Nicola Sebastianelli & Robin Bürgi

Project Report - Predicting Skill Level of Player based on Actions

Using Recorded Matches of Counter Strike: Global Offensive Games to Predict the Skill Level of New Players

- Business Understanding:

The goal of this project is to predict the skill level of a player in the online multiplayer game Counter-Strike: Global Offensive or CS:GO in short. This game is a round based first person shooter that is played on a 5 players against 5 players basis. When a player is looking for a new match, the game tries to find teammates that are of similar skill level to provide a fun and challenging experience.

Players that are new to the game don't have yet an assigned skill level. To provide the best gameplay for new players it is necessary to determine their rank quickly so that they are matched against opponents of equal skill. This is the problem that this analysis tries to tackle. A classification model should be created that allows to determine the rank of a player based only on a few actions.

The analysis is based on an open source dataset on Kaggle. The dataset contains over a million instances of gameplay actions from different players and their according rank. The dataset can be found here: https://www.kaggle.com/skihikingkevin/csgo-matchmaking-damage .

- Data Exploration:

The dataset is composed of about 1 million entries and 33 attributes, initially we are going to remove some of the attributes that are useless for our aim and then we are going to make some test to the remaining attributes to select the more significative ones.

```
import math
import pandas as pd
from sklearn import neighbors, datasets , preprocessing,svm
from sklearn.metrics import accuracy_score ,classification_report,
confusion_matrix
from sklearn.utils import shuffle
from sklearn.model_selection import StratifiedShuffleSplit,
cross_val_score, train_test_split
from sklearn.multiclass import OneVsRestClassifier
import matplotlib.pyplot as plt
import seaborn as sn
import numpy as np


df = pd.read_csv('mm_master_demos.csv') # Read the DataSet

# Removing not used attributes
del df['Unnamed: 0']
del df['file']
del df['date']
del df['seconds']
del df['att_team']
del df['att_side']
del df['vic_team']
del df['vic_side']
del df['winner_team']
del df['winner_side']
del df['att_id']
del df['vic_id']
del df['award']
del df['is_bomb_planted']
del df['bomb_site']


data=df.copy() # Creating a working copy of the original DataSet
data=shuffle(data) # Randomizing the DataSet
data # Printing the DataSet
```

	map	round	tick	hp_dmg	arm_dmg	hitb
618791	de_cache	17	107432	15	5	Chest

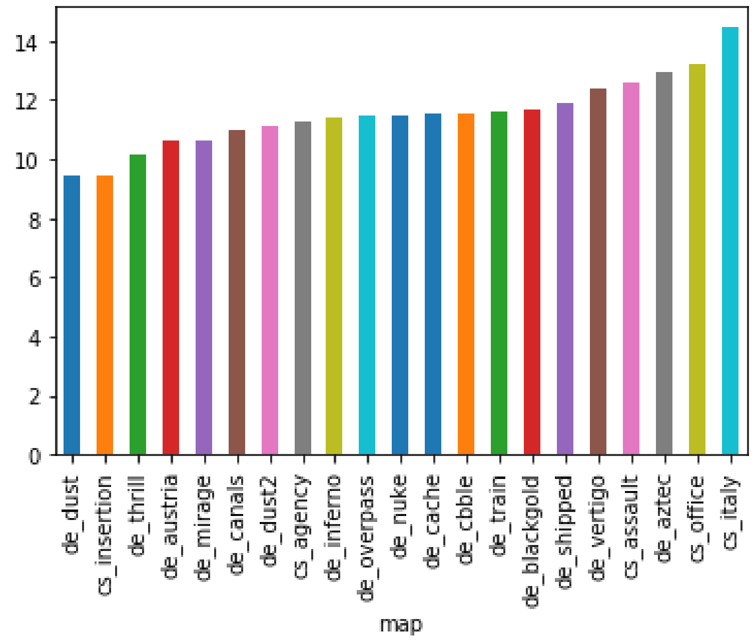
	map	round	tick	hp_dmg	arm_dmg	hitb
947890	de_cbble	15	92860	27	5	Stoma
160767	de_mirage	1	11603	13	7	Stoma
308813	de_dust2	13	99666	26	1	RightA
254760	de_mirage	5	30279	22	4	Chest
96578	de_cache	1	7555	100	0	Head
926397	de_dust2	20	113052	7	0	Generi
186785	de_cache	24	152884	9	0	LeftArı
412159	de_overpass	20	129543	16	8	Chest
93519	de_inferno	19	124392	21	4	Chest
597901	de_mirage	3	21498	24	5	Stoma
919729	de_inferno	21	137828	22	4	Chest
567154	de_mirage	1	10232	27	1	Chest
724635	de_cache	21	138582	100	15	Head
374476	de_inferno	11	78291	23	3	Chest
2769	de_cache	28	158554	22	4	RightA
58712	de_cache	25	152237	3	0	Generi
492441	de_dust2	16	102339	11	6	Chest
112031	de_mirage	3	17428	32	0	Stoma
146361	de_cache	18	110411	26	0	RightL
451382	de_inferno	3	21577	8	0	Generi
585141	de_mirage	19	140210	15	3	Chest
422676	cs_office	17	124168	25	0	Chest
558945	de_dust2	5	43554	22	4	Chest

	map	round	tick	hp_dmg	arm_dmg	hitb
345057	de_overpass	25	155607	14	5	Stoma
662354	de_mirage	22	144843	16	3	Chest
216958	de_mirage	5	42247	4	0	Generi
756218	de_inferno	21	144796	19	0	RightL
344564	de_overpass	9	51409	24	1	Chest
797119	de_inferno	4	27717	16	3	Chest
...	...	...	...	...	...	...
726125	de_dust2	19	117925	100	0	Head
354126	de_nuke	7	48376	20	0	Chest
362412	de_cache	17	86683	16	3	Chest

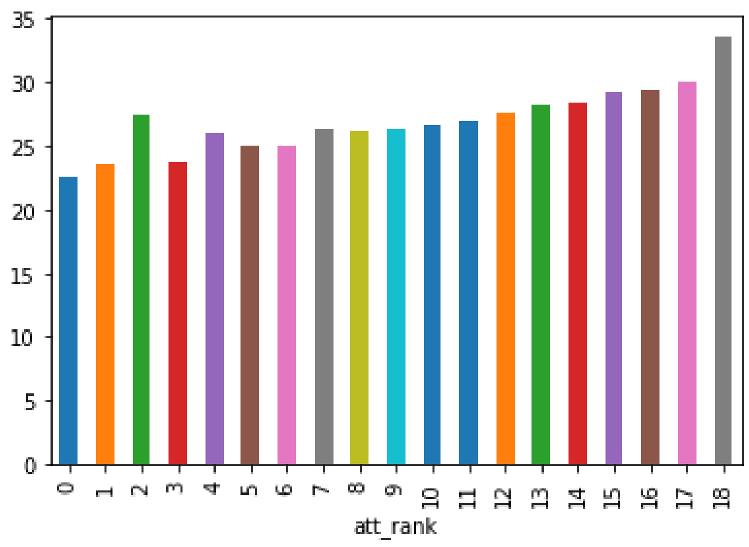
955466 rows × 18 columns

We are now going to plot some graphs to understand the nature and distribution of our attributes and data.

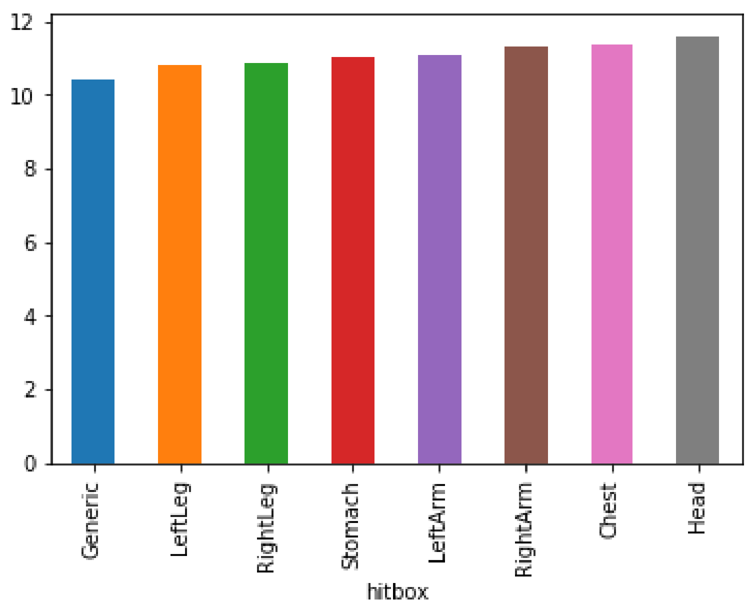
```
# Plotting sorted played map based on average rank of the players
data.groupby('map')['att_rank'].mean().sort_values().plot(kind='bar')
```



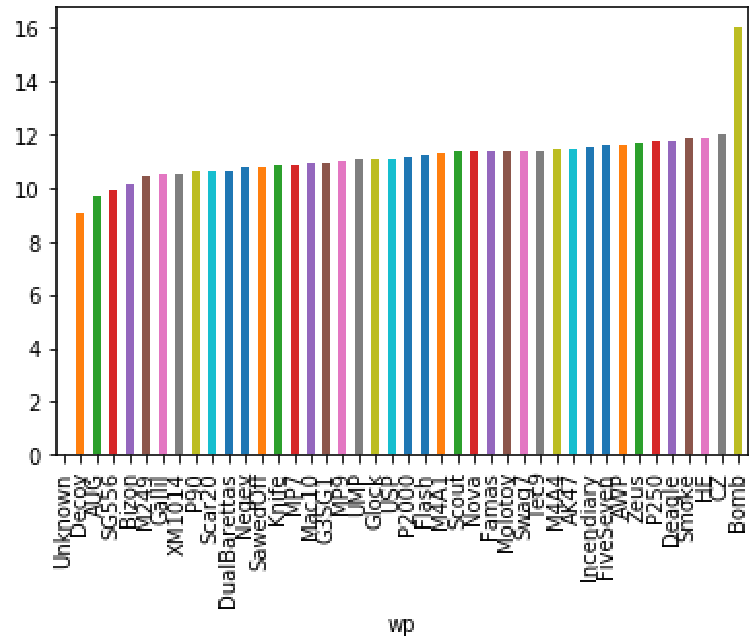
```
# Plotting distribution of rank based on average damages inflicted to othe
data.groupby('att_rank')['hp_dmg'].mean().plot(kind='bar')
```



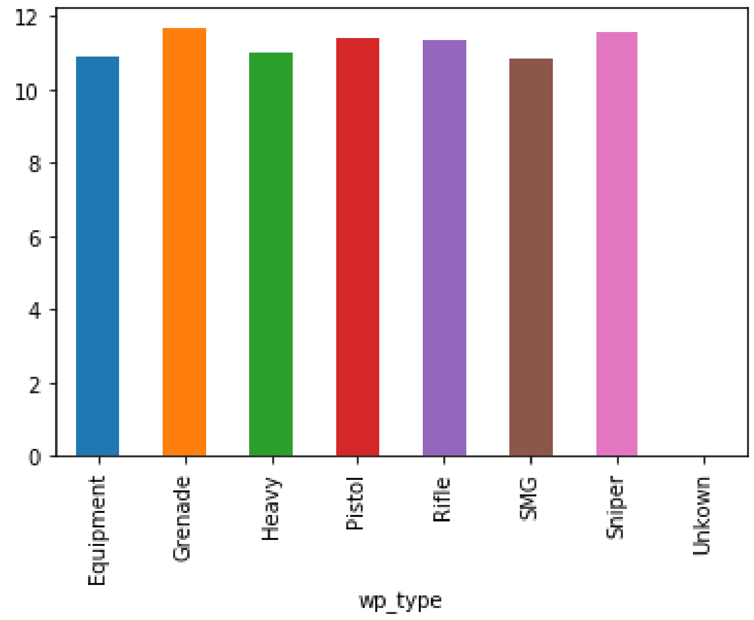
```
# Plotting sorted hitbox based on average rank of the players
data.groupby('hitbox')['att_rank'].mean().sort_values().plot(kind='bar')
```



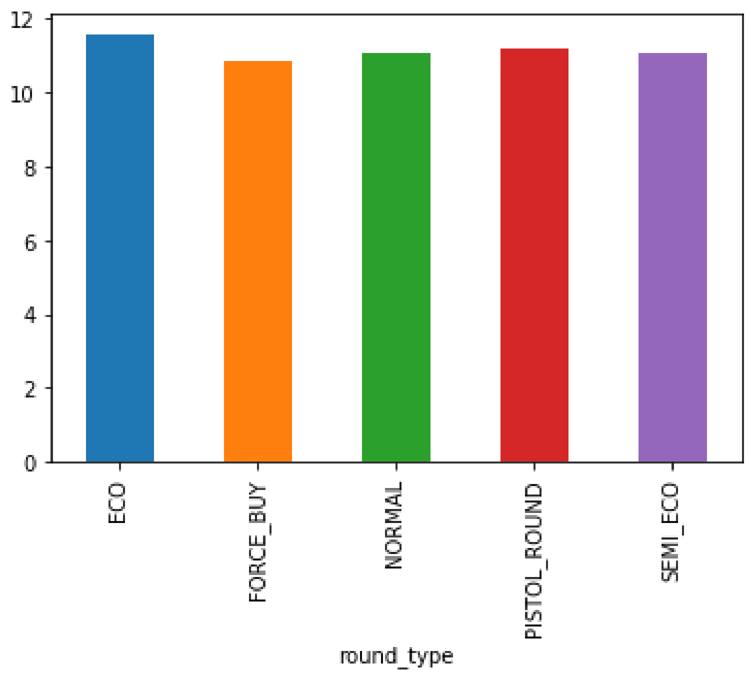
```
# Plotting sorted weapon based on average rank of the players
data.groupby('wp')['att_rank'].mean().sort_values().plot(kind='bar')
```



```
# Plotting weapon type based on average rank of the players
data.groupby('wp_type')['att_rank'].mean().plot(kind='bar')
```



```
# Plotting type of round based on average rank of the players
data.groupby('round_type')['att_rank'].mean().plot(kind='bar')
```



```
data.dtypes # Printing the type of the attributes of the DataSet
```

```
map                 object
round                int64
tick                 int64
hp_dmg               int64
arm_dmg              int64
hitbox              object
wp                  object
wp_type             object
att_rank             int64
vic_rank             int64
att_pos_x          float64
att_pos_y          float64
vic_pos_x          float64
vic_pos_y          float64
round_type          object
ct_eq_val            int64
t_eq_val             int64
avg_match_rank     float64
dtype: object
```

- Data Preparation

The data preparation process includes the conversion of categories that are text values to numerical values and the normalization of all values.

Convert Categories

- Hitbox

- Head and Chest --> UpperBody --> 0
- Stomach and Arms --> CenterBody --> 1
- Legs and Generic --> LowerBody --> 2

- Round Type

- ECO --> 0
- NORMAL, PISTOL, SEMI_ECO --> 1
- FORCE_BUY --> 2

```
# Converting hitbox and round_type text data to numerical data grouping the
numeric_cats = {
    "hitbox": {"Head": 0, "Chest": 0, "Stomach": 1, "RightArm": 1,
    "LeftArm": 1, "Generic": 2, "RightLeg": 2, "LeftLeg": 2},
    "round_type": {"ECO": 0, "NORMAL": 1, "PISTOL_ROUND": 1, "SEMI_ECO": 1,
    "FORCE_BUY": 2}
}
data.replace(numeric_cats, inplace=True)


# Applying one-hot-encoding to map, weapon and weapon type attributes
data = pd.get_dummies(data, columns=['map', 'wp_type', 'wp'])
data.dtypes
```

```
round                   int64
tick                    int64
hp_dmg                  int64
arm_dmg                 int64
hitbox                  int64
att_rank                int64
vic_rank                int64
att_pos_x             float64
att_pos_y             float64
vic_pos_x             float64
vic_pos_y             float64
round_type              int64
ct_eq_val               int64
t_eq_val                int64
avg_match_rank        float64
map_cs_agency           uint8
map_cs_assault          uint8
map_cs_insertion        uint8
                       ...

wp_Unknown              uint8
wp_XM1014               uint8
wp_Zeus                 uint8
```

```
Length: 86, dtype: object
```

```
# Data normalization for every attributes but the rank class
rankData = data['att_rank']
normalizedData = (data-data.min())/(data.max()-data.min())
normalizedData['att_rank']=rankData.values
normalizedData
```

	round	tick	hp_dmg	arm_dmg	hitbox	att_ranl
618791	0.551724	0.458995	0.15	0.05	0.0	11
947890	0.482759	0.392974	0.27	0.05	0.5	12
160767	0.000000	0.024824	0.13	0.07	0.5	9
308813	0.413793	0.423810	0.26	0.01	0.5	14
254760	0.137931	0.109439	0.22	0.04	0.0	10
96578	0.000000	0.006483	1.00	0.00	0.0	12
926397	0.655172	0.484457	0.07	0.00	1.0	11
186785	0.793103	0.664924	0.09	0.00	0.5	9
412159	0.655172	0.559173	0.16	0.08	0.0	16
93519	0.620690	0.535835	0.21	0.04	0.0	12
597901	0.068966	0.069655	0.24	0.05	0.5	12
919729	0.689655	0.596710	0.22	0.04	0.0	7
567154	0.000000	0.018612	0.27	0.01	0.0	9
724635	0.689655	0.600126	1.00	0.15	0.0	13
374476	0.344828	0.326966	0.23	0.03	0.0	11
2769	0.931034	0.690613	0.22	0.04	0.5	11
58712	0.827586	0.661993	0.03	0.00	1.0	14
492441	0.517241	0.435920	0.11	0.06	0.0	14
112031	0.068966	0.051215	0.32	0.00	0.5	12

	round	tick	hp_dmg	arm_dmg	hitbox	att_ranl
146361	0.586207	0.472492	0.26	0.00	1.0	9
451382	0.068966	0.070013	0.08	0.00	1.0	6
585141	0.620690	0.607502	0.15	0.03	0.0	9
422676	0.551724	0.534821	0.25	0.00	0.0	11
558945	0.137931	0.169584	0.22	0.04	0.0	13
345057	0.827586	0.677261	0.14	0.05	0.5	14
662354	0.724138	0.628493	0.16	0.03	0.0	8
216958	0.137931	0.163662	0.04	0.00	1.0	9
756218	0.689655	0.628280	0.19	0.00	1.0	10
344564	0.275862	0.205172	0.24	0.01	0.0	12
797119	0.103448	0.097831	0.16	0.03	0.0	10
...	...	...	...	...	...	...
726125	0.620690	0.506536	1.00	0.00	0.0	7
354126	0.206897	0.191431	0.20	0.00	0.0	9
362412	0.551724	0.364988	0.16	0.03	0.0	14

955466 rows × 86 columns

```
# Printing data correlation of attributes with the player rank
normalizedData.corr()['att_rank']
```

```
round                 0.012081
tick                  0.006935
hp_dmg                0.057941
arm_dmg              -0.035189
hitbox               -0.096196
att_rank              1.000000
vic_rank              0.645005
att_pos_x             0.027995
att_pos_y             0.043982
vic_pos_x             0.026662
vic_pos_y             0.044857
```

```
round_type          -0.072260
ct_eq_val            0.020356
t_eq_val             0.010572
avg_match_rank       0.786182
map_cs_agency        0.001691
map_cs_assault       0.017514
map_cs_insertion    -0.016428
                       ...
wp_Unknown          -0.359904
wp_XM1014           -0.016920
wp_Zeus              0.002330
Name: att_rank, Length: 86, dtype: float64
```

- Modeling:

Two approaches were taken to create a classifier for the player rank. The first approach uses a K-Nearest Neighbours algorithm to classify the instances and the second approach uses a neural network for the classification.

KNN:

We are going to try first to apply the KNN using all the 86 attributes but we also expect a low accuracy level cause of the large number of attributes and the low correlation between them.

```
y = normalizedData['att_rank'] # Creating class array
X = normalizedData.copy()
del X['att_rank'] # Creating attributes matrix using all the attributes

nEntries = 300000 # Number of training entries

knn=neighbors.KNeighborsClassifier(n_jobs=-1,
n_neighbors=int(math.sqrt(nEntries))) # Creating an instance
# of KNN Classifier using as K the square root of nEntries
knn.fit(X.head(nEntries), y.head(nEntries)) # Training the KNN with
#the firsts nEntries

prediction= knn.predict(X.iloc[nEntries:nEntries+10000])
# Predicting the classes of the next 10000 entries
print accuracy_score(y.iloc[nEntries:nEntries+10000],prediction)
# Printing the accuracy comparing the true classes with the predicted ones
```

```
0.282
```

As expected we have a really low accuracy level, now we are going to try to use just some significative attributes that are:

- hp_dmg
- hitbox
- att_rank
- vic_rank
- avg_match_rank

```
data = normalizedData[['hp_dmg', 'hitbox','att_rank','vic_rank',
'avg_match_rank']].copy()
# Creating new DataSet taking just the useful attributes

y = data['att_rank'] # Creating class array
X = data.copy()
del X['att_rank'] # Creating attributes matrix

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,random_state=0)
# Splitting data in 80% for training and 20% for testing

print X_train.shape, y_train.shape # Printing number of training entries
print X_test.shape, y_test.shape # Printing number of testing entries
```

```
(764372, 4) (764372,)
(191094, 4) (191094,)
```

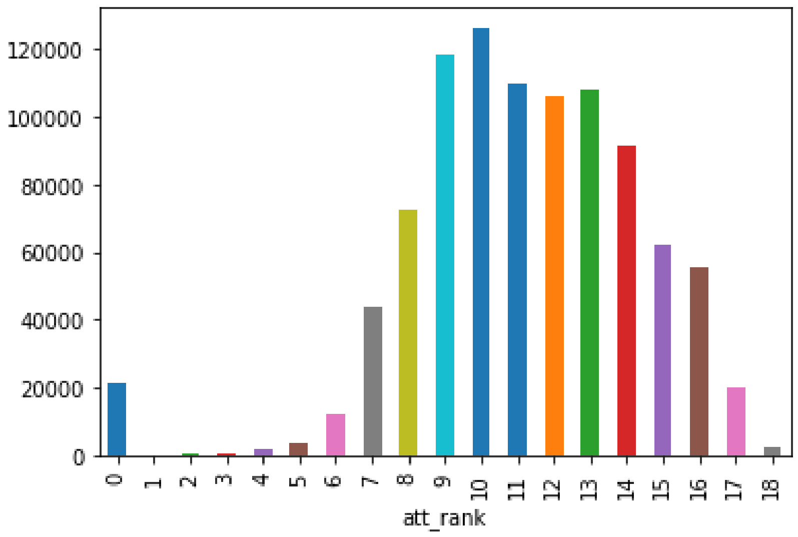
```
knn=neighbors.KNeighborsClassifier(n_jobs=-1,
n_neighbors=int(math.sqrt(len(X_train.index))))
# Creating an instance of KNN Classifier using as K the square root of the

knn.fit(X_train, y_train) # Training the KNN

y_pred= knn.predict(X_test)
# Predicting the classes of testing entries

print accuracy_score(y_test,y_pred)
# Printing the accuracy comparing the true classes with the predicted ones
```

```
0.42515201942499503
```

Option for improvement --> reduce output classes

The accuracy is improved but is still not enough for our aim. Is not necessarry that we get the exact rank level, the important thing is that players could play with other players that have similar skill level, so we decided to reduce the number of ranks level to three (high, mid,low level) according to the following plotted graph. This decision drastically impove our model accuracy level and it will still being valid for the business case.

```
df.groupby('att_rank')['hp_dmg'].count().plot(kind='bar')
# Plotting the number of players for each rank
```



```
n_classes = 3 # Number of new total classes

# Grouping the classes
numeric_cats = {
    "att_rank": {0:"0", 1:"0", 2:"0", 3:"0", 4:"0", 5:"0", 6:"0", 7:"0",
    8:"0",
                9:"1", 10:"1", 11:"1", 12:"1",13:"1", 14:"1",
                15:"2", 16:"2", 17:"2", 18:"2"}
}
data.replace(numeric_cats, inplace=True)
data.head()
```

	hp_dmg	hitbox	att_rank	vic_rank	avg_match_rank
175379	0.28	0.0	1	0.555556	0.222222
903816	0.44	0.5	1	0.611111	0.444444
104196	0.13	0.0	2	0.833333	0.888889
33692	0.10	1.0	1	0.722222	0.666667

	hp_dmg	hitbox	att_rank	vic_rank	avg_match_rank
247244	0.15	0.0	2	0.888889	1.000000

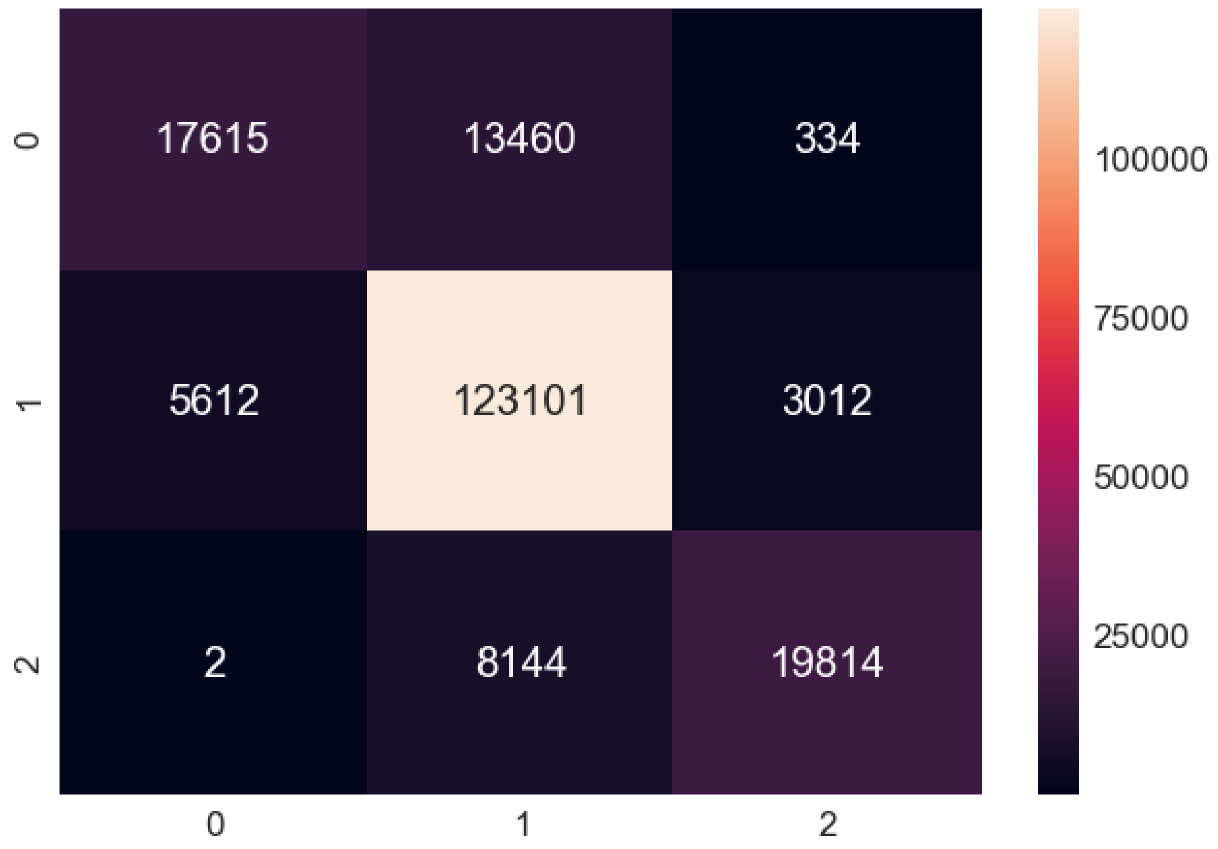
```
y = data['att_rank'] # Creating class array
X = data.copy()
del X['att_rank'] # Creating attributes matrix using all the attributes

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,rar
knn=neighbors.KNeighborsClassifier(n_jobs=-1,
n_neighbors=int(math.sqrt(len(X_train.index))))
 # Creating an instance of KNN Classifier using as K the square root of the

knn.fit(X_train, y_train) # Training the KNN

y_pred= knn.predict(X_test) # Predicting the classes of testing entries
print accuracy_score(y_test,y_pred)
# Printing the accuracy comparing the true classes with the predicted ones
```

```
0.8400577726145247
```

- Confusion Matrix:

```
print classification_report(y_test, y_pred)
# Printing the accuracy comparing the true classes with the predicted ones

# Creating the confusion matrix and visualize it
cm=confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, range(n_classes),range(n_classes))
plt.figure(figsize = (10,7))
sn.set(font_scale=1.7)
sn.heatmap(df_cm, annot=True, fmt='g')
```

```
             precision    recall  f1-score   support

          0       0.76      0.56      0.64     31409
          1       0.85      0.93      0.89    131725
          2       0.86      0.71      0.78     27960

avg / total       0.84      0.84      0.83    191094
```



- Cross Validation:

```
clf = neighbors.KNeighborsClassifier(n_jobs=-1,
n_neighbors=int(math.sqrt(len(X_train.index))))
# Creating an instance of KNN Classifier using as K the square root of the

sss = StratifiedShuffleSplit(n_splits=10)
# Provides train/test indices to split data in 10 train/test sets

scoresSSS = cross_val_score(clf, X, y, cv=sss, scoring='accuracy')
# Execute the cross validation

print(scoresSSS) # Print cross validation accuracy array
```

```
[0.84226611 0.84211959 0.84191026 0.84008917 0.84228704 0.84124044
 0.839524   0.84009964 0.84067527 0.83962866]
```

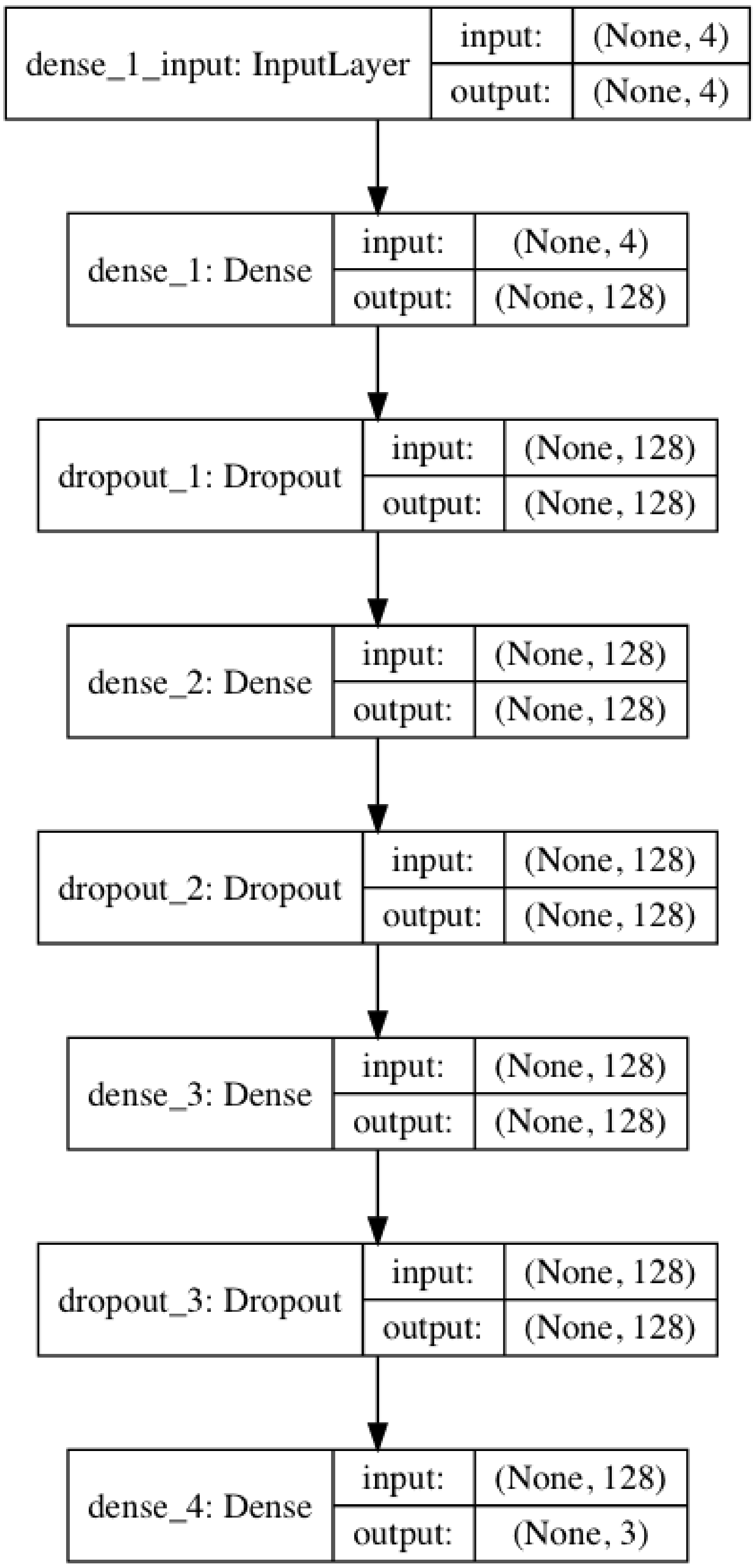
Neural Network:

The Keras framework in combination with a TensorFlow backend was used to create the neural network. The architecture of the neural network is a feed forward network with 4 layers, one input and one output layer as well as 2 fully connected hidden layers. The network is trained using ~700’000 instances and validated using a 10-fold cross-validation.

```
import keras
import graphviz
import pydot_ng as pydot
from keras.models import Sequential
from keras.utils import to_categorical, plot_model
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
```

```
Using TensorFlow backend.
```

```
model = Sequential() # Initialazing new Keras sequential model
model.add(Dense(128, activation='relu', input_dim=4)) # Adding first layer
model.add(Dropout(0.5))
# Adding Dropout layer by randomly selecting nodes to be dropped-out with a

for i in range(1,3):
    model.add(Dense(128, activation='relu')) # Adding two mid layer to the
    model.add(Dropout(0.5))
    # Adding Dropout layer by randomly selecting nodes to be dropped-out wi
model.add(Dense(3, activation='softmax'))
# Adding output layer to the Neural Network with 3 neurons

sgd = SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
# Initializing the optimizer for the model compiling

model.compile(loss='categorical_crossentropy', optimizer=sgd,
metrics=['accuracy']) # Compiling the model for a multi-class classificatio

pydot.find_graphviz()
plot_model(model, to_file='model.png',
show_shapes=True, show_layer_names=True) # Plotting neural network layers
```



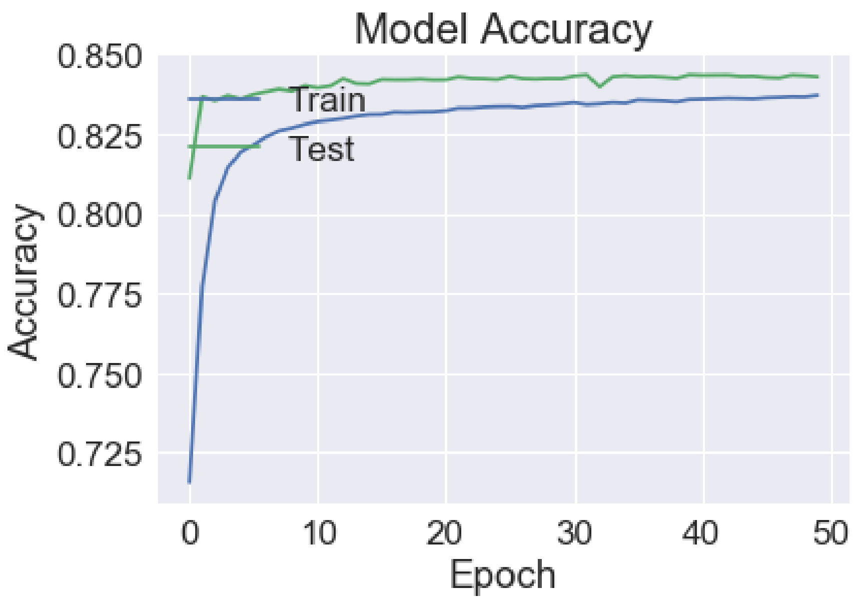
```
y_trainBinary = to_categorical(y_train)
# Converting a class vector to binary class matrix.
hist = model.fit(X_train, y_trainBinary,validation_split=0.33,epochs=50,bat
# Training the Neural Network using 33% of training entries for validation
```

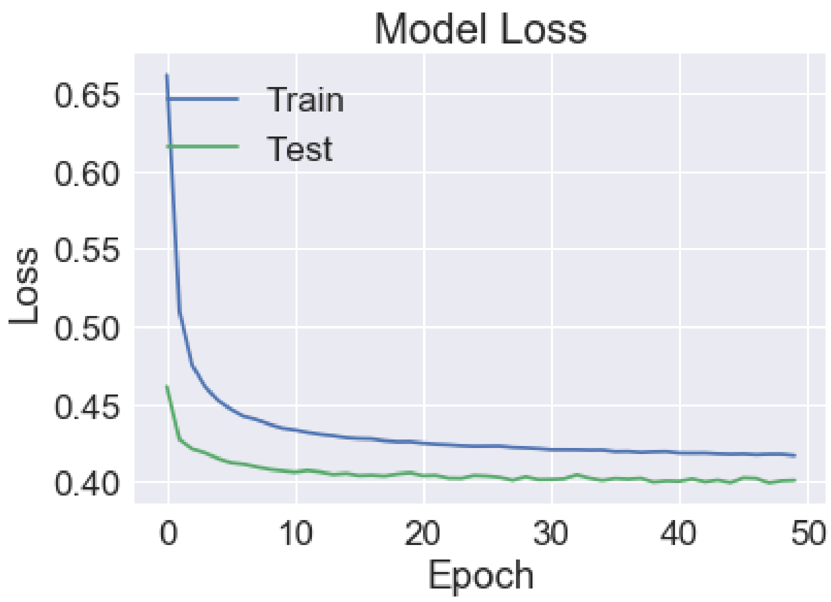
```
WARNING:tensorflow:Variable *= will be deprecated. Use variable.assign_mul
Train on 512129 samples, validate on 252243 samples
```

```
Epoch 1/50
512129/512129 [===] - 15s 30us/step - loss: 0.6617 - acc: 0.7156 - val_loss
Epoch 2/50
512129/512129 [===] - 14s 28us/step - loss: 0.5094 - acc: 0.7771 - val_loss
Epoch 3/50
512129/512129 [===] - 14s 28us/step - loss: 0.4748 - acc: 0.8040 - val_loss
Epoch 4/50
512129/512129 [===] - 14s 28us/step - loss: 0.4611 - acc: 0.8144 - val_loss
Epoch 5/50
512129/512129 [===] - 14s 28us/step - loss: 0.4524 - acc: 0.8193 - val_loss
Epoch 6/50
512129/512129 [===] - 14s 28us/step - loss: 0.4467 - acc: 0.8215 - val_loss
Epoch 7/50
512129/512129 [===] - 14s 28us/step - loss: 0.4422 - acc: 0.8241 - val_loss
Epoch 8/50
512129/512129 [===] - 14s 28us/step - loss: 0.4400 - acc: 0.8260 - val_loss
Epoch 9/50
512129/512129 [===] - 14s 28us/step - loss: 0.4370 - acc: 0.8269 - val_loss
Epoch 10/50
512129/512129 [===] - 14s 28us/step - loss: 0.4345 - acc: 0.8280 - val_loss
Epoch 11/50
512129/512129 [===] - 14s 28us/step - loss: 0.4333 - acc: 0.8289 - val_loss
Epoch 12/50
512129/512129 [===] - 14s 28us/step - loss: 0.4317 - acc: 0.8294 - val_loss
Epoch 13/50
512129/512129 [===] - 14s 28us/step - loss: 0.4304 - acc: 0.8299 - val_loss
Epoch 14/50
512129/512129 [===] - 14s 28us/step - loss: 0.4295 - acc: 0.8306 - val_loss
Epoch 15/50
512129/512129 [===] - 14s 28us/step - loss: 0.4283 - acc: 0.8311 - val_loss
Epoch 16/50
512129/512129 [===] - 14s 28us/step - loss: 0.4278 - acc: 0.8312 - val_loss
Epoch 17/50
512129/512129 [===] - 14s 28us/step - loss: 0.4276 - acc: 0.8319 - val_loss
Epoch 18/50
512129/512129 [===] - 14s 28us/step - loss: 0.4263 - acc: 0.8318 - val_loss
Epoch 19/50
512129/512129 [===] - 14s 28us/step - loss: 0.4257 - acc: 0.8319 - val_loss
Epoch 20/50
512129/512129 [===] - 14s 28us/step - loss: 0.4257 - acc: 0.8320 - val_loss
Epoch 21/50
512129/512129 [===] - 14s 28us/step - loss: 0.4248 - acc: 0.8323 - val_loss
Epoch 22/50
512129/512129 [===] - 14s 28us/step - loss: 0.4242 - acc: 0.8331 - val_loss
Epoch 23/50
512129/512129 [===] - 14s 28us/step - loss: 0.4238 - acc: 0.8331 - val_loss
Epoch 24/50
512129/512129 [===] - 14s 28us/step - loss: 0.4232 - acc: 0.8334 - val_loss
Epoch 25/50
512129/512129 [===] - 14s 28us/step - loss: 0.4229 - acc: 0.8336 - val_loss
Epoch 26/50
512129/512129 [===] - 14s 28us/step - loss: 0.4229 - acc: 0.8336 - val_loss
Epoch 27/50
```

```
512129/512129 [===] - 14s 27us/step - loss: 0.4230 - acc: 0.8333 - val_loss
Epoch 28/50
512129/512129 [===] - 14s 28us/step - loss: 0.4221 - acc: 0.8339 - val_loss
Epoch 29/50
512129/512129 [===] - 14s 28us/step - loss: 0.4218 - acc: 0.8341 - val_loss
Epoch 30/50
512129/512129 [===] - 14s 28us/step - loss: 0.4213 - acc: 0.8344 - val_loss
Epoch 31/50
512129/512129 [===] - 14s 28us/step - loss: 0.4206 - acc: 0.8349 - val_loss
Epoch 32/50
512129/512129 [===] - 14s 28us/step - loss: 0.4205 - acc: 0.8342 - val_loss
Epoch 33/50
512129/512129 [===] - 14s 28us/step - loss: 0.4205 - acc: 0.8345 - val_loss
Epoch 34/50
512129/512129 [===] - 14s 28us/step - loss: 0.4203 - acc: 0.8349 - val_loss
Epoch 35/50
512129/512129 [===] - 14s 28us/step - loss: 0.4204 - acc: 0.8347 - val_loss
Epoch 36/50
512129/512129 [===] - 14s 28us/step - loss: 0.4193 - acc: 0.8357 - val_loss
Epoch 37/50
512129/512129 [===] - 14s 28us/step - loss: 0.4194 - acc: 0.8355 - val_loss
Epoch 38/50
512129/512129 [===] - 14s 28us/step - loss: 0.4188 - acc: 0.8354 - val_loss
Epoch 39/50
512129/512129 [===] - 14s 28us/step - loss: 0.4192 - acc: 0.8351 - val_loss
Epoch 40/50
512129/512129 [===] - 14s 27us/step - loss: 0.4194 - acc: 0.8358 - val_loss
Epoch 41/50
512129/512129 [===] - 14s 28us/step - loss: 0.4184 - acc: 0.8359 - val_loss
Epoch 42/50
512129/512129 [===] - 14s 28us/step - loss: 0.4183 - acc: 0.8361 - val_loss
Epoch 43/50
512129/512129 [===] - 15s 29us/step - loss: 0.4184 - acc: 0.8362 - val_loss
Epoch 44/50
512129/512129 [===] - 14s 28us/step - loss: 0.4179 - acc: 0.8361 - val_loss
Epoch 45/50
512129/512129 [===] - 14s 28us/step - loss: 0.4176 - acc: 0.8360 - val_loss
Epoch 46/50
512129/512129 [===] - 14s 28us/step - loss: 0.4178 - acc: 0.8364 - val_loss
Epoch 47/50
512129/512129 [===] - 14s 28us/step - loss: 0.4174 - acc: 0.8365 - val_loss
Epoch 48/50
512129/512129 [===] - 14s 28us/step - loss: 0.4177 - acc: 0.8367 - val_loss
Epoch 49/50
512129/512129 [===] - 14s 28us/step - loss: 0.4176 - acc: 0.8366 - val_loss
Epoch 50/50
512129/512129 [===] - 14s 28us/step - loss: 0.4168 - acc: 0.8372 - val_loss
```

```python
# Plotting history of training for accuracy
plt.plot(hist.history['acc'])
plt.plot(hist.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plotting history of training for loss
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```





```
evals = model.evaluate(X_train, y_trainBinary,
batch_size=len(X_train.index)/10)
# Evaluating the training model

print evals # Printing the evaluation result
```

```
764372/764372 [==============================] - 2s 2us/step
[0.40210509932774696, 0.842413116560774]
```

- Conclusion

In the beginning of the project it was found out that most of the attributes have no real correlation to the player rank. We decided to focus on those attributes that had a high correlation. Reducing the number of output classes from nineteen to three improved the accuracy of our model drastically while still being valid for the business case. The accuracy of the KNN and the neural network was comparable at around 84%. While it takes more time to train the neural network, the prediction is much faster than the KNN wich makes the neural network the better solution for this business case.

The model is now mostly based on how much damage a player does and what the rank of the victim player was. This could result in new players hunting the best player in a match to achieve a better skill rank and would interfere with the objective of this team based game. However as the rank of all opponents is hidden until the match is finished, this is only a hypothetical problem.