ML project

Matúš Vojčik

1. Data preparation

- First thing I noticed there was a lot of missing values (some data estimation might help here).
- Some columns had wrong values (double kills).
- I checked if the data were consistent and I found 2 errors. Game id 2864 and 2846 had wrong team id, so I fixed it because 5 players from this game had different team id.
- I wanted to keep it simple and low dimensional for the sake of this project, so I picked those columns, which I believe, are the most important (games played(GP), kills(K), deaths(D), assists(A), gold(G), wins(W), loses(L)).
- I left out matches that didn't have any kill, death… info

1. Data preparation

- I created statistics for each **game, player, team** from the given files:

1. **Game** (winner_K, loser_K, winner_D, loser_D…)
2. **Team** (GP, K, D, A, G, W, L)
3. **Player** (GP, K, D, A, G, W, L)

Team K is the sum of all players' kills for that team.

Game winner_K is the sum of all winner players' kills in that game. Etc.

2. Features

- The stats of each game are left out for creation of its features (it would not be ok to use the stats of a game for its prediction).
- I treated every game as if it were the last game played (Other option would be to treat the games chronologically, i.e. use only the data that was available at the time the game was played.

2. Features

- To keep it simple I give an example how a feature ***f*** and class ***c*** is computed for a game ***g*** with winner team ***w*** and loser team ***l*** from the "kills" stat

```
w_k = (w['K'] – g['winner_K'])/(w['GP'] - 1)
l_k = (l['K'] – g['winner_K'])/(l['GP'] - 1)

If random < 0.5
        f = w_k – l_k, c = 1
else
        f = l_k – w_k, c = 0
```

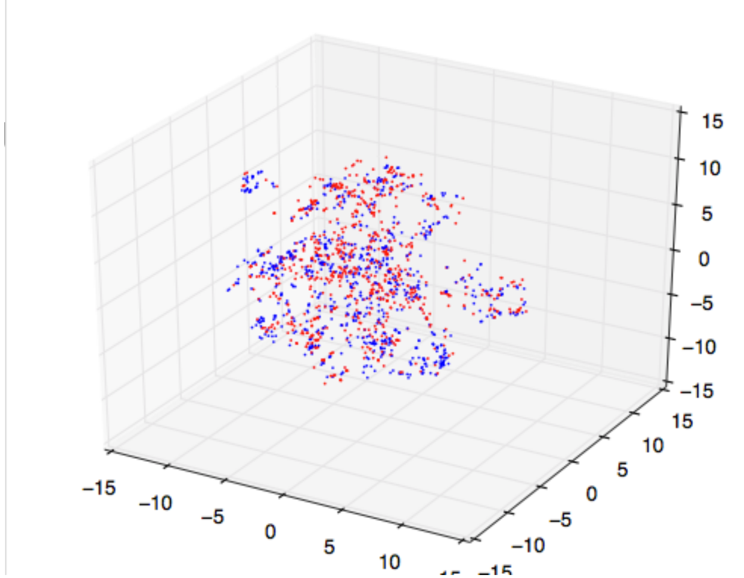
- The random < 0.5 is there in order to have roughly the same amount of 0 and 1 classes. Otherwise there might be a bias towards 1 class
- The difference of stats cuts the features in half and makes the features symmetric for the ML model. If I kept the features separate the ML model might not treat the features the same and if I switched them on the input the result might not be the same either.
- make_features_teams.py

2. Features

- The features of a game are calculated from the stats of each team.
- Stats of each team are calculated from all the players that played for that team.
- It might be worth investigating other options also e.g. stats for each team for a game would be calculated only from the 5 players that are going to play. Then there is option to count all of their games or only the games that they played for that team or some kind of weighted average of all the methods.

3. Visualization

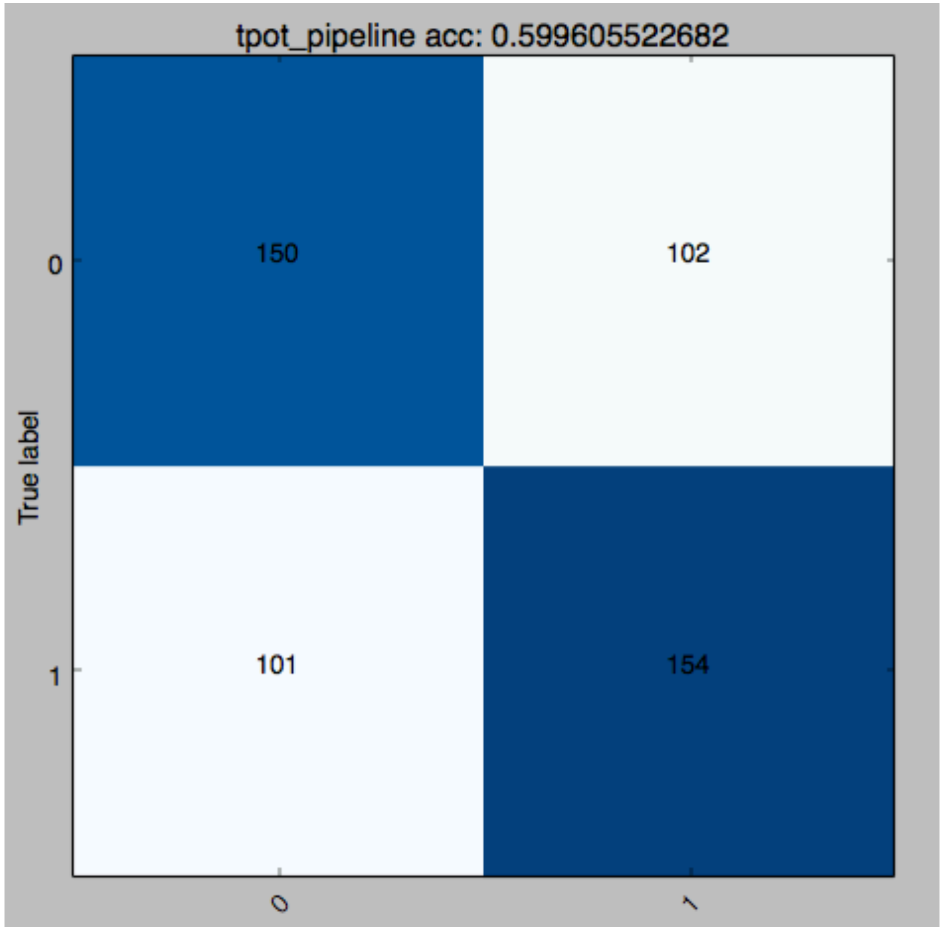
- I scaled each feature to zero mean and unit variance.
- Reduced dimensions to 3 with TSNE.
- I used both from scikit-learn library.
- I plotted the result with matplotlib
- There were visible some little clusters so I hoped the data wo be more separable in higher dimensions.
- visualize_features.py

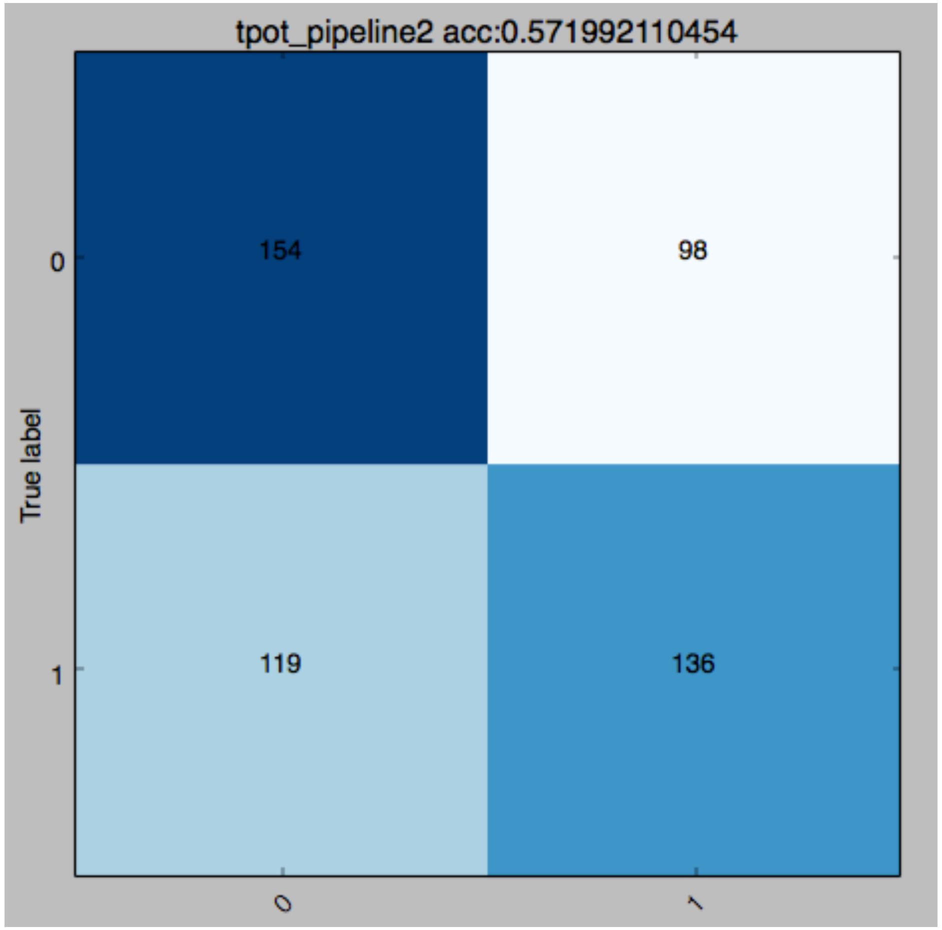


4. tpot

- Tpot is an open source library. It uses genetic algorithm to try different combinations of ML models.
- I have ran tpot few times and the result has always been some combination and variation of randomized forests.
- The percentage of correctly classified matches from the test set was around 58%-60%.
- tpot_train.py, tpot_pipeline.py, tpot_pipeline2.py

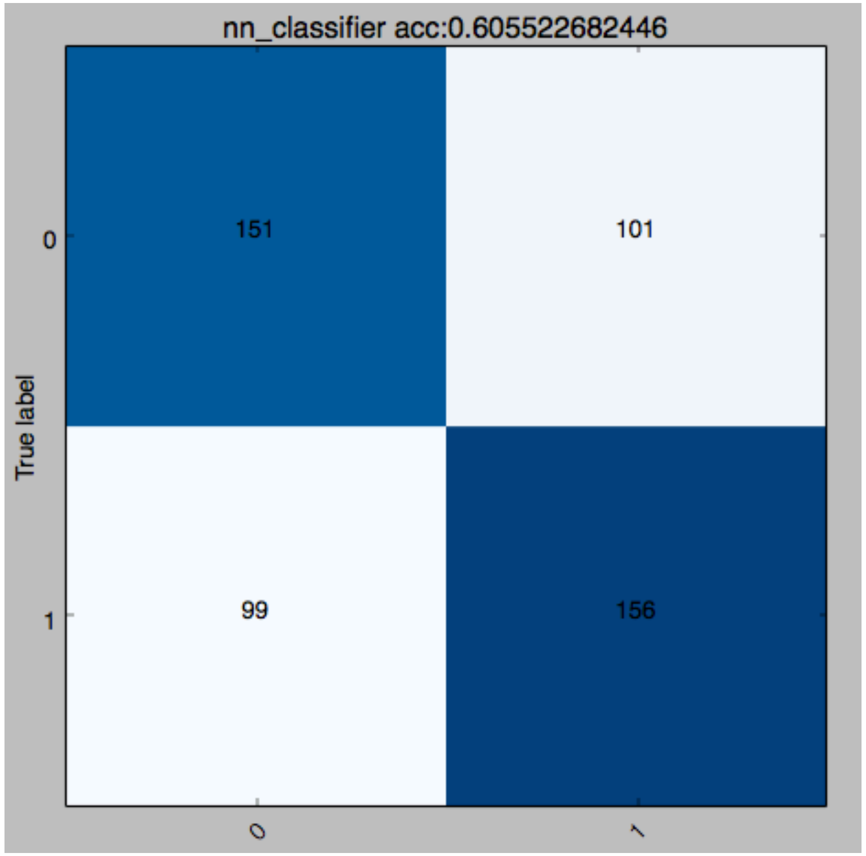
4. Results from tpot





5. Neural network classifier

- I iteratively tried different settings for number of neurons in hidden layers to see how it behaves.
- 60% correctly predicted from the test set.
- nn_classifier.py



6. Neural network regression

- So far I treated the problem as a classification, but someone might want to know the chances of each team winning.
- I transformed the labels into 1 hot labels (1 -> [1,0], 0 ->[0,1]) and fed them into a neural network.
- The output were 2 numbers which represented the chances of each team winning.
- I tried different kinds of hidden layer settings, loss functions and optimizers. I managed to get around 61% of correctly predicted matches.

7.Player oriented features

- So far, the features were team oriented. That means team stats were the computed from all the games that the team played.
- Player oriented features means that the stats for a team were computed from all the games that each of the 5 players played.
- I tried all of the models described earlier and the results were a bit better

7.Player oriented features results

- Tpot pipeline – 60.08%
- Tpot pipeline 2 – 61.46%
- Neural network classifier – 62.45%
- Neural network regression – 62.85%

Conclusion

- I managed to get ~63% in predicting matches from the test set.
- To make it better I would investigate all of the available features and try different feature selection techniques and missing value estimation.
- What might help the most would be adding the “first” statistics (first blood, first tower kill etc.).
- Fine tuning parameters might add few % as well.
- Arranging the data differently (e.g. separate features for players) and designing more sophisticated DNN model with many hidden layers (e.g. separate layer only for players’ features to let the neural network find the connections by itself) might also increase the accuracy of predictions.