Predicting the Winner in CS:GO Matches

(STATS/CSE 780 course project)

Yan Min (Student number: 400546093)

2023-12-12

Abstract

This study explores the application of advanced machine learning techniques, specifically Random Forest and Deep Neural Network(DNN), in predicting the winners of CS:GO matches. We aim to use the past data of matches to forecast future match results. Leveraging the dataset provided by Skybox as part of their CS:GO AI Challenge, we preprocessed the data and then fed the data into the models. The performance of these two models are evaluated by accuracy, precision, and recall metrics. Our results yielded relatively high accuracies for both models, up to about 75%. The two machine learning models we developed improve the prediction and analysis of future match results, which is beneficial for the development of eSports.

Introduction

The dataset("CS:GO Round Winner Classification" 2020) was originally published by Skybox as part of their CS:GO AI Challenge, running from Spring to Fall 2020. The dataset consists of around 700 demos from high level tournament play in 2019 and 2020. Warmup rounds and restarts have been filtered, and for the remaining live rounds a round snapshot have been recorded every 20 seconds until the round is decided. We want to use this dataset to predict the final winner, which can be seen as a classification problem.

There are totally 122410 entries and 97 colums in this dataset. The features have one boolean type, 94 float types and 2 object types. The number of entries is greater than the number of parameters. There are no null values in this dataset. We choose `round_winner` as the output value, which is consisted of the winners `CT` and `T`. There are other different types of features, such as `time_left`, `ct_score` and `ct_health`, which will be used as the input features. Figure 1 shows the distrubutions of selected features after feature slection.

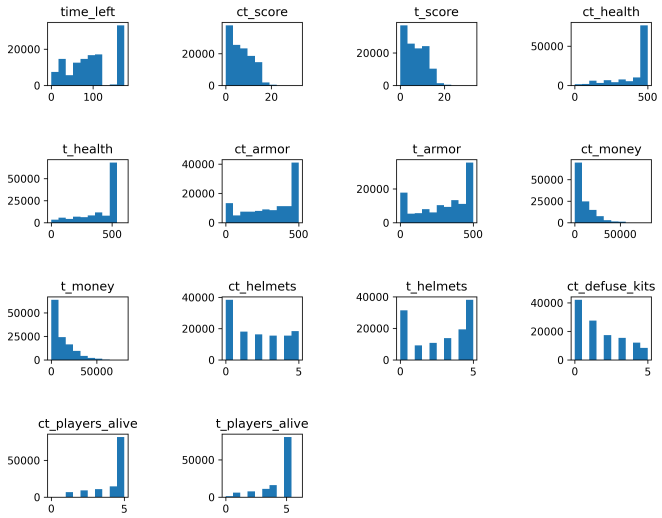


Figure 1: The distribution of selected variables after feature selection

Methods

Firstly, we will do Exploratory Data Analysis(EDA) to find out the relationships among different features. Then, we will drop unnessaray columns and transform the values of some features based on the analysis. Afterwards, we will do scaling.

The reasons for choosing random forest are that it can capture the complex interactions and non-linear relationships among the features, and that it is suitable for classification problems(`CT` or `T`). For the model of random forest, beforing fitting the data, we do the principal component analysis(PCA) for the features. Regarding to the tuning of the parameters, we do grid searching to find the best parameters for the model and the best accuracy score. Then we use the training dataset to check whether the model is overfitting and the test dataset to get the accuracy score.

The reasons for choosing deep neural network(DNN) are that it suits for large datasets, which in this case consists of `122410` entries, and that we can benefit from the trained model for future use. For the model of DNN, we set the number of layers as `4`, the number of nodes of each layer as `300`, and the number of epochs as `50`. Between each layer, we also add batch normalization layers, in order to normalize the activations. We choose the binary cross entropy as the loss function and the `Nadam` as the optimizer. Then we build the DNN model and render the accuracy scores for the training set and test set.

Results

Fistly, we do feature selection. As there are 97 variables in this dataset, we need to decrease the number of features in order to reduce the training time. As we can see, over eighty features are about how much of each weapon is left in the game and we can know that these features are high correlated about `ct_money` and `t_money`. So we drop them off and select 15 of them. Then we draw the correlation matrix of the selected features. From figure 2, we can tell that `ct_players_alive` is high correlated with `ct_health`, and `t_players_alive` is highly correlated with `t_money`. So we drop off the colomns of `ct_players_alive` and `t_players_alive`. Then we can use the left 13 features as inputs and `round_winner` as outputs. To do the prediction, in the colomn of `round_winner`, we convert `CT` to `0`, and `T` to `1`.

For the column of `bomb_planted`, at first, the values are boolean types and we convert `true` to `1` and `false` to `0`. For others features, we do data scaling. For the column of `round_winner`, the number of `CT` and the number of `T` are close to `50%` of the total entries. So we can directly split the dataset. After this, we divide the dataset into training set and test set, according to the test size of `0.3`.

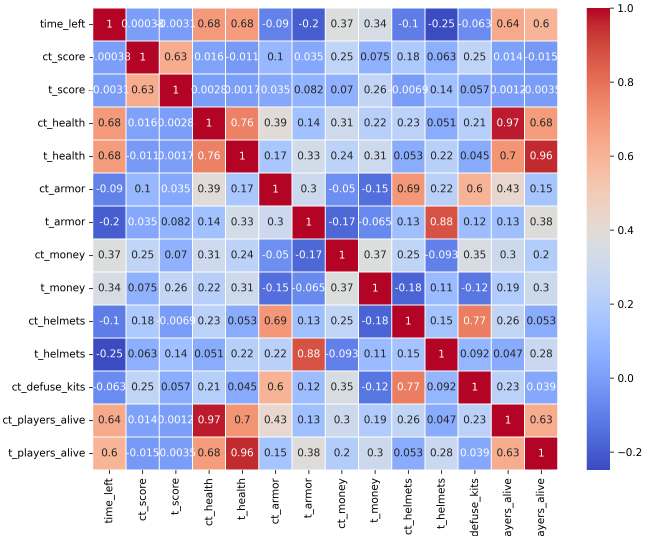


Figure 2: The correlation matrix of selected features

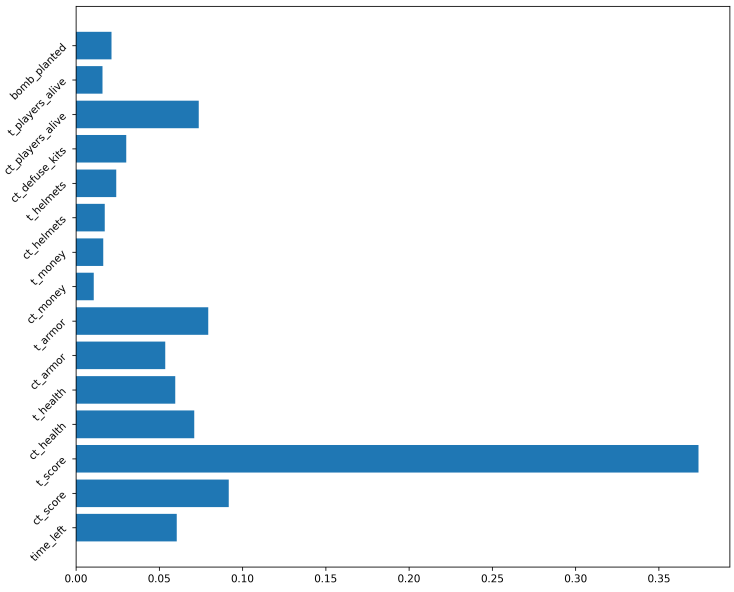


Figure 3: The Feature Importance Derived from the Random Forest Model

Deriving from grid searching, we have that the best parameter for `min_samples_split` is `8`. We can obtain feature importance from the Random Forest mode, as figure 3 shows. We can see that `t_score` is the most important among all features, following by `ct_score`. We can also see that the scores of both team make the biggest impact on final results, as the scores may impact each team's morale. As table 1 shows, the accuracy score for the training set is `0.791`, which is accepetable since it's not overfitting, and the accuracy score for the test set is `0.764`, which is pretty good for prediction. Plus, the F1 score of the random forest model is `0.765`, which indicates a good balance between precision and recall in the model's predictions. In a real game, this accuracy is significant in predicting the outcome of the game.

	Random Forest	**DNN**
Accuracy for Test Set	0.764	0.752
Accuracy for Training Set	0.791	0.750
F1 Score	0.764	0.767

Table 1: Accuracies and f1 scores for both models

For the DNN model, the accuracy on the test set is `0.752`, while on the training set, it is `0.750`. There is no strong sign of overfitting. And the F1 score is 0.767, which indicates a strong balance for precison and recall in this model. The results from the DNN model are promising and could significantly benefit team strategies. Since the DNN model consists of many layers and neurals,

which allows it to deal with large datasets like this.

Conclusion

From the results, we can conclude that both models are suitable for predicting the winners trained on this dataset. And the accuracy scores for two models are close, but the random forest model are bad at overfitting compared to DNN model in this case. But random forest model can give us some valuable insights on the feature importance. We obtain that we should pay more attention on scores and health points during the games. So the team can try different strategies based on this. Besides, game updates can significantly impact the performance of the models. But to DNN, DNN can keep learning the new features or new values so that it can predict the winner well even in new versions of CS:GO matches. To be specificly, we can use the techniques of transfer learning, using the weight matrice obtained from the previous model and use it as initial weight matrices for the DNN model.

While the random forest model offers insights into feature importance, the DNN model is good at adapting to new features and updates of games, due to its ablility to continuously learn. There a tradeoff between interpretability and adaptability when it comes to choose random forest or DNN. Besides, training and deploying DNN models, especially with transfer learning, can be computationally intensive. Also, grid searching can be time-consuming when it comes to large models.

Supplementary material

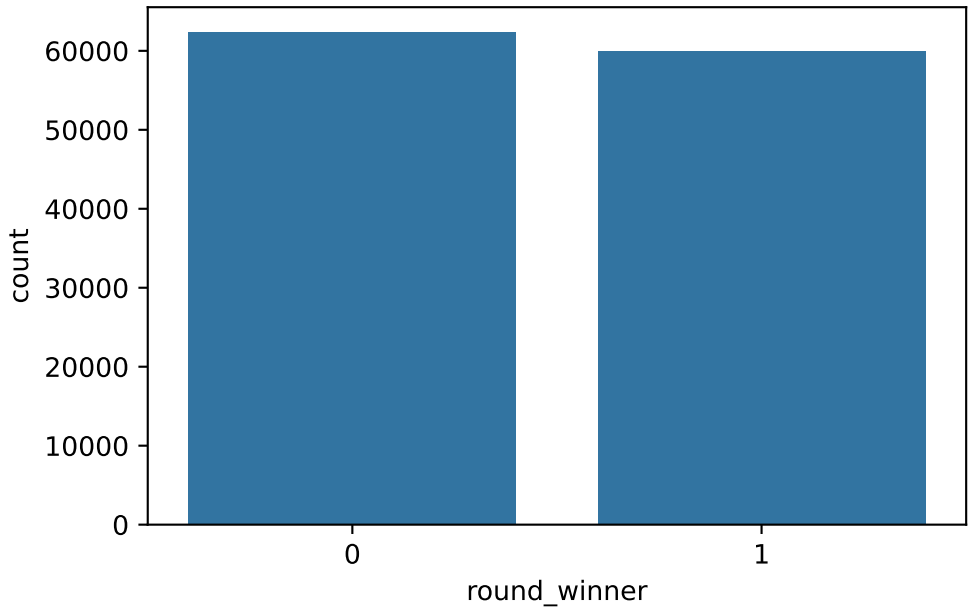
1. Import the libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import scale
```

2. Read the datasets

```
df = pd.read_csv('./csgo_round_snapshots.csv')
#df.info()

df.isnull().sum()

df['round_winner'] = df['round_winner'].apply(lambda x: 1 if x == 'CT' else 0)
print(df['round_winner'].value_counts())

sns.countplot(x='round_winner', data=df)
```

```
round_winner
0    62406
1    60004
Name: count, dtype: int64

<Axes: xlabel='round_winner', ylabel='count'>
```



3. Data preprocessing

```
df['bomb_planted'] = df['bomb_planted'].astype(int)
n = 16
df_selected = df.iloc[:, list(range(n)) + [-1]]
df_selected = df_selected.drop('map', axis=1)
df_selected.head(5)
```

4. Scaling

```
X = df_selected.drop(['round_winner', 'bomb_planted'], axis=1)
Y = df_selected['round_winner']

num_rows, num_cols = 4, 4  # Specify the grid size
fig, axs = plt.subplots(num_rows, num_cols, figsize=(10, 8))
fig.tight_layout(pad=5.0)  # Adds padding between plots

# Iterate over the DataFrame columns and plot
for i, col in enumerate(X.columns):
    row = i // num_cols
```

```
    col_index = i % num_cols
    axs[row, col_index].hist(X[col])
    axs[row, col_index].set_title(col)

# Hide the empty subplots
for i in range(len(X.columns), num_rows * num_cols):
    axs[i // num_cols, i % num_cols].axis('off')

plt.savefig("all_data.png")
plt.show()

correlation_matrix = X.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=.5)
plt.savefig('corr.png')
plt.show()

from sklearn.preprocessing import StandardScaler
std = StandardScaler()
std.fit(X)
X = pd.DataFrame(std.transform(X), columns=X.columns)
X['bomb_planted'] = df_selected['bomb_planted']
X.shape
```

5. Splitting

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=8964)
```

6. PCA

```
from sklearn.decomposition import PCA

pca_X = PCA()
```

```
X_pca = pd.DataFrame(pca_X.fit_transform(X), columns=X.columns, index=X.index)
X_pca

#Train/test for decision trees
X_train_p, X_test_p, Y_train_p, Y_test_p = train_test_split(
  X_pca, Y, test_size=0.3, random_state=8964)
```

7. Random Forest

```
rf = RandomForestClassifier(n_estimators=100,
  random_state=8964, max_depth=10, min_samples_split=8)  # For classification
rf.fit(X_train_p, Y_train_p)
Y_pred = rf.predict(X_test_p)

# For classification
accuracy = accuracy_score(Y_test_p, Y_pred)
accuracy

# from sklearn.model_selection import GridSearchCV
# rf = RandomForestClassifier(random_state=8964)
# param_grid = {
#     'min_samples_split': [2, 4, 6, 8, 10]
# }
# grid_search = GridSearchCV(estimator=rf,
#   param_grid=param_grid, cv=5, scoring='accuracy')
# grid_search.fit(X_train_p, Y_train_p)
# print(f"Best Parameter: {grid_search.best_params_}")
# print(f"Best Score: {grid_search.best_score_}")

Y_pred_1 = rf.predict(X_train_p)
accuracy = accuracy_score(Y_train_p, Y_pred_1)
```

```
accuracy


plt.figure(figsize=(10, 8))
plt.barh(X_train.columns,rf.feature_importances_)
plt.yticks(rotation=45)
plt.tight_layout()  # Adjust layout
plt.savefig('feature.png')

from sklearn.metrics import f1_score

f1 = f1_score(Y_test_p, Y_pred, average='binary')

print("F1 Score:", f1)
```

8. DNN

```
from tensorflow import keras

# Set model parameters
n_layers = 4
n_nodes = 300
regularized = False
dropout = True
epochs = 50

# Make a Keras DNN model
model = keras.models.Sequential()
model.add(keras.layers.BatchNormalization())
for n in range(n_layers):
    if regularized:
        model.add(keras.layers.Dense(n_nodes, kernel_initializer="he_normal",
         kernel_regularizer=keras.regularizers.l1(0.01), use_bias=False))
```

```
    else:
        model.add(keras.layers.Dense(n_nodes,
         kernel_initializer="he_normal", use_bias=False))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Activation("elu"))
    if dropout:
        model.add(keras.layers.Dropout(rate=0.2))
model.add(keras.layers.Dense(1, activation="sigmoid"))
model.compile(loss='binary_crossentropy',
  optimizer='Nadam', metrics=['accuracy'])

# Make a callback that reduces LR on plateau
reduce_lr_cb = keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                                                 patience=5, min_lr=0.001)

# Make a callback for early stopping
early_stopping_cb = keras.callbacks.EarlyStopping(patience=5)

# Train DNN.
history = model.fit(np.array(X_train), np.array(Y_train), epochs=epochs,
     validation_data=(np.array(X_test), np.array(Y_test)),
      callbacks=[reduce_lr_cb, early_stopping_cb], batch_size=128)
```

```
model.summary()
model.evaluate(X_test, Y_test)

predictions = model.predict(np.array(X_test))
predicted_labels = (predictions > 0.5).astype(int)

f1 = f1_score(Y_test, predicted_labels, average='binary')
```

```
print("F1 Score:", f1)
```

References

"CS:GO Round Winner Classification." 2020. https://www.kaggle.com/datasets/christianlillelund/csgo-round-winner-classification.