Deep Learning Group Project: PUBG Dataset Exploration, Visualization and Prediction

*** **
School of Data and Computer Science, Sun Yat-sen University
Guangzhou, China
*****@mail2.sysu.edu.cn

****** ***
School of Data and Computer Science, Sun Yat-sen University
Guangzhou, China
*******@mail2.sysu.edu.cn

***** ***
School of Data and Computer Science, Sun Yat-sen University
Guangzhou, China
*****@mail2.sysu.edu.cn

Abstract

PLAYERUNKNOWN'S BATTLEGROUNDS is drawing more and more attention in recent years, whose matching dataset is available in Kaggle for contestants worldwide to explore. However, only a few neural network model is applied to the prediction of the player's win percentage, and no convolution neural network (CNN) is witnessed in top-ranked submissions. In this paper, we implement a series of networks, including basic MLP and various CNN with fine tuning. Then, we analyze their performances and conclude their drawbacks in PUBG dataset. After walking through other high-ranked submissions, we then visualize the dataset and figure out the inner relation between them by feature engineering. Finally, we implement GDBT and random forest (RF) regression with extra features we built in feature engineering. Our results show not only the shortcomings of CNN in handling dataset with strong and sparse relation but also the overwhelming advantages of feature engineering. The performances of classic machine learning models then prove our hypothesis.

1. Introduction

We choose PUBG Finish Placement Prediction in Kaggle as the group project for the Deep Learning course in SYSU. Motivated by the course objective, we do a huge amount of experiments and investigations towards this topic and our main contributions are as followed:

1. We implement a series of (six) neural network models, including basic MLP, ResNet, SE-ResNet, Vgg, DenseNet, ResNeXt, in PyTorch. The data loader of PUBG dataset is also implemented.

2. We fine tune all the neural network models mentioned above in a small train set, testing them with different optimizers, different model structures and different learning rate (enabled by learning rate scheduler).

3. We do feature engineering to the whole dataset, constructing several new features for the future appliance. Also, visualization to the feature engineering is added.

4. We implement random forest (RF) regression and logistic regression in sklearn, with the constructed feature to learn from the whole PUBG dataset. And, we visualize the importance of the top features.

5. We try to draw a Conclusion of the failure of the network in PUBG dataset, and the reason why classic machine learning methods work perfectly in such circumstance. An overall performances comparison is given.

The rest of paper is organized as follows. In Section 2, a brief description of the PUBG dataset is given. In Section 3, we introduce previous works that we apply in this paper. Then, the structures of the network models are listed in Section 3. In Section 4, we do a lot of experiments using

various model mentioned above. A comparison to them is also pointed out in this section. In Section 5, our feature engineering and result visualization are performed. Using the new features we get from the previous engineering, we conduct Random Forest (RF) regressions and Gradient Boosting Decision Tree (GDBT) to the whole dataset in Section 6. The whole paper is concluded in Section 7, and a detailed contribution of each team member is listed in Section 8. References and source code will be attached after the end of this paper.

2. Dataset Description

In a PUBG game, up to 100 players start in each match (matchId). Players can be on teams (groupId) which get ranked at the end of the game (winPlacePerc) based on how many other teams are still alive when they are eliminated. In the 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.

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 players per group.

We are supposed to create a model which predicts players' finishing placement based on their final stats, on a scale from 1 (first place) to 0 (last place).

Three files are given, whose name, description and size are available in Table 1. The data field of the train set and test set is available in Table 2.

3. Related Works

The history of the neural network must date back to 1961 when Rosenblatt *et al*. [11] first introduced the definition of perceptrons. In the past half century, due to the limit of computation resource, a neural network is left unknown to the public. In 2012, Alex *et al*. [7] used full-connected layers and convolution layer to build a convolution neural network (CNN), which leads to the development of the neural network. The effect of the convolutional network depth is first investigated in the work of Simonyan *et al*. [13]. In their implementation, the Very deep convolutional network (Vgg) does great in large-scale image recognition. The density of connected layers is also studied. Huang *et al*. [5] introduced DenseNet, which use the output of all the previous layers as its input.

He *et al*. [3] solve the higher error with deeper network problem by residual blocks, and present a residual learning framework (ResNet) to ease the training of networks that are substantially deeper than those used previously. Its depth can exceed Vgg without adding its complexity easily. Hu *et al*. [4] add a Squeeze-and-Excitation (SE) block to ResNet, which adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels. Xie *et al*. [14] divided ResNet into several paths, called increasing cardinality. Such a method do not increase the complexity of the model but still have a better result in image recognition.

Although all the CNN is applied to image recognition, we are curious whether it works in the PUBG dataset. As a control group, two tree theory models is used, including a classic random forest (RF) regression introduced by Breiman [1] and Gradient Boosting Decision Tree (GDBT)-based algorithm by Ke *et al*. [6].

4. Network Model

4.1. Multilayer Perceptron

Two multilayer perceptron (MLP) is implemented in our mid-term report. To validate the feasibility for MLP to address the win place prediction problem, the first simple MLP is defined. In this model, there is only 1 input layer and 1 output layer, all of them are linear layers. The input layer accepts a 23×1 array as input and giving a 10×1 array as output. The output layer accept a 10×1 array as input and giving a scaler as output.

In the following implementation, a second MLP is applied. In this MLP, ReLU is used as an activation function after each full connect layer, while BatchNorm1d is also introduced. This MLP has 2 hidden layers with the same setting – taking 28×1 array as input, giving 28×1 array as output. And, the input and output layers are exactly the same as the previous MLP.

4.2. ResNet

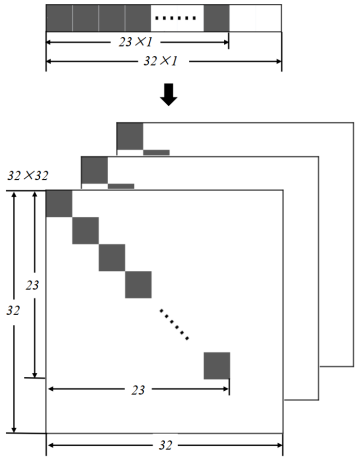

Figure 1. 23×1 array to $1 \times 3 \times 32 \times 32$ matrix transform

File Name	description	size
train_V2.csv	the training set	660MB
test_V2.csv	the test set	273MB
sample_submission_V2.csv	a sample submission file in the correct format	32.9MB

Table 1. Table of input files

Data Name	description
DBNOs	Number of enemy players knocked.
assists	Number of enemy players this player damaged that were killed by teammates.
boosts	Number of boost items used.
damageDealt	Total damage dealt. Note: Self inflicted damage is subtracted.
headshotKills	Number of enemy players killed with headshots.
heals	Number of healing items used.
Id	Player's Id
killPlace	Ranking in match of number of enemy players killed.
killPoints	Kills-based external ranking of player.
killStreaks	Max number of enemy players killed in a short amount of time.
kills	Number of enemy players killed.
longestKill	Longest distance between player and player killed at time of death.
matchDuration	Duration of match in seconds.
matchId	ID to identify match. There are no matches that are in both the training and testing set.
matchType	String identifying the game mode that the data comes from.
rankPoints	Elo-like ranking of player.
revives	Number of times this player revived teammates.
rideDistance	Total distance traveled in vehicles measured in meters.
roadKills	Number of kills while in a vehicle.
swimDistance	Total distance traveled by swimming measured in meters.
teamKills	Number of times this player killed a teammate.
vehicleDestroys	Number of vehicles destroyed.
walkDistance	Total distance traveled on foot measured in meters.
weaponsAcquired	Number of weapons picked up.
winPoints	Win-based external ranking of player.
groupId	ID to identify a group within a match.
numGroups	Number of groups we have data for in the match.
maxPlace	Worst placement we have data for in the match.
winPlacePerc	The target of prediction.

Table 2. Table of data description

Before we can apply the ResNet into PUBG dataset, some transformation has been made to suit the norm of ResNet and all the following CNN. We apply an 23×1 array to $1 \times 3 \times 32 \times 32$ matrix transform, which is illustrate in Fig. 1. In such a transform, only the black blocks are non-zero, and the three channel of the transformed matrix is exactly the same.

We implement a ResNet18 with three layers, and each layer contains two residual blocks illustrated in Fig. 2. Each residual block contains two convolution layer, followed by BatchNorm2d layer and ReLU as the activation function. Beware that the residual is added before the second ReLU.

In the end of ResNet, an average pooling is appended,

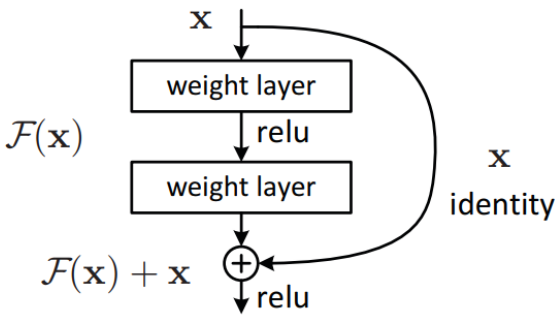

Figure 2. residual block

followed by a n to 1 full connect layer as output prediction.

4.3. SEResNet

The SEResNet has almost the same structure as ResNet does. The only difference is that it has a Squeeze-and-Excitation (SE) block appended to the residual block. The SE block contains two full connect layers, and a ReLU and a sigmoid as the activation function. Before the first full connect layer, a pooling squeezes the global information embedding of each channel. The embedding is then become the "weight" of each channel in the excitation step.

Since the principle of SE block is rather easy and the other details are the same as ResNet, any over-detailed investigation is omitted here.

4.4. VGG

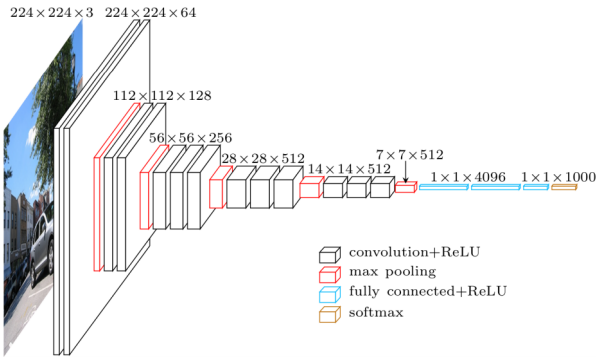

Figure 3. Vgg structure

Vgg is a CNN with many convolution layers. With deepening into a smaller matrix using convolution layers, the higher features of images are captured. After a BatchNorm2d and a ReLU, a MaxPooling with stride larger than 1 reduces the size of the input matrix. The structure is illustrated in Fig. 3. We implement Vgg11, Vgg13, Vgg16 and Vgg19. The main difference between them is the number of convolution layers.

4.5. DenseNet

In the architecture of a DenseNet shown in Fig. 4, for each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. That is to say, even though with swallower layers in DenseNet, it can still learn feature from each preceding layer. Different from Vgg, the size of the feature map does not change during the dense block, until the output layer.

For a typical densenet52 structure, it has four dense blocks and four transition layers. Each dense block contains six bottlenecks, with two convolution layers and two BatchNorm2d layers. Each transition layer has one convolution layer and one BatchNorm2d layer. In different DenseNet instances, the main difference is the number of bottlenecks a dense block holds.

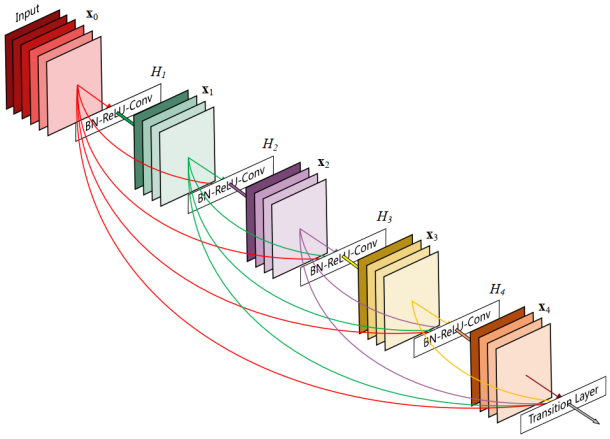

Figure 4. DenseNet structure

4.6. ResNeXt

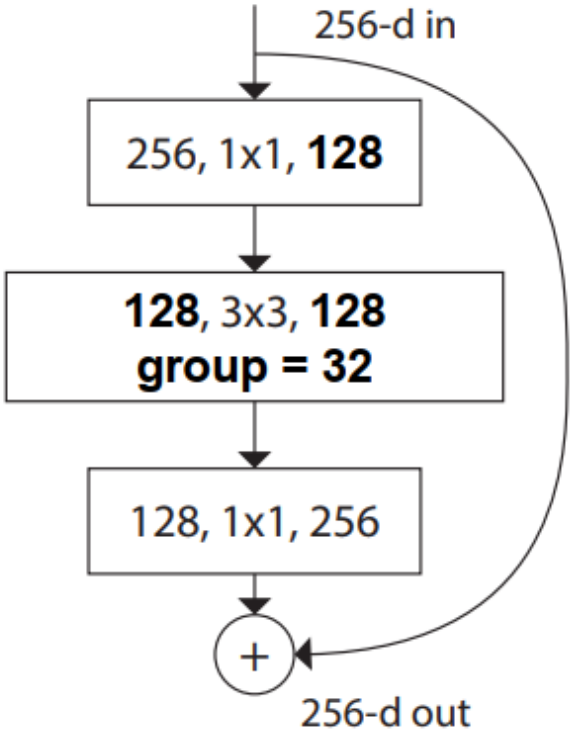

Figure 5. ResNeXt structure

The ResNeXt showed in Fig. 5 is constructed by repeating a building block that aggregates a set of transformations with the same topology. The simple design results in a homogeneous, multi-branch architecture that has only a few

hyper-parameters to set. This strategy exposes a new dimension, which we call "cardinality" (the size of the set of transformations), as an essential factor in addition to the dimensions of depth and width.

In our implementation, only resnext32_16x8d is given due to the memory limit of the devices (GPUs). The resnext32_16x8d includes three layers, while each layer contains three blocks. In each block, three convolution layers and three BatchNorm2d layers repeat in turns.

5. Network Experiments

In this section, we conduct a series of experiments to evaluate the performances of our MLP and CNN models. Each model is fine tuned to show their best result. Readers can go through the source code in the appendix. Notice that, due to the huge size of the original dataset (it takes 30 minutes to train ONE epoch using GTX1080Ti in the first MLP), we only evaluate them using a smaller dataset in this section.

5.1. Multilayer Perceptron

Two MLPs are both evaluated. In the first simpler MLP, we set the learning rate to $1e-5$, using SDG with momentum 0.9 as an optimizer, to train the model. The result is shown in Fig. 7. After 100 epochs of training, the training loss si 0.1411, while the test loss is 0.3444.

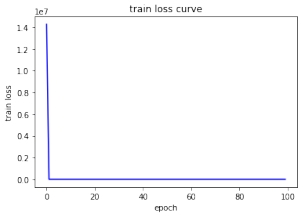

Figure 6. first MLP training loss

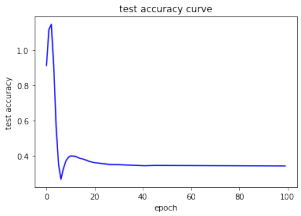

Figure 7. first MLP test loss

Things are different when the MLP goes deeper. In the second MLP with 3 hidden full connect layers, the results are shown in Fig. 9. We set the learning rate to $1e-6$, using SDG with momentum 0.9 as an optimizer, to train the model. After 100 epochs of training, the training loss is 164.3882, while the test loss is 10.0996. Even though we try different hyperparameters, it just does not work.

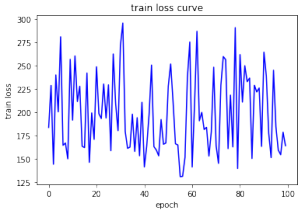

Figure 8. second MLP training loss

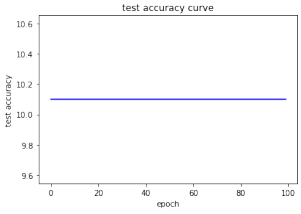

Figure 9. second MLP test loss

All the code and results of MLPs are available in appended file **group-project-MLP-1**.

5.2. ResNet

Only resnet18 is evaluated in this part. We set the learning rate to 0.01, using SDG without momentum as an optimizer, to train the model. After 100 epochs of training, the training loss is 0.0944, while the test loss is 0.2755. The result is shown in Fig. 11.

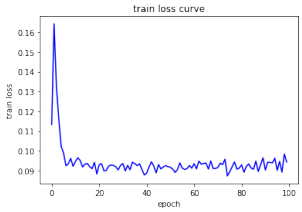

Figure 10. resnet18 training loss

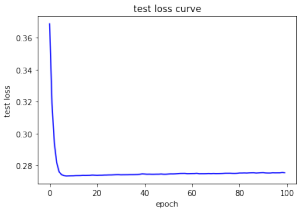

Figure 11. resnet18 test loss

We try different optimizers and learning rate scheduler in resnet18, and only the optimal result is shown. To figure out all the results of resnet18, please refer to appended file **group-project-ResNet-1**.

5.3. SEResNet

Only seresnet18 is evaluated in this part. We set the learning rate to 0.01, using SDG without momentum as an optimizer, to train the model. Every 100 epochs, the learning rate decrease to its one-tenth because of the preseted scheduler. After 400 epochs of training, the training loss is 0.0907, while the test loss is 0.2788. The result is shown in Fig. 13.

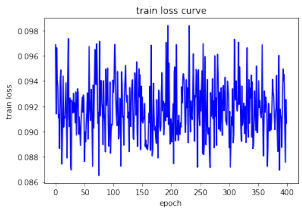

Figure 12. seresnet18 training loss

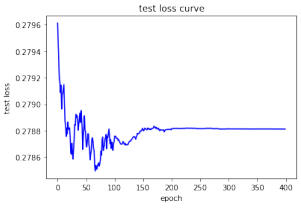

Figure 13. seresnet18 test loss

We try different optimizers and learning rate scheduler in resnet18, and only the optimal result is shown. To figure out all the results of resnet18, please refer to appended file **group-project-SEResNet-1**.

5.4. VGG

In this part, the performances of vgg11, vgg13, vgg16 and vgg19 are all evaluated. Readers can walk through **group-project-VGG-1** to have a close look. We also test them with different optimizers. The best combination is proved to be RMSprop and vggnet11 in learning rate of

0.01. Every 100 epochs, the learning rate decrease to its one-tenth because of the preset scheduler. After 400 epochs of training, the training loss is 0.0919, while the test loss is 0.2773. The result is shown in Fig. 15.

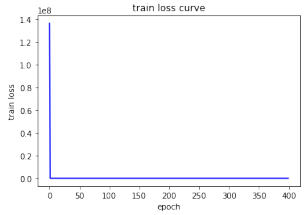

Figure 14. vgg11 training loss

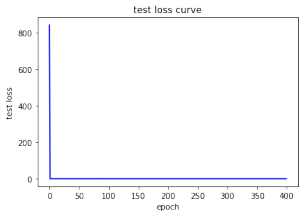

Figure 15. vgg11 test loss

5.5. DenseNet

In this part, the performances of densenet52, densenet121, densenet169, densenet201 and densenet264 are all evaluated. Readers can walk through **group-project-DenseNet-1** to have a close look. We also test them with different optimizers. The best combination is proved to be Adam and densenet169 in learning rate of 0.01. Every 100 epochs, the learning rate decrease to its one-tenth because of the preset scheduler. After 400 epochs of training, the training loss is 0.0919, while the test loss is 0.2788. The result is shown in Fig. 17.

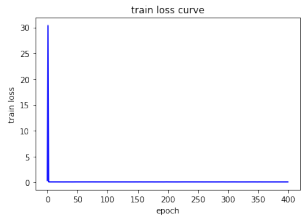

Figure 16. densenet169 training loss

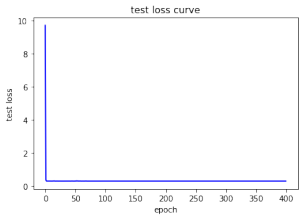

Figure 17. densenet169 test loss

5.6. ResNeXt

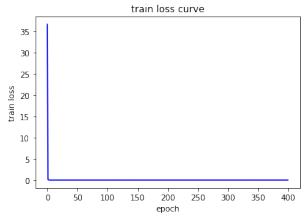

Figure 18. resnext32_16x8d training loss

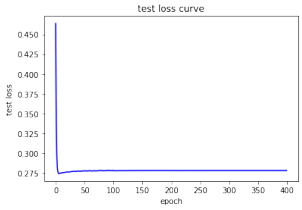

Figure 19. resnext32_16x8d test loss

Only resnext32_16x8d is evaluated in this part. We set learning rate to 0.01, using SDG without momentum as an optimizer, to train the model. Every 100 epochs, the learning rate decrease to its one-tenth because of the preseted scheduler. After 400 epochs of training, the training loss is 0.0912, while the test loss is 0.2782. The result is shown in Fig. 19.

To figure out all the results of resnext32_16x8d with different hyperparameters, please refer to appended file **group-project-ResNeXt-1**.

5.7. Analysis

Table 3 shows the optimal results of each neural network models. Omitted the abnormal mlp2, we can see that the performance of each network is quite close – although they do have a great difference in their structures. Moreover, the first-ranked kernel of the PUBG submission has a test loss less than 0.025, which means that we are still far away from the best solution.

Model	Training Loss	Test Loss
mlp1	0.1411	0.3444
mlp2	164.3882	10.0996
resnet18	0.0944	0.2755
seresnet18	0.0907	0.2788
vgg11	0.0919	0.2788
densenet169	0.0919	0.2788
resnext32_16x8d	0.0912	0.2782

Table 3. Table of performances of MLPs and CNNs

6. Visualization and Feature Engineering

Inspired by the data visualization [9], [2] and Feature Engineering [10] kernels in Kaggle, we apply some of their methods to develop our own features.

6.1. Visualization

In this part, we try to analyze the relation of all the stats to locate valuable information to do further prediction. The source code and more figures are available in appendix **visualization** file.

6.1.1 Kill and Damage

It is commonly believed that the top killers are likely to be the battle royals (winners of the game). We walk through the dataset by pandas and figure out that the average person kills 0.9248 players, 99% of people have 7.0 kills or less, while the most kills ever recorded is 72. The Fig. 20 and Fig. 21 show the distribution of kill count and damage stats respectively.

From the kill-winPlacePerc plot in Fig. 22, we can draw a conclusion that a player with more kill counts is more likely to win the game.

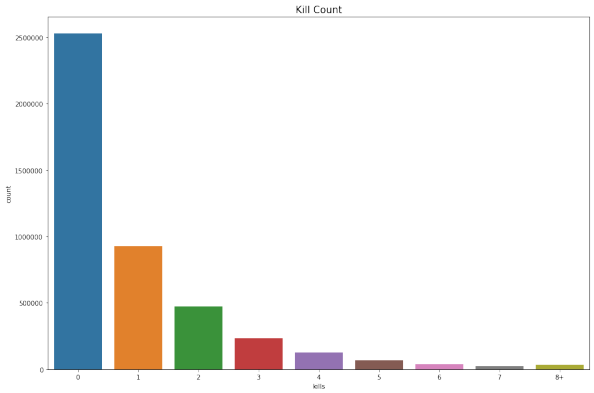

Figure 20. distribution of kill count

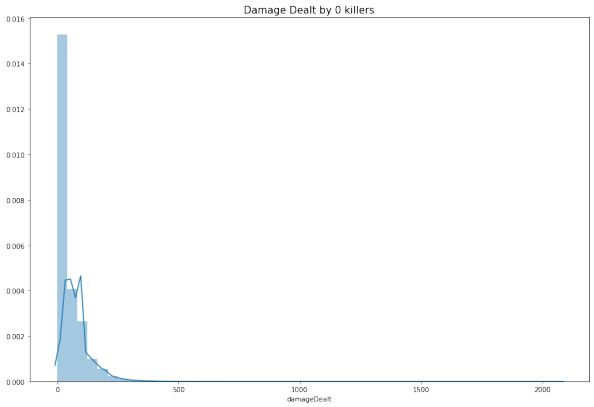

Figure 21. distribution of damage stats

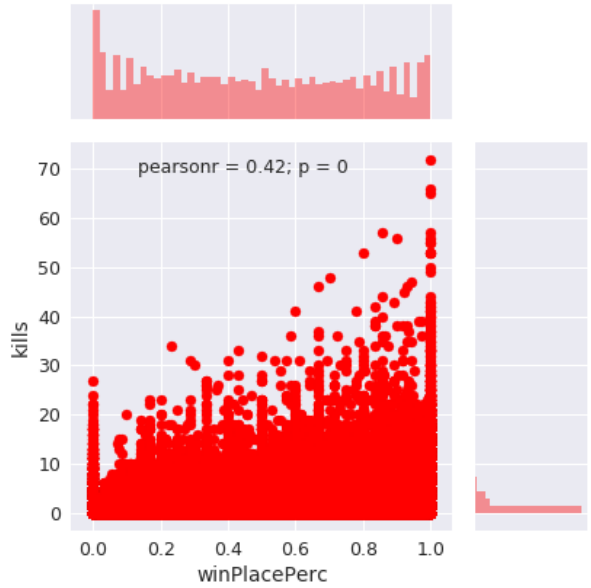

Figure 22. distribution of damage stats

6.1.2 Walking distance

Longer walking distance means longer life, and thus the player should have a larger winPlacePerc. The average person walks for 1154.2m, 99% of people have walked 4396.0m or less, while the marathoner champion walked for 25780.0m. We can see the distribution of walkDistance in Fig. 23

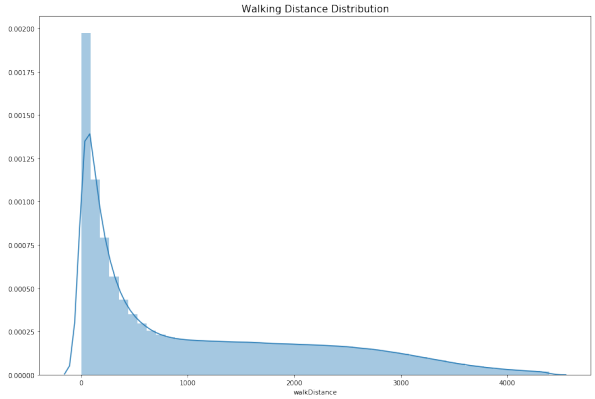

Figure 23. distribution of damage stats

From the walk-winPlacePerc plot in Fig. 22, we can draw a conclusion that a player with longer walking distance is more likely to win the game.

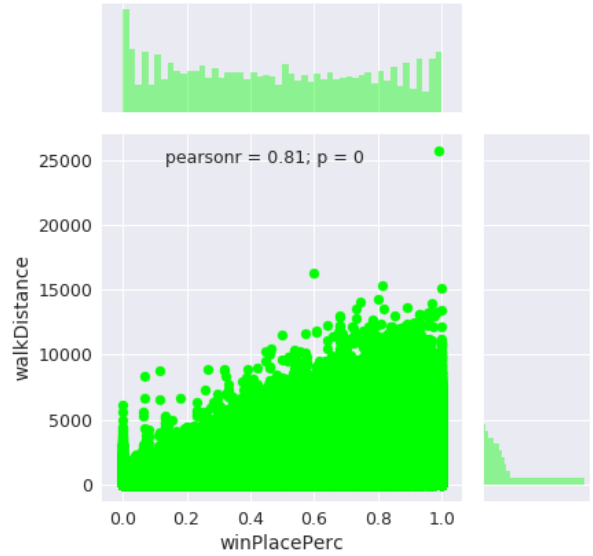

Figure 24. distribution of damage stats

6.1.3 Healing and Boosting

The winner always deals with his wound on his own. No wonder why a good doctor can usually survive from a game. As we can see from Fig. 25 the more heal/boost items are used, the higher average winPlacePerc the play has.

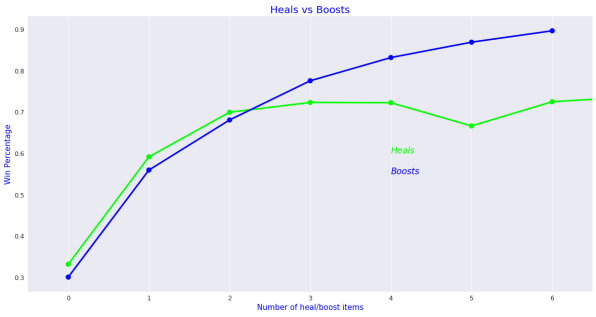

Figure 25. distribution of damage stats

6.1.4 Pearson correlation between variables

Finally, we draw a +table to show the Pearson correlation between winPlacePerc and other variables. As we can see from the last column of Fig. 26, the walkDistance, weaponsAcquired and boost has the highest positive relation to winPlacePerc, while the killPlace has the highest negative relation to it.

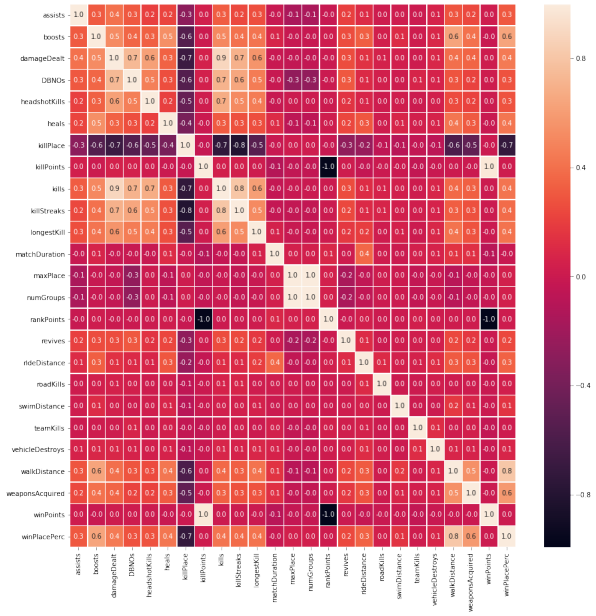

Figure 26. distribution of damage stats

6.2. Feature Engineering

In this part, a new feature is added and cheater stats are removed. Please go through **scikit-RF** for cheater detection and new feature.

There are two main methods to do feature engineering. The extra features are listed in Table 4.

1. Aggregation: Aggregate the metrics with the same meaning in reality.
2. Normalization: Use a metric to normalize the other to make the comparison more reasonable.

Cheating players are also the aim of feature engineering – we should remove them in case these stats affect the prediction to normal ones. However it is a rather empirical way to detect cheater, thank god we have a professional PUBG player in our group. The cheat methods are listed in Table 5.

7. Machine Learning Experiments

In this section, two tree-theory-based classic machine learning methods are applied to predict the winPlacePerc of PUBG dataset. Benefit from the advanced scikit-learn [12] and lightgbm [8] packages, we will not bother implement a efficient model of them. And, as it is a Deep Learning course report, I will not give any detailed implementation in the paper. If readers are curious about the source code, please go through **lightgbm-GDBT** for GDBT and **scikit-RF** for RF regression.

7.1. Random Forest Regression

Random Forest (RF) Regression is implemented in scikit-learn in Python. The hyper parameters of RF regression training is $estimators = 200$, $minimun\_sample\_leaf = 3$ and $max\_feature = 0.5$. Within 10 minutes, the RF regression finished the FULL dataset training, with a training loss at 0.00202852 and test loss at 0.00628085. The feature importance is visualized in Fig. 27.

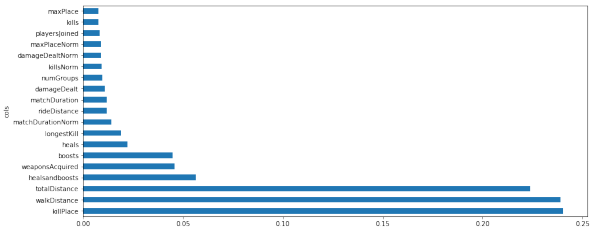

Figure 27. distribution of damage stats

7.2. Gradient Boosting Decision Tree

Gradient Boosting Decision Tree (GDBT) is enabled by lightgbm packages in Python. The hyper parameters of GDBT training is $estimators = 20000$, $early\_stopping\_rounds = 200$, $learning\_rate = 0.05$, $bagging\_fraction = 0.7$, $colsample_bytree = 0.7$, $threads = 4$, and $leaves = 31$. Within 2 hours, the GDBT finished the FULL dataset training, with a training loss at 0.0196855 and test loss at 0.0271666.

Extra Feature	description
playersJoined	the number of players in a game
killsNorm	Normalize the kill count with the playersJoined
damageDealtNorm	Normalize the damageDealt with the playersJoined
maxPlaceNorm	Normalize the maxPlace with the playersJoined
matchDurationNorm	Normalize the matchDuration with the playersJoined
healsAndBoosts	add heals and Boosts
totalDistance	add walking, swimming and riding distance
boostsPerWalkDistance	count of boosts per WalkDistance
healsPerWalkDistance	count of heals per WalkDistance
healsAndBoostsPerWalkDistance	count of healsAndBoosts per WalkDistance
killsPerWalkDistance	count of kills per WalkDistance

Table 4. Table of Extra Feature

Extra Feature	description
Illegal match	winPlacePerc value is NaN
Kills without movement	Kills without movement
Aim bot 1	More than 30 kills
Aim bot 2	100% headshot and more than 10 kills
Aim bot 3	longestKill longer than 1000
Fast travel	totalDistance longer than 10000
Fast loot	weaponsAcquired more than 80
Fast healing and boosting	healsAndBoosts more than 40

Table 5. Table of Cheat Method

8. Conclusion

In this paper, we have done a comprehensive investigation of PUBG dataset. At the very beginning of the paper, we try to apply MLPs and CNNs to the prediction of winPlacePerc. However, we fail to tell the advantages of a more complex CNN in predicting PUBG dataset, even though we have already fine tuned the networks. Being surprised by this situation, we walk through the top-ranked kernels in Kaggle and find the significance of Feature Engineering. Therefore, detailed visualization of every important metrics has been shown and a feature engineering has been done in Section 6. We continue to the tree-theory-based machine learning methods, GDBT and RF regression, to be precise, and find it far better and quicker than ANY neural network models. Let the final result table 6 and its cold-blooded metrics end this long long discussion.

Model	Training Loss	Test Loss
mlp1	0.1411	0.3444
mlp2	164.3882	10.0996
resnet18	0.0944	0.2755
seresnet18	0.0907	0.2788
vgg11	0.0919	0.2788
densenet169	0.0919	0.2788
resnext32_16x8d	0.0912	0.2782
RF regression	0.00202852	0.00628085
GDBT	0.0196855	0.0271666

Table 6. Table of performances of all models

9. Contributions

The percentage followed by the name is the contribution rate of this group project.

1. studentID: 1634. Contributions:implement dataloader of PUBG dataset, filter and cutthe dataset, **implement all six neural networks** (MLP,ResNet, SEResNet, Vgg, DenseNet, ResNext, thanksTA), instruct teammates to fine tune CNNs, analyze the models' structures and their performances, **visualize dataset**, **feature engineering**, **implement RF regression and GDBT**, analyze the failure of neural networks, **write the whole paper**.

2. studentID: 1532. team leader, hand in report, help implement some neural networks, **fine tune neural networks**, write the mid-term report.

3. studentID: 1634. help implement some neural networks, **fine tune neural networks**, write the mid-term report, try an embedding model (but failed, not shown is this paper).

References

[1] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[2] Dimitrios Effrosynidis. Eda for the popular battle royale game pubg. https://www.kaggle.com/deffro/eda-is-fun, 2018.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[5] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[6] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[8] lightgbm. Lightgbm. https://lightgbm.readthedocs.io/en/latest/Python-API.html#training-api, 2019.

[9] Pedro Marcelino. Comprehensive data exploration with python. https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python, 2019.

[10] Rejasupotaro. Effective feature engineering. https://www.kaggle.com/rejasupotaro/effective-feature-engineering, 2019.

[11] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[12] scikit. scikit. https://scikit-learn.org/stable/index.html, 2019.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.