Predicción de posición final en una partida de PUBG

Por:

Jhon Alexander Bedoya Carvajal

Maria Camila Arcila Ramírez

Curso:

Introducción a la Inteligencia Artificial

Docente:

Raúl Ramos Pollan

Universidad de Antioquia

Facultad de Ingeniería

Medellín

2022

Contenido

1. Introducción

Uno de los pilares más importantes para la Inteligencia Artificial, es precisamente el poder analizar como nosotros los seres humanos podemos buscar distintas soluciones para problemas que se nos presentan día con día en todo lugar y aún más importante el saber cómo somos capaces de poder seleccionar una solución entre tantas posibles y poder resolver los imprevistos que se nos presentan. En este trabajo se tiene a disposición más de 65,000 juegos en datos de jugadores anónimos, divididos en conjuntos de entrenamiento y prueba, y se busca predecir la posición en el ranking final a partir de las estadísticas finales del juego y las calificaciones iniciales de los jugadores.

2. Planteamiento el problema

Los videojuegos estilo Battle Royale han conquistado el mundo. 100 jugadores caen en una isla con las manos vacías y deben explorar, buscar y eliminar a otros jugadores hasta que solo quede uno en pie, todo mientras la zona de juego continúa reduciéndose. PlayerUnknown's BattleGrounds (PUBG) ha disfrutado de una gran popularidad. Con más de 50 millones de copias vendidas, es el quinto juego más vendido de todos los tiempos y tiene millones de jugadores activos mensuales. Se busca predecir la posición en el ranking final a partir de las estadísticas finales del juego y las calificaciones iniciales de los jugadores. ¿Cuál es la mejor estrategia para ganar en PUBG? ¿Deberías sentarte en un lugar y esconderte en tu camino hacia la victoria, o necesitas ser el mejor tirador? ¡Dejemos que los datos hablen!

2.1. Dataset

El Dataset proviene de una competencia de Kaggle en la que se proporciona datos de más de 65 mil partidas de PUBG. Este Dataset cuenta con 29 columnas y 4446966 filas.

Debido a la cantidad tan extensa de datos, se tomó una muestra aleatoria de 200.000 filas para trabajar en este proyecto. Inicialmente, se buscó cumplir con los requisitos establecidos para el Dataset (al menos el 10% de las columnas han de ser categóricas y al menos ha de tener un 5% de datos faltantes en al menos 3 columnas). Las columnas 'killPlace', 'maxPlace' y 'matchType' se transformaron a categóricas. Por último, se eligieron 3 columnas aleatoriamente para eliminar entre el 5% y 10% de las entradas para cumplir con el requisito de los datos faltantes.

2.2. Métrica

La métrica de evaluación para esta competencia es el error absoluto medio (MAE) entre el winPlacePerc predicho y el winPlacePerc observado. El MAE es una medida de errores entre observaciones emparejadas, se calcula como:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

Es un promedio aritmético de los errores absolutos $|e_i| = |y_i - x_i|$, donde y_i es la predicción y x_i el verdadero valor.

En cuanto a la métrica de negocio, nuestro modelo de predicción de la posición final del jugador debería tener un $MAE \leq 10\%$, ya que se usará el modelo para determinar cuál es la mejor estrategia para ganar el juego, lo que se busca es mediante las posiciones de ranking de los jugadores, ver si se tiene más éxito cuando el jugador se esconde, o si es mejor estrategia salir a competir con los demás jugadores, por lo que es importante tener el MAE por debajo del 10%

2.3. Variable Objetivo

La variable que se quiere predecir es la posición final a partir de las estadísticas finales del juego y las calificaciones iniciales de los jugadores. Basados en esto, determinar cuál puede ser la mejor estrategia para ganar una partida de PUBG. Esta ubicación es un valor percentil que se calcula a partir de la variable 'maxPlace'.

3. Exploración descriptiva del Dataset

3.1. Análisis de variable objetivo

Para el análisis de la variable objetivo se graficó la distribución de esta y frente a otras variables de interés como las muertes y las posiciones en los diferentes rankings.

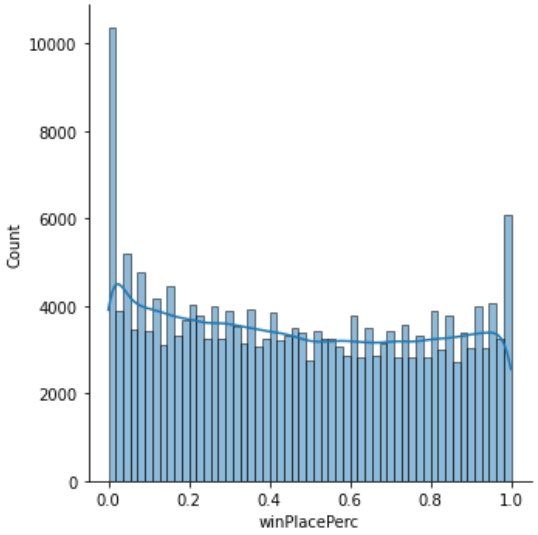

***Gráfico 1**. Distribución de la variable objetivo.*

El *Gráfico 1* muestra la distribución de la ubicación final y evidencia que las observaciones se reúnen más en la parte izquierda, aunque no es un sesgo muy marcado, esto lo podemos confirmar con un coeficiente de asimetría de 0.0983. Además, la mediana o cuantil del 50% da un valor de 0.4583, lo que quiere decir que el 50% de los jugadores quedan en una posición menor a 0.4583.

Los jugadores se ubican en una posición final promedio de 0.4725.

3.2. Variable Kills.

Para explorar la variable kills se creó una variable temporal que define rangos de acuerdo con el número de kills, esta variable toma el nombre de “kill ranges” y en las Gráficos 2 y 3 se pueden ver los diferentes rangos que hacen parte de esta.

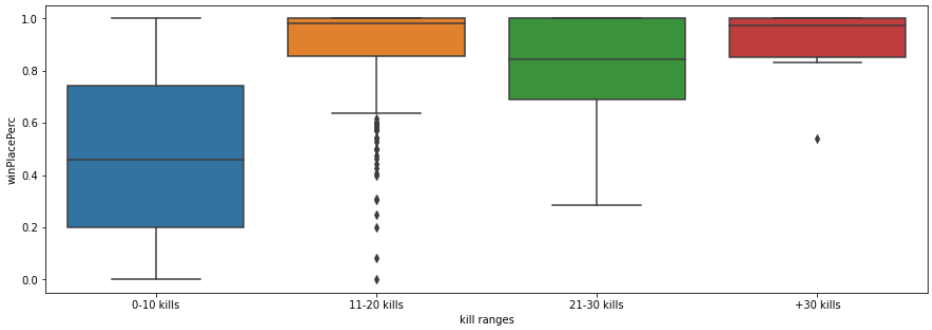

Gráfico 2. *Relación entre el número de kills y la variable objetivo.*

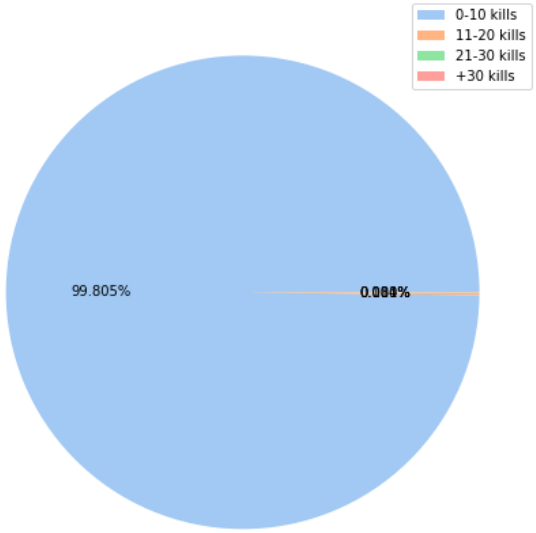

Gráfico 3. *Proporción de rangos de kills.*

De la *Gráfico 3* podemos concluir que el rango con más jugadores es de 0 a 10 kills, siendo una proporción del 99.805% de los datos. De la Gráfico 2, se concluye que estos jugadores normalmente terminan en una posición final entre 0.2 y 0.75, además, que los jugadores con un número de kills mayor al rango 0-10 suelen quedar en los primeros puesto al final de la partida, es decir, posiciones finales entre 0.8 y 1.

3.3. Variables walkDistance y swimDistance.

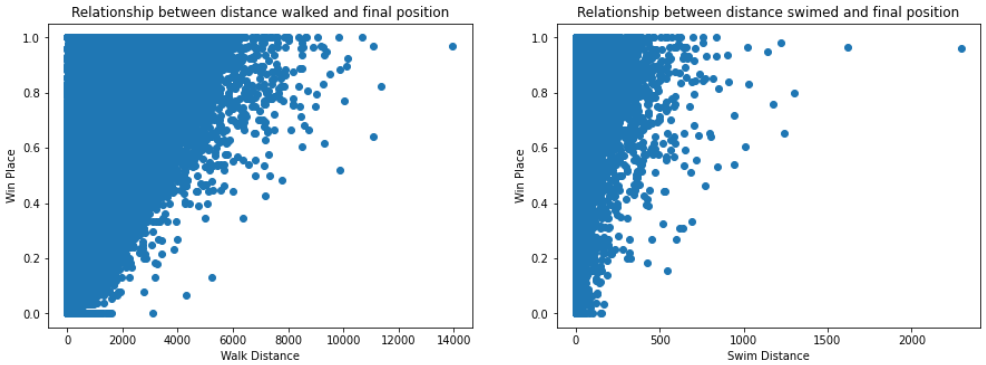

Gráfico 4. *Relaciones entre distancia caminada y distancia nadada con la variable objetivo.*

En la *Gráfico 4* se puede observar que sí hay, aunque poca, una relación entre la distancia caminada y nadada en la partida con la posición final. Los jugadores con mejores posiciones al final de la partida suelen caminar y nadar distancias largas, aunque muchas veces esto no se cumple, pues también hay una gran acumulación de jugadores con buenas posiciones sin haber caminado o nadado, aunque sea cortas distancias. Observemos esto numéricamente:

La proporción de jugadores que quedan en posiciones finales superiores al promedio sin caminar es de: 0.062 %
La proporción de jugadores que ganan la partida sin caminar es de: 0.0155 %

La proporción de jugadores que ganan la partida sin caminar más de mil metros es: 0.11449999999999999 %

La proporción de jugadores que quedan en posiciones finales superiores al promedio sin nadar es de: 42.803999999999995 %

La proporción de jugadores que ganan la partida sin nadar es de: 2.436 %

3.4. Variables boosts y heals.

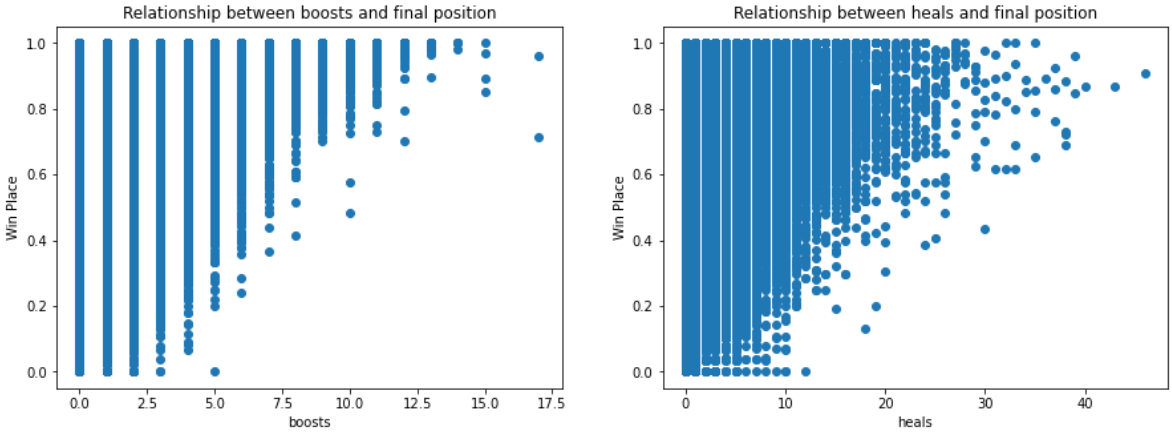

Gráfico 5. *Relaciones entre refuerzo y curaciones utilizadas con la variable objetivo.*

En la *Gráfico 5* podemos evidenciar que pasa igual que son distancia caminada y distancia nadada, los jugadores que quedan en las mejores posiciones al final de la partida suelen usar más refuerzos (boosts) y más curaciones (heals), pero también hay muchos casos en los que quedan en las mejores posiciones sin usar ningún refuerzo o usar alguna curación. Lo que sí es muy evidente es que los jugadores que quedan en las últimas posiciones usan pocos refuerzos y curaciones o no usan ninguno.

En promedio, el número de refuerzo utilizados por los ganadores de las partidas es: 3.884

La proporción de jugadores que ganan la partida sin usar ningún refuerzo es de: 0.214 %

En promedio, el número de curaciones utilizadas por los ganadores de las partidas es: 3.377

La proporción de jugadores que ganan la partida sin usar ninguna curación es de: 0.469 %

3.5. Variables rideDistance, roadKills y vehicleDestroys.

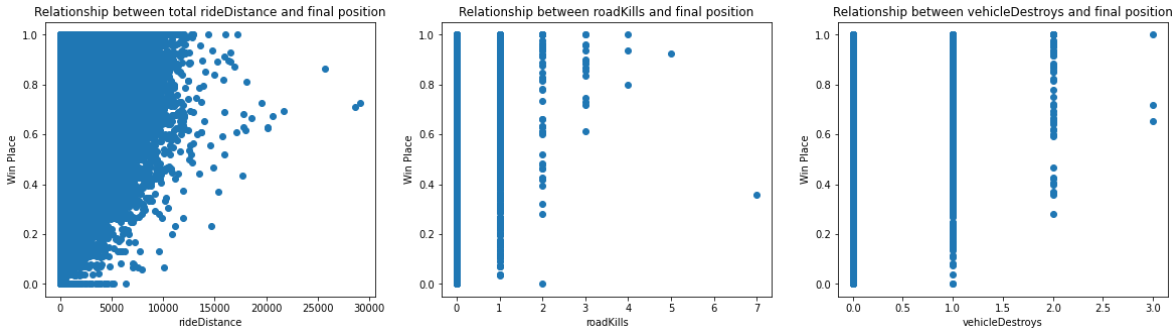

Gráfico 6. *Relaciones entre distancia recorrida en vehículos, muertes hechas desde un vehículo y vehículos destruidos con la variable objetivo.*

En la *Gráfico 5* se puede observar que tenemos una relación muy similar a las anteriores, aunque esta vez no es tan marcada. Las variables rideDistance, roadKills y vehicleDestroys no tienen mucha relación con la variable objetivo winPlacePerc.

3.6. Variable matchType.

```
squad-fpp           79132
duo-fpp             44844
squad               27909
solo-fpp            24313
duo                 14135
solo                 8102
normal-squad-fpp      727
crashfpp              272
normal-duo-fpp        254
flaretpp              139
normal-solo-fpp        92
flarefpp               23
normal-squad           22
crashtpp               15
normal-solo            15
normal-duo              6
Name: matchType, dtype: int64
```

Tabla 1. *Frecuencia absoluta de matchType.*

En la *Tabla 1* se observa la frecuencia absoluta de mayor a menor de los diferentes tipos de juego que hay en la variable matchType. El tipo de juego “squad-fpp” es por mucho el más frecuente, seguido del “duo-fpp” y “squad”.

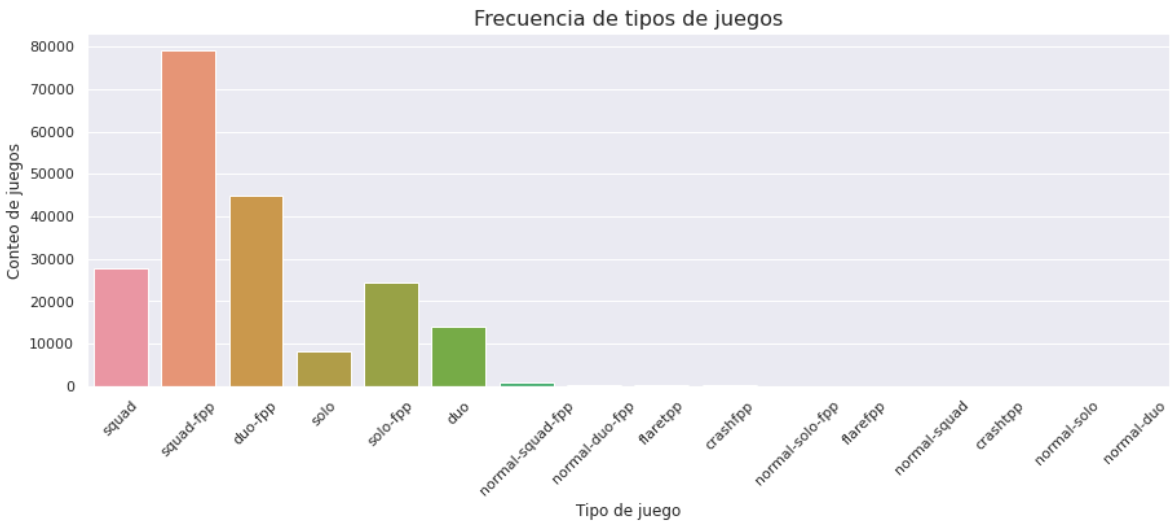

Gráfico 7. *Frecuencia absoluta de matchType.*

En la *Gráfico 7* se presenta gráficamente lo observado en la tabla 1. Ciertamente, el tipo de juego "squad-fpp" es por mucho el más frecuente, seguido del "duo-fpp" y "squad".

3.7. Variables restantes

Gráfico 8. *Relación entre el resto de variables.*

En la *Gráfico 8* se observa la relación del resto de variable numéricas con la variable objetivo winPlacePerc. No hay alguna relación destacable con la variable objetivo.

3.8. Distribución de variables numéricas

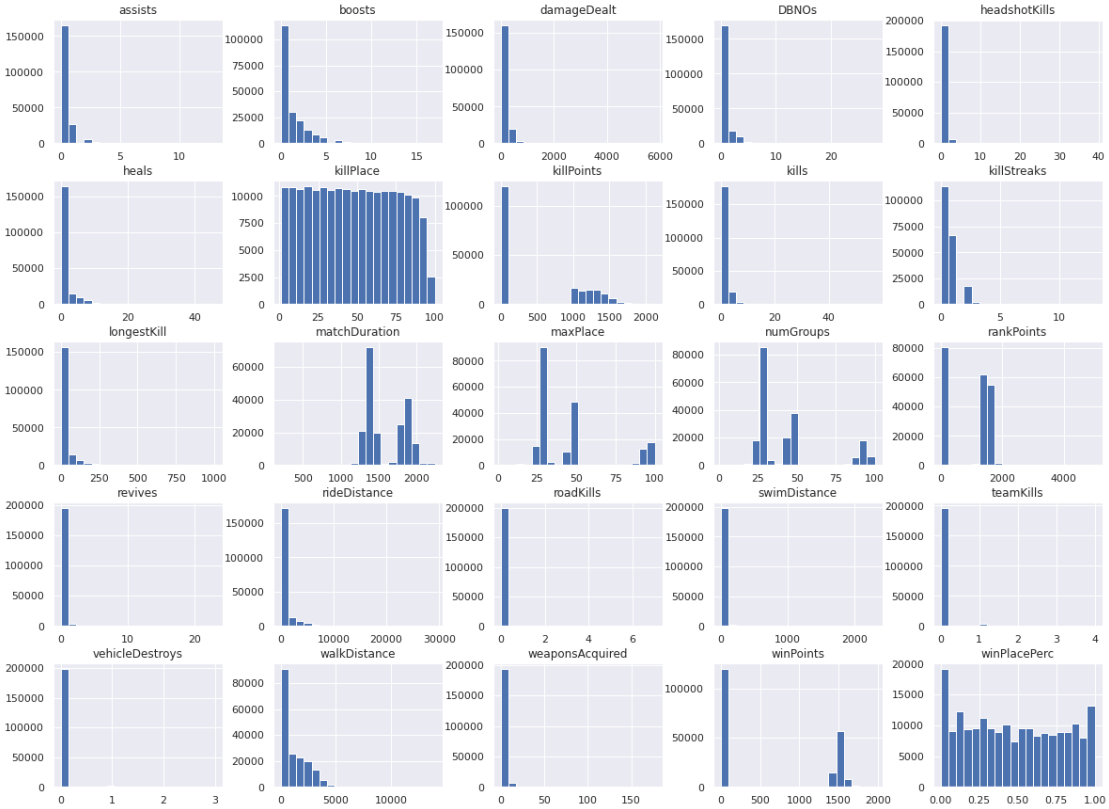

Gráfico 9. *Distribución de las variables numéricas.*

3.9. Correlación de variables

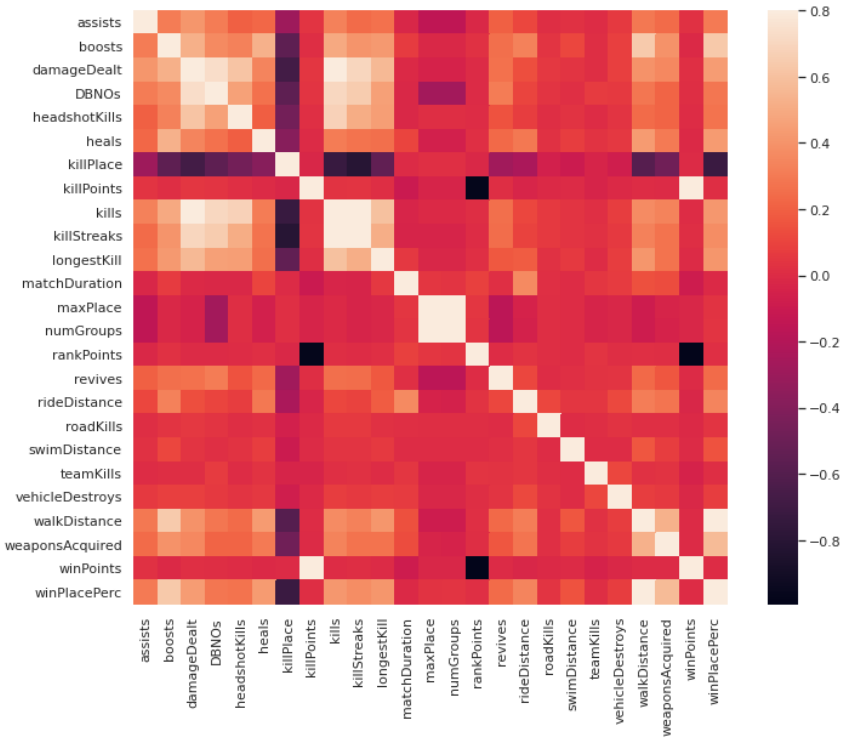

Gráfico 10. *Nivel de correlaciones.*

	winPlacePerc
winPlacePerc	1.000000
walkDistance	0.810308
boosts	0.634664
weaponsAcquired	0.572598
damageDealt	0.440375
heals	0.428850
kills	0.417856
longestKill	0.409437
killStreaks	0.376265
rideDistance	0.340665
assists	0.298843
DBNOs	0.281746
headshotKills	0.274252
revives	0.242931
swimDistance	0.149027
vehicleDestroys	0.074130
numGroups	0.038468
roadKills	0.038391
maxPlace	0.035890
rankPoints	0.015420
teamKills	0.013980
killPoints	0.011410
winPoints	0.005528
matchDuration	-0.005940
killPlace	-0.717686

Tabla 2. *Valores de correlación entre las variables y la variable objetivo.*

En el *Gráfico 10* se observa el nivel de correlación que tienen todas las variables entre sí. Mientras que la *Tabla 2* muestra el valor de la correlación entre las variables y la variable objetivo. De esta tabla se obtienen las variables que tienen poca correlación, para este caso serán las que estén entre -0.1 y 0.1 y se observan en la tabla 3.

```
['vehicleDestroys',
 'numGroups',
 'roadKills',
 'maxPlace',
 'rankPoints',
 'teamKills',
 'killPoints',
 'winPoints',
 'matchDuration']
```

Tabla 3. *Variables que tienen poca correlación con la variable objetivo.*

4. Tratamiento de datos

4.1. Eliminar columnas que no aportan información

Las columnas Id, groupId y matchId no aportan información. Se decide eliminarlas.

```
df = df.drop(['Id'], axis=1)
df = df.drop(['groupId'], axis=1)
df = df.drop(['matchId'], axis=1)
df.head()
```

Imagen 1. *Eliminación de variables.*

4.2. Definir variables categóricas

Las variables 'killPlace', 'maxPlace' y 'matchType' y se manipularán como variables categóricas, ya que son propiedades cualitativas.

```
df['killPlace'] = df['killPlace'].astype('category')
df['maxPlace'] = df['maxPlace'].astype('category')
df['matchType'] = df['matchType'].astype('category')
```

Imagen 2. *Transformación de variables.*

4.3. Valores faltantes

Para el tratamiento de datos se hace un análisis de qué método de reemplazo (remplazar por cero, reemplazar por la media, reemplazar por valores de una normal equivalente) es el que mejora más significativamente el modelo. Con pruebas de hipótesis e iteración de modelos como Arboles de decisión, de encontró que los tres métodos de reemplazo son significativos.

En la tabla 4 se muestran las columnas con valores faltantes.

```
col_nulls = df.isnull().sum()
col_nulls[col_nulls!=0]
```

Imagen 3. *Encontrar variables con valores faltantes.*

```
damageDealt     16400
longestKill     17000
walkDistance    19200
dtype: int64
```

Tabla 4. *Variables con valores faltantes.*

4.3.1. Tres técnicas de sustitución

- Por un valor fijo (cero)
- Por un valor fijo (media)
- Muestreo de una normal equivalente (misma media y desviación estándar)

4.3.2. Análisis Gráfico de las diferentes técnicas de reemplazo de valores faltantes.

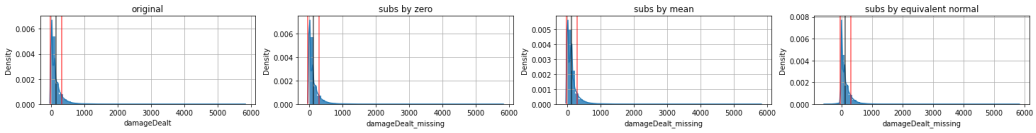

Gráfico 11. *Reemplazo de valores faltantes en damageDealt.*

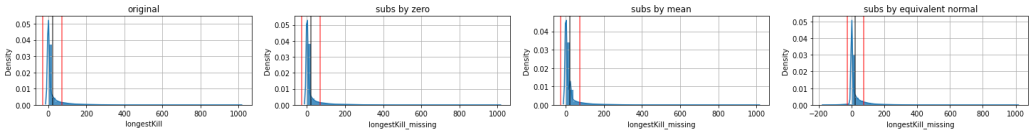

Gráfico 12. *Reemplazo de valores faltantes en logentkills.*

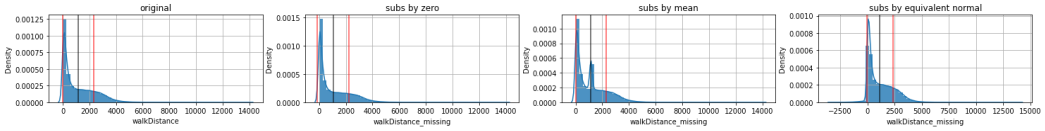

Gráfico 13. *Reemplazo de valores faltantes en walkDistance.*

4.3.3. Análisis de pruebas de hipótesis para valores faltantes.

- e_1: grupo de control, modelos entrenados sin **damageDealt**
- e_2: grupo de población, modelos entrenados sin **damageDealt** con cualquiera de las tres técnicas de reemplazo aplicadas

Nuestra hipótesis nula (no hay ningún efecto en el uso de la nueva variable):

$$H_0: \mu_{e_1} - \mu_{e_2} = 0$$

Nuestra hipótesis de prueba (aplicar la técnica mejora los modelos):

$$H_1: \mu_{e_1} - \mu_{e_2} < 0$$

La anterior hipótesis se aplica también para las técnicas de reemplazo por la media, por un muestreo de una normal estándar equivalente y para cada variable con valores faltantes.

4.3.3.1. Pruebas de hipótesis para técnica de reemplazo de valores faltantes para la variable damageDealt.

```
100% (4 of 4) |########################| Elapsed Time: 0:14:49 Time:  0:14:49
Ttest_indResult(statistic=4.466452523844365, pvalue=0.0002982831806199279)
Ttest_indResult(statistic=5.459949721096866, pvalue=3.469174879193539e-05)
Ttest_indResult(statistic=6.020834689173713, pvalue=1.0799498903555681e-05)
```

Imagen 4. *Resultado de hipótesis de técnicas de reemplazo de valores faltantes en damageDealt.*

4.3.3.2. Pruebas de hipótesis para técnica de reemplazo de valores faltantes para la variable longestKill.

```
100% (4 of 4) |########################| Elapsed Time: 0:14:26 Time:  0:14:26
Ttest_indResult(statistic=15.707045569864079, pvalue=5.949707898443066e-12)
Ttest_indResult(statistic=15.01049323866388, pvalue=1.2740279450854496e-11)
Ttest_indResult(statistic=14.802588733946148, pvalue=1.6085885364922523e-11)
```

Imagen 5. *Resultado de hipótesis de técnicas de reemplazo de valores faltantes en logentkills.*

4.3.3.3. Pruebas de hipótesis para técnica de reemplazo de valores faltantes para la variable walkDistance.

```
100% (4 of 4) |########################| Elapsed Time: 0:16:01 Time:  0:16:01
Ttest_indResult(statistic=157.260482869588, pvalue=1.0565086518310392e-29)
Ttest_indResult(statistic=179.6382108488179, pvalue=9.64894786330019e-31)
Ttest_indResult(statistic=153.28249059031282, pvalue=1.675012618544972e-29)
```

Imagen 6. *Resultado de hipótesis de técnicas de reemplazo de valores faltantes en wlakDistance.*

Para las 3 columnas con datos faltantes, usar cualquiera de las 3 alternativas de técnicas para llenado de datos faltantes brinda una mejora en los resultados de modelos como un Random Forest Regressor con 20 árboles. No hay una diferencia notable para decidir qué técnica es mejor. Se decide reemplazar los valores faltantes con la media.

```
1 for col in col_nulls[col_nulls!=0].index:
2     df[col] = df[col].fillna( df[col].mean())
```

Imagen 7. *Reemplazar valores faltantes por la media.*

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 26 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   assists         200000 non-null  int64
 1   boosts          200000 non-null  int64
 2   damageDealt     200000 non-null  float64
 3   DBNOs           200000 non-null  int64
 4   headshotKills   200000 non-null  int64
 5   heals           200000 non-null  int64
 6   killPlace       200000 non-null  category
 7   killPoints      200000 non-null  int64
 8   kills           200000 non-null  int64
 9   killStreaks     200000 non-null  int64
 10  longestKill     200000 non-null  float64
 11  matchDuration   200000 non-null  int64
 12  matchType       200000 non-null  category
 13  maxPlace        200000 non-null  category
 14  numGroups       200000 non-null  int64
 15  rankPoints      200000 non-null  int64
 16  revives         200000 non-null  int64
 17  rideDistance    200000 non-null  float64
 18  roadKills       200000 non-null  int64
 19  swimDistance    200000 non-null  float64
 20  teamKills       200000 non-null  int64
 21  vehicleDestroys 200000 non-null  int64
 22  walkDistance    200000 non-null  float64
 23  weaponsAcquired 200000 non-null  int64
 24  winPoints       200000 non-null  int64
 25  winPlacePerc    200000 non-null  float64
dtypes: category(3), float64(6), int64(17)
memory usage: 35.7 MB
```

Tabla 5. *Resultado de reemplazar valores faltantes.*

5. Modelos e iteraciones

5.1. Modelos supervisados

5.1.1. La primera iteración se hace con los siguientes 3 estimadores:

```
estimator1 = LinearRegression()
estimator2 = DecisionTreeRegressor(max_depth=5)
estimator3 = RandomForestRegressor(n_estimators = 2,max_depth = 5)
```

Imagen 8. *Estimadores de la primera iteración.*

5.1.2. Partición de los datos

Es necesario dividir los datos en datos para entrenamiento y otros para pruebas. De los datos que queden para entrenamiento también se tomará una parte para hacer el test en las validaciones y la parte de los datos para prueba se usarán solo al final, cuando se haya hecho la selección de modelos, de modo que se pueda probar el desempeño los modelos seleccionados con datos que no conoce. En la *Imagen 9* se puede observar el código para hacer la partición.

```python
from sklearn.model_selection import train_test_split

test_size = 0.3
val_size  = test_size/(1-test_size)

print (X.shape, y.shape)
print ("test size %.2f"%test_size)
print ("val size is %.2f (relative to %.2f) "%(val_size, 1-test_size))

Xtv, Xts, ytv, yts = train_test_split(X, y, test_size=test_size)
print (Xtv.shape, Xts.shape)
```

Imagen 9. *Código de partición de datos.*

5.1.3. Seleccionar modelos (primera iteración)

Partiendo de los 3 estimadores que se definieron en la Imagen 8, se implementa una validación cruzada para escoger los dos modelos que tengan un mejor rendimiento en esta primera iteración. El código utilizado para esta validación se observa en la *Imagen 10*.

```python
zscores = []
estimators = [estimator1, estimator2, estimator3]
for estimator in estimators:
    print('\n---------', estimator, '----------\n')
    z = cross_validate(estimator, Xtv, ytv,
                       return_train_score=True, return_estimator=False,
                       scoring='neg_mean_absolute_error',
                       cv=ShuffleSplit(n_splits=10, test_size=val_size))

    report_cv_score(z)
    zscores.append(np.abs(np.mean(z["test_score"])))

best = np.argmin(zscores)
print ("\nSeleccionado:", best+1)
best_estimator = estimators[best]
print ("\nModelo seleccionado:")
print (best_estimator)
```

Imagen 10. *Código para validación cruzada.*

```
--------- LinearRegression() ----------

MAE test   0.098 (±0.0002) with 10 splits
MAE train  0.098 (±0.0002) with 10 splits

--------- DecisionTreeRegressor(max_depth=5) ----------

MAE test   0.093 (±0.0007) with 10 splits
MAE train  0.093 (±0.0007) with 10 splits

--------- RandomForestRegressor(max_depth=5, n_estimators=2) ----------

MAE test   0.092 (±0.0010) with 10 splits
MAE train  0.092 (±0.0009) with 10 splits

Seleccionado: 3

Modelo seleccionado:
RandomForestRegressor(max_depth=5, n_estimators=2)
```

Imagen 11. *Resultado de la validación cruzada.*

En la *Imagen 11* se observa el MAE promedio de cada modelo tanto en el Entrenamiento como en el Test de los 10 splits. Además, se concluye que los dos modelos con el mejor rendimiento fueron el Decisión Tree Regressor y el Random

Forest Regressor. Con estos dos modelos se utilizó el módulo GridSearchCV para encontrar los parámetros que brinden el mejor rendimiento a estos dos modelos.

5.1.4. Encontrar mejores parámetros para los modelos Decision Tree Regressor y Random Forest Regressor.

5.1.4.1. Decision Tree Regressor

Para encontrar los mejores parámetros para el Decision Tree se hieron 5 split iterando entre los parámetros max_depth de 5, 10, 15, 20 y 25, como se muestra en la *Imagen 12*.

```
parametros = {'max_depth': [5,10,15,20,25]}

decission_tree = GridSearchCV(estimator = estimator2,
                              param_grid = parametros,
                              cv = ShuffleSplit(n_splits= 5, test_size=val_size),
                              scoring = 'neg_mean_absolute_error',
                              verbose = 1,
                              return_train_score = True,
                              n_jobs = -1)
decission_tree.fit(Xtv, ytv)

print("\nMejor estimador Decision Tree: ",decission_tree.best_estimator_)
print("Mejores parámetros para el estimador Decision Tree: ", decission_tree.best_params_)

Des_tree = decission_tree.best_estimator_
Des_tree.fit(Xtv, ytv)

print('\nMAE del Decision Tree en entrenamiento: ',"{:.5f}".format(mean_absolute_error(ytv, Des_tree.predict(Xtv))))
print('MAE del Decision Tree seleccionado: ',"{:.5f}".format(mean_absolute_error(yts , Des_tree.predict(Xts))))
```

Imagen 12. *Código para encontrar mejores parámetros.*

El resultado del código presentado en la *Imagen 12* se puede observar en la *Imagen 13*.

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

Mejor estimador Decision Tree:  DecisionTreeRegressor(max_depth=10)
Mejores parámetros para el estimador Decision Tree:  {'max_depth': 10}

MAE del Decision Tree en entrenamiento:  0.07188
MAE del Decision Tree seleccionado:  0.07467
```

Imagen 13. *Resultado de mejores parámetros para Decission Tree.*

De la *Imagen 13* se concluye que el mejor parámetro encontrado para el Decision Tree es un max_depth = 10, dándonos un MAE promedio de 0.07188 en entrenamiento y 0.07467 en el Test.

5.1.4.2. Random Forest Regressor.

Para encontrar los mejores parámetros para el Random Forest Regressor se hiceron 5 split iterando entre los parámetros, tanto para max_depth como para $n\_estimators$, de $5, 10, 15, 20$ y 25; como se muestra en la *Imagen 14.*

```
parametros = { 'n_estimators': [5,10,15,20,25],
               'max_depth':[5,10,15,20,25]}

forest_reg = GridSearchCV(estimator = estimator3,
                          param_grid = parametros,
                          cv = ShuffleSplit(n_splits= 5, test_size=val_size),
                          scoring = 'neg_mean_absolute_error',
                          verbose = 1,
                          return_train_score = True,
                          n_jobs = -1)
forest_reg.fit(Xtv, ytv)

print("\nMejor estimador Random Forest: ",forest_reg.best_estimator_)
print("Mejores parámetros para el estimador Random Forest: ", forest_reg.best_params_)

Rdm_forest = forest_reg.best_estimator_
Rdm_forest.fit(Xtv, ytv)

print('\nMAE del Random Forest en entrenamiento: ',"{:.5f}".format(mean_absolute_error(ytv, (Rdm_forest.predict(Xtv)))))
print('MAE del Random Forest seleccionado: ',"{:.5f}".format(mean_absolute_error(yts , Rdm_forest.predict(Xts))))
```

Imagen 14. *Código para encontrar mejores parámetros.*

El resultado del código presentado en la *Imagen 14* se puede observar en la *Imagen 15.*

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits

Mejor estimador Random Forest:  RandomForestRegressor(max_depth=15, n_estimators=25)
Mejores parámetros para el estimador Random Forest:  {'max_depth': 15, 'n_estimators': 25}

MAE del Random Forest en entrenamiento:  0.04873
MAE del Random Forest seleccionado:  0.06452
```

Imagen 15. *Resultado de mejores parámetros para Random Forest.*

De la *Imagen 15* se concluye que los mejores parámetros encontrados para el Random Forest son un $max\_depth = 15$ y un $n\_estimator = 25$, dándonos un MAE promedio de 0.04873 en entrenamiento y 0.06452 en el Test.

5.2.Modelo no supervisados

5.2.1. PCA con Random Forest

Iterando entre un número de componentes de 5,10,15, y 20 para el PCA, se obtuvieron datasets reducidos en cada iteración, para luego correr nuestro modelo Random Forest seleccionado anteriormente con cada uno de estos Dataset y comparar el rendimiento para escoger el mejor PCA.

```
from sklearn.decomposition import PCA
components = [5,10,15,20]
test_size = 0.3
val_size  = test_size/(1-test_size)
perf = []
Rdm_forest = RandomForestRegressor(n_estimators = 20,max_depth = 15)
for i in components:
    pca = PCA(n_components = i)
    X_t = pca.fit_transform(X2)

    Xtv, Xts, ytv, yts = train_test_split(X_t, y2, test_size=test_size)
    print (Xtv.shape, Xts.shape)

    Rdm_forest.fit(Xtv, ytv)
    perf.append(mean_absolute_error(yts , Rdm_forest.predict(Xts)))
    print('MAE del modelo con ', i ,'elementos: ',"{:.5f}".format(mean_absolute_error(yts , Rdm_forest.predict(Xts))))
    print('------------')

print('Mejor MAE: ',"{:.5f}".format(np.min(perf)),'  ; obtenido con ',components[np.argmin(perf)],' componentes para PCA')
```

Imagen 16. *Código de iteración para componentes de PCA.*

Del código de la *Imagen 16* se obtuvo que el mejor PCA con nuestro Random Forest es el de 20 componentes con un MAE de 0.07612.

.

```
(140000, 5) (60000, 5)
MAE del modelo con  5 elementos:  0.11355
------------
(140000, 10) (60000, 10)
MAE del modelo con  10 elementos:  0.10205
------------
(140000, 15) (60000, 15)
MAE del modelo con  15 elementos:  0.08035
------------
(140000, 20) (60000, 20)
MAE del modelo con  20 elementos:  0.07612
------------
Mejor MAE:  0.07612   ; obtenido con  20  componentes para PCA
```

Imagen 17. *Resultado de mejor número de componentes para PCA.*

Con el mejor PCA seleccionad, se corre este PCA con 20 componentes, para luego encontrar los mejores parámetros para el Random Forest con el Dataset resultante del PCA. La búsqueda de estos parámetros se lleva a cabo de la misma manera que cuando se hizo en los modelos supervisados y se encontró que los mejores parámetros fueron max_depth = 20 y n_estimators = 25, obteniendo así un MAE de 0.03377 en entrenamiento y 0.07501 en Test.

5.2.2. NMF con Random Forest

De la misma forma que con el PCA, se encontró que el mejor parámetro para este modelo es de 20 componentes y los mejores parámetros para el Random Forest con la matriz resultantes del NMF con 20 componentes son max-depth = 15 y n_estimators = 25, obteniendo así un MAE de 0.06748 en entrenamiento y 0.07862 en Test.

6. Curvas de aprendizaje

6.1. Modelos supervisados

6.1.1. DecissionTreeRegressor(max_depth=10)

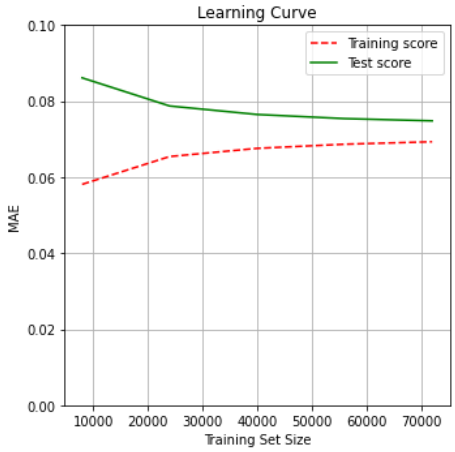

Gráfico 14. *Curva de aprendizaje para Decission Tree.*

En el *Gráfico 14* se evidencia que a medida que se le dan más datos al Decisión Tree Regressor el erro en test vas disminuyendo hasta parecer estabilizarse con el de entrenamiento, pero ambos MAE son malos.

6.1.2. RandomForestRegressor(n_estimators = 25,max_depth = 15)

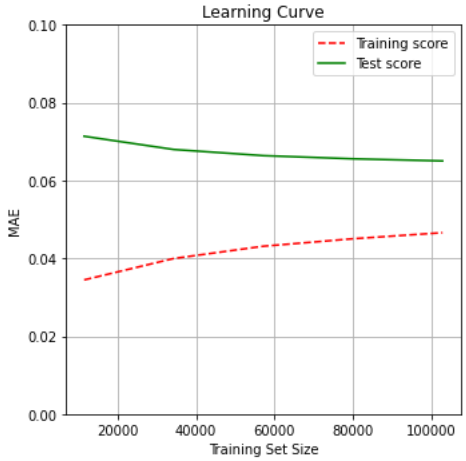

Gráfico 15. *Curva de aprendizaje para Random Forest.*

En el *Gráfico 15* se evidencia un posible caso de overfitting, el MAE en entrenamiento empieza muy bien y el MAE en test empieza mal y mejora un poco, pero ninguno de los dos tiene un cambio drástico. Es posible que el modelo sea muy complejo o que se requieran más datos.

6.2.Modelos no supervisados + Random Forest

6.2.1. PCA con Random Forest

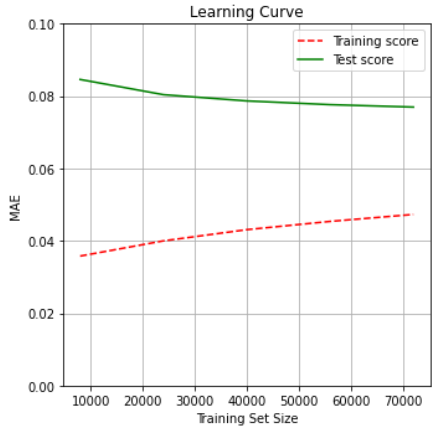

Gráfico 16. *Curva de aprendizaje para PCA con Random Forest.*

En el *Gráfico 16* se evidencia un posible caso de overfitting, el MAE en entrenamiento empieza muy bien y el MAE en test empieza mal y mejora un poco, pero ninguno de los dos tiene un cambio drástico. Es posible que el modelo sea muy complejo o que se requieran más datos.

6.2.2. NMF con Random Forest

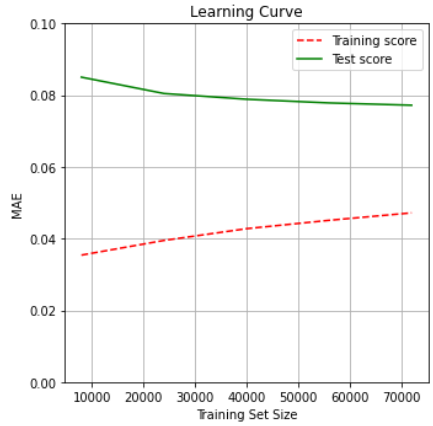

Gráfico 17. *Curva de aprendizaje para NMF con Random Forest.*

En el *Gráfico 17* también se evidencia un posible caso de overfitting, el MAE en entrenamiento empieza muy bien y el MAE en test empieza mal y mejora un poco, pero ninguno de los dos tiene un cambio drástico. Es posible que el modelo sea muy complejo o que se requieran más datos.

7. Retos y condiciones de despliegue

Como reto se tiene que las estadísticas de las partidas deben obtenerse al tiempo que estas se llevan a cabo para que la estrategia de los jugadores vaya adaptándose de acuerdo esa

posición final predicha, es decir, si sus estadísticas actuales van dando predicciones de últimas posiciones, es necesario cambiar la estrategia de juego.

En cuanto al despliegue del modelo, es necesario obtener constantemente todas las estadísticas durante las partidas para ir generando las predicciones y que la estrategia de juego se vaya adaptando de acuerdo con estas predicciones. También es necesario evaluar el desempeño del modelo constantemente y entrenarlo en caso de que el desempeño deje de cumplir con lo esperado.

8. Conclusiones

- Hay que buscar reducir el error, ya sea analizando la importancia de las variables en los modelos seleccionados o agregando variables que brinden más información a los modelos.
- El modelo Random Forest genera curvas de aprendizaje que presentan Overfitting.
- Es necesario encontrar mejores parámetros y con menos complejidad para el modelo Random Forest.
- Se requiere encontrar la manera de reducir el overfitting, reduciendo la complejidad del modelo o haciendo iteraciones con más datos
- Los modelos no supervisados con el Random Forest se compartan igual debido a que el Random Forest utilizado es muy complejo.

9. Bibliografía

- PUBG-finish-placement-prediction | Kaggle. (2022). Retrieved 11 November 2022, from https://www.kaggle.com/competitions/pubg-finish-placement-prediction
- https://rramosp.github.io/ai4eng.v1/content/M00_intro_udea.html