Documentation BE World Cup Pipeline

Sources

On the internet are several sources to fetch. In the very beginning we thought about reliable and good features how to predict an outcome for a soccer match. Being a data analytics team we quickly came to simple conclusion: Just gather *everything* related to international soccer matches. Dataformat was not of first importance. As long as it is legal to fetch that data it is fine.

Several sources turned out to be fine:

1. Wikipedia, e.g as archive for world cup squads
2. Repositories in GitHub, e.g Openfootball/World-Cup
3. APIs, e.g Football-Data
4. Official Fifa website FIFA
5. Soccer archives , e.g Linguasport , RSSSF
6. Other ranking sites , e.g Fifaindex
7. Kaggle , e.g FIFA 2017 Player Dataset , FIFA 2018 Player Dataset

Already existent datasets in csv-format were mostly deprecated. Hence we built our own web-scrap-scripts to fetch data from the above mentioned websites and many more.

Fifa World Cup Matches from 1930 until 2017

Since Linguasport seems to be a reliable and structured source we gathered historical fifa world cup data and after preprocessing them (e.g remove badly encoded strings, data-formats etc.) we finally received following snippet of datatable:

edition	round	team1	team2	hst1	hst2	fst1	fst2	fs.t1.winnerflag	team1iso	team2iso
1930-URUGUAY	GROUP_STAGE	France	Mexico	4	1	4	1	W	FR	MX
1930-URUGUAY	GROUP_STAGE	USA	Belgium	3	0	3	0	W	US	BE
1930-URUGUAY	GROUP_STAGE	Serbia	Brazil	2	1	2	1	W	RS	BR
1930-URUGUAY	GROUP_STAGE	Romania	Peru	3	1	3	1	W	RO	PE
1930-URUGUAY	GROUP_STAGE	Argentina	France	1	0	1	0	W	AR	FR
1930-URUGUAY	GROUP_STAGE	Chile	Mexico	3	0	3	0	W	CL	MX
2018-RUSSIA	PRELIMINARY	Italy	Sweden	0	0	0	0	D	IT	SE
2018-RUSSIA	PRELIMINARY	Ireland	Denmark	1	5	1	5	L	IE	DK
2018-RUSSIA	PRELIMINARY	Burkina Faso	Cape Verde	4	0	4	0	W	BF	CV
2018-RUSSIA	PRELIMINARY	Senegal	South Africa	2	1	2	1	W	SN	ZA
2018-RUSSIA	PRELIMINARY	Australia	Honduras	3	1	3	1	W	AU	HN
2018-RUSSIA	PRELIMINARY	Peru	New Zealand	2	0	2	0	W	PE	NZ

Fifa Euro Cup Matches from 1960 until 2014

In addition to fifa world cup data we gathered as well fifa eurocup data from Linguasport in order to extract more feature-information for later modelling. After scraping data using same logic as before, we received following snippet datatable:

edition	round	team1	team2	hst1	hst2	fst1	fst2	fs.t1.winnerflag	team1iso	team2iso
1960-FRANCE	PRELIMINARY	Ireland	Czechia	2	0	2	0	W	IE	CZ
1960-FRANCE	PRELIMINARY	Czechia	Ireland	1	0	4	0	W	CZ	IE
1960-FRANCE	GROUP_STAGE	Russia	Hungary	3	0	3	1	W	RU	HU
1960-FRANCE	GROUP_STAGE	France	Greece	3	0	7	1	W	FR	GR
1960-FRANCE	GROUP_STAGE	Romania	Turkey	0	0	3	0	W	RO	TR
1960-FRANCE	GROUP_STAGE	Greece	France	0	0	1	1	D	GR	FR
2016-FRANCE	1/4_FINAL	Wales	Belgium	1	1	3	1	W	GB-WLS	BE
2016-FRANCE	1/4_FINAL	Germany	Italy	NA	NA	1	1	D	DE	IT
2016-FRANCE	1/4_FINAL	France	Iceland	4	0	5	2	W	FR	IS
2016-FRANCE	1/2_FINAL	Portugal	Wales	0	0	2	0	W	PT	GB-WLS
2016-FRANCE	1/2_FINAL	Germany	France	0	1	0	2	L	DE	FR
2016-FRANCE	FINAL	Portugal	France	0	0	1	0	W	PT	FR

Fifa International Rankings from 1993 until 2018

Alright, so we gathered historical soccer data from 1930 until 2017. But this data does not *directly* tell us relevant/deterministic **features** for a country. One can say that a country with lots of goals scored might be a aggressive team but we will elaborate this idea later in *href:FeatureEngineering*. In order to get a measure for a country we collected international rankings for almost any country in the world directly from FIFA and some archives. Under following FIFA link you can see the current world ranking table. Gathering and preprocessing the files for rankings from 1993 until 2018 we received following country ranking table where we list *iso2-countrycodes* and the *Total_Points* for each country by year:

iso	total_points	year
AE	32	1993
AG	7	1993
AL	16	1993
AO	14	1993
AR	56	1993
AT	40	1993
DE	1609	2018
BR	1489	2018
PT	1360	2018
AR	1359	2018
BE	1337	2018
PL	1228	2018
ES	1228	2018
CH	1197	2018
FR	1185	2018

Ranking from fifaindex.com

Since our base idea was to gather as much data as possible in order to enrich our later prediction model, to forecast the outcome of a football match we needed more direct features which significantly contribute into the football outcome. Considering correlation of features, e.g if a country has a high *total_toints* value we consider the country team to be a strong one. But what kind of features would suggest a country to be strong as well? After some research we found that fifaindex lists country strengths from 2005 until 2018. The *features* they provide are *AttackStrength*, *MidfieldStrength*, *DefenseStrength* and an *OverallStrength*. One problem we realized after scraping data is that fifaindex.com does not provide much *strengths* values for all countries. In the following table are the scraped results:

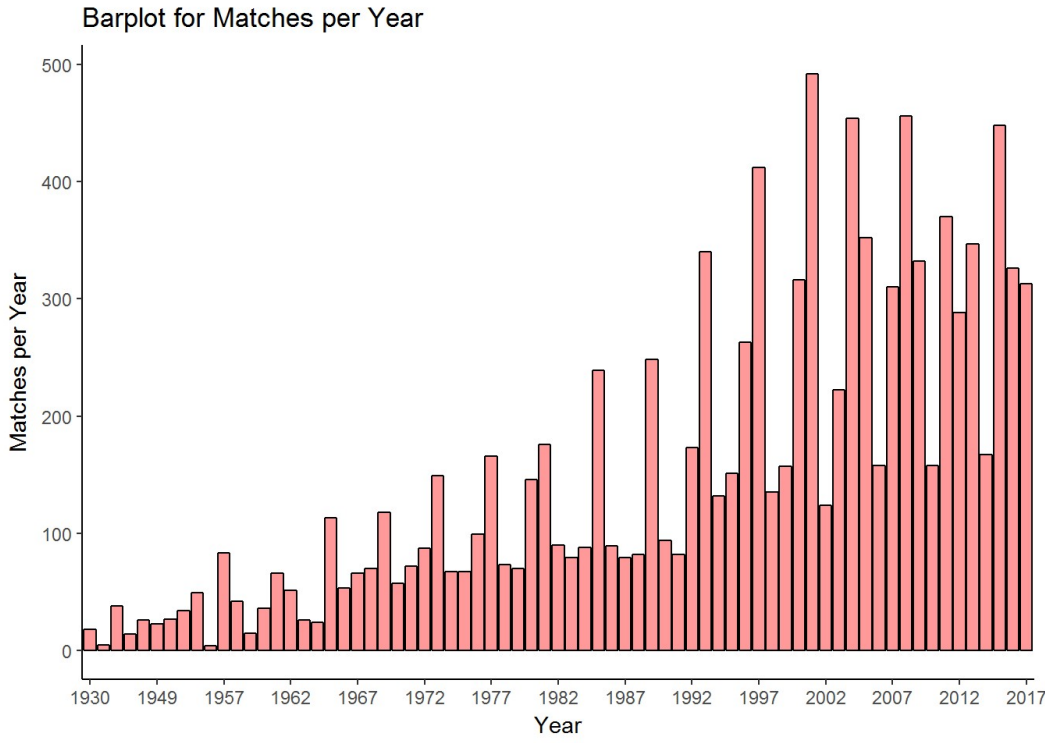
country	att	mid	def	ovr	year	iso
France	94	89	84	88	2005	FR
Brazil	92	88	87	88	2005	BR
Spain	90	88	86	88	2005	ES
England	89	88	88	88	2005	GB-ENG
Italy	92	83	86	87	2005	IT
Argentina	86	85	87	85	2005	AR
Bulgaria	70	70	69	70	2018	BG
China PR	70	69	70	70	2018	CN
Canada	69	72	66	69	2018	CA
New Zealand	71	68	67	68	2018	NZ
Bolivia	69	67	66	67	2018	BO
India	63	58	58	60	2018	IN

Exploratory Data Analysis

In this section we show some exploratory data analysis regarding historical Fifa World and Euro Cup data.
For the basis of exploratory data analysis we concatenated (`rbind`) the two historical Fifa World and Euro Cup data tables into one table and later filtered that resulting dataset according to participating team countries for fifa worldcup 2018.

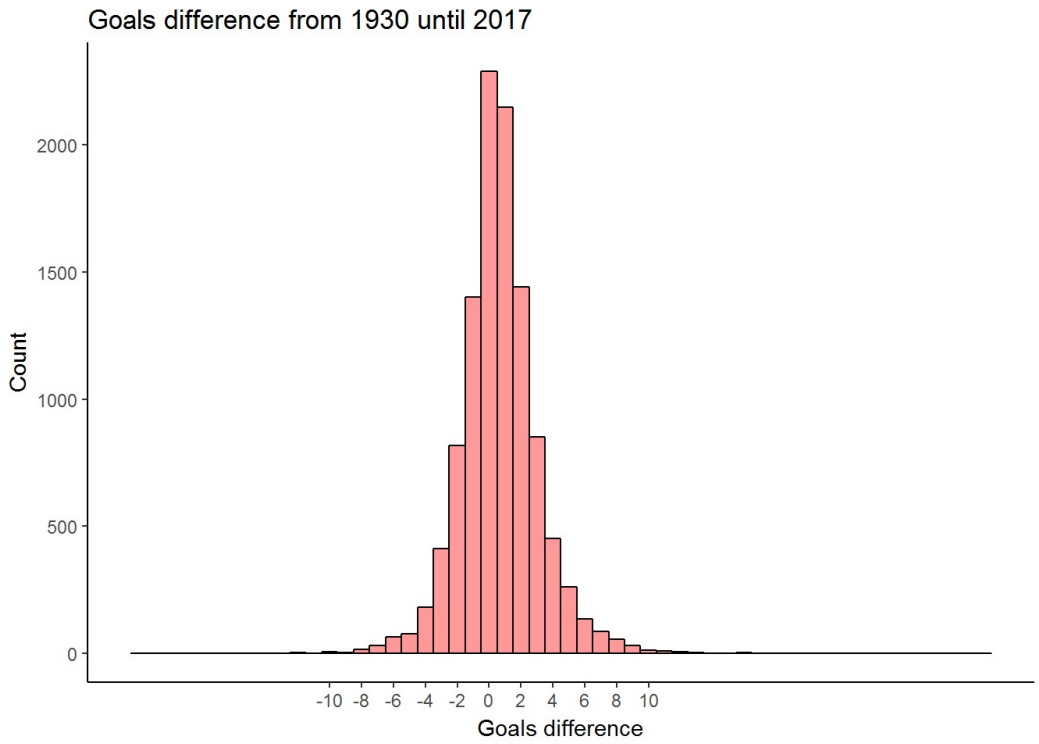
General Overview

In the following plot the overall matches per year are plotted. In this case we used the concatenated dataset without applying the filter to the respective World Cup 2018 squads.



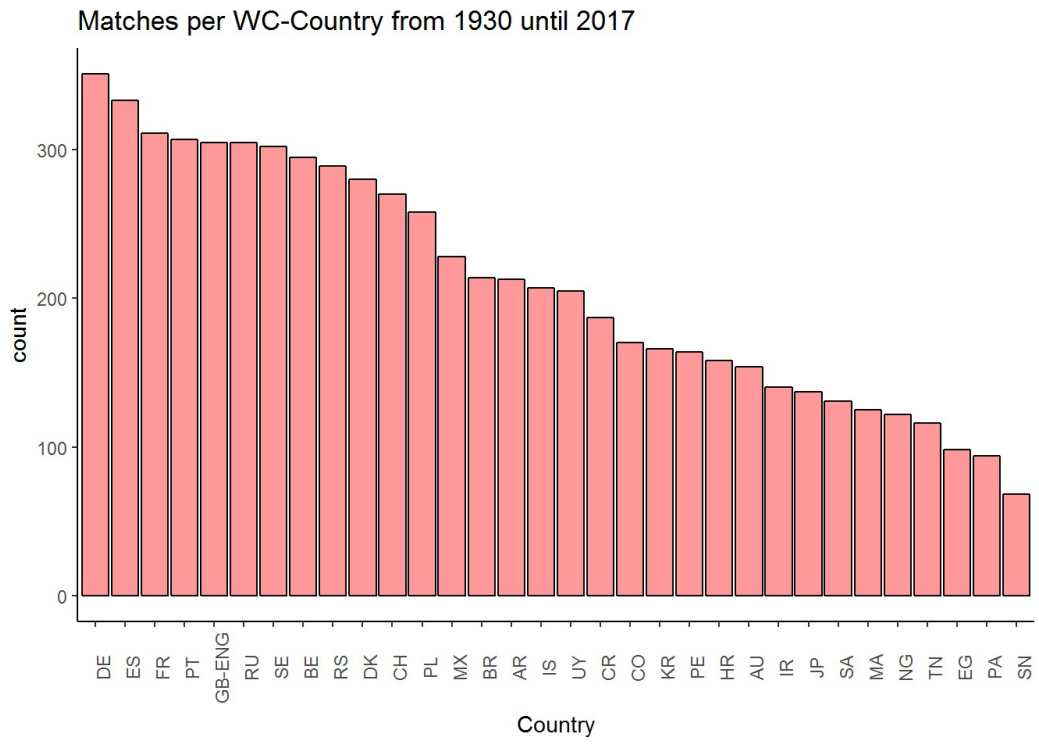
In the next plot we show the goal difference for all matches from 1930 until 2017. From the first view one might think that the distribution for this data, namely `difference = score_team1 - score_team2`, might be gaussian. Note that `difference`can only

take **discrete** values. Since the goals for each team are **counts**, with the assumption of independence of both scores the `difference` random-variable could be modelled as [skellam distribution](#).



Analysis for Competing Squads involved in World Cup 2018

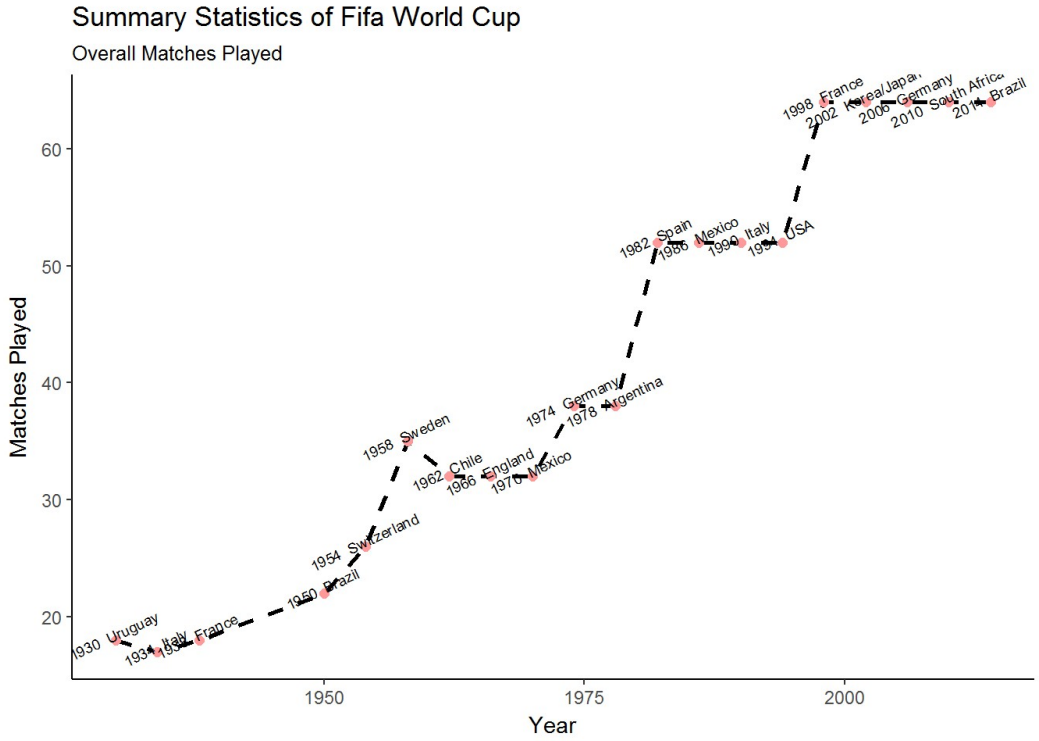
As we only gathered World and Euro Cup data, the current model lacks data from other confederations like Asia, Africa, South and North America and Oceania. This can be seen in the bar plot below for *Matches per WC-Country from 1930 until 2017*. Countries within Europe have a high amount of overall matches whereas Iran, Japan, Nigeria and Tunisia in our dataset are not represented much.
Futher improvements would be to gather data from other confederations and preprocess them into proper format as the current data.



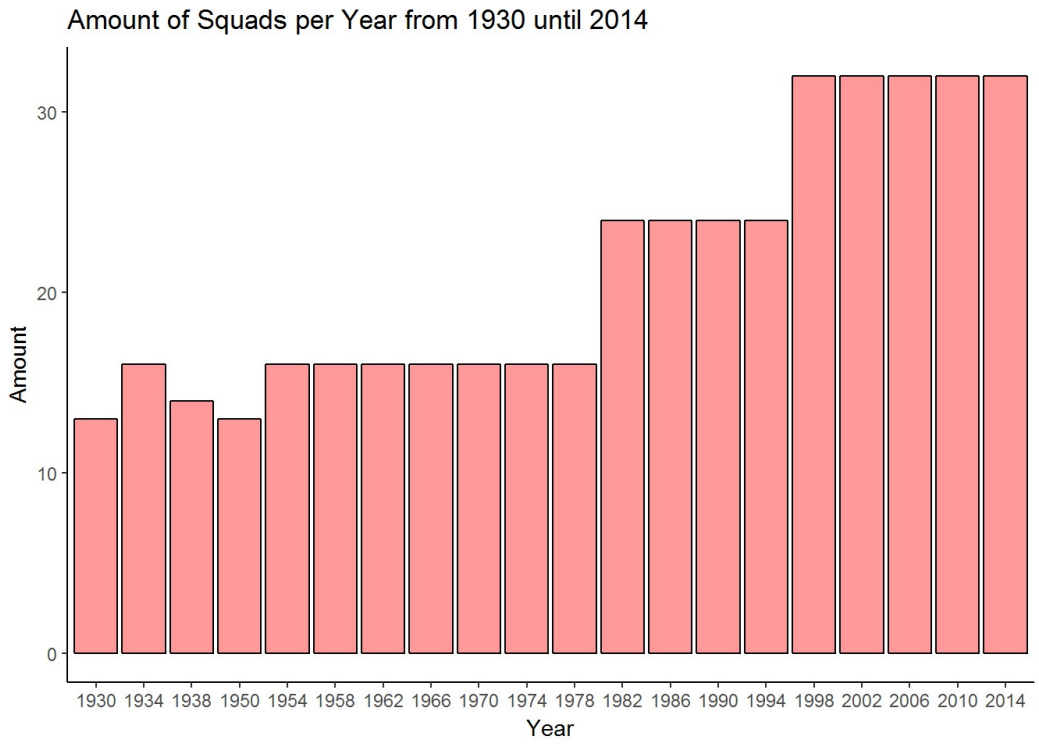
Overall statistics FIFA World Cup 1930 - 2014

For the world cups in the past we summarized the matches which took place within the championships (excluding preliminary matches).

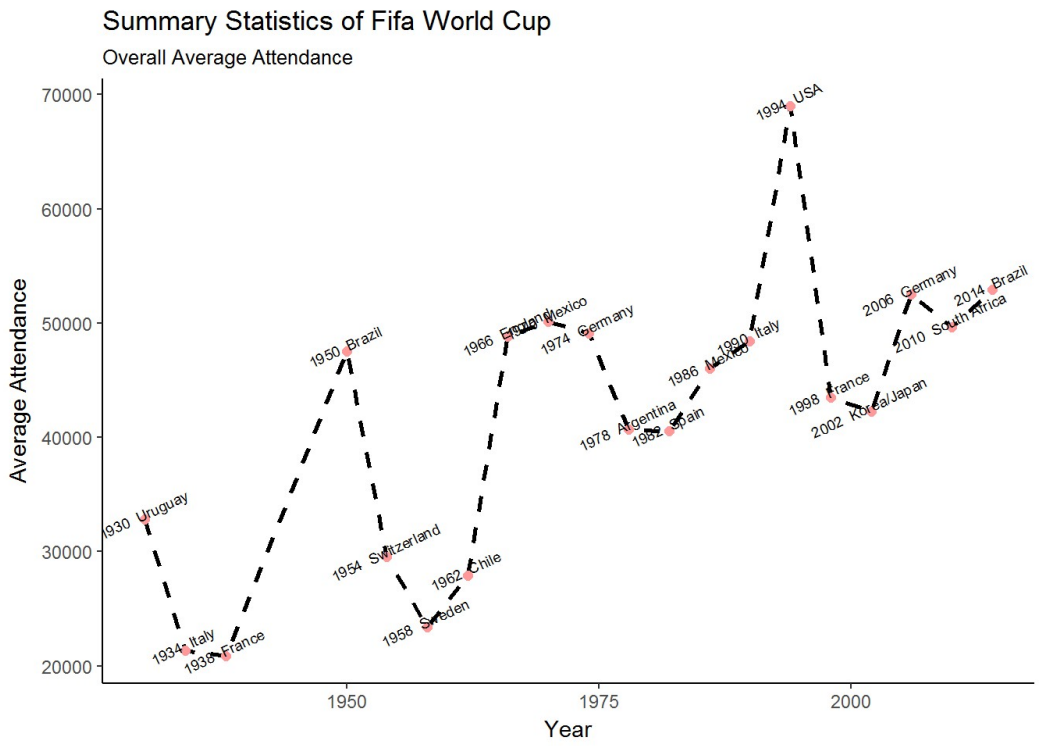
In total during the 20 worldcups there were 836 matches played.



The reason why the amount of matches increased is because during the years the number of competing squads rose as well the format/schedule for the matches with respect to group-phase and knockout-phase changed. In 1930 for example, the group phase consisted of 4 groups of 3 or 4 teams and the upcoming matches after that group-phase were a knockout of 4 teams directly, in which that team among the 4 with the most win became world champion.
For a more detailed version of the format in each year the history of fifa world cup (wikipedia) explains it well. In the upcoming barplot the number of squads in each year will be shown:



The average attendance of visitors for the FIFA World Cup increased during the years as expected:



Feature Engineering

What is Feature Engineering

Feature Engineering is the combination and transformation of features to improve the predicitive performance.
In classical introductory machine learning examples feature engineering does not have a big role in the ml-pipeline because mostly in the data generating process relevant features were already gathered together.
An example would be the Prediction of real-estate prices. In this case all relevant features are available and still it is possible to create new features which might improve the predictive performance of the machine learning model.

- In general, for supervised learning, we try to learn a relationship between "input" x and "output" y .
- For learning, there is training data with labels (the outcome y) available.
- Mathematically, we face a problem of function approximation:
  search for an f , such that, for all points in the training data, and also all newly observed points, fullfill: $y \approx f(x)$

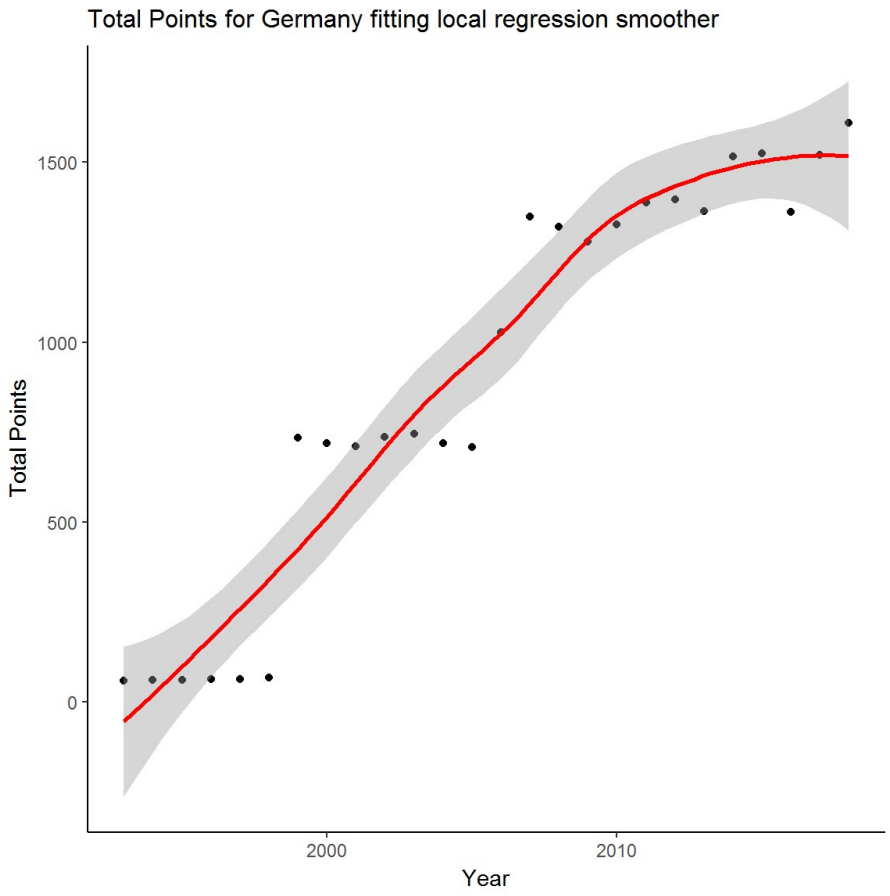
For the FIFA World Cup Prediction our task is a **Multiclass Classification** Task because our outcome y can take 3 values, namely: W, D, L for *win, draw* and *loss* for a team. Note that in either group-phase the outcome really can be 1 of the 3 possibilities. In the knockout-phase a *draw* would be the outcome after regular 90 minutes (full-time like in the quarterfinal in 2014) playtime.

Created Features for FIFA World Cup Prediction

Our general problem in the FIFA World Cup Prediction is that we **lack features**. Through data scraping and preprocessing we received information for the two competing teams, when the match took place, what kind of round it was (e.g *preliminary* or *group phase* etc) and the halfscores and finalscores.
In order to get more features one can simply gather more data. For example the weather conditions at a daymatch, the lineup for each team as well as betting odds.

FIFA-Ranking

We crawled FIFA Ranking for each competing country and since the rankings are published on a monthly basis due to our data which lists the matches in a year and not precisely by month in a year we average all 12 monthly FIFA Rankings for a year respectively. As created features `team1_total_points` and `team2_total_points` we join the historical matches data to the the fifa ranking based on the `team1iso` and `team2iso` and the `year` the match took place. Additionally we created a feature `total_points_diff` which calculates the difference for the Total Points of team1 and team2. Averaging all monthly based FIFA-Rankings from 1993 until 2018 for Germany and leads to following plot:

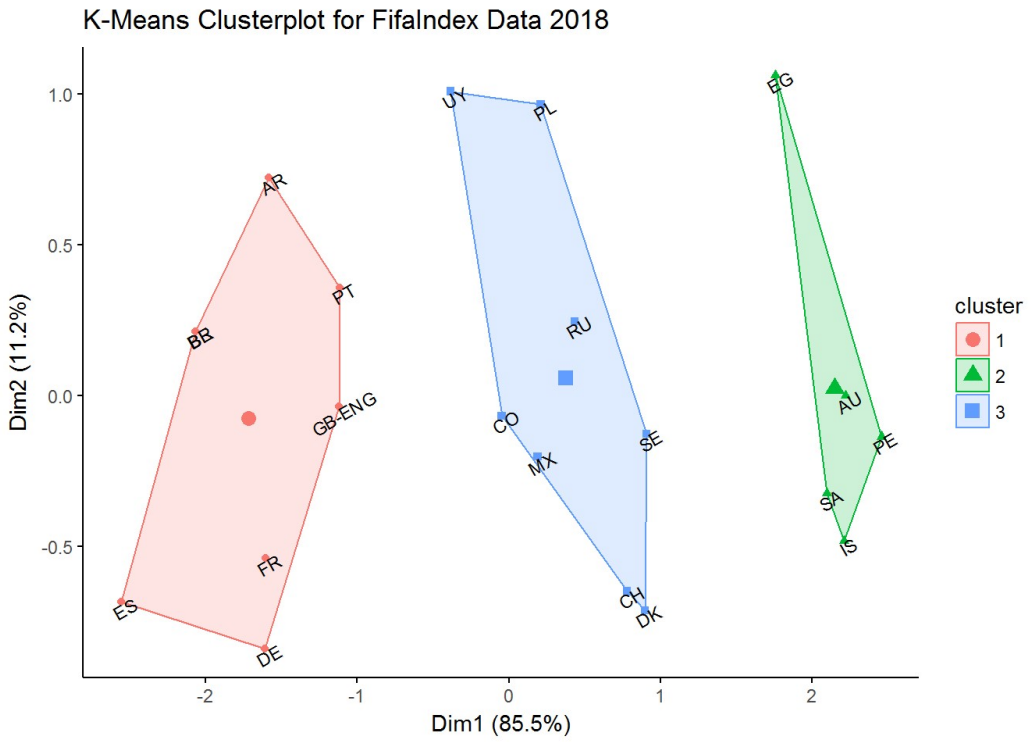


The reason why the total points for Germany changed is due to the different calculation methods as listed in the history of calculation methods. For the German ranking data we fit a *local regression smoother* in order to show that even for the different calculation methods (1993-1998, 1999-2006 and 2007-2018) Germany in each period is still likely to improve its total points during the years.

FIFA-Index

Since the FIFA-Ranking summarizes the total strengths of a country we tried to specify this strengths even more with raw data on the internet. One can describe the strengths of a team into terms of *attack*, *midfield* and *defense*. Scraping data from Fifaindex.com delivered us the wanted features. We still faced the problem that this website does not provide strength values for every country and is limited to the year ranges from 2005 until 2018. The first problem might cause issues in the later model training since not all machine learning algorithm can handle missing values. Restricting to only years above 2005 might lead to a poor model performance as well.

Filtering the fifaindex datatable for the competing teams of 2018 and applying *k-means clustering* under the assumption to group the countries into three groups: **weak**, **middle**, **strong** we receive following plot:



Note that before applying the clustering-algorithm we first calculated a principal component analysis in order to transform the 3 dimensional space (features: `att`, `mid`,`def`) into a 2 dimensional space, such that we can plot the result. The two resulting principal components explain in total 96.7% of variance. One can think the two principal components as `forward` and `backward` in which the `mid` strengths flows somewhat into `att` and `def` leading to respective new features `forward` and `backward`.
The plot indicates the strong squads. Having this in mind with more data (since fifaindex only provides strength features from 2005 on and not for all competing World Cup squads) we could either use the gathered data to enrich and feed our machine learning or create from the clustering results, which assigns for each country a class membership, a new feature `strengthClassification` which either is -1, 0 or 1, representing weak, middle and strong countries.
Since the scraped data from fifaindex is insufficient and leads to a high amount of missing values in our final dataset which we split into train, validate and testing , **we do not include the fifaindex data to our final dataset**.

Create Statistics and Features from past games

In order to enrich our prediction model and feed the learning algorithm with external information, e.g FIFA Total Points Ranking or Fifaindex.com Ranking, the history of world and europe cup matches still conceal information which can affect the outcome of a football match. For each competing country within the matches historical data we computed features like `avgShot`, `avgReceived`, `attackStr`, `defenseStr`, `avgWins`, `avgTurnovers` and so on. In order to adapt these features to train the prediction model these values ideally should be calculated with respect to each year from 1930 until 2018 on. Additionally it is intuitive that a team which had won 5 games in a row is likely to win the 6th match again. Since we believe that the ranking provided by Fifa is a reliable feature for the strength of a team we filtered the historic matches from 1993 on, neglecting all matches from 1930 until 1992.

Computing these features and joining tables we received a dataset with **31 features** and **7016 observations (matches)**.

Feature Selection

In the above section we computed and gathered features in order to feed our machine learning model. For training the model it is still questionable if all features really contribute into the outcome/prediction for a match. There are several reasons to do feature selection, namely:

1. **Interpretability and applicability:**
   A reduction of features for the resulting model makes it easier (even for machine learning models) to understand it.
2. **Scalability:**
   Reducing the number of features for large datasets can improve running time for training and evaluation a machine learning model.
3. **Algorithmic necessity:**
   A few methods cannot handle the state when the number of features is above the number of observations. Additionally not every multiclass classifier algorithm can handle any type of data. In our case we have numeric and categorical data. Since we want to make use of the Fifaindex.com ranking we have quite a lot missing values (NAs) for the years 1993 until 2013. A possible way to handle this problem is to do *imputation*, use machine learning algorithms which can handle missing value, or drop the features from fifaindex.com.
4. **Curse of Dimensionality:**
   Datasets with a lots of features might have mathematical properties like correlation or (perfect) collinearity. E.g computing `total_points_diff` in the direct way does not gain more insights, since this feature is already within `team1_total_points` and `team2_total_points` . In some classical statistical methods like linear regression the computation of coefficients/weigths for the final model cannot be solved due to perfect collinearity. Therefore dropping features which cause this problem is wanted. But for other machine learning models the `total_points_diff` might be a important feature, which contributes much into the outcome of a match (as we will see later). Having come to this point for feature selection we want to recap the k-means clustering in the **Fifaindex-subsection**. Note that we first applied a principal component analysis in order to shrink the 3-dimensional space into a 2-dimensional space. In order to shrink the feature space even more we suggested to classify each country to get either a weak, middle or strong label in the new feature `strengthClassification`. With this we would have reduced the number of **3 numeric features** into **1 categorical feature** which the machine learning algorithm might be better of using for prediciting the outcome of a match. This thought, of course, should be validated through several methods like computing the information gain, which we will do in the next subsection.

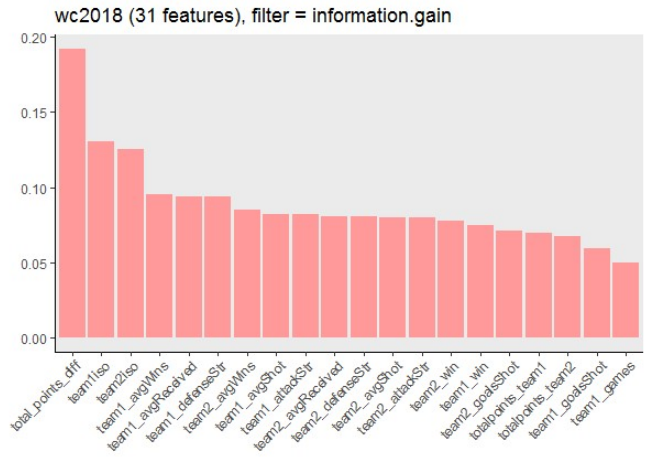
Selecting Features

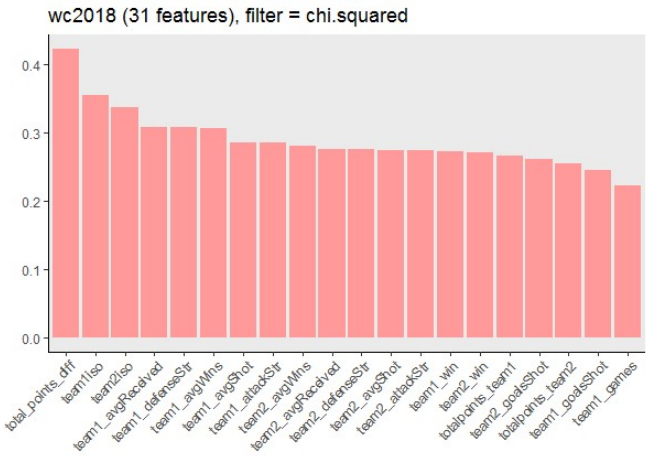
Some learning algorithms already contain feature selection, like **Lasso**, **Ridge**, **Elastic-Nets GLM**, **Decision Trees** or **Random Forest** which can shrink or even drop features that do not contribute much into the outcome y .
We applied filter methods to assign an importance value to each future. Based on these values the features can be ranked and a feature subset can be selected. As filter methods we used:

- **information gain** which computes an Entropy-based information gain between feature and target
- **chi squared** which computes the independence between feature and target

Subset Filter Results

Computing these values in the following we plot the features with the 20 highest values:





Having these results we select `year`, `round`, `total_points_diff`, `team1iso`, `team2iso`, `team1_avgWins`, `team2_avgWins`, `totalpoints_team1`, `totalpoints_team2`, `team1homeEdition` and `team2homeEdition` as features for our machine learning model.
The reason we also want to include the totalpoints for each team is because of the *magnitude* of the ranking. As mentioned earlier, throughout the years there were different calculation methods for the totalpoints. Therefore the magnitude changed. Also having a `total_points_diff`of 10 for example does not explain much about the two teams. It only indicated that team 1 is "stronger" than team 2. It might make a difference if Germany (totalpoints 1600) plays against Brazil (totalpoints 1590) which leads to a totalpoints difference of 10 or Panama (totalpoints 510) against Russia (totalpoints 500). The last 2 features mentioned are flag columns which can take values -1 (no home country), 0 (game is preliminary, hence not a championship match) or 1 (home country).
For our prediction model in 2018 Russia would be flagged as 1 and all other countries as -1.

Model Training

For the predicting the outcome of a world cup match we trained two different models.
In terms of model-class our primary aim is a **good prediction** over understanding the underlying model. Hence we will rather choose a machine learning algorithm than an classical statistical model. Statisical models should be used if the priority layes on **valid insights** over good prediction (e.g. understanding the influence of a feature).
The first model, we call it **no-live-model**, contains following features: `year`, `round`, `total_points_diff`, `team1iso`, `team2iso`, `team1_avgWins`, `team2_avgWins`, `totalpoints_team1`, `totalpoints_team2`, `team1homeEdition` and `team2homeEdition`.
The second model, we call it **live-model**, contains following features: `year`, `round`, `total_points_diff`, `team1iso`, `team2iso`, `hst1`, `hst2`, `totalpoints_team1`, `totalpoints_team2`, `team1homeEdition` and `team2homeEdition`.
The difference between the models is, that the no-live-model contains average wins for each team, whereas the live-model contains current half/live-scores of a match.

Train/Validate and Test Datasets

For the upcoming computations (benchmark, parameter tuning and model building) we split the final dataset of **7016 matches** into a train/validate dataset of **6952 matches** in order to train and validate the model. Furthermore we want to **test** and predict the Fifa World Cup 2014, consisting of **64 matches**. Hence one can say that we do the "classical" test-step 2 times. Namely in the validation step (after building the model) *validate* the model with data, it has not seen before. In addition we then predict the Fifa World Cup 2014.

No-Live-Model

The current probabilities displayed on our website are predictions from the no-live-model. In the following we will explain the technical details of model building.

Choosing the correct Machine Learning Algorithm

Sometimes people are wondering whether there is exactly **one** algorithm which works best for a problem. The clear answer is no. As in many blogpost, like this one from Towards Data Science, one always have to look at the data and its features. Since our final dataset contains of **7016 observations (matches)** and **11 features**, which are numeric and categorical, there are several algorithms/methods which we can apply. Also neglecting the fifaindex data we received a dataset with no missing values. Otherwise we should have choosen (machine learning)-algorithms which can handle missing values well. To have a first glance of the performance for classical statistical methods and machine learning algorithms we ran a benchmark study on our dataset, in order compare the different algorithms. Our resampling strategy was a **5-fold Cross-Validation**. This means in a nutshell, that for each algorithm we split the 6952 observations randomly in 5 equal-sized chunks, but with stratification, such that all classes (W,D,L) are represented correctly in each chunk. Training will be performed on the 4 chunks and testing on the 1 chunk left. This procedure will be repeated 5 times. For a more detailed explanation of cross validation have a look at this blogpost.

Benchmark Results

For a first glance we chose 7 different methods (5 machine learning and 2 classical statistical algorithms [naive bayes and glm elastic net]) with its default settings for performing the benchmark. We received following **aggregated performance measures** from the 5 runs for each method:

task.id	algorithm	validate.acc	validate.mmce	time.train	time.predict
wc2018	gradient boosting machine	0.6371207	0.3628793	1.022	0.026
wc2018	naive bayes	0.6189920	0.3810080	0.020	0.334
wc2018	kernel support vector machine	0.6454673	0.3545327	14.424	2.836
wc2018	glm elastic net	0.6384178	0.3615822	25.250	0.114
wc2018	decision tree	0.6374139	0.3625861	0.256	0.100
wc2018	random forest	0.6208641	0.3791359	5.432	1.560
wc2018	neural network	0.4227673	0.5772327	13.588	1.560

As a result one can see that the aggregated test accuracy for the support vector machine is highest, but training takes in comparison to the decision tree, which has almost the same aggregated test accuracy, quite long. We believe the reason for the fast training time of the decision tree is, the java implementation of the function weka.classifiers.trees.J48 in contrast to the slower implementation of the svm from the kernlab package.

Hyperparameter Optimization

Many machine learning algortihms have hyperparameters that need to be set. In the benchmark results we applied the several methods with its default parameters provided by the developer who programmed the functions. Often suitable parameter values are not obvious and it is preferable to tune / optimize the hyperparameters, that is automatically identify values / sets that lead to the best performance. In order to perform parameter tuning one must specify the search space, namely a discrete/continous set of values, which a parameter can take. In addition to that there are different approaches, how one can conduct the tuning process. We will stick to **random search** because the search space for the different algorithms can be quite big (remember the *curse of dimensionality*). As an example the support vector machine, provided with an Gaussian-Kernel has at least two hyperparameter to be tuned for good performance. A regularization constant C, which playes a role in the lagrangian

optimization to find the support-vectors, and a kernel hyperparamter γ . Both values are continuous, so to conduct the tuning one can specifiy the search space to be $C \in [1, 100], \gamma \in [0.1, 10]$. If we were to perform grid search we would have to run through the cartensian product of $[1, 100] \times [0.1, 10]$ which can be very costly. Also it can occur that we *blow* up computational resources if we walk in parameter combinations, which results to bad performance (low accuracy) since grid search just executes the model training with the parameter defined. For example all values from $C > 10$ and $\gamma > 5$ are leading to low accuracies. Hence we would rather not use computational resources on performing the tuning on values above it. But grid search does it, because the search space is defined. For that reason and for the sake of tuning parameters for more algorithms we stick to **random search** which randomly picks values from the defined search space.
We used **5-fold Cross-Validation** again and for the random search set **10 Iterations**. This means in each iteration out of 10 we randomly pick a set out of $[1, 100] \times [0.1, 10]$ and then perform the 5-fold-cross-validation. In this section we just performed the tuning for gradient boosting machine, kernel support vector machine, glm elastic net, decision tree and random forest. Extracting the **aggregated cross-validation results** with best performance, we received:

task.id	algorithm	validate.acc	validate.mmce	time.train	time.predict
wc2018	gradient boosting machine	0.6368370	0.3631630	1.976	0.024
wc2018	kernel support vector machine	0.6368357	0.3631643	21.646	4.776
wc2018	glm elastic net	0.4850361	0.5149639	24.070	0.086
wc2018	decision tree	0.6374097	0.3625903	0.244	0.096
wc2018	random forest	0.6276218	0.3723782	2.412	1.482

Comparing the validation accuracies for each algorithm we see that with parameter tuning the classical glm elastic net performed worse than with its default algorithms. This happened since applied random search and all iterations picked “worse” parameters than the default arguments. Since we prioritized **good prediction**, we choose the kernel support vector machine as our machine learning algorithm. In fact gradient boosting machine, kernel support vector machine, decision tree and random Forest perform almost the same way in terms of accuracy. We decide to choose the support vector machine, since we tuned this model only with respect to the 2 parameters C and γ . The other three algorithms contain more parameter. Gradient boosting machines and random forest are both ensemble techniques, which are based on decision trees. We will see in the later live-model, how decision trees in general can perform better, when half/live scores are included in the prediction.

Choose optimal Support Vector Machine and Predict the Fifa World Cup 2014

From the parameter tuning we extract the optimal parameters and model for the support vector machine. Since the performance measures in the table above are **aggregated** measures for the 5 cross-validation iterations, we conduct another 5-fold cross-validation with the optimal parameter in order to extract the svm models (the reason why we have to do this step (again through resampling) is because the parameter tuning function does not provide keeping the models from the cross-validation iterations).

Displaying the results for the cross-validation with respect to the support vector machine we receive:

iter	validate.acc	validate.mmce	timetrain	timepredict
1	0.6446043	0.3553957	15.59	3.76
2	0.6266187	0.3733813	15.77	3.19
3	0.6506111	0.3493889	16.86	3.28
4	0.6359712	0.3640288	17.13	3.03
5	0.6277898	0.3722102	16.00	3.47

Our best support vector machine model (from iteration 3) has a validation accuracy of **65.06 %** and misclassification rate of **34.94 %**.

So how does our model predict the Fifa World Cup 2014 ? Note that we “cheated” a little bit, since in the train/validate step for the training phase, data above 2014 was used. We received following results in predicting the World Cup 2014:

On predicting the Fifa World Cup 2014 our no-live support vector machine classifier has an (test) accuracy of **64.0625 %**

Live-Model

The motivation for the live-model lays in embedding live/half scores of a match. It might be, that apart from the `total_points_diff` the current/half score can change the probability for a team to win a match. If for example, Germany in the past often lost the first half of a match but in the second half prevailed and won the match. Also it is interesting to examine the magnitude of goals. It would be incredible if the machine learning algorithm can learn that germany in past lost the first half with many goals behind but then won in final time.
For the live-model we used the same pipeline with benchmarking, hyperparameter tuning and model building as before including 5-fold Cross-Validation and the test/validate and test dataset.

Benchmark Results

As before we ran a **5-fold Cross-Validation** to gain first insights on how different methods perform on the train/validate datasets. For the learners with its implemented default hyperparameters we received follwing **aggregated performance measures**:

task.id	algorithm	validate.acc	validate.mmce	time.train	time.predict
wc2018	gradient boosting machine	0.6839672	0.3160328	1.072	0.026
wc2018	naive bayes	0.7500020	0.2499980	0.018	0.318
wc2018	kernel support vector machine	0.9127755	0.0872245	8.722	1.378
wc2018	glm elastic net	0.9088825	0.0911175	13.440	0.094
wc2018	decision tree	0.9067168	0.0932832	0.266	0.128
wc2018	random forest	0.8607247	0.1392753	4.678	1.808
wc2018	neural network	0.7948390	0.2051610	17.270	1.578

The aggregated test accuracy on the validation sets for the live-model is in contrast to the no-live-model for any method *higher*. It is intuitive that with actual (half) scores for each team the outcome of the match after final time can be predicted easier.

Hyperparameter Optimization

Setting the environment of hyperparameter tuning just like the no-live model we receive following **aggregated cross-validation results**:

task.id	algorithm	validate.acc	validate.mmce	time.train	time.predict
wc2018	gradient boosting machine	0.8917203	0.1082797	3.194	0.030
wc2018	kernel support vector machine	0.9130575	0.0869425	8.562	1.564
wc2018	glm elastic net	0.4860151	0.5139849	16.594	0.090
wc2018	decision tree	0.9107515	0.0892485	0.178	0.116
wc2018	random forest	0.8826376	0.1173624	3.160	1.544

Again the support vector machine and decision tree are both good classifiers. In the next step we will investigate the support vector machine, gradient boosting machine and random forest. The reason why we want to include the last two algorithms over the decision tree lays in their background. Both, the gradient boosting machine and random forest are ensemble learning

methods. In a nutshell ensemble methods means that it creates a multitude of weak classifiers (in both cases the basis are decision trees) and by combination become a strong classifier. For more insights have a look at this blogpost.

Choose the optimal classifier and predict the Fifa World Cup 2014

We conducted another **5-fold Cross-Validation** again and trained a gradient boosting machine, support vector machine and random forest with its optimal parameters. After training the gradient boosting machine we received for following validation results:

iter	acc	mmce	timetrain	timepredict
1	0.8890490	0.1109510	3.360	0.11
2	0.9026676	0.0973324	3.480	0.03
3	0.9020173	0.0979827	3.290	0.05
4	0.8744589	0.1255411	3.750	0.03
5	0.8846431	0.1153569	4.060	0.03
aggregated gbm	0.8905672	0.1094328	3.588	0.05

And for the support vector machine:

iter	acc	mmce	timetrain	timepredict
1	0.9135447	0.0864553	10.700	1.31
2	0.9178082	0.0821918	7.610	1.31
3	0.9293948	0.0706052	8.010	1.37
4	0.8961039	0.1038961	7.800	1.28
5	0.9077145	0.0922855	8.110	1.33
aggregated svm	0.9129132	0.0870868	8.446	1.32

And finally the random forest:

iter	acc	mmce	timetrain	timepredict
1	0.8739193	0.1260807	2.880	1.580
2	0.8868061	0.1131939	2.880	1.580
3	0.8984150	0.1015850	2.860	1.530
4	0.8744589	0.1255411	2.870	1.540
5	0.8788753	0.1211247	2.920	1.610
aggregated rF	0.8824949	0.1175051	2.882	1.568

Comparing the aggregated results for each methods, the support vector machine performs best. For the live-model we will not choose the support vector machine because on predicting the fifa world cup 2014 it performs from all three worst with **87.5 %** accuracy. The random forest has an test accuracy of **93.75 %** and the gradient boosting machine is able to predict all games of the world cup 2014 correct, leading to an accuracy of **98.4375 %**.
The gradient boosting machine predicting all almost all World Cup matches from 2014 correctly made us suspicious. In order the gain more insights we will compare the predictions for the random forest and gradient boosting machine.

In the upcoming table the predictions from the random forest will be displayed:

It is interesting to investigate the matches, which our random forest missclassified. It turns out that 4 games are missclassified. Following matches are missclassified by our random forest:

year	round	team1iso	team2iso	hst1	hst2	truthTeam1	predictTeam1	prob.Team1Win	prob.Team2Win	prob.Draw
2014	GROUP_STAGE	RU	KR	1	1	D	W	0.4663	0.0777	0.4560

year	round	team1	iso	team2	iso	hst1	hst2	truth	Team1	predict	Team1	prob.Team1Win	prob.Team2Win	prob.Draw
2014	GROUP_STAGE	JP		GR		0	0	D		L		0.0949	0.5264	0.3787
2014	GROUP_STAGE	DE		GH		2	2	D		W		0.6562	0.1250	0.2188
2014	GROUP_STAGE	US		PT		2	2	D		W		0.3962	0.3585	0.2453

In total there are **13 draws** in the world cup. It turned out that our random forest misclassified 4 out of 13 draw matches. Intuitively one can argue that draws are indeed less likely and therefore don't occur much in a world cup. So how does the gradient boosting machine seperate between win/loss and a draw regarding the 4 misclassified match? Selecting the probabilities for the gradient boosting machine and filtering to draw matches only leads to following table:

Apparently the gradient boosting machine always predict a draw outcome, if the halfscore for both teams are even unless one participating team is germany (good for us actually). Although the score between Germany and Ghana is 2-2 the gradient boosting machine predicts that Germany wins with a probability of 0.407, indicating that in this case the `total_points_diff` contributes more into the predicted outcome. After filtering the data tables and analysing it might be, that the gradient boosting machine is giving "more weight" into the half scores, rather than the `total_points_diff` (Note that this is also a feature included into the model, but not displayed for the sake of clarity).

As a final conclusion and with the belief in mind that the live-model should really model the live scores instead of half score we will eventually **choose the gradient boosting machine** as our live-model.