UNIVERSITÀ DI PISA

Data Mining
Academic Year 2021/2022

Data Mining Project: Tennis Matches

Luca Santarella - Alessandro Puccia (Group 6)

Contents

Chapter 1

Introduction

The first task required to explore a data set of tennis matches in order to understand the data and to assess its quality and also to make the necessary adjustments to correct errors and abnormalities. First thing that we noticed is that the data set had several problems typical of the preprocessing phase, such as the presence of outliers, wrong or completely missing values and unnecessary attributes. After the ”cleaning” phase was over we computed indicators of the tennis players in order to create profiles for each one of them.

Next we used different clustering techniques and clustering understanding means to better understand the kind of players in our data set based on different algorithms and metrics.

The third task required to perform a classification of the players to predict if they are high or low skilled players, so we explored several models and compared their results, for most models we achieved an accuracy performance of over 94%.

The final task required to work with time series, we were given a data set of temperatures of one hundred cities for the period going from 2000 to the beginning of 2010, so we performed clustering on this data and obtained four different clusters one for every kind of climate.

Chapter 2

Task 1: Data Understanding & Data Preparation

2.1 Data integration

We were given three separate csv files:

- **tennis_matches.csv**: a CSV (Comma-Separated Values) file containing matches of tournaments from January 2016 to August 2021, [186128 rows x 50 columns]
- **male_players.csv**: a CSV file containing the names and surnames of male tennis players, [55202 rows x 2 columns]
- **female_players.csv**: a CSV file containing the names and surnames of female players [46166 rows x 2 columns]

One of the first things done was the data integration of the three data sets into a single one. First of all we merged name and surname and dropped the duplicates inside the data sets about the players, then we added a column with the sex ('m' for male and 'f' for female). The two datasets were then merged into a single one containing all male and female players.

Next two left join were performed using the names of the winner and the loser as key, the result was a final data set containing the tennis matches with the addition of the sex for each player.

There were also some players in the matches file that were not listed in the players file and so after the join operations their sex was missing. Given the fact that the missing values were few we decided to update them by using an external source (WTA and ATP official website).

2.2 Data Quality

In this phase we assess the data quality of our data set and look for errors or inaccurate data such as:

- **missing values**: we selected a list of attributes that were considered essentials to describe the performance of a player in a match and then dropped all the matches that had all these attributes as missing values. This was necessary because we noticed how using interpolation algorithms for such amount of missing data lead us to strange behaviours in the following tasks.
- **syntactic and semantic accuracy**: we discovered valid values that were not included in the domain specified by the requirements, such attributes are:
  - *tourney_level*: which had the values 'W' which stands for WTA (Women Tennis Association) tournaments and 'O' which stands for Olympics
  - *winner_entry*/*loser_entry*: which had the values 'SE' which stands for Special Exempt, 'Alt' which stands for 'Alternate', 'SR' for 'Special Ranking', 'JE' for 'Junior Exempt', 'Jr' for 'Junior', 'IR' and 'P' for 'ITF reserved acceptance'. The domain has been expanded to also include 'D' which stands for 'Default' in case one player is not included in a special category.

  We also discovered that the data set had extra attributes which were not described, which are:
  - *score*: the result of the tennis matches set represented as a string object.
  - *round*: specify the round of the match that was played ('F', 'SF', 'QF', 'R16', 'R32', 'Q1', 'Q2', 'Q3', 'R64', 'R128', 'RR', 'BR').
  - *tourney_spectators*: total spectators of a particular tourney.
  - *tourney_revenue*: total revenue of a particular tourney.

The attribute score was processed and then dropped to get other attributes that describe the number of games won (*w_games* and *l_games*), sets won (*w_sets* and *l_sets*) and the total amount of games played (*total_games*). Processing this attribute we found that some matches were not played due to forfeit or withdraw before the start of the match (*W/O* or *Walkover*), some other were played but a disqualification occurred (*DEF* or *Def.*) and some matches contained in the score attribute the word *RET* (this means that the loser gave up the match). We decided to discard the matches in the first case because they missed the important information about the performance of the players (in these attributes all the other values were set as 0) while for the other two cases we decided to keep them because they were played for a bit and so they participate in the measurement of the performance of a player.

We dropped the complete duplicate rows and checked that every column has the appropriate type. In particular, only the attribute *tourney_date* is represented as a float64 type and so we converted it to datetime format.

Some features (*match_num*, *tourney_id*, *loser_id*, *winner_id*) were discarded because we found them unnecessary for the following operations. We removed also the semantically redundant features (e.g. ace has two attributes *w_ace* for the winner and *l_ace* for the loser) doubling the number of rows but almost halving the number of dimensions.

To create a significant profile in the following task, we decided to drop the matches of players that have played less than 5 matches. With this operation we cut the number of players that will be used next but this allowed us to obtain more meaningful results.

Next we analysed how the matches are distributed by considering the number of matches that were played by female/male players and their favorite type of hand (see 2.1). We also found an unbalanced number of matches with regard to the nationalities of the players.

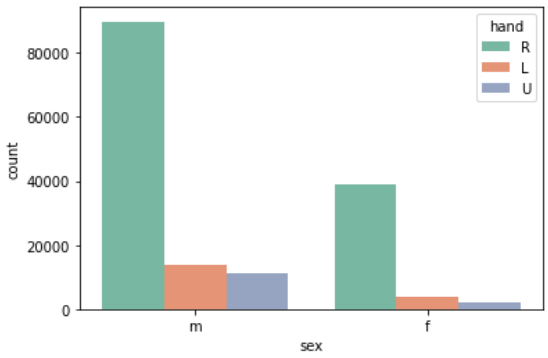


Figure 2.1: Count of matches by sex grouping by type of serving hand
(*R*: right hand, *L*: left hand and *U*: unknown).

The attributes for height and age had inconsistent values in the dataframe, this was due to the nature of the attribute that is expressed in relation with the tournament date. The matches of the tournaments are held in a timeframe of five years so the age will increase accordingly whereas the height (especially in the younger players) will increase too. We decided to take the maximum value in the dataframe for the age and the height which represents the most updated one.

2.3 Outliers

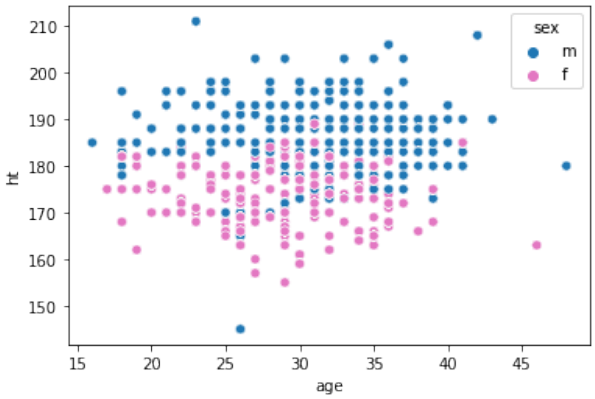


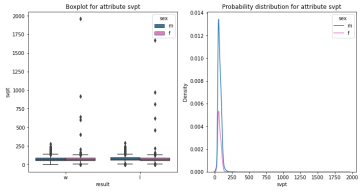
Figure 2.2: Outliers for age and ht grouped by sex

We only handled 1D-outliers by deciding to drop them only when we found some extremely strange situations (e.g. the minutes played or service points scored), in other situations (like for the height) we decided to fix them by considering external sources (WTA and ATP official website).

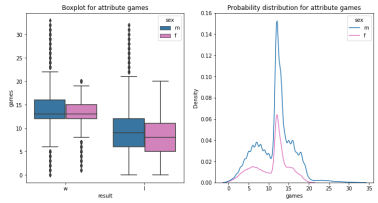
To help us in the process, we used boxplots and probability distribution plot to see graphically what is happening considering female/male and wins/losses.

During these analyses we found the following interesting situations:

- matches that were played (all the attributes like games won, service points scored etc. are different than 0) but have a match length equal to 0. In this cases we set the attribute minutes with the mean computed by considering all the other matches.
- matches with duration in minutes too high even considering the presence of best of 5 matches. In this case we set an upper bound of 300 minutes (6 hours) and dropped the matches where the attribute minutes is higher.
- for attributes like *totalGames* and *games* (games won) we found matches with a lot of games played/won and matches with few games played/won as we can see in figure 2.3b. These are for two reasons: the first case is due to head-to-head matches where there is an interleaving of sets won/lost for each player, the second one is due to the presence of matches where one of the two players retired during the match (if this happens in the first sets then we got less games played). In both cases we decided to not drop them because they refer to matches that were regularly played.
- considering the attribute *svpt* in figure 2.3a we found matches where it was set as 0 (and also the other features equal to 0) so we decided to drop them. Other matches had these attribute with value that seemed too high considering also the amount of minutes played, as before we set a threshold and dropped the matches with an higher amount.
- we did not found extremely strange situations in the remaining attributes and so we did not modify or drop these matches. This was also done to not disrupt possible patterns between strong and weak players thus we did not modify or drop outliers for the new metrics.



(a) Outliers for the attribute *svpt*.



(b) Dense outliers considered by IQR for the attribute games.

Figure 2.3: Outliers analysis for the attribute *svpt*.

2.4 Missing values

The tennis matches dataset presented various attributes which had missing values, in order to deal with these we used various techniques. First we established a set of attributes which deeply characterized the matches and thus are considered

important to do any type of analysis, then we used this set of features to drop all the rows which had missing values for these set. A total number of matches were deleted in this way.

Next we managed the remaining missing values in three main ways, which are:

- Handling missing values of static attributes: some features of the tournaments are static values which do not change and so we replaced the missing values taking the values from other matches from the same tournament, these attributes are: *tourney_name*, *tourney_date*, *tourney_level*, *tourney_spectators*. Furthermore, there were also static attributes for the players which had missing fixed in the same way as the tournament and also by integrating with an external source (WTA and ATP official website), these attributes are *ioc*, *hand*, *rank* and *ht*.
- Handling missing values which have default values, i.e. attributes like entry and rank had default values which are respectively ”D” which stands for ”Default”, meaning that the player did not have a particular type of entry and ”-1” for the feature rank, meaning that the rank has not participated in one of the 19 major tournaments.
- Handling missing values using central tendency: for values such as *surface*, *round*, *hand*, *sets*, *games* and *totalGames* we used the mode which we believed was better suited as replacement, whereas the values of the attributes ht and age were replaced with the mean which was computed separately for males and females.
- Handling missing values using interpolation: attributes such as *minutes*, *ace*, *1stIn*, *bpFaced*, *bpSaved* etc. We proceeded by first computing the interpolated data by using a linear, polynomial of second order interpolation and nearest neighbour interpolation, as well as substitution using mode and mean, then we analyzed the distributions of the interpolated and the variation in percentage associated with them. In the end we always chose the linear interpolation.

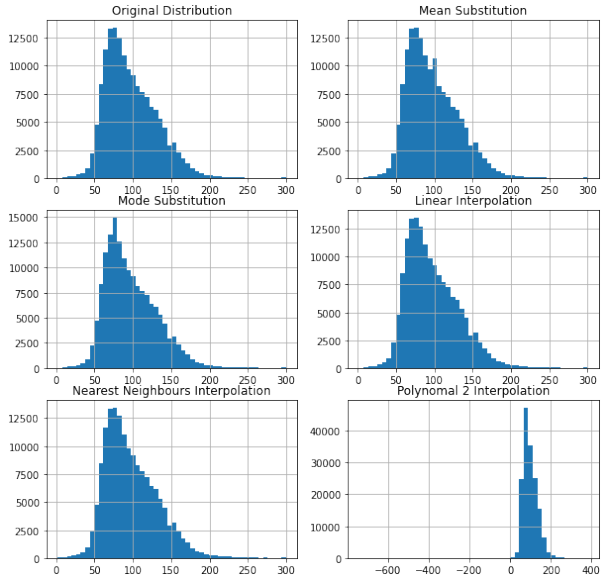


Figure 2.4: Handling of missing values using interpolation for attribute *minutes*

2.5 Creating a new dataset

In order to find players profiles, we created a new dataset where we added some categorical attributes and computed several metrics that were changed during the project. At the end, we decided to compute for each of the players:

- *ioc, sex, hand*: nationality, sex and serving hand of the player.
- *rank*: rank standing of the player computed as an average.
- *ht* and *age*: height and age of the player.
- *tourPlayed*: amount of tournaments played.
- *maxAceSvpt*: for each tournament we sum the number of aces scored and divide them by the amount of service points, then we get the maximum.

$$maxAceSvpt = \max\left(\left(\sum_{i=1}^{tourPlayed} \frac{ace_i}{svpt_i}\right) \times spect\_weight\right) \tag{2.1}$$

- *avgDfSvpt*: for each tournament we sum the number of double faults and divide them by the amount of service points, then we get the weighted average.

$$avgDfSvpt = \frac{\left(\sum_{i=1}^{tourPlayed} \frac{df_i}{svpt_i} \times spect\_weight_i\right)}{\sum_{i=1}^{tourPlayed} spect\_weight_i} \tag{2.2}$$

- *1sv2svWon*: for each tournament we sum the number of 1st serves won and the number of 2nd serves won, then we divide them by the amount of service points. Finally we consider the weighted average.

$$1sv2svWon = \frac{\left(\sum_{i=1}^{tourPlayed} \frac{1svWon_i + 2svWon_i}{svpt_i} \times spect\_weight_i\right)}{\sum_{i=1}^{tourPlayed} spect\_weight_i} \tag{2.3}$$

- *bpS_bpF*: for each tournament we sum the number of break points saved and we divide them by the amount of breakpoint faced, then we consider the weighted average.

$$bpS\_bpF = \frac{\left(\sum_{i=1}^{tourPlayed} \frac{bpSaved_i}{bpFaced_i} \times spect\_weight_i\right)}{\sum_{i=1}^{tourPlayed} spect\_weight_i} \tag{2.4}$$

- *win_ratio*: win ratio of the player by considering all its matches. We decided to not consider the year because not all the players have played in all these 6 years.

In particular *maxAceSvpt*, *avgDfSvpt*, *1sv2svWon* and *bpS_bpF* are all indicators that are weighted by considering the feature *tourney_spectators* as an estimator of the importance of the tournaments where the player has participated.

For the attribute *bpS_bpF* we found for certain players that they have never faced a breakpoint so for them we got a missing value. We decided to drop these players because we noticed that using a default value (like "-1") lead us to strange behaviour in the following tasks especially for clustering. As a result we got a dataset containing a total of 1971 players.

Next we analyzed the correlation between these new numerical indicators. We didn't found extremely high correlation, as we can see in figure 2.5 the highest ones were between *maxAceSvpt*, *tourPlayed* and *1sv2svWon*, *bpS_bpF* but given the fact that these are semantically different attributes and since we wanted to capture all the performance of a player we decided to keep them.

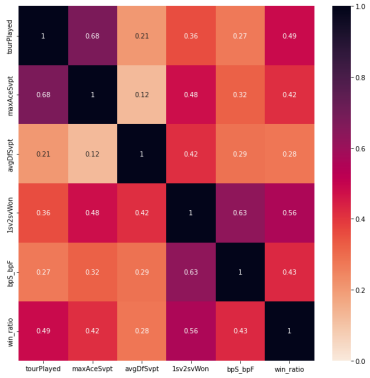


Figure 2.5: Correlation for the newly defined attributes.

2.6 PCA

Before using PCA we normalized the numeric data (excluding *ht*, *age* and *rank*) using a MinMax scaler. We start by considering the explained variance ratio for different number of components to try to understand what is the optimal number of components to use that allows to lose less information and obtain some meaningful plot.

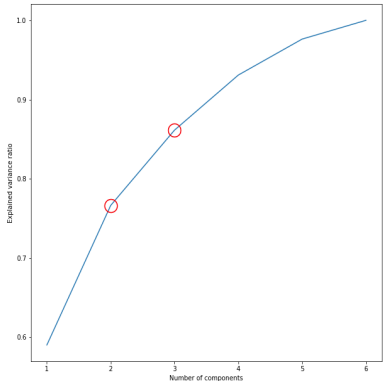


Figure 2.6: Explained variance ratio for PCA, highlighting the results for 2 and 3 components.

As we can see in figure 2.6, with 3 components we lose little information, nonetheless plotting a 3D scatterplot of the PCA components we didn't get particular information about the data that could guide us for the next task.

Chapter 3

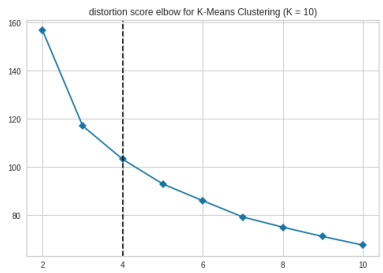
Task 2: Clustering analysis

For all the following algorithms, we used these attributes for clustering: *tour_played*, *maxAceSvpt*, *avgDfSvpt*, *1sv2svWon*, *bpS_bpF* and *win_ratio*. All these attributes were normalized using a MinMax scaler.

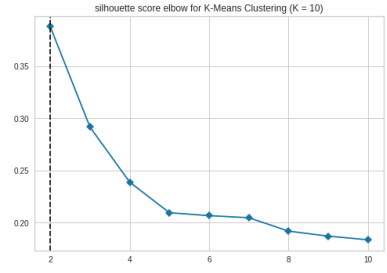
3.1 K-Means

The first algorithm considered is K-Means, it needs the number of clusters to be found and then in each step tries to identify them with a centroid and assign each point to the cluster which has the closest centroid.

The main metric that was considered is the Silhouette, an estimator of how clusters are dense and well separated. Other metrics were computed like SSE (measure of variation within a cluster), Calinski-Karabasz (estimates how well the clusters are defined), Davies Bouldin (estimate the similarity as a distance between clusters). For each of them we use the elbow method for K in range between 2 and 10. In figure 3.1a there is an example of it is applied for the SSE metric whereas in figure 3.1b we can notice how the silhouette decreases while increasing the value of K.



(a) Elbow method applied for the SSE metric.



(b) Silhouette evaluation for different values of k.

Figure 3.1: SSE and Silhouette for K between 2 and 10

Comparing these metrics we found that the optimal number of clusters is 2. With this setting we get the best silhouette

and still a good value for the other metrics. The two clusters result to be unbalanced.

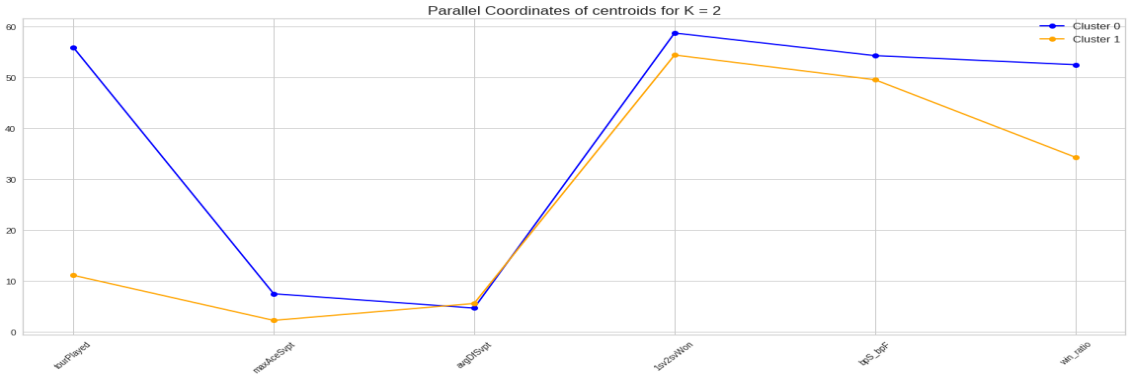


Figure 3.2: Parallel coordinates for K=2.

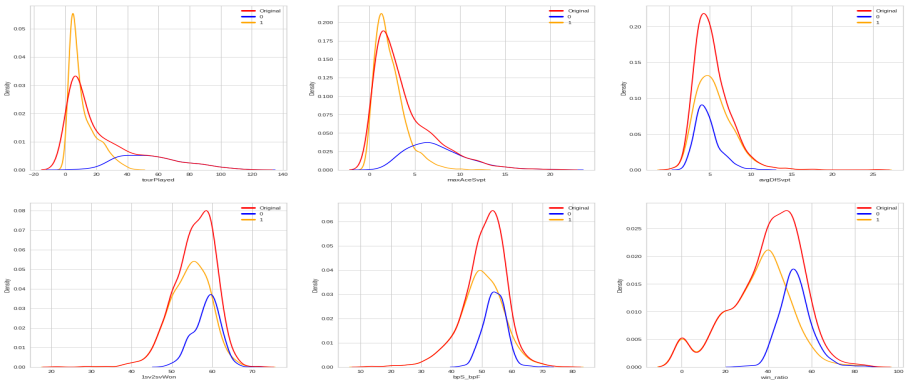


Figure 3.3: Distributions of the values for the considered attributes.

From the distribution plots in figure 3.3 and the parallel plots in figure 3.2 we can derive that *bpS_bpF*, *1sv2svWon* and especially *avgDfSvpt* seems to not contribute well to the clustering algorithm. In general, in figure 3.4 we found that the within cluster 0 we have players that have higher values for the attribute *maxAceSvpt*, *tourPlayed* and *win_ratio*. So we can distinguish the clusters in the following way:

- **Cluster 0**: it is related to *high tier* players, in particular experienced players that played a lot of tournaments and that have a win ratio above the average.
- **Cluster 1**: it is related to *low-normal tier* players.

Considering the categorical attributes we found that in the high-tier players there are more males in proportion to the low-normal tier players. In both clusters there is no much difference in terms of the type of serving hand used. Exploiting the distribution of the attribute *rank* (see figure 3.5) inside the two clusters, we can see how the cluster 0 have players with lower rank than the ones in cluster 1. For what concerns the nationality, seems there are more high-tier Spanish players in proportion to low-tier players with respect to other nationalities. After Spain, Germany and French are the most common nationalities.

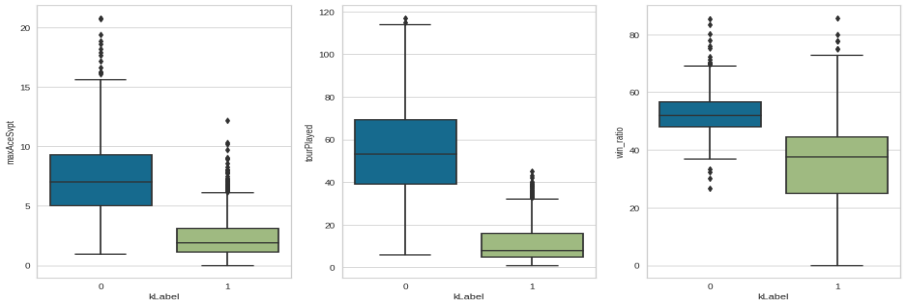


Figure 3.4: Highlighting the differences of the 3 most important attributes used for clustering.

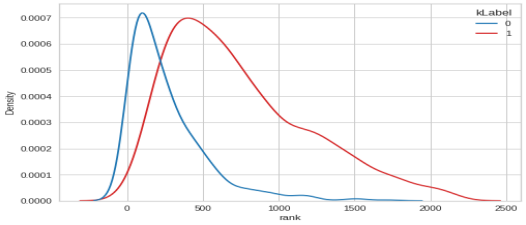


Figure 3.5: Distribution of the rank attribute within the two found clusters.

3.2 Hierarchical clustering

For hierarchical clustering we used the following methods to define inter-cluster similarity: Single, Complete, Average and Ward. All of these were applied by considering only the euclidean metric.

After that we had an insight using the dendograms and the default cut-off provided by sklearn. As we can see in figure 3.6 found out that the Single method does not seem appropriate for our problem because it only creates a big cluster, as a consequence we only concentrated on the other methods where we got 5 as the maximum amount of clusters.

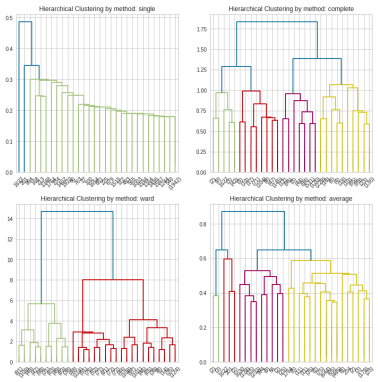


Figure 3.6: Dendograms using the main methods and euclidean metric.

Like with K-Means we try to understand what is a good value for the number of clusters by employing the Agglomerative Clustering algorithm (bottom-up) for values between 2 and 5. For all three methods we got that 2 clusters gave us the best results in terms of Silhouette and Davies-Bouldin score.

Cluster 0 refers to **low-normal tier** players while cluster 1 refers to **high tier** players. We noticed that Ward and Average methods tend to reduce a bit the unbalanceness of the size of the clusters. Differently from K-Means, we got that the attribute *win_ratio* (see figure 3.7) separate in a worse way the two clusters (especially for the complete method) whereas instead *tourPlayed* (see 3.8) and *avgDfSvpt* seems to lead us to more explainable results.

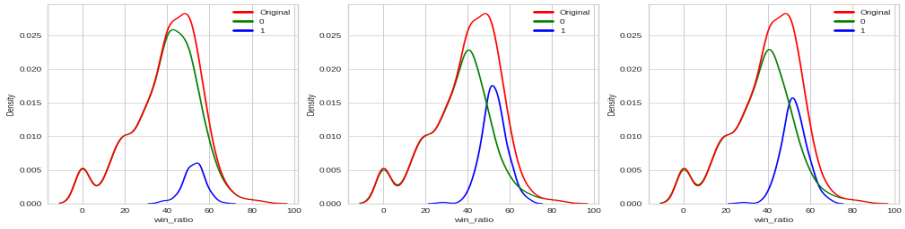


Figure 3.7: Distribution of the values of attribute *win_ratio* for the main methods considered.

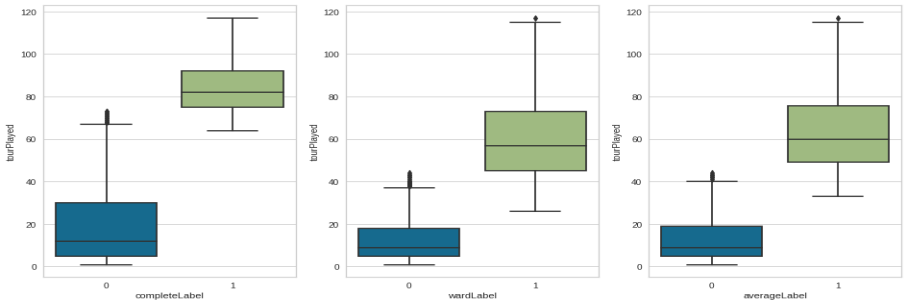


Figure 3.8: Differences of the attribute *tourPlayed* between the two clusters.

Considering the sex we noticed how using the Complete method leads us to have no females in the *high-tier* cluster while using the other methods we got a better distribution of the females in both clusters.

3.3 DBSCAN

Next we focused on density based clustering, we used the DBSCAN algorithm that considers clusters as dense region of points that are separated by lower or higher density regions. DBSCAN works well when clusters have different shapes and/or may be overlapping, we discovered that this is not the case with our dataset. We computed the distances between the points using the euclidean metric, these values will be used in the knee method to establish which are the best parameters for the algorithm. In fact, in order to have similar densities inside the clusters we search for the k-th neighbour which is more distant from the others. Since the algorithm takes in input *eps*, which is the radius of the points to be considered and *min_samples*, which is the minimum number of points to take in consideration, we first try to use knee method with k=4 and try to use eps=0.18. Using DBSCAN with these parameters we obtain 32 noise points and a single cluster of 1939 points (see figure 3.9.

We try a more elaborate search using as metric the mean noise point distance and the number of clusters, by doing this we try a grid search with values for *eps* that go from 0.05 to 0.01 and *min_samples* that go from 2 to 20. The optimal values

are similar values to the previous approach but the result now is with two clusters: one having 1931 points and the other one 3 points, the noise points are 37. The results are disappointing and we concluded that the dataset is not suited for this kind of clustering.

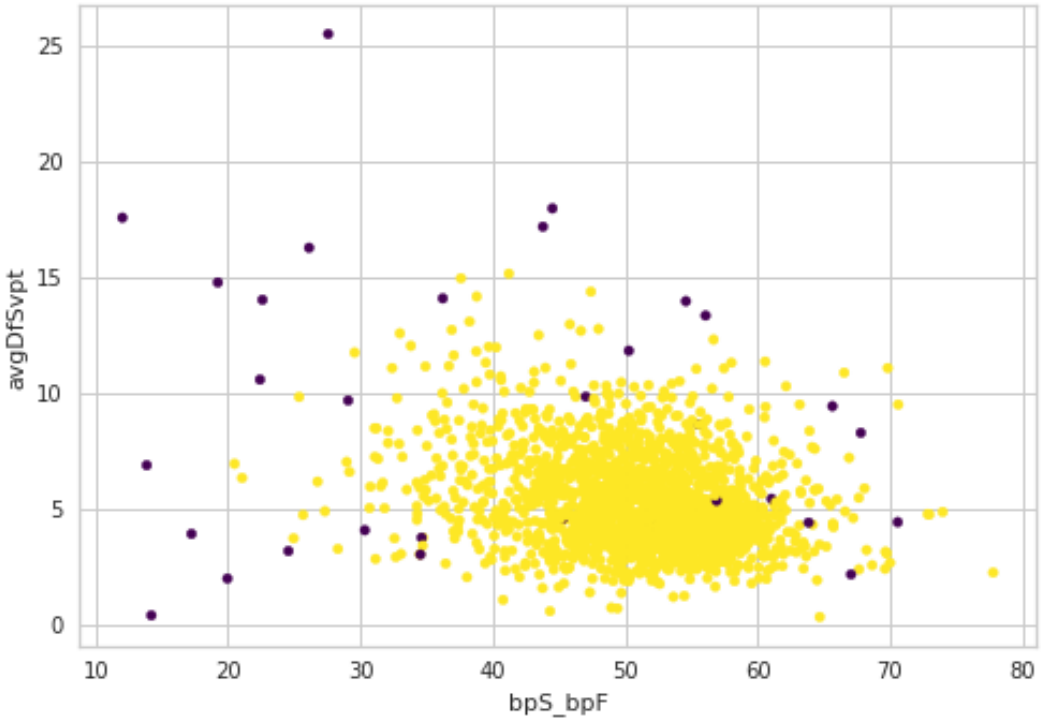


Figure 3.9: DBSCAN algorithm performed on the dataset

3.4 Extra clustering algorithms

We considered the following extra clustering algorithms provided by the pyclustering package: **X-Means**, **G-Means** and **Fuzzy C-Means**. X-Means clustering algorithm extends K-Means by trying to automatically determine the number of clusters based on BIC score as splitting criterion. G-means is also an extension of K-Means and similar to X-Means starts with a small number of centers, and grows the number of centers by performing splitting that is based on the Anderson-Darling statistical test. Fuzzy C-Means algorithm uses two general formulas for cluster analysis. The first is to update membership of each point, the second formula is used to update centers in line with obtained centers. With this algorithm we have to specify the number of expected clusters and to do so we used K-Means++.

While the first two doesn't require the amount of clusters to search, with Fuzzy C-Means we need to specify the amount of cluster to find. X-Means and G-Means provided really bad results with practically all the clusters overlapping, in particular X-Means found 7 clusters while G-Means found 138 clusters. The only algorithm that performed well was Fuzzy C-Means which had practically the same results, in terms of important attributes, as K-Means and Hierarchical clustering. We choose 2 as the number of clusters because by using higher values we got two or more overlapping clusters.

Chapter 4

Task 3: Predictive analysis

4.1 Preprocessing for classification

The binary classification task required to predict if a certain player was a high skilled or a low skilled player using the indicators computed in the previous task with also height, age.

The first thing that we needed to do was to create the label by using the rank of the players (since we had more than one value we used the average of the the matches), so we decided to consider the top hundred rank players as high skilled players (which corresponded to the 15 percentile) giving it the value 1 and the others as low skilled player with the label value 0.

This process created the targets for each instance the dataset that was used for the classification, but having only 15% of the dataset representing only one class was an issue because the data that we had was heavily unbalanced. In order to fix this we adopted an oversampling solution which generates new samples by randomly sampling with replacement the current available samples, after this process we have 45% representation for the positive (high skilled players) class and the 55% for the negative one.

The original dataset was then divided into development set (80% of the original) and test set (20%) which is kept for the final evaluation of the models in order to estimate the generalization error, the development set will have fraction of 20% used for the cross validation during the grid search for the model selection.

We explored several different classifiers and evaluated according to different metrics such as: accuracy, precision and recall on the test set, number of false positives and false negatives, AUC (Area Under Curve) value for the ROC (Receiver operating characteristic) curve, confusion matrix and the importance of the feature for decision models.

4.2 Decision Tree

The first model that we used is the decision tree classifier, this supervised model is used to predict the target (in this case if the tennis player is high or low skilled) of the instances using simple decision rules. In order to find the best hyper parameters we define an initial grid search with sparser values to find a good range to setup a second grid search with more accurate values. The grid search utility also performed a k-fold (in our case k=10) cross validation for the model selection using a validation set composed of 20% of the development set. After the second search was concluded we selected the best model according to the mean error on the ten folds and refitted the model on the whole development set, at this point we started to evaluate the model.

Figure 4.1 shows the a graphical representation of our decision tree model, the first split is based on the win ratio (separating players that have win ratio greater than 53% as high skilled players), which is logically one of the most important feature to classify an high tier player from a low tier.

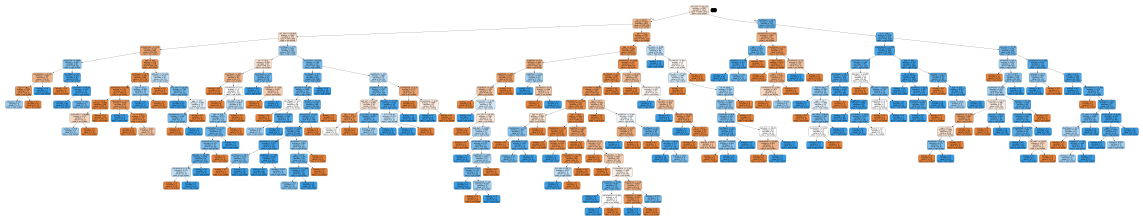


Figure 4.1: Graphical representation of the decision tree

Next we examine the feature importance for the classification task, as shown in 4.2 it is evident that the attributes *win ratio*, *tourPlayed* and *maxAceSvpt* are the most important one for the classification, we also notice that the importance of the attributes remain almost the same also for the random forest model.

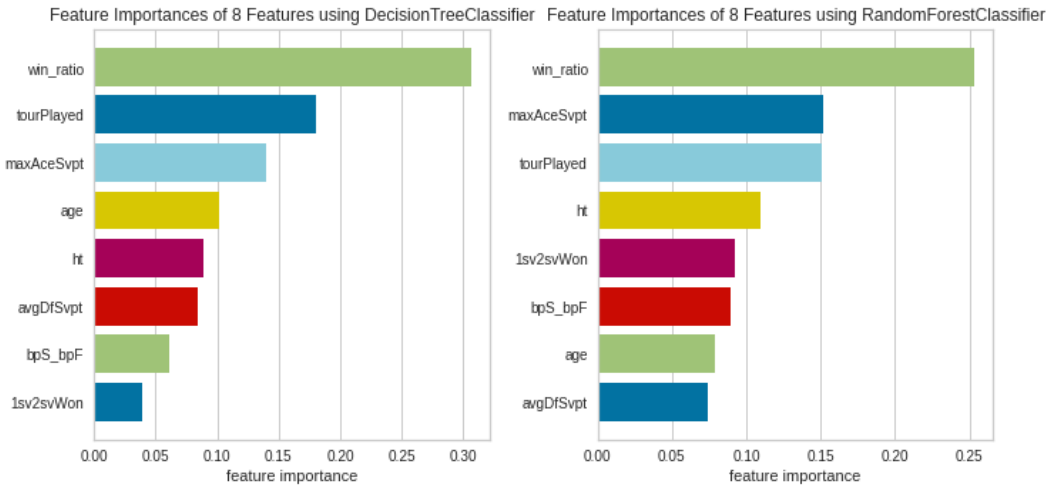


Figure 4.2: Feature importance for decision tree and random forest models

The accuracy for the decision tree on the test set is 94% and 99% on the training set. It is evident from figure 4.4 that the precision for the high skilled players (89%) and the recall on low skilled players (90%) is quite low, these two values tend

to be low also in the other models, this is caused by the high amount of false positive (10%) on the low skilled class (see figure 4.3).

We believe that for the task it is easy to classify high skilled players since they have distinctive better match statistics whereas the low skilled players fall in more categories from very good to above average players, thus a much wider spectrum w.r.t. to the top players. Furthermore, the model can be tricked into thinking that a certain player could be an high skilled player by attributes such as *win_ratio* with high values (e.g. 55%-60%) which are easier to achieve when matches are not as competitive as in the top ranks.

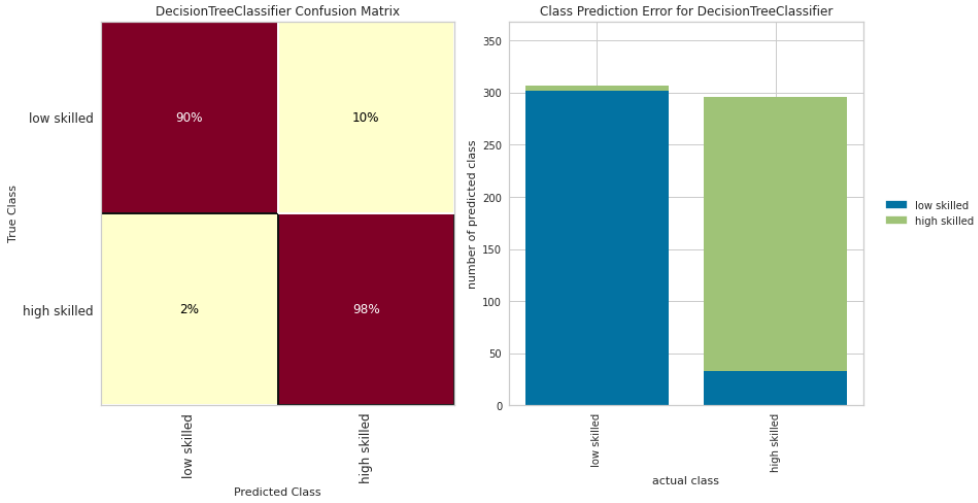


Figure 4.3: Confusion matrix for the decision tree model

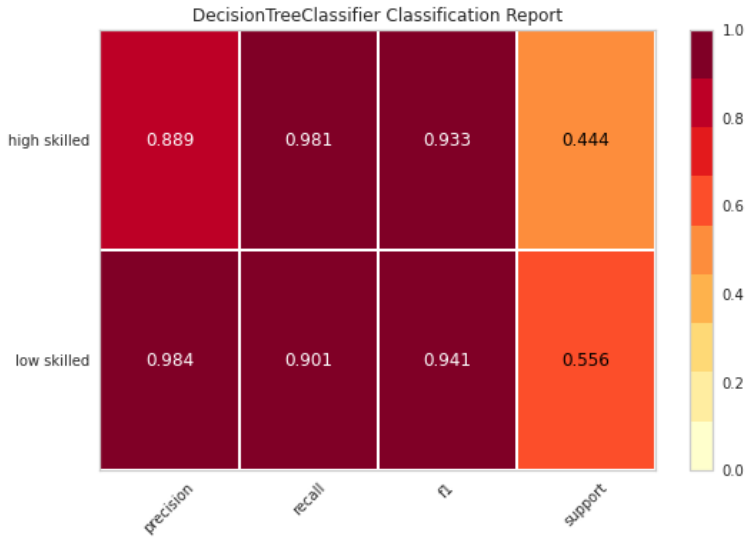


Figure 4.4: Precision, Recall, F1 measure and relative support for the decision tree

4.3 Random Forest

Next we explore Random Forest which is a type of ensemble model which uses various decision trees which are fitted on various sub-samples of the dataset, usually it improves the predictive accuracy and it helps control the overfitting. This time we proceed using a randomized search which works similarly to the grid search but it can use a continuous interval of values for the hyper-parameters. The randomized search is lighter in terms of number of computations because it is not fitting on every possible combination but just a subset.

The Random Forest model gives 96% accuracy on the test set and 100% accuracy on the training set, we can observe that the generalization error is lower w.r.t. the decision tree confirming that random forest tends to have a better accuracy but less interpretability. We also notice that this model has higher precision (92%) and recall (93%) on high skilled players, due to the fact that the amount of false positive is now lower (7%).

4.4 Naive Bayes

Next we try the Naive Bayes classifier which uses the conditional probability to estimate the outcome, the assumption is that attributes are conditionally independent given the class label of the target. We suspect that this is not the case for our dataset but we tried the model anyway and obtained poor results, the model is underfitting with only 80% of accuracy on the training set and 77% accuracy on the test set. We believe that the attributes of our dataset are not conditionally independent given the label of the target, in fact there is dependence between age and height given that the player is an high skilled one (e.g. a certain age and height can give an advantage for the player).

4.5 Rule Based classifier

Next we try the RIPPER algorithm which is a direct method used to learn classification rules, as usual we perform the hyper parameters tuning using the grid search. The rule based classifier performs very poorly with an accuracy of 75% on the test set and 66% on the training showing a clear example of underfitting, also in this case we have an high amount of false positives.

4.6 K Nearest Neighbors

The next model is K Nearest Neighbors which predicts the next instance comparing it with instances seen during the training phase based on a certain metric. In this model we have few parameters, so we perform a grid search to find the best ones for our dataset. The evaluation of the model is quite positive, the KNN gives a 95% accuracy on the test set and a 100% accuracy on the training set, the number of misclassified low skilled players are slightly lower (9%) w.r.t. to the decision tree.

4.7 SVM

SVM (Support Vector Machines) is a model with statistical learning roots that can have promising results in different applications. Since SVM requires more computing to fit we then perform a randomized search over a subset of values for the hyperparameters. This model is the best one we have found with an incredible result of 99% accuracy on the test set and 100% accuracy on the training set, the major difference is in the low number of false positives which cause high values for the precision and recall metrics (as in figure 4.5). We suspect that such high accuracy is due to the fact that SVM is less noise tolerant than the other models, so it tends to be too flexible on the noise introduced by the indicators coming from the dataset, since these attributes do not describe perfectly the high skilled players, this leads to a situation of overfitting on the noise, we believe that using a different test set taken from another dataset the generalization error would be higher.

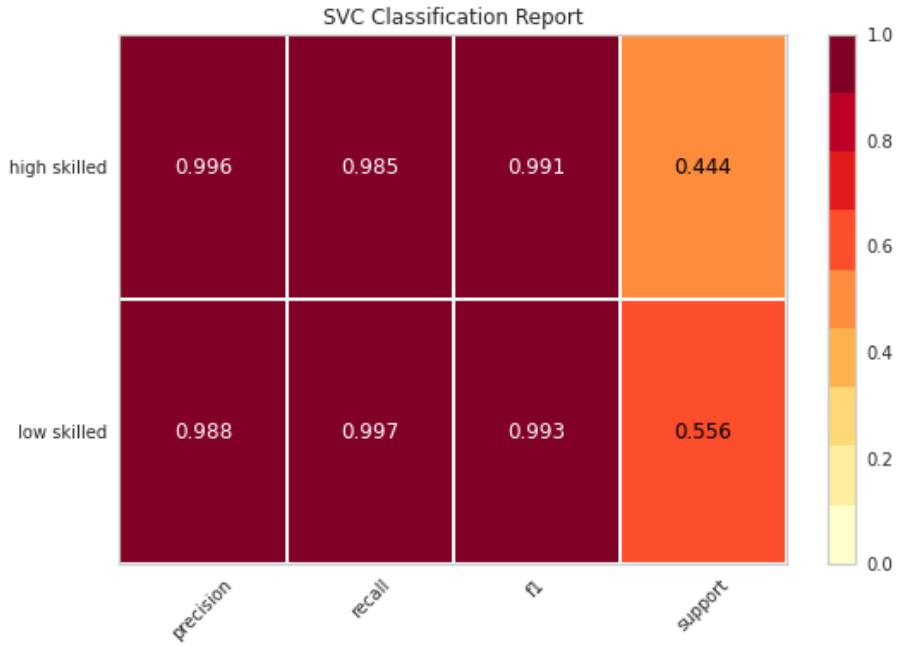


Figure 4.5: Classification report for the SVM model

4.8 Neural Networks

The neural network model also uses a flexible approach towards the classification task, in this case there are more hyper parameters to be tuned so we perform a initial randomized search to find a good range of values and a second one to refine the result. Figure 4.6 shows the learning curves for the neural network which are quite unstable but the result is overall good. The accuracy on the test set is 93% and on the training set and 98% which could indicate a possible situation of underfitting, also in this model the amount of false positives is the same to the one of decision tree (10%).

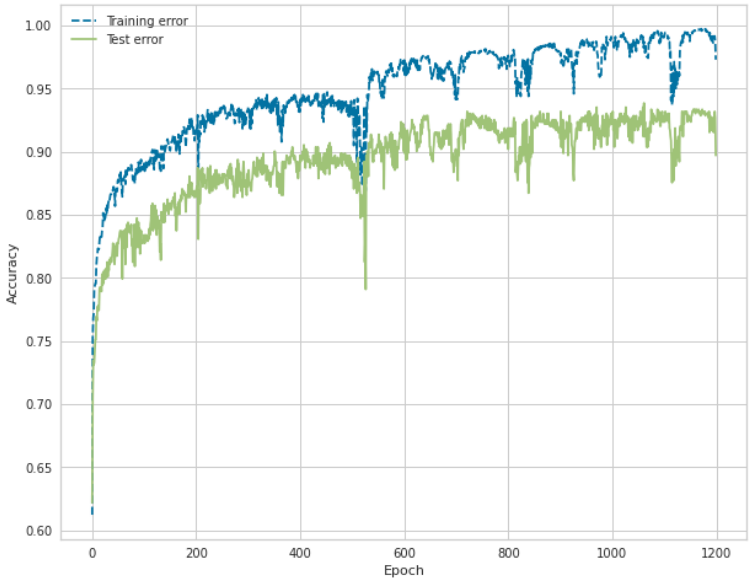


Figure 4.6: Learning curves for the neural network

4.9 Ensemble of all models

Finally we consider an ensemble voting classifier using all the previous models (except for gnb and ripper which performed very poorly), this final model gives a 97% accuracy on the test set and 100% on the training set. The percentage of misclassified low skilled players is only 5%, which is one of the best values besides the SVM model.

A final comparison of all the models analyzed is in table 4.1, which shows the accuracy and percentage of false positives and false negatives w.r.t. the high skilled class, whereas figure 4.7 shows the ROC curve of each model and 4.8 shows the confusion matrix for all models.

Model	Accuracy	False Positives(H)	False Negatives(H)
Decision Tree	94%	10%	2%
Random Forest	96%	7%	1%
Naive Bayes	77%	20%	26%
RIPPER	76%	10%	42%
K Nearest Neighbors	95%	9%	1%
SVM	99%	0%	1%
Neural network	93%	10%	3%
Ensemble of all models	97%	5%	1%

Table 4.1: Summary of the models analyzed.

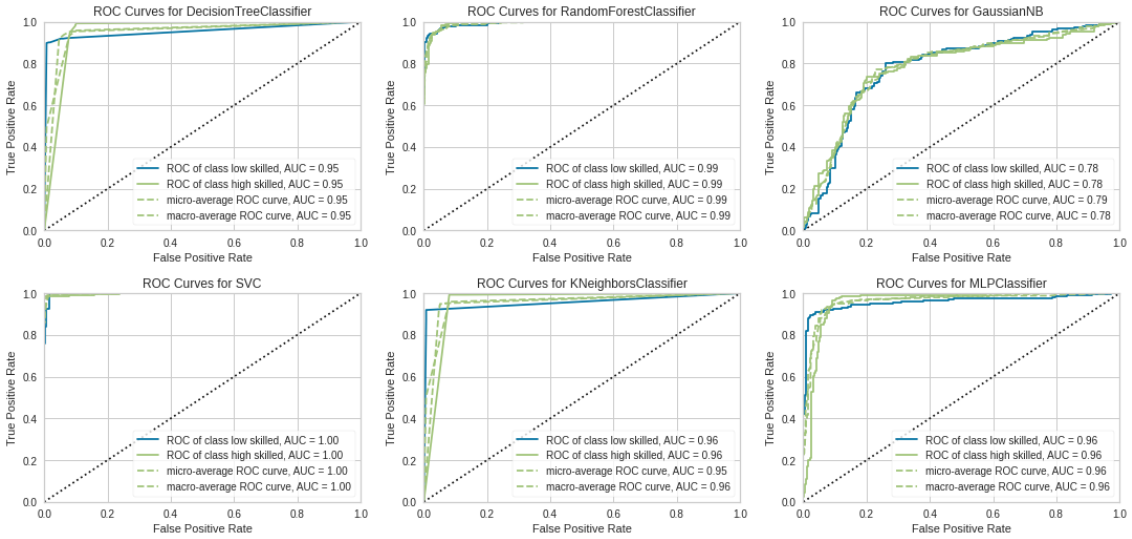


Figure 4.7: ROC curves of all models.

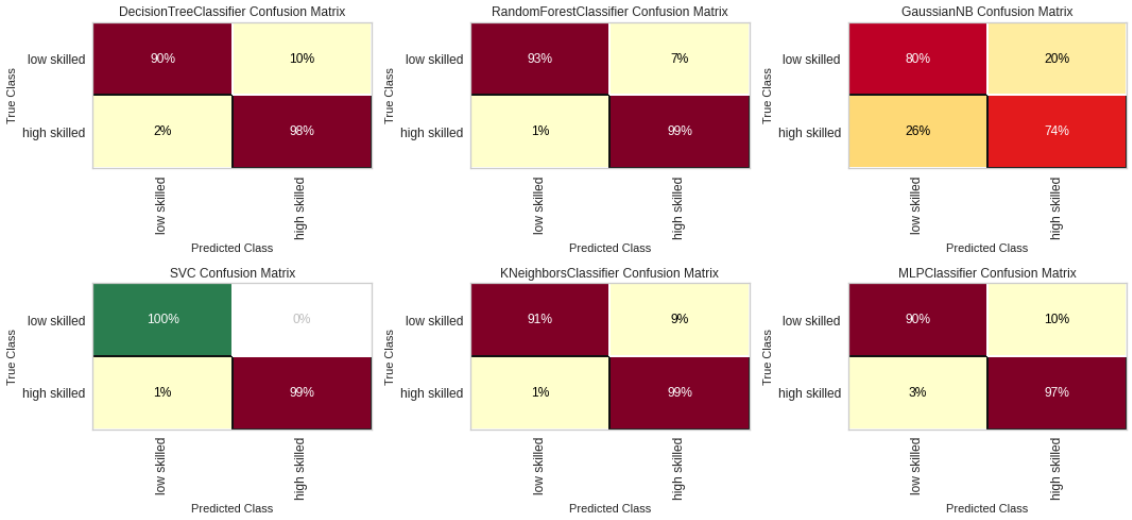


Figure 4.8: Confusion matrix for all models.

Chapter 5

Task 4: Time series analysis

For this task we considered the analysis of the average and standard deviation of temperatures in 100 cities around the globe. Each timeseries was defined for a given month and they all span over 10 years, from 2000 to the beginning of 2010.

First of all, exploiting the attribute Latitude, we divided the timeseries in two datasets: timeseries related to the austral hemisphere and the ones related to the boreal hemisphere. For plotting purposes then we transformed the timeseries in order to have two dataset for each hemisphere, one containing the average and the other containing the standard deviation where each column refers to a particular month.

After this transformation we noticed how some timeseries missed the measurement for January 2000 and the measurements from February to December 2010. We decided then to drop the rows related to the year 2010 and to estimate the missing measurements for January 2000 as the mean for the same city of the other years.

5.1 Clustering on *AverageTemperature*

We started by searching the best k to be used in conjunction with the algorithm TimeSeriesKMeans. Using euclidean and DTW metrics, we computed the SSE and silhouette score for values of k in the range between 2 and 7. In figure 5.1 we can see the results considering the dataset related to the austral hemisphere.

For the boreal hemisphere we applied the elbow method on the SSE and decided to choose 4 as the number of clusters. At first, for the austral hemisphere, it seems that 3 could be a good value for clustering because we obtain the lowest SSE and still a good value for the silhouette. Instead we decided to choose 4 as the number of clusters because it allowed us to spot a particular city (Santiago, capital of Chile) that seemed to have really strange values.

Exploiting the centroids in figure 5.2 and searching the climate of the major cities in the obtained clusters, we decided to name the clusters in the following way:

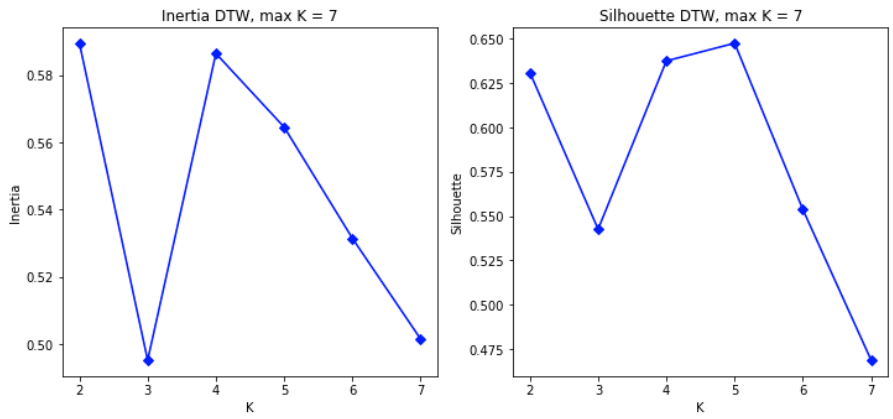


Figure 5.1: DTW and euclidean values for inertia and silhouette in austral dataset.

- For the boreal hemisphere:
  - Centroid 0, **Cold continental** climate: measurements of cities like Moscow, Saint Petersburg, Toronto.
  - Centroid 1, **Tropical** climate: measurements of cities like Calcutta, Singapore, Bangkok.
  - Centroid 2, **Temperate** climate: measurements of cities like Los Angeles, Madrid, Rome.
  - Centroid 3, **Semi-arid** climate: measurement of cities like Alexandria, Delhi, Cairo.
- For the austral hemisphere:
  - Centroid 0, **Tropical** climate: measurements of cities like Dar Es Salaam, Fortaleza, Jakarta.
  - Centroid 1, **Temperate** climate: measurements of cities like Cape Town, Melbourne, Sidney.
  - Centroid 2, **Sub-tropical** climate: measurements of cities like Rio De Janeiro, São Paulo.
  - Centroid 3, **Tundra** climate: only one city, Santiago.

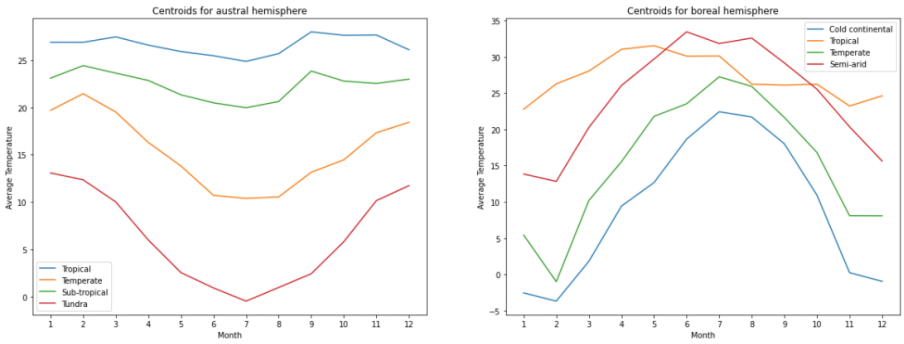


Figure 5.2: Centroids for austral and boreal hemisphere.

We decided to not remove the city of Santiago because searching its coordinates lead us to a mountainous region that is between Chile and Argentina. Maybe the meteorological station was collocated in this zone and so we got these strangely low values.

There were countries and cities that had measurements belonging to different type of climate. Considering the country in figure 5.3 we found how within China and India there are two different climate regions. The same results were made by considering the singular cities, for example New York had some measurements belonging to Cold continental and other belonging to Temperate climate.

Next we considered the different seasons in both the hemisphere to spot outliers that could be cities that had a more drastically lowering of temperatures or cities with too high temperatures.

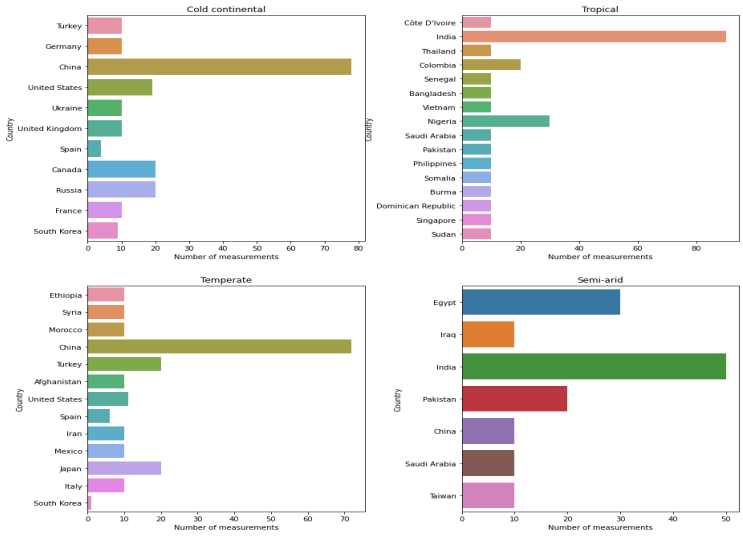


Figure 5.3: Country measurements of boreal hemisphere cities.

5.2 Clustering on *AverageTemperatureUncertainty*

Considering the climate type of tropical regions and all the other regions, this lead us to cluster on the standard deviation considering two clusters for both the hemispheres:

- **Low defined seasons**: cities where between the seasons there isn't a sudden change of temperatures, mostly tropical and sub-tropical.
- **High defined seasons**: cities where there is an huge change between summer/winter and where in spring/autumn there is a moderate rise/decline of temperatures.

With two clusters we obtained better results on the austral hemisphere, we think that this is due to the presence of more tropical cities with respect to cities belonging to other climate regions.