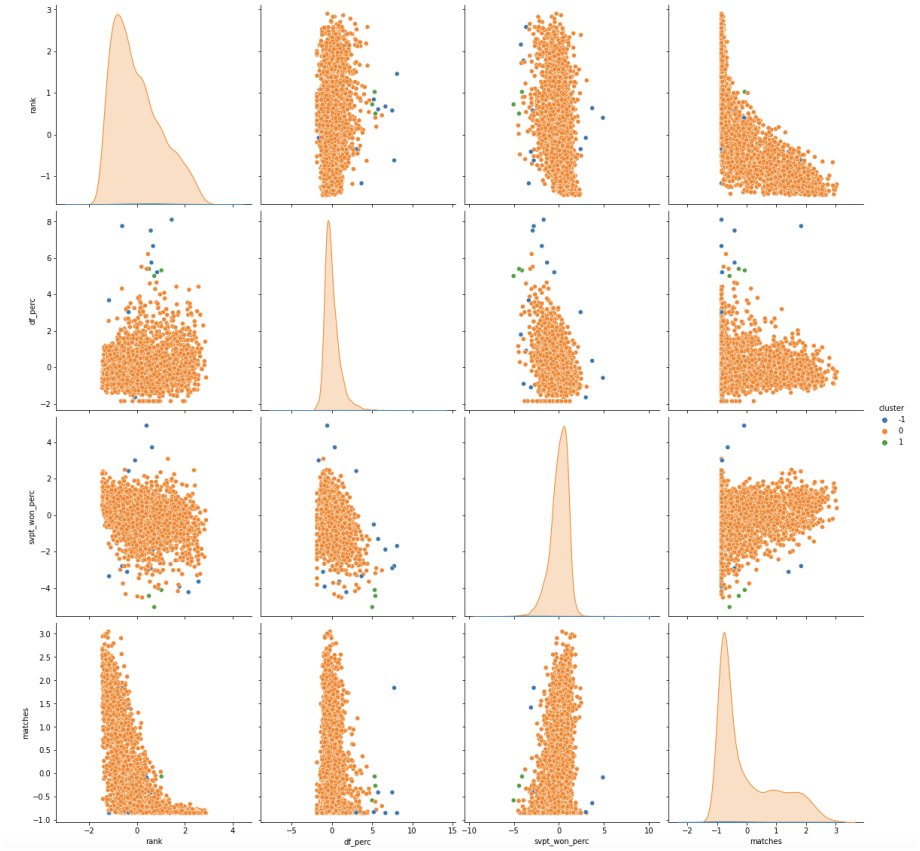
From this plot we can see that with the variation of k, the final shape of the curve does not change significantly, so we decided to plot the Median and analyze it.



In red we can see the lines that delimit the elbow created by the curve. We found that the low and high eps of the elbow are at x=3000 and x=3105, so our eps' range goes from 1,011 to 1,461. Then to get the best value in this range and to select also the best minSamples value, we applied a grid search by computing the number of clusters and the Silhuette score for each combination of minimum sample (we used a range of *min_sample* from 3 to 10) and eps distance (we discretize the range between 1,011 and 1,461 with a step of 0.015).

3.3.2 Results

Given the grid search result we want to use the highest possible value for the Silhouette score. In this case the higher value is with an `eps` between 1,417 and 1,447 and with `minSamples` equals to 6. This configuration leads to only one cluster contained all the points. So, we tried to decrease the `eps` to 1,012 and the `min_sample` to 3 obtaining the 2-cluster result reported below.



We were not particularly satisfied with the results. The algorithm did not associate enough points to the second cluster making it not usable for our purpose. We think that our data form a dense agglomerate of points leading the DBSCAN algorithm to identifies only on big cluster with some noisy points. For this reason, in our opinion the Density based clustering is not the most suitable clustering approach that can be applied.

3.4 Hierarchical Clustering

The Hierarchical Clustering produce a set a nested clusters organized as a Hierarchical tree. There are two main type of hierarchical clustering:

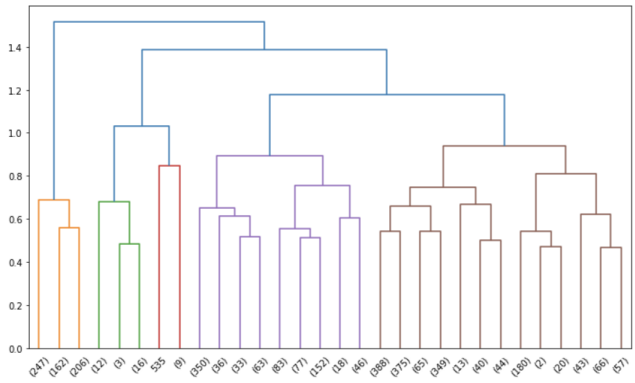
- *Agglomerative*: starts with the points as individual clusters and at each step merge the closest pair of clusters until only one cluster left.
- *Divisive*: starts with one, all-inclusive cluster and at each step it split it until each cluster contains a single element.

3.4.1 Implementation

In our implementation we applied the agglomerative approach in the following way:

- Compute the distance matrix by running the `pdist` function
- We merge the clusters bu applying the `linkage` function that returns a matrix in which every line represent a merging step and the function for each step returns the clusters id merged, the distance between these two and the new cluster dimension

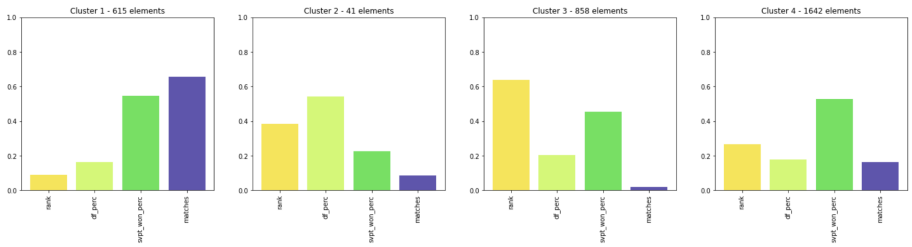
The resulting dendogram is the following:



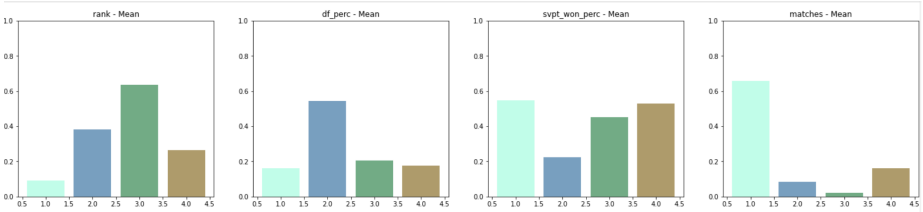
We decided to cut the dendogram at the distance value of 1.1, obtaining 4 clusters different clusters with the following cardinality: 4: 1642, 3: 858, 1: 615, 2: 41. The function `fcluster` allows us to transform the hierarchical algorithm result into a numpy array in which for every point on our dataset, the relative cluster identifier assigned by the algorithm is reported.

3.4.2 Results

We plotted the result by using two opposite representation: the first one in which for every cluster a subplot with the mean for every attribute is represented through a bar plot.

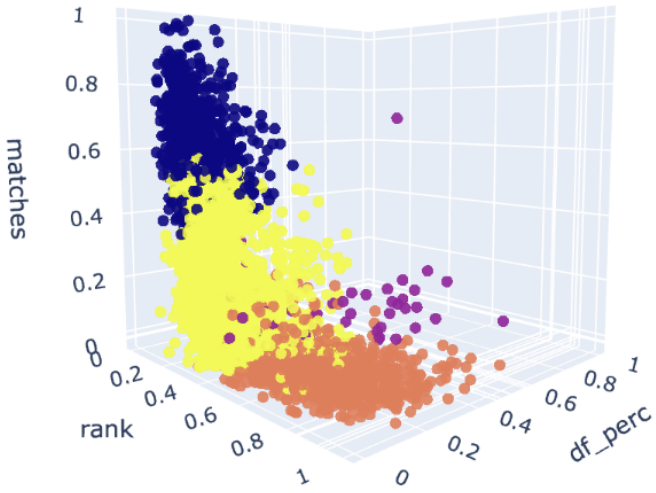


In the second representation, for every attribute in our dataset, the cluster average attribute's value for each cluster is plotted using a bar plot.



By looking the last results, it is interesting to notice how the data points have been clustered. In particular we can observe that the strongest players (with respect to our data) are inserted in the cluster 0 (the light blue one). In fact, the rank of this cluster is very low as well as the number of double faults. The service points won percentage and the number of matches is instead very high. The cluster 2 (the green one) is instead the cluster of the worst players: they have an high rank and an high number of double faults while they played very few matches and they have a low number of service point won. The last cluster (cluster 3, the brown one) represents the mid-ranked players. The cluster 1 (the dark blue one) contains very few data points that we decided not to consider during the analysis.

In the 3d plot below we can see that we obtained results similar to K-means.



4 Predictive Analysis

The predictive task of this project has the goal of classifying the players as high-ranked players and low-ranked players. We used several classification models:

- `Decision tree`
- `K Nearest Neighbour (KNN)`
- `Rule based classifier`
- `Naive bayes`
- `Random Forest`
- `Support Vector Machines (SVM)`
- `Ada Boost`

4.1 Pre-processing

During the pre processing phase we want first of all select the set of attributes on which we can rely for the prediction task. We decided to use the following attributes: *birth, gender, hand, bp_save_ratio, svpt_won_perc, ace, df_perc, matches, percent_won*.

Then, we created the label which is based on the average rank of the player. In particular, we computed the mean average rank of all the players and we label with 1 the players with the average rank above the mean and we label with 0 the others.

Once we have created the label for each player, the rank attribute has been dropped in order to prepare the dataset for the classification tasks.

4.2 Models

Decision tree

The parameters used in the decision tree classifier have been chosen by analyzing the train set and test set accuracy. The initial parameters configuration that we used lead the model to over-fitting. So, we decided to reduce model complexity by reducing the max_depth of the decision tree.

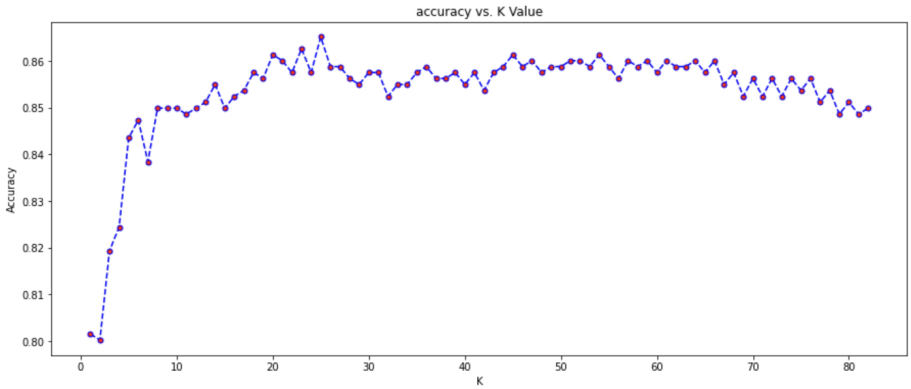
Rule based classifier

As rule based classifier we used the RIPPER algorithm. We find the best model parameters (`k` and `prune_size`) by exploiting a grid search.

Naive Bayes

We applied the `GaussianNB` model from *sklearn* library and we check the model accuracy on the training set and on the test set in order to verify that the model was not in over-fitting.

K Nearest Neighbour



Given the reported plot, we analyzed what is the best k value for our model. We also tried different algorithm and we found that the bast one is the `kd_tree`

Random Forest

Random Forest model is defined by a lot of parameters and in order to find the best one we applied a grid search. Applying the grid search result to a Random Forest model we test the model on the test-set.

Support Vector Machines

We tested different kind of kernels (e.g. *sigmoid*, *poly*..) and we found that the best kernel for our dataset is the *linear* one.

AdaBoost

AdaBoost was tested by using several type of base estimator:

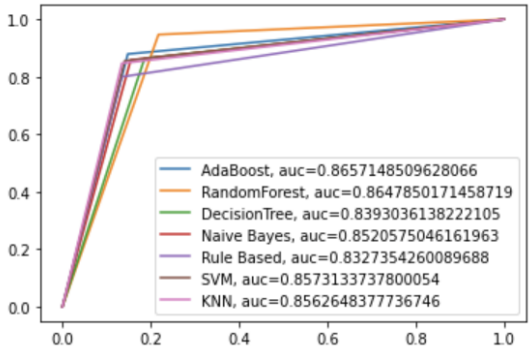
- `Default`: the default base estimator is a decision tree classifier with *max_depth* equals to 1
- `Decision tree`: our decision tree model described before
- `Naive Bayes`: our naive bayes model
- `Random Forest`: our random forest model

4.3 Models evaluation

In the following table we reported the test accuracy score for each of the model that we applied. In our opinion, since we test all the models on the same datatset split, the accuracy score is valid metric to evaluate the models.

Model	DT	KNN	Rule-based	NaiveBayes	RF	SVM	AdaBoost
Accuracy	84%	84%	84%	85%	87%	86%	87%

4.3.1 Area Under The Curve - AUC

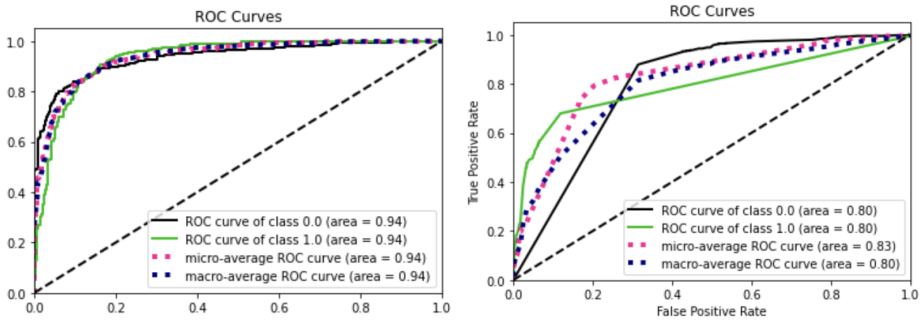


AUC provides an aggregate measure of performance across all possible classification thresholds, it could have a value between zero and one and tells how much the model can distinguish between classes. Higher the AUC, the better the model is at predicting 'high rank player' classes as 'high rank player' and 'low rank player' classes as 'low rank player'.

So, the best models are those with AUC closest to one: AdaBoost, RandomForest, NaiveBayes, KNN.

4.3.2 Receiver Operating Characteristic - ROC

In this section we compare the best model we have obtained based on the accuracy on the test set with respect to the the worst one by exploiting the ROC curve.



The above plots represent the ROC curves of the AdaBoost and Rule Based methods respectively. The ROC Curve shows you the relationship between two metrics called fall-out and sensitivity (or recall), which express respectively the False Positive Rate and the True Positive Rate of a classifier. Even if the accuracy metric between the two models is not too different, the ROC curves shows us that the input data is better classified by the AdaBoost model.

Following the analysis carried out by this report, we think the best predictive model is the AdaBoost using Random Forest as base classifier.

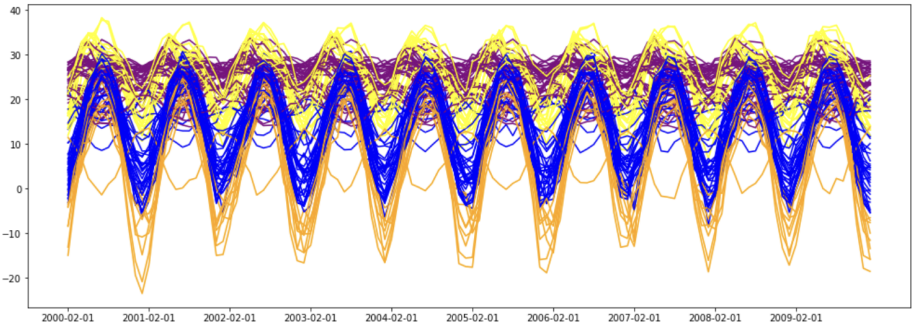
5 Time Series Analysis

Considering the dataset of time series CityGlobalTemperature2000-2009.csv containing for 100 cities the temperature measurements (mean and standard deviation over a month) we want to find groups of similar cities with respect to the temperature trends.

In order to address this problem we applied three different approach:

1. **Clustering original data-set**: given the set of individual time series data, we grouped similar time series into the same cluster;
2. **Feature based clustering**: we first created features and then used them to cluster the time series;
3. **Compression-based clustering**: we compressed time series and ran the clustering algorithm on the compressed version.

We have applied the K-Means algorithm with a K value set to 4 to all of the methods, and we look to the inertia metric, which intuitively measure how far away the points within a cluster are, in order to understand which is the best approach. By analyzing the inertia metric we obtained the best clusterization by using a compressed version of the data in input to the K-Means clustering algorithm. The compressed version of the time series is given by applying the Piecewise Aggregate Approximation.



The plot above shows the different time series contained in the dataset that are colored based on the cluster assigned to each of them. In particular we can observe that the orange cluster, for example, represent the cities that reach the lower average temperature during the year. The yellow one represents the cities in which during the year the higher temperature is reached but they have more variations during the change of the seasons with respect to the purple cluster which seems to be more stable. To give a more concrete representation, we can think the purple cities as the more tropical ones, in which the temperature remains more or less the same (typically quite high) but for example the changing of the season takes more rain.

To visualize in a more human-comprehensible the result we plot the cities contained in the dataset on a map and represent them with a color based on the cluster assigned to it.