Report on the Final Project on Machine Learning

Omar Backhoff Larrazolo IMAT 03655997
Evgeny Agamirzov IMAT 03647819
Oleksandr Shchur IMAT 03646788

All the source code and data can be downloaded from https://github.com/shchur/ml_project

Predicting sports results is usually a complicated task given that they rely on human effort, and it is the challenge that drove us into this project. We want to determine how well some Machine Learning algorithms can perform in this area.

Our choice is football. More specific, we want to predict the results of the **English Premier League (EPL)**. Three major challenges were to be faced:

- Get statistical data of the EPL (for last eight seasons including the current one);
- Build our training set with the features based on the gathered data;
- Compare different classifiers and determine which one performs better.

1. Obtaining the data

The EPL statistics has been collected from wildstat.com. For this purpose a web-scraper was written (*project_location/parser/scraper.php*).

It gathers the information about all the table standings and matches that have happened during the last eight seasons giving **2280** matches and **228** tables in total (one table and ten matches per matchday).

The scraper reads the *html* tags and writes the data into *.csv* files (*project_location/data/tables.csv, project_location/data/matches.csv*). The scraper can be run by executing *scrape.bash*.

2. Building the training set

Each training example corresponds to one match. To build a training example for one particular match we are looking back to the history and collect the statistics of the teams before this match has happened. To be precise, we are gathering the information about all the

matches that have occurred this season and two seasons back before the game. We use the result of the match as an outcome for this training example.

For **each team** participating in the match we are collecting the following information (statistics of the current season and two last seasons **before the match**):

- Average points per match;
- Average away points per match (away matches);
- Average home points per match (home matches);
- Average goals scored per match;
- Average goals received match;
- Average points per match against this particular opponent;
- Average goals scored per match against this particular opponent;
- Team position in the table before the match has happened.

This gives eight features for each team participating in the match. Thus, resulting training example would look like this:

Team 1 statistics	Team 2 statistics	Outcome

The outcome has only three options:

- Team 1 wins;
- Team 2 wins;
- Draw.

The statistics gathered by the scraper contains the data for eight seasons (from 2007 to 2015). We start computing our training set from the season 2009 - 2010 because for each game we have to look two seasons back (found to be the optimal value for the number of past seasons to consider). We also apply custom weights to regulate the influence of the previous seasons, e.g. the information of old seasons should not be as influential as the most recent ones. Eventually our training set consists of training examples for each match starting from season 2009 - 2010 till now.

In order to build the training set execute *build_training_set.bash*. Make sure you have *python-numpy* and *python-pandas* installed. The training set is then saved as *.csv* file (*project/data/training/training_set.csv).*

Some of the entries usually have incomplete information. This happens because some of the teams did not participate in previous seasons. These entries were ignored in order to reduce uncertainty, leaving the training set with a total number of around **1900** entries.

3. Making predictions

In our project we have applied the following classification algorithms from scikit-learn and neurolab libraries:

- Logistic regression
- kNN
- SVM
- Random Forests
- Feed Forward Neural Network

3.1. Prediction quality estimation

In order to estimate the efficiency of specific algorithms we used **10-fold cross validation**. The data set was split into 10 even parts, then the algorithm was trained on 9 of them and the predictions were made for the left-out part. The procedure was repeated for each of the parts and the correct guesses percentage was then averaged among the trials. The obtained value was used as the accuracy measure for the algorithm.

3.2. Data normalization

Data may differ by orders of magnitude in the different fields of our training vector, which may lead to worse performance for some algorithms. Normalization was performed by dividing each element of the training set matrix A by the largest value of the column it is in:

$$a_{ij} = \frac{a_{ij}}{max_k |\, a_{kj}|}$$

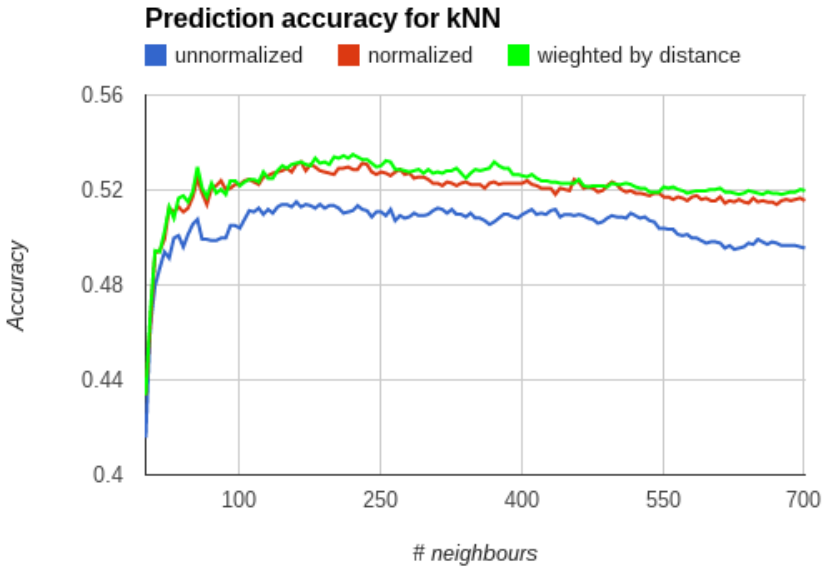
The results for normalized data are included where appropriate.

3.3. Optimal hyperparameter evaluation

Behavior of each algorithm is defined by the choice of the hyperparameters. For each combination of the hyperparameters cross-validation has been applied and the best choice has been saved.

3.3.1 K-Nearest Neighbours

The parameter with the highest effect on the prediction quality is *k* - the number of neighboring data points to consider. Values in range [1, 700] were tested.



As expected, normalization substantially increases the algorithm's accuracy. The regular implementation of the algorithm determines the probability of a point x belonging to class *c* as:

$$p(z = c \mid x, K) = \frac{1}{K} \sum_{i \in N(x)} I(z_i = c)$$

An idea for improvement of the algorithm is to weight each neighboring point's influence by normalized distance to that point:

$$n(x, x_i) = \frac{d(x, x_i)}{\sum_{k \in N(x)} d(x, x_k)}$$

$$p(z = c \mid x, K) = \sum_{i \in N(x)} I(z_i = c) \cdot n(x, x_i)$$

The latter approach gives a minor increase in accuracy, however it's also more computationally intensive. Table below represents the **best achieved results**:

	Best K	Prediction accuracy [%]
Unnormalized data	161	**51.5**
Normalized data	166	**53.1**
Normalized data with weighting by distance	221	**53.2**

3.3.2 SVM

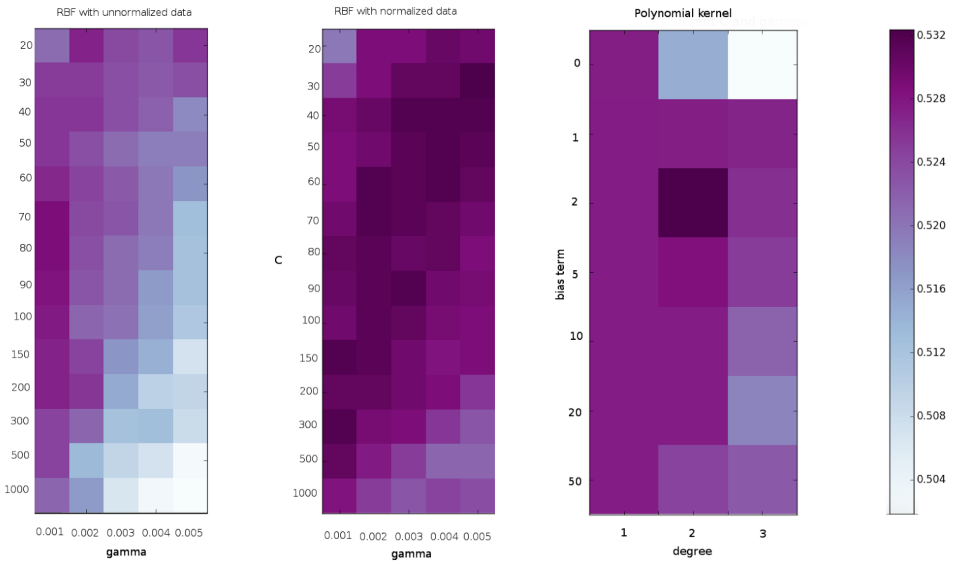
The behavior of the SVM algorithm is largely defined by the choice of the kernel function. In our project we considered **polynomial kernel** and **radial basis function kernel**.

RBF kernel: $exp(-\gamma \cdot |x - x'|^2)$

In this case the behavior is conditioned by the kernel parameter γ which determines how far influence of single example reaches and the penalty parameter for the error term ***C***. The predictions were computed for different combinations of the hyperparameters.

Polynomial kernel: $(< x, x' > + r)^d$

The results were computed for varying values of the bias parameter ***r*** and the degree ***d*** up to 3. Higher degrees were much more expensive to compute and didn't provide a substantial increase in the prediction quality.



From the plots it can be seen that normalization has positive effects on the classifier results for our data set. One should, however, keep in mind that this is not true for every problem.

	gamma	C	Prediction accuracy [%]
RBF unnormalized data	0.001	70	**52.9**
RBF normalized data	0.005	30	**53.1**
Polynomial normalized data	r = 221	d = 2	**53.3**

3.3.3. Random Forest Classifier

There are a couple of parameters that we can change in order to obtain different results. These parameters are: number of estimators and the criterion of the quality of split.

We worked with 1000 estimators, or trees, all the time and we tested the only two possible criterions for the quality of split: *gini*, which measures the impurity and *entropy*, which measures information gain.

Two major observations when using this classifier: the results are quite inconsistent and using *entropy* as a criterion is considerably slower.

The best results we were able to obtain were using the *gini* criterion with an accuracy of 0.531830 with normalized data and 0.53231 with unnormalized data.

3.3.4 Logistic Regression

For this classifier we can adjust the parameter *C* which is the inverse of the regularization strength. Also we can select the norm used in the penalization, either L1 or L2.

Trying with the different combinations we obtained the following results:

	C	Prediction accuracy [%]
Unnormalized data (L1 norm)	0.15	**53.4**
Unnormalized data (L2 norm)	0.9	**53.7**
Normalized data (L1 norm)	0.36	**53.3**
Normalized data (L2 norm)	0.9	**52.9**

3.3.5. Neural Network

For this part the implementation of a feedforward neural network from the *neurolab* library has been used. The network consists of 16 input layer neurons (corresponding with the number of elements in the training vector), a hidden layer consisting of 10 neurons and one output neuron. Prediction accuracy was obtained by 10-fold cross-validation.

# neurons in input layer	# neurons in hidden layer	# neurons in output layer	Prediction accuracy [%]
16	10	1	**46.2%**

3.4. Real problem prediction

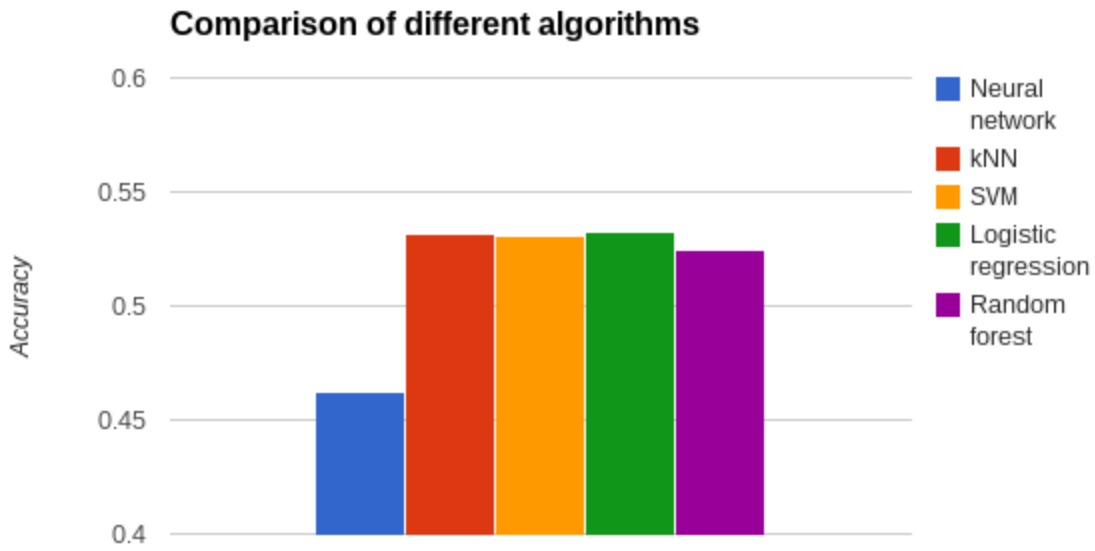
After estimating the optimal hyperparameters we want to test the algorithm on a real problem and see how accurate it is. It is also interesting to see whether the predictions make sense to the regular football fan who can actually say which team is more likely to win the game. The table below shows the pairs played in the last match day of EPL, which actually finished on the 22.02.2015.

Team 1	*vs.*	Team 2	Actual result	Prediction
Sunderland	*vs.*	*West Bromwich*	Draw	Team 2
Aston Villa	*vs.*	*Stoke City*	Team 2	Team 2
Chelsea	*vs.*	*Burnley*	Draw	Team 1
Crystal Palace	*vs.*	*Arsenal*	Team 2	Team 2
Swansea City	*vs.*	*Manchester United*	Team 1	Team 2
Hull City	*vs.*	*Queens Park Rangers*	Team 1	Team 1
Manchester City	*vs.*	*Newcastle United*	Team 1	Team 1
Tottenham	*vs.*	*West Ham United*	Draw	Team 1
Everton	*vs.*	*Leicester City*	Draw	Team 1
Southampton	*vs.*	*Liverpool*	Team 2	Team2

To predict these outcomes we used an ensemble of algorithms (SVM, kNN, RF, LR) and let them vote. All algorithms voted equally and agreed on the final result. We observe the 50% prediction accuracy (as it was expected).

4. Conclusions

With finely tuned parameters our algorithms produced predictions of the following accuracy



Even though we got exactly 50% of the real problem predictions correct, these were impressively accurate in terms of common sense (strong teams were supposed to win against weak ones); someone who knows the English Premier League could have had similar predictions.

In reality, predicting football results is very difficult because there are many factors involved which we cannot control and which we did not take into account for this particular project: injuries, priorities with respect to other tournaments, human factor, etc. But we showed that using the correct training set one can achieve reasonable predictions and while having 53% of accuracy is not exactly reliable, it is definitely better than a random choice out three possible options.

Another observation we have made is that the data set has a major impact on the quality of the predictions. It can also be seen that with a reasonable choice of hyperparameters most of the algorithms are producing similar outputs.