PUBG
Final Placement Prediction

STUDENTS GROUP

Francesco Falleni, Francesco Gemignani,
Cristiano Landi, Ivan Ogando

Distributed Data Analysis & Mining

Data Science & Business Informatics

Academic Year 2021/2022

1 Introduction

1.1 The game

In a PUBG game, up to 100 players start in each match. Players can be on teams which get ranked based on how many other teams are still alive when they are eliminated. The last person or team alive wins the match.

In game, players can pick up different munitions, revive downed-but-not-out (knocked) teammates, drive vehicles, swim, run, shoot, and experience all of the consequences – such as falling too far or running themselves over and eliminating themselves.

1.2 The project

We are provided with a large number of anonymized PUBG game stats, formatted so that each row contains one player's post-game stats. The data comes from matches of all types: solos, duos, squads, and custom; there is no guarantee of there being 100 players per match, nor at most 4 player per group.

We must learn a model which predicts teams finishing placement based on their final stats.

Note, when we say teams we mean a single player in the case of a 'solo' match and a group of players in the case of 'duo' and 'squad' matches. Later on, we will describe how we have transformed the records in the dataset so that there is only one record even in the case of squad and solo matches.

2 Data Understanding and Preprocessing

The dataset consists of 4,411,698 records and 29 attributes described in Table 1. The distributions of the numerical attributes are shown in Figure 1.

We can identify 47,185 matches and 2,014,090 teams. We have one missing value that we removed early on.

The Id attribute is not useful as it uniquely identifies each record, so it was removed immediately. The groupId attribute is a team identifier and determines several attributes: matchId, matchDuration, winPlaceClass, matchtype, maxPlace, numGroups, winPlacePerc.

We analyzed the distribution of matchType -it's a string that describes the game modes, the numbers of players in each team and if they are playing in first or third person- and we removed all matchType that have a very low frequency.

In addition, for each type of match, the value of the matchType attribute has been changed to represent the number of players of which each team is composed (1, 2 or 4) and a new attribute has been created to identify if the match is played in first or third person (*isFirstPerson*).

At the end of this transformation, we went from 16 to 3 matchType categories (ignoring the view type): solo, duo and squad (Figure 2).

The target variable, winPlacePerc, is presented as a percentile of the final placement and therefore can take a value between 0 (last position) and 1 (first position). This variable, however, depends on the number of teams in the match, which, remember, is not guaranteed: a match of the same type may be composed of 99 teams instead of 100 and, therefore, generate different values. To predict the final placement of a player we decided to discretize winPlacePerc in bins, creating a new variable called winPlaceClass (Figure 3):

1. $winPlacePerc == 1$
2. $0.75 < winPlacePerc < 1$
3. $0.5 < winPlacePerc \leq 0.75$
4. $0.25 < winPlacePerc \leq 0.5$
5. $0 < winPlacePerc \leq 0.25$
6. $winPlacePerc == 0$

In this way it will be then possible to use the models for the classification with only 6 target classes: if then we will obtain very interesting performances ($Accuracy > .90$) we will proceed with the increase of the number of bins.

Note 1: we decided not to try any regression algorithm on *winPlacePerc* in order not to lose the discrete value constraint.

Note 2: A possible alternative we evaluated is to create a model for each distinct value of the number of teams in each match; however, it would have been necessary to create and evaluate 100 models that would be unusable in the case of a match composed of a total number of teams that is not present in the data (e.g., a new type of match).

The dataset provided is composed of records related to the statistics of a player within a match, even in the case of matches in teams composed by different numbers of players: in the case of a 'squad' match type, for example, there would be 4 times more features than in a 'solo' match type. Another problem is the management of disconnections: each disconnection and reconnection generates a new player in the same team, but with all zero attribute values. Since the data is anonymized, it is impossible to deduce which team member the data belonged to before the disconnection except for the 'solo' mode. So, to have a dataset with a unique granularity (one record per team) we decided to transform the dataset aggregating by teams (groupId and attributes determined by it). For all the remaining attributes we calculated the average, sum and maximum (generating a new dataset of 57 features) grouping by groupId: at this stage, even analyzing the semantics of the various attributes, it is not easy to deduce which of these aggregation functions is the most appropriate to represent the statistics of a team.

Because of disconnections, the average calculation could be wrong (think of a player in 'solo' mode that disconnects three times), for this reason, the average is calculated dividing by the number of players in the teams only if that number is less than the maximum number of players per team in that game mode.

In this step, we also added a boolean attribute that indicates whether the group has had any disconnections by checking if $COUNT(*) > matchType$[1].

[1]in case of a match in 'squad' the team can be composed of only three players, this problem could also negatively affect the calculation of the average, but we believe that the average of the attributes can be an important source of information

Name	Domain	Description
Id	String	unique identifier
groupId	String	identify a group within a match
matchID	String	identify a match
assists	Integer	number of times a player helps to make a kill
boosts	Integer	number of boost items used
damageDealt	Integer	total damage dealt, means how many bullets have ever been hit
DBNOs	Integer	number of enemy players knocked
headshotKills	Integer	number of enemy players head-shotted's
heals	Integer	number of healing items used
killPlace	Integer	ranking in match of number of enemy players killed
killPoints	Number	kills-based external ranking of player.
kills	Integer	number of enemy players killed Composed by headshotKills, roadKills, not teamKills
killStreaks	Integer	max number of enemy players killed in a short amount of time
longestKill	Number	longest distance between player and player killed at time of death
matchDuration	Integer	duration of whole single/teammate match in seconds
matchType	String	string identifying the game mode that the data comes from
maxPlace	Integer	worst placement we have data for in the match
numGroups	Integer	number of groups we have data for in the match
rankPoints	Number	Elo-like ranking of player
revives	Integer	number of times this player revived teammates
rideDistance	Number	total distance traveled in vehicles measured in meters
roadKills	Integer	number of kills while in a vehicle
swimDistance	Number	total distance traveled by swimming measured in meters
teamKills	Integer	number of times this player killed a teammate
vehicleDestroys	Integer	number of vehicles destroyed
walkDistance	Number	total distance traveled on foot measured in meters
weaponsAcquired	Integer	number of weapons picked up
winPoints	Number	win-based external ranking of player
winPlacePerc	Number	the target of prediction. This is a percentile winning placement

Table 1: Attributes description

3 Clustering Analysis

We performed clustering with K-Means to search for different types of players within the original dataset.

The set of features to run the algorithm was chosen by removing those attributes that had a correlation value greater than a certain threshold, $\theta = 0.5$.

The spark dataframe with the selected features was subsequently assembled and then normalised with MinMaxScaler, as the domains of the variables were substantially different.

At this point, we want to determine the best number of clusters to run K-Means. We started searching in a range of $k = [2, 15]$ on a random sample of players (20%, approx. $900,000$ record), keeping track of the SSE and Silhouette score.

We chose the number of clusters with the best trade-off between the two values, thus $k = 6$, and re-run K-Means on the entire dataset. The value of SSE is 97278.73 and silhouette = 0.5955. Each cluster has the following dimensions: $[23.23\%, 10.09\%, 23.06\%, 11.53\%, 5.92\%, 26.98\%]$.

We analysed the centroids resulting from clustering to characterise them on the base of the features. In Figure 6 we observe that *matchDuration* and *maxPlace* are the attributes that most characterise the clusters, for the remaining attributes no significant differences are noted.

Looking more closely at the distribution of *matchDuration* it is possible to identify two groups of values that reflect the two different sizes of the game maps. Moreover, remembering how the feature *maxPlace* is generated, it is easy to see that there are basically 3 groups determined by the composition of the team (also confirmed by the Figure 5). Putting these two observations together, it is easy to identify the combinations of the distributions of these attributes in the six clusters.

The distributions of the target class *winPlaceClass* within each cluster were also analysed but no significant differences were observed.

We performed a second clustering by manually choosing the features to be used among those with high correlation. In addition, all features describing the type and dynamics of the match were excluded.

The selected features are: *assists, headshotKills, heals, kills, killStreaks, longestKill, rideDistance, roadKills, swimDistance, vehicleDestroys, walkDistance.*

We selected $k = 5$ as the best value by looking at SSE and Silhouette in Figure 7. The value of SSE is 20241.51 and silhouette = 0.4882. Each cluster has the following dimensions: $[55.49\%, 14.14\%, 5.51\%, 6.54\%, 18.3\%]$.

Next, we analysed the distribution of the *matchtype* feature within the clusters (Figure 8): it is possible to verify how the three classes are equally distributed in the clusters, an indication that the type of match is irrelevant to characterize the similarities between players.

On the other hand, by analyzing the distribution of *winPlaceClass* (Figure 8) within the clusters we can observe some interesting behaviour:

- Loser's cluster: Most of the records where winPlaceClass is worth 6 or 5 are in cluster 0
- Winner's cluster: most of the winning players (winPlaceClass = 1) are in cluster 2
- Mixed cluster: clusters 1, 3, and 4 are similar to each other and show particular differences in the distribution of WinPlaceClass values

In addition, we have analyzed the centroids (Figure 9), from which we draw the following observations:

- *roadKills*, *swimDistance*, and *vehicleDestroys* are not useful features for the characterization of clusters.
- The Loser cluster (0) has the lowest value of each feature, compared to the other clusters. It contains most of the players.
- The Winner cluster (2) has the highest value for almost every feature, compared to the other clusters, so they are very active players in the game dynamics.

- Cluster 1 is the closest to the loser cluster. In contrast to the latter, we have much more active players, who take care of themselves more often and assist other team members. Nevertheless, they fail to enter into the dynamics of the game.
- Cluster 3 is characterized by "crashing players": they heal often and travel a long distance in vehicles (they are also the ones who destroy the most vehicles along with the winner's cluster)
- Cluster 4 is the one that, ignoring *longestKill*, is closest to the Winner's cluster: it is therefore characterized by quite active players who make a lot of short-range kills.

4 Final Placement Classification Models

4.1 Ovierview

In this section, we describe the models we applied to solve the placement group position task on the transformed dataset, i.e., the dataset where a single row represents the entire group.

Some attributes highly correlated with the target variable, such as killPlace and rankPoints, were removed at this stage in order not to have trivial models: testing including all variables resulted in models that relied only on attributes correlated with the target variable.

Regardless of the model, most of the steps are shared. The macro steps followed are the following:

1. Loading the dataset
2. Feature selection: treat categorical attributes according to model and remove attributes that are highly correlated with others
3. Data Normalization: MinMaxScaler or StandardScaler
4. Parameter tuning on a sample of the dataset (using CrossValidator or other techniques to get the best hyperparameters) For each model we identify the starting range for the parameters, we then perform the search for hyperparameters specifying, for each hyperparameter, three values (min, middle, max). According to the values used by the best models, we choose the new hyperparameters range and repeat the search until we identify, for each hyperparameter, a unique value.
5. Splitting (of the whole dataset or a substantial part of it) into evaluation and training sets
6. Learning phase: fitting and prediction
7. Evaluation on the test set and analysis of the results

For some more advanced models we used additional steps listed later. For example, we performed univariate feature reduction in LinearSVC to further reduce the number of variables. We also implemented an EarlyStopping procedure in NNs to handle the overfitting problem.

4.2 Implementations

We tried basic classifiers such as **Naive Bayes** in the hope that, being less algorithmically complex, we could use the entire dataset. In this case, we identified and removed the highly correlated features pairs.

For the **Support Vector Machines** it was necessary to use *OneVsRest* with 6 binary LinearSVC classifiers. We performed the univariate feature selector to search for the minimum number of features (25) without penalizing the final performance. The learning phase was performed on the entire dataset.

In the **Logistic Regression** model, we performed univariate feature selection to reduce the number of features to at most then 30 ($k = 30$), without penalizing model performance. As for SVM, the learning phase was done on the whole dataset.

As for the model based on **Decision Tree**, it was not necessary to eliminate the highly correlated features by the construction of the model[2]. For further comments on the obtained model, please refer to the Explainability Section.

For the **Random Forest** approach, we opted for a grid search to find the best values for the hyperparameters: *maxDepth*, *minInstancesPerNode*, *numTrees* and *impurity*. Finally we kept the selection of features that gave the best results once cross-validated.

Since we could only use **Gradient Boosted Trees** for binary classification tasks, we again had to use OneVsRest consisting of 6 GBT classifiers. We did not take any steps other than those described at the beginning of the chapter.

Finally, for neural networks, an important additional step was needed. The MLlib implementation -**Multilayer perceptron classifier**- does not provide the same functionality as the one in the Keras library, specifically there is no advanced mechanism to avoid underfitting and overfitting of the model. While for underfitting it is sufficient to increase the number of epochs, for overfitting the only indirect mechanism provided by MLlib is the optimizer. For this reason, a function *customEarlyStoppingNN* was created that performs, for n iterations the following steps:

0. Initialize the following lists: *loss_tariningSet*, *loss_validationSet*, *acc_trainingSet*, *acc_validationSet*, *modelli*
1. If the models' list is not empty, extract from the most recent model the weights relative to the arcs and use these values as initial weights for the new neural network.
2. Perform training of the new neural network for m epochs ($1 < m << n$)[3]
3. Extract from the trained neural network the parameters of loss and accuracy on the training set
4. Calculate the loss and accuracy values on the validation set (passed as a parameter to the function)
5. If the loss value on the validation set is increased compared to the neural network with the minimum value and that model is a certain number (function parameter) of iterations away, then the function stops returning a pandas dataframe with all the measurements and the best model.
6. Save in the list *modelli* and return to the first step

[2]Remember that at each step the information gain is recalculated

[3]The condition that $m > 1$ is necessary since it would seem necessary to run at least two epochs at the optimizer to act effectively.

Different configurations of neural networks were manually tested and, for the same performance, the model with fewer perceptrons was preferred. Data were scaled using StandardScaler.

As you can see from the Table 2, the best neural network is the one with only one relatively small hidden layer. It is interesting to note that even a neural network without any hidden layers gets negligibly lower results: this allowed us to make an interesting analysis in the Explainability section. Looking at the Figure 10 and 11 it is possible to see how both networks do not present the phenomenon of overfitting or underfitting.

4.3 Results

The Table 2 compares the performance of the models tested. Note how the less complex models achieved poor performance. From the classification report, we can see that SVM does not perform well, the first and last classes are not predicted and we have a maximum f1 just below 0.70 only for classes 2 and 5. The performance of Logistic Regression model is better than LinearSVC, all classes have a higher f1 than SVM, although the last class are still not detected. In Figure 15 we can see that for each class predicted, the errors are mostly committed with the adjacent classes.

5 Explainability

5.1 Decision Tree, Random Forest and Gradient Boosted Trees

In this section, we want to try to understand which were the most significant features to solve the task. Having used tree-based models is possible to get easily the importance of each attribute. Starting with the Decision Tree (Figure 12a), we can see that the most important attributes are *walkDistance_AVG* and *walkDistance_MAX*. It is natural to think that a player who has survived longer has covered large distances unlike one who, by ending the game earlier, has covered short distances.

We tried to remove the most important features related to distance (*walkDistance_** and *distance_**) obtaining an accuracy, f1, precision and recall of .58. The new feature importance are shown in Figure 12b.

We evaluated the feature importance of the Random Forest (As depicted in Figure 12c). We could observe that, in contrast to the DT, there were several features with relative importance. Although, we noticed that still the distance indicators were the most relevant to the model as it was to be expected.

Since the GBT-based model consists of OneVsRest classifier, we decided to deepen the analysis by looking at the feature importances of each GBT that compose up the OVR model (Figure 13).

It can be observed that for classes 2, 3, 4, 5 the attributes *walkDistance_AVG* and *walkDistance_MAX* prevail, similarly to DT. For class 1 (first position) we observe that *kills_SUM* is the most important feature, this makes us think that in order to get first, it is not only necessary to walk great distances, but also to kill other players.

Finally, for class 6 (last position) we have slightly different importance. In particular, we note that the feature *hasDisconnected* has significant importance, and thus, makes us guess that teams that had disconnected players might have come last in the ranking. Depending on how the disconnections were handled during data collection, we could attribute this phenomenon to the fact that when reconnected, some players are found to be dead, or more simply, disconnections are indicators of a poor internet connection that negatively affects the in-game performance.

5.2 Neural Network

Since it was possible to find a neural network that, despite not being composed of hidden layers, can make predictions of a quality substantially similar to the best models, it was decided to try, by analyzing the weights on the arcs of the individual output nodes, to understand which features were crucial to the prediction.

Our intuition is to, in order to evaluate the model with respect to all six classes, observe the standard deviation of each weight on the arc associated with the same feature.

There are 51 features provided to the model, thus we have 52 weights for each output node. Analyzing in detail the documentation we concluded that the bias was associated with the fifty-second incoming arc. At this point, it was easy to calculate for each feature the standard deviation of the associated weights and export the chart in Figure 14. From the chart, we can observe that highly correlated features (such as most <sum, maximum, mean> triplets) have different standard deviations, which highlights that the NN-based model can handle correctly these kinds of features.[4]

The features that most influence prediction are: *roadKills_AVG*, *killStreaks_MAX*, *heals_AVG*, *hasDisconnected*, *DamageDealt_MAX*, *assist_SUM* and *DBNOs_MAX*.

[4]In the Jupyther Notebook named *"Neural Network input_output MinMaxScaler"*, a second neural network was trained with data scaled with MinMaxScaler. Although this neural network scored lower performance (Accuracy of 70%) we continued with the same type of analysis, with the difference of having multiplied the standard deviation of the scaled data to that of the weights. This, if our reasoning is correct, should allow a more complete analysis of the neural network.

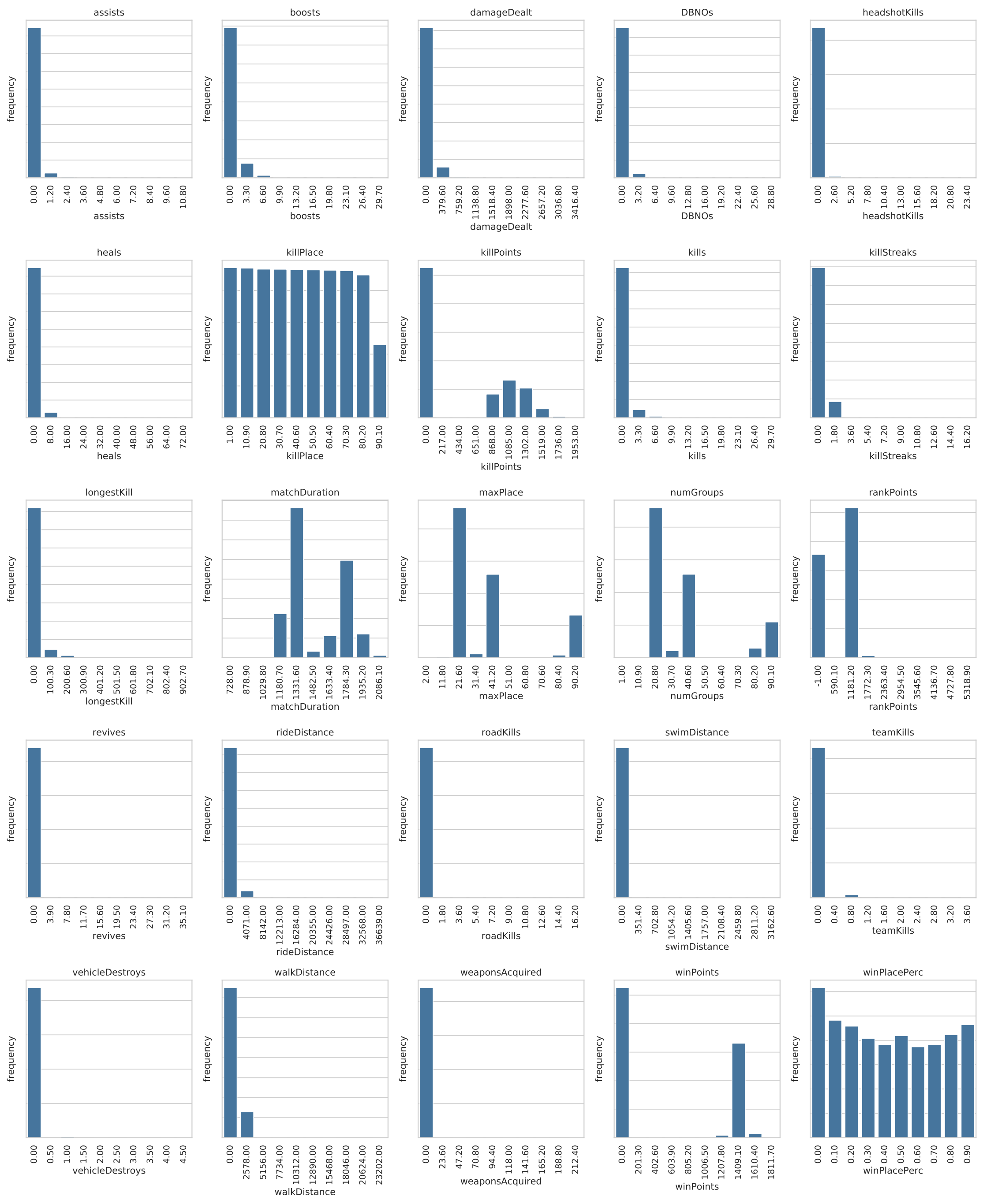


Figure 1: Attributes distribution

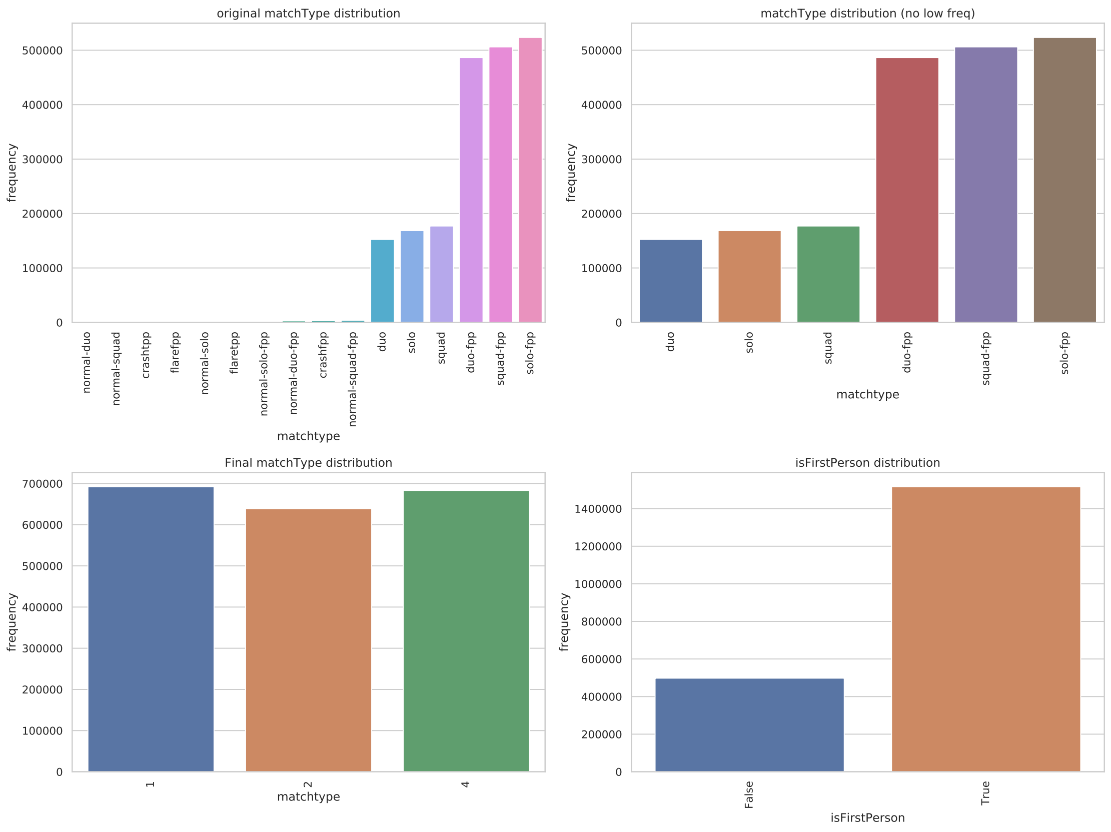
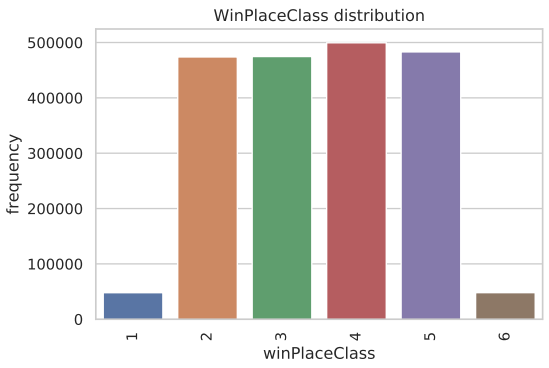
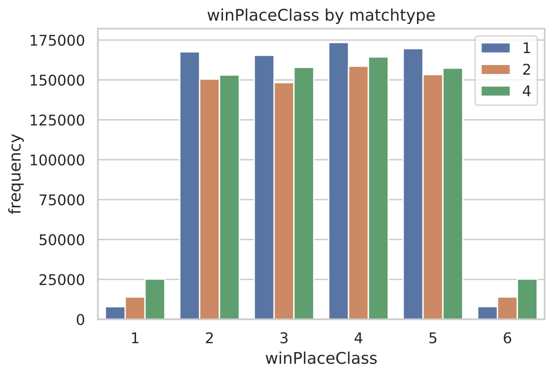


Figure 2: matchType and isFirstPerson distribution



(a)



(b)

Figure 3: winPlaceClass Distribution

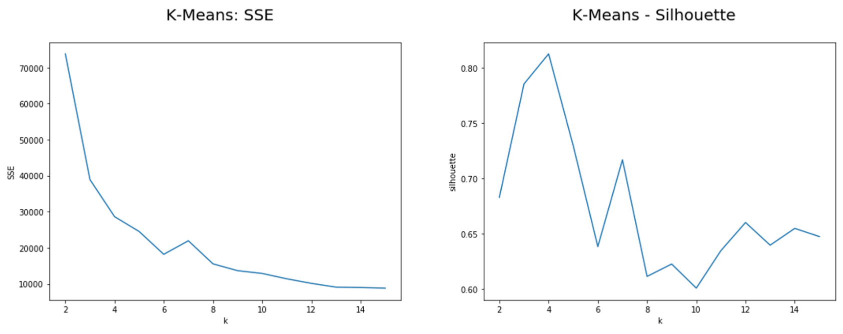


Figure 4: SSE and Silhouette score as the number of clusters increases

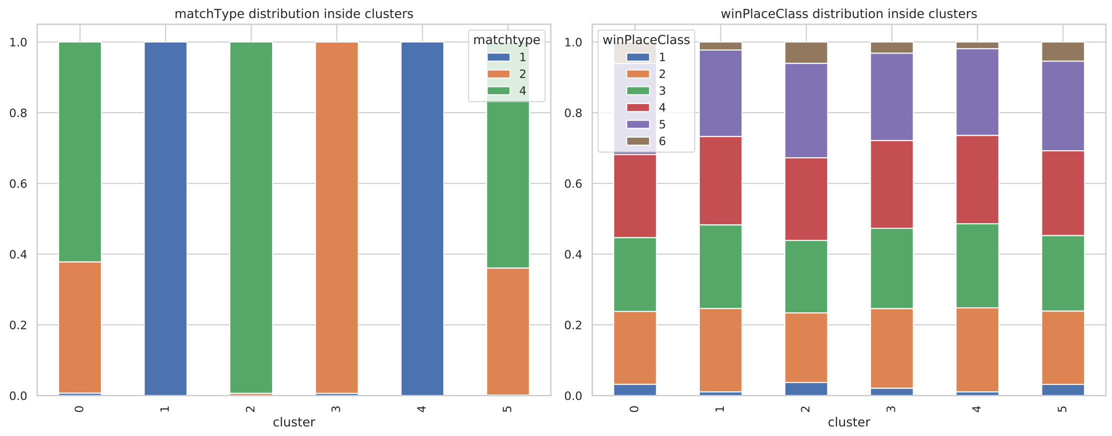


Figure 5: Analysis of distributions within clusters

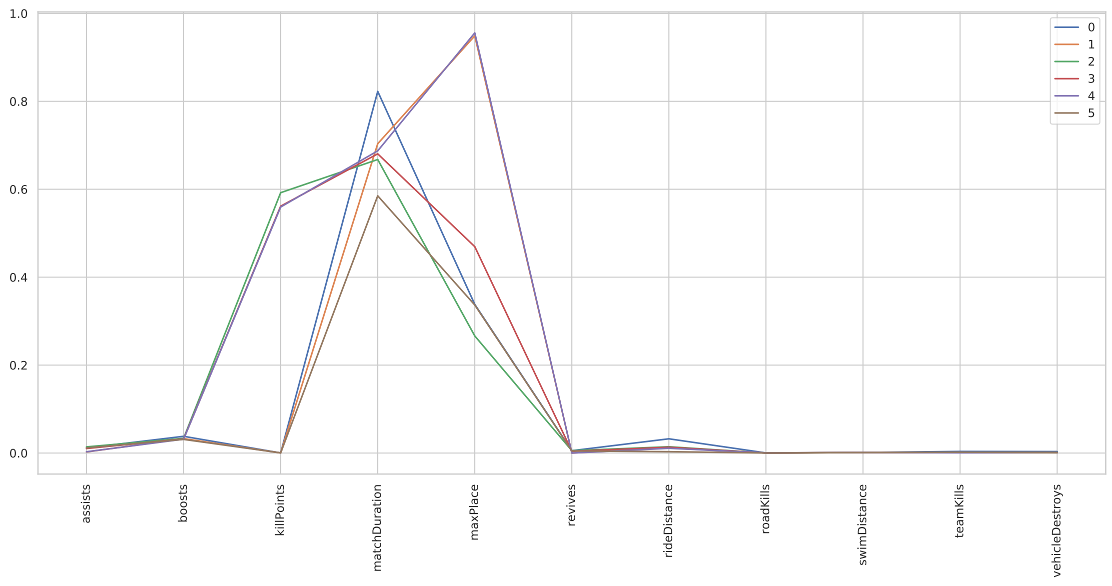
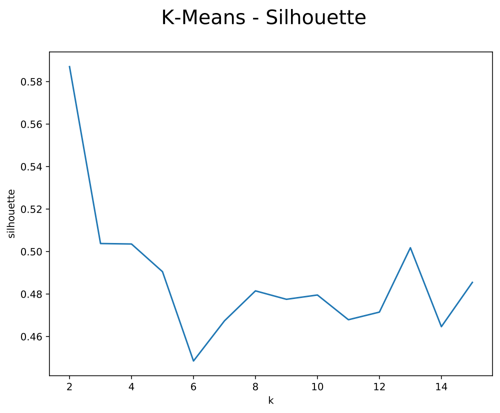
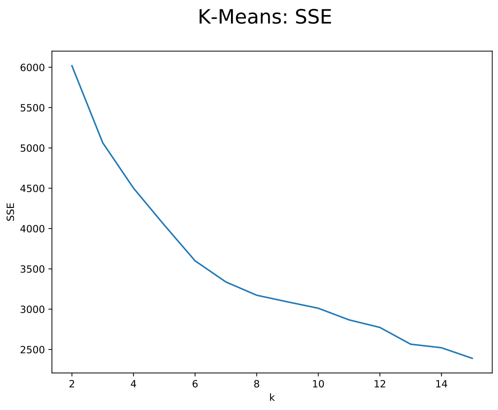


Figure 6: Centroids Clustering 1



(a) Silhouette



(b) SSE

Figure 7: SSE and Silhouette score as the number of clusters increases

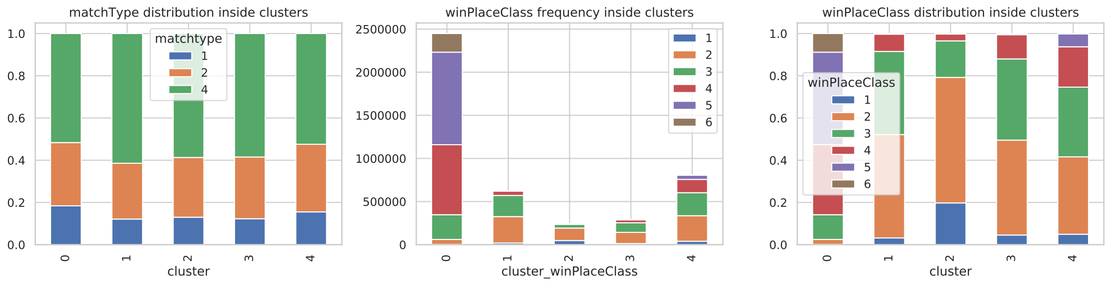


Figure 8: Analysis of distributions within clusters

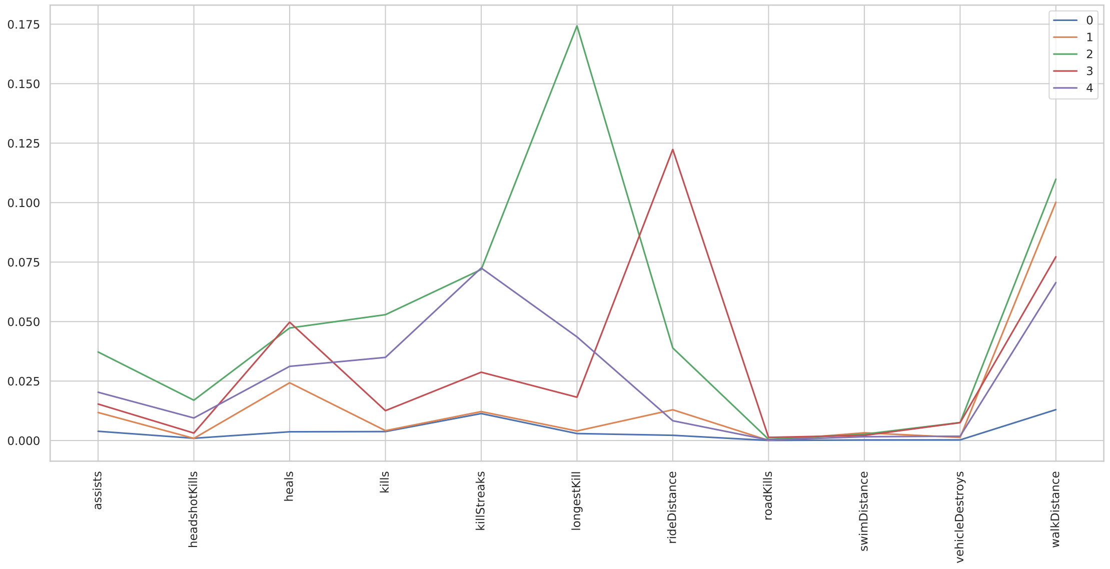
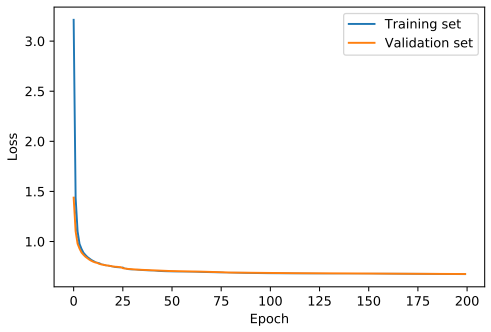
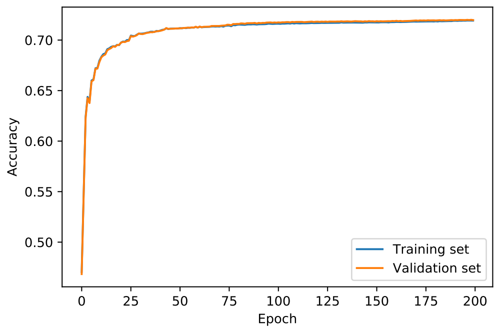


Figure 9: Centroids Clustering 2

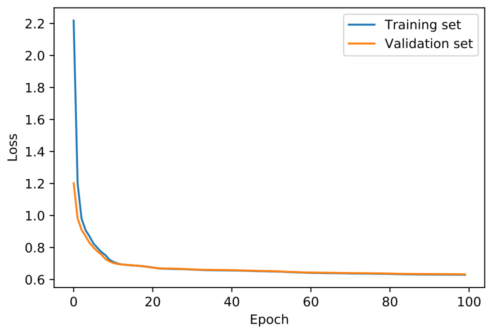


(a) Loss

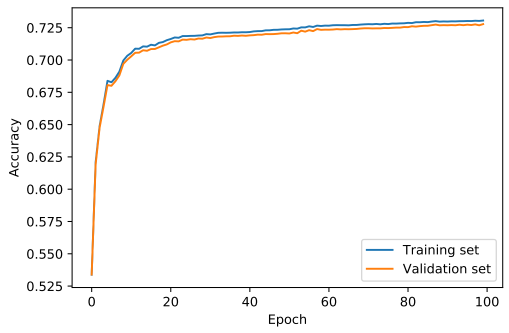


(b) Accuracy

Figure 10: Loss and Accuracy for the NN [in, out]

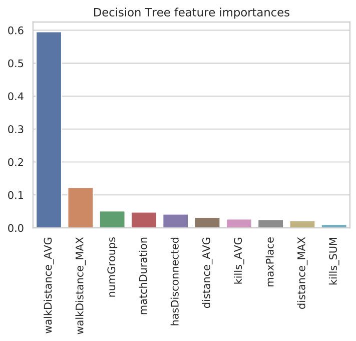


(a) Loss

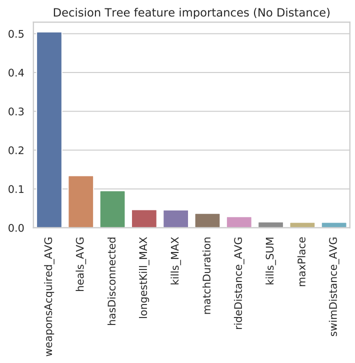


(b) Accuracy

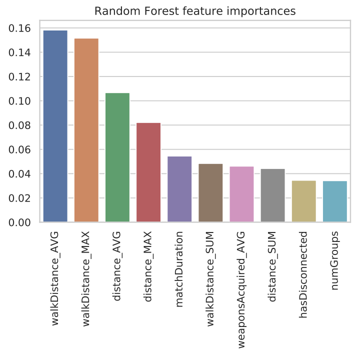
Figure 11: Loss and Accuracy for the NN [in, 80, out]



(a) DT



(b) DT (No Distance)



(c) RF

Figure 12: Feature importance

		class						**avg**	
		1	2	3	4	5	6	macro avg	weighted avg
Naive Bayes smoothing=0.0 modelType=gaussian	accuracy							.34	
	precision	.24	.63	.40	.34	.35	.11	.34	.42
	recall	.65	.37	.31	.29	.30	.91	.47	.34
	f1 score	.35	.46	.35	.31	.33	.19	.33	.36
Logistic Regression maxIter=100, regParam=0.01	accuracy							.64	
	precision	.36	.69	.57	.59	.70	.00	.49	.62
	recall	.13	.75	.51	.63	.78	.00	.47	.64
	f1 score	.19	.72	.54	.61	.74	.00	.47	.63
Decision Tree maxDepth=11, maxBins=256 minInstancesPerNode=50 minInfoGain=0	accuracy							.72	
	precision	.64	.74	.64	.69	.80	.74	.71	.72
	recall	.24	.78	.64	.70	.82	.58	.63	.72
	f1 score	.35	.76	.64	.70	.81	.65	.65	.72
Random Forest maxDepth=15, numTrees=50 minInstancesPerNode=5 impurity=gini	accuracy							.73	
	precision	.64	.74	.66	.71	.82	.78	.72	.73
	recall	.20	.81	.64	.72	.82	.54	.62	.73
	f1 score	.30	.77	.65	.71	.82	.64	.65	.73
GBT maxIter=120, maxDepth=8 minInstancesPerNode=10 minInfoGain=0.0, maxBins=64 subsamplingRate=0.9	accuracy							.73	
	precision	.63	.75	.66	.71	.82	.77	.72	.73
	recall	.31	.79	.65	.72	.83	.58	.65	.73
	f1 score	.41	.77	.65	.71	.82	.66	.67	.73
SVM maxIter=100, regParam=0.05	accuracy							.53	
	precision	.00	.54	.50	.57	.52	.00	.35	.50
	recall	.00	.97	.11	.19	.97	.00	.37	.53
	f1 score	.00	.69	.17	.29	.67	.00	.30	.43
NN [in, 80, out]	accuracy							.73	
	precision	.65	.75	.66	.71	.79	.71	.71	.73
	recall	.28	.79	.65	.70	.85	.45	.62	.73
	f1 score	.39	.77	.66	.71	.82	.55	.65	.73
NN [in, out]	accuracy							.72	
	precision	.64	.76	.67	.70	.76	.63	.69	.72
	recall	.30	.78	.64	.68	.86	.36	.60	.72
	f1 score	.40	.77	.65	.69	.81	.46	.63	.72

Table 2: Results

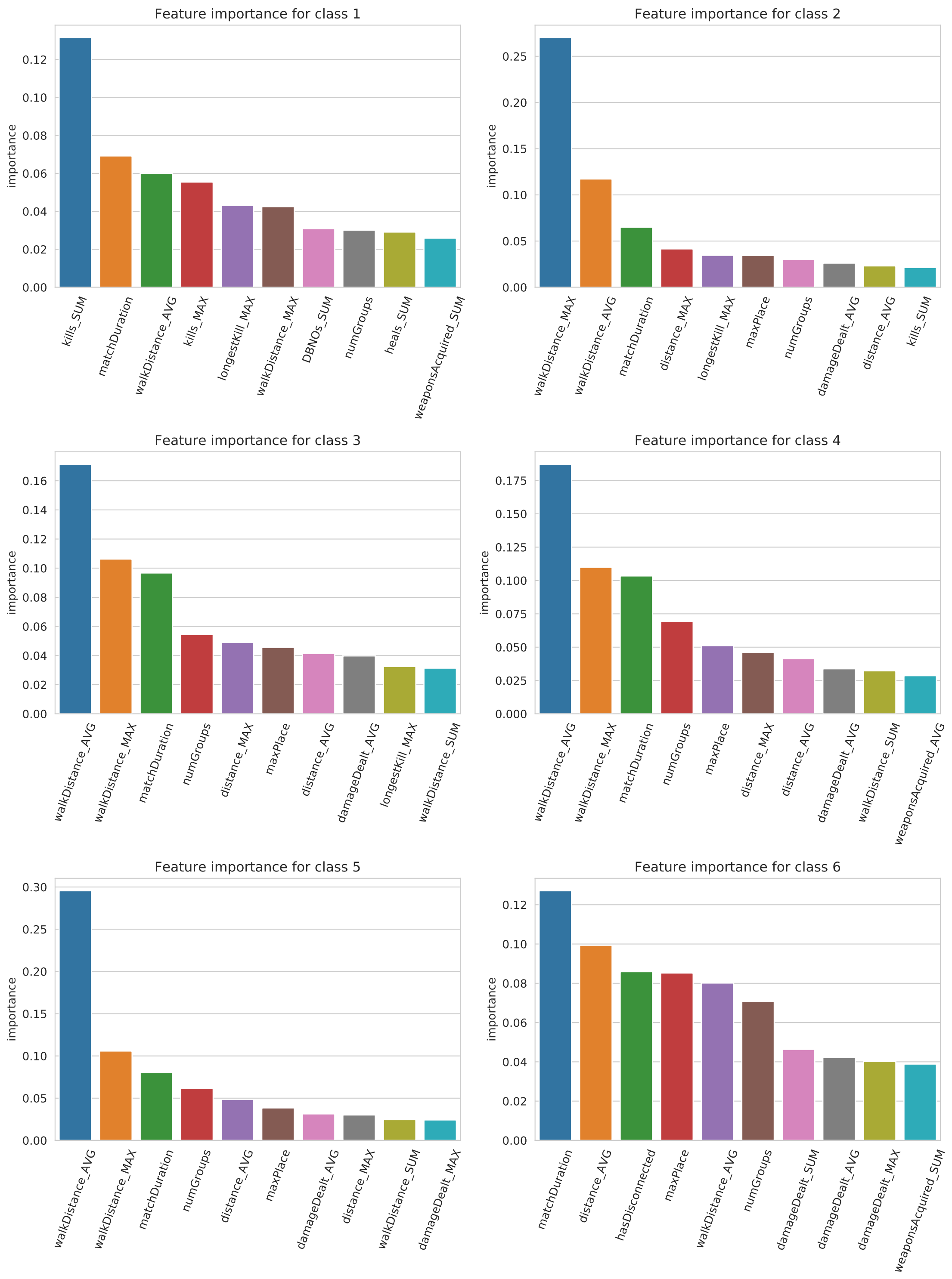


Figure 13: GBT Feature importances

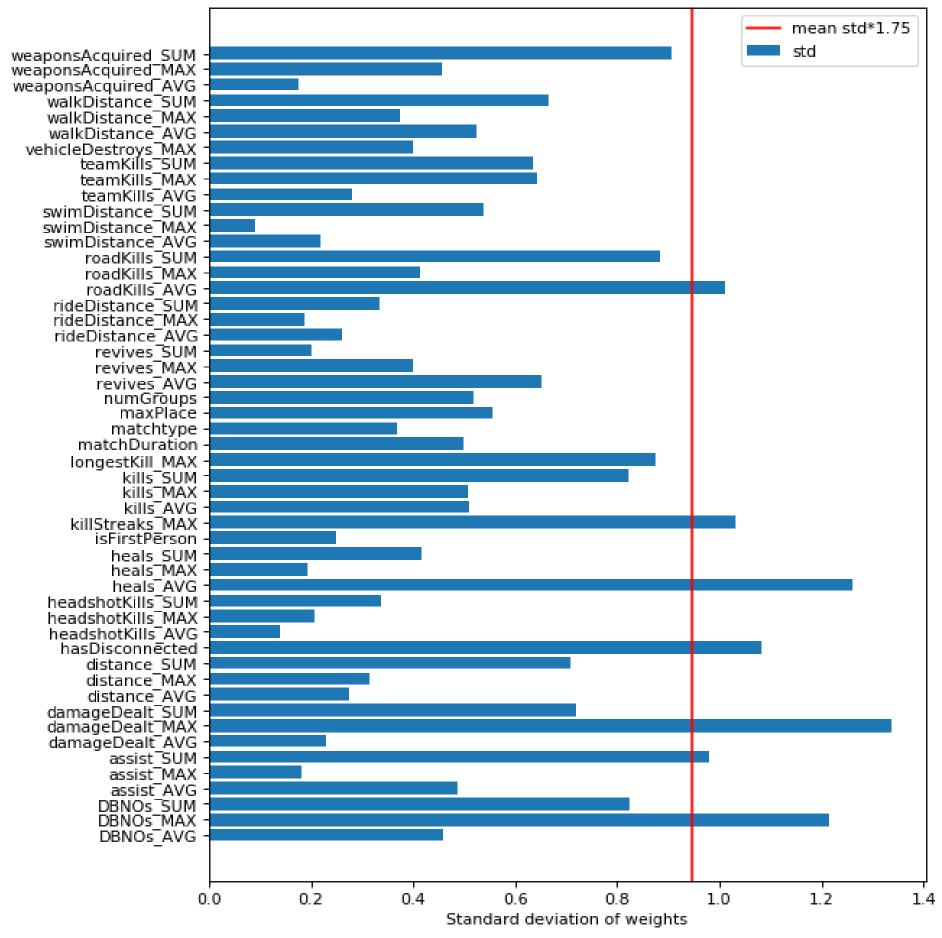
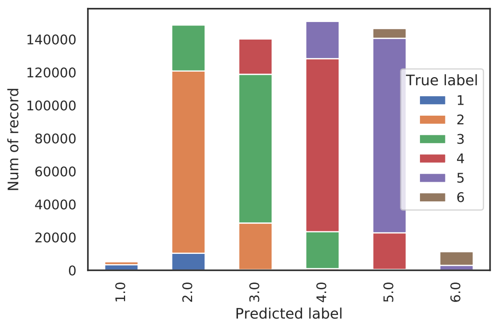
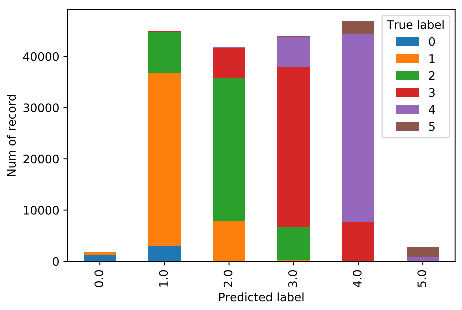


Figure 14: NN weight std



(a) Bar chart prediction label DT



(b) Bar chart prediction label NN

Figure 15: Confusion Matrix analysis