Mateusz monikowski
mateuszmonikowski@gmail.com

esportsLABgg Counter-Strike Data Challenge

Data analysis

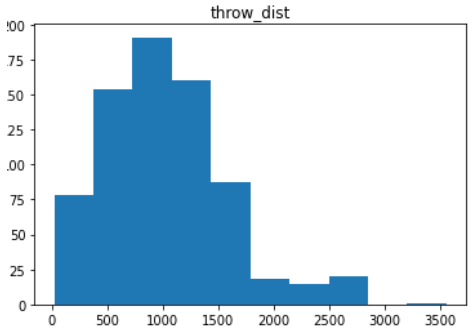
The solution is a python project, with it's virtual environment and classify.py implemented as specified.

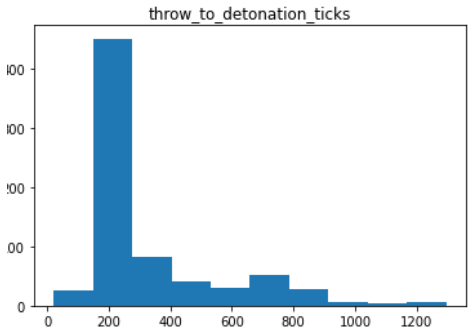
Exploratory data analysis has been performed and documented in two jupyter notebooks, 'Grenade Data Analysis' and 'ML Model with coords' (in that order).

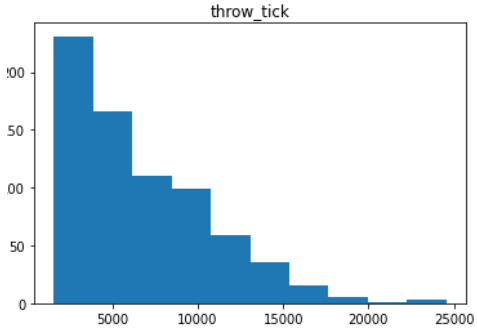
In the first notebook, I performed statistical analysis of the data; explored the correlations, distributions, biases. I extracted some new features – 'throw_dist' and 'throw_to_detonation_ticks', with the latter eventually being rejected – due to strong correlation with 'throw_dist', and very biased distribution.

```
print(grenades_num.loc[grenades["TYPE"] != "flashbang"].corr())
```

	throw_dist	throw_to_detonation_ticks	throw_tick
throw_dist	1.000000	0.754852	-0.130857
throw_to_detonation_ticks	0.754852	1.000000	-0.087012
throw_tick	-0.130857	-0.087012	1.000000







Mateusz monikowski
mateuszmonikowski@gmail.com

The categorical variables were changed into one-hot columns, so that the statistical and neural network models could make use of them.

```
dummies = pd.get_dummies(grenades[['team', 'TYPE', 'map_name']])
dummies
```

	team_CT	team_T	TYPE_flashbang	TYPE_molotov	TYPE_smoke	map_name_de_inferno	map_name_de_mirage
0	1	0	1	0	0	1	0
1	0	1	0	0	1	1	0
2	0	1	0	0	1	1	0
3	0	1	1	0	0	1	0
4	0	1	1	0	0	1	0
...	...	...	...	...	...	...	...
719	0	1	0	0	1	0	1
720	1	0	1	0	0	0	1
721	1	0	1	0	0	0	1
722	1	0	0	0	1	0	1
723	1	0	0	0	1	0	1

724 rows × 7 columns

Eventually, after rejecting seemingly useless coordinates and detonation tick and performing standardization on numerical variables…

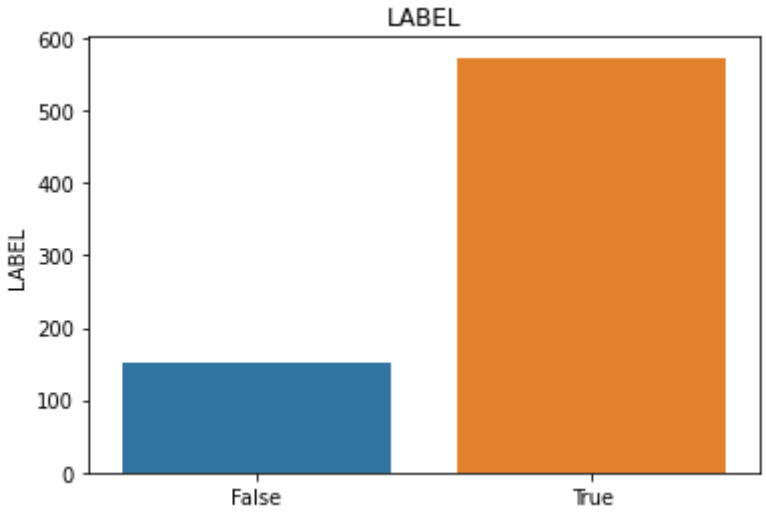
	team_CT	team_T	TYPE_flashbang	TYPE_molotov	TYPE_smoke	map_name_de_inferno	map_name_de_mirage	throw_dist	throw_tick_norm
0	1	0	1	0	0	1	0	0.089648	1.472557
1	0	1	0	0	1	1	0	-0.113771	1.145835
2	0	1	0	0	1	1	0	1.889142	-0.997096
3	0	1	1	0	0	1	0	-0.166697	1.043068
4	0	1	1	0	0	1	0	0.002041	-1.179434
...	...	...	...	...	...	...	...	...	...
719	0	1	0	0	1	0	1	-0.944650	1.595435
720	1	0	1	0	0	0	1	0.008079	0.852346
721	1	0	1	0	0	0	1	0.129028	1.205619
722	1	0	0	0	1	0	1	-0.880028	-1.085291
723	1	0	0	0	1	0	1	2.906014	0.021665

724 rows × 9 columns

This was the input for several statistical and eventually neural network model. The results, however, were not great at all.

Mateusz monikowski
mateuszmonikowski@gmail.com

The train data is greatly inbalanced, which made both statistical models and neural network greatly biased towards only giving 0s or 1s. You can see the distribution of invalid/valid grenades below.



What can be done in this situation is limiting the greater class to be about equal to the smaller class in the training data. And this was done:

```
X_a_Y_t = X_and_Y.loc[X_and_Y["LABEL"] == 1].sample(frac=0.28)
Y_t = X_a_Y_t["LABEL"]
X_t = X_a_Y_t.drop("LABEL", axis=1)

X_a_Y_f = X_and_Y.loc[X_and_Y["LABEL"] == 0].sample(frac=0.9)
Y_f = X_a_Y_f["LABEL"]
X_f = X_a_Y_f.drop("LABEL", axis=1)

X_even = X_t.append(X_f)
Y_even = Y_t.append(Y_f)
```

28% of valid and 90% of invalid grenades were used to train the models. The results were not any better, though – only neural network saw any improvement. Below you can see some of the results different models produced.

Mateusz monikowski
mateuszmonikowski@gmail.com

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter = 2000)
cv = cross_val_score(lr,X_scaled,Y,cv=5)
print(cv)
print(cv.mean())
```

```
[0.79310345 0.79310345 0.79310345 0.7862069  0.79166667]
0.7914367816091954
```

Seems good, right? Well…

```
lr.predict(X_scaled)
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
      dtype=int64)
```

This was the case with all the other models – everything was either 0 or 1

Mateusz monikowski
mateuszmonikowski@gmail.com

```
from sklearn import tree
dt = tree.DecisionTreeClassifier(random_state = 1)
cv = cross_val_score(dt,X_even,Y_even,cv=5)
print(cv)
print(cv.mean())
```

```
[0.51666667 0.50847458 0.55932203 0.52542373 0.6779661 ]
0.5575706214689264
```

```
dt.fit(X_even, Y_even)
confusion_matrix(Y, dt.predict(X))
```

```
array([[151,   0],
       [573,   0]], dtype=int64)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
cv = cross_val_score(knn,X,Y,cv=5)
print(cv)
print(cv.mean())
```

```
[0.74482759 0.75172414 0.77931034 0.74482759 0.77083333]
0.7583045977011496
```

```
knn.fit(X_even, Y_even)
confusion_matrix(Y, knn.predict(X))
```

```
array([[151,   0],
       [573,   0]], dtype=int64)
```

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state = 1)
cv = cross_val_score(rf,X,Y,cv=5)
print(cv)
print(cv.mean())
```

```
[0.69655172 0.72413793 0.71724138 0.75172414 0.77083333]
0.7320977011494253
```

```
rf.fit(X_even, Y_even)
confusion_matrix(Y, rf.predict(X))
```

```
array([[151,   0],
       [573,   0]], dtype=int64)
```

Mateusz monikowski
mateuszmonikowski@gmail.com

Neural networks, however, saw some improvement when the training data was balanced. Before, it was predicting everything as 1. After giving it balanced data…

```
from sklearn.metrics import confusion_matrix
confusion_matrix(Y, estimator.predict(X_scaled))
```

```
array([[ 81,  70],
       [165, 408]], dtype=int64)
```

```
confusion_matrix(Y_even, estimator.predict(X_even))
```

```
array([[74, 47],
       [16, 99]], dtype=int64)
```

```
confusion_matrix(Y, estimator.predict(X_scaled)) - confusion_matrix(Y_even, estimator.predict(X_even))
```

```
array([[  7,  23],
       [149, 309]], dtype=int64)
```

First are the predictions on the entire dataset, then on the training (X_even, Y_even) dataset, and then on entire – training. There are some hits on both 1s and 0s now, the model is clearly trying. Definitely though, it did not have enough data – just as the statistical models below. That made me think.

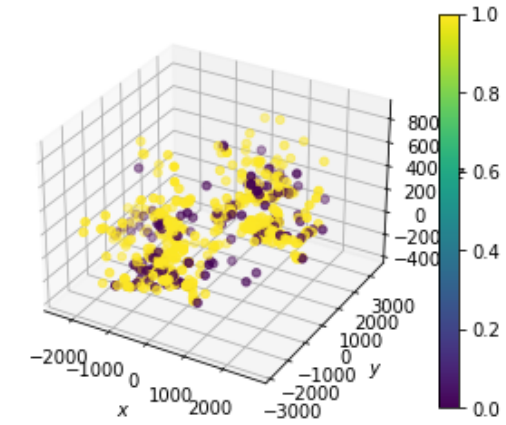
Counter Strike is a tactical shooter with intricate tactics, and these tactics often include throwing grenades in specific places on the map, to block passageways, vision, or blind camping enemies. This led me to a hypothesis, that the detonation coordinates might actually have a great effect on the data, and so I tested it.

Mateusz monikowski
mateuszmonikowski@gmail.com

Now, moving on to “ML Model with coords” notebook:

The first thing I’ve done was plotting the detonation coordinates along with information, whether the grenade was valid or not.

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
cax = ax.scatter(grenades['detonation_raw_x'], grenades['detonation_raw_y'],
                 grenades['detonation_raw_z'], c=grenades['LABEL'])
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$z$')
fig.colorbar(cax)
plt.show()
```



The clusters of yellow (valid) grenades are clearly visible, with purple (invalid) having sort of a more random spread. And so this time I left the coordinates in.

	team_CT	team_T	TYPE_flashbang	TYPE_molotov	TYPE_smoke	map_name_de_inferno	map_name_de_mirage	detonation_raw_x	detonation_raw_y	detonation_raw_z	throw_dist	throw_tick_norm
0	1	0	1	0	0	1	0	448.495	3380.300	222.4680	1078.051779	9.498897
1	0	1	0	0	1	1	0	778.686	2274.620	137.7930	963.940743	9.309733
2	0	1	0	0	1	1	0	1447.970	896.635	144.4740	2087.509684	8.069029
3	0	1	1	0	0	1	0	2717.180	318.481	552.7410	934.250694	9.250234
4	0	1	1	0	0	1	0	313.638	2291.220	396.4490	1028.907444	7.963460
...	...	...	...	...	...	...	...	...	...	...	...	...
719	0	1	0	0	1	0	1	-1462.400	728.349	-45.9688	497.844023	9.570041
720	1	0	1	0	0	0	1	-756.819	-2198.980	-159.1930	1032.294524	9.139811
721	1	0	1	0	0	0	1	-696.054	-1541.940	59.4860	1100.143034	9.344347
722	1	0	0	0	1	0	1	529.241	-1621.910	-262.0000	534.094957	8.017967
723	1	0	0	0	1	0	1	-1151.840	-628.823	-165.9690	2657.942144	8.658866

724 rows × 12 columns

Mateusz monikowski
mateuszmonikowski@gmail.com

The coordinates had normal distributions, so they could be standardized right away. Once again, I split the data so that I had about equal distribution of classes in the train set

```
X_a_Y_t = X_and_Y.loc[X_and_Y["LABEL"] == 1].sample(frac=0.28)
Y_t = X_a_Y_t["LABEL"]
X_t = X_a_Y_t.drop("LABEL", axis=1)

X_a_Y_f = X_and_Y.loc[X_and_Y["LABEL"] == 0].sample(frac=0.9)
Y_f = X_a_Y_f["LABEL"]
X_f = X_a_Y_f.drop("LABEL", axis=1)

X_even = X_t.append(X_f)
Y_even = Y_t.append(Y_f)
```

And went on to checking out different models (this time with grid optimization) .

This was the best result:

```
knn = KNeighborsClassifier()
param_grid = {'n_neighbors' : [3,5,7,9,11],
              'weights' : ['uniform', 'distance'],
              'algorithm' : ['auto', 'ball_tree','kd_tree'],
              'p' : [1,2,3]}
clf_knn = GridSearchCV(knn, param_grid = param_grid, cv = 5, verbose = True, n_jobs = -1)
best_clf_knn = clf_knn.fit(X_even,Y_even)
clf_performance(best_clf_knn,'KNN')
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
KNN
Best Score: 0.5640677966101695
Best Parameters: {'algorithm': 'auto', 'n_neighbors': 7, 'p': 1, 'weights': 'distance'}
```

```
confusion_matrix(Y, clf_knn.predict(X_scaled)) - confusion_matrix(Y_even, clf_knn.predict(X_even))
```

```
array([[ 10,   5],
       [145, 268]], dtype=int64)
```

Up here, you can see around 65% accuracy, which with how little data there is to work with, was very impressive for me (especially that it's OUTSIDE of test data, so there is no way it comes from overfit!)

Mateusz monikowski
mateuszmonikowski@gmail.com

A neural network allowed for slight improvement, though. Using the same train set, These were the results (on unseen data, of course):

```
array([[  9,   6],
       [128, 285]], dtype=int64)
```

60% accuracy on invalid grenades, 69% accuracy on valid grenades, 68,6% accuracy total. This model was saved and later used in classify.py script.

The manouver of limiting train data to make classes equal paid off, however results were still far from perfect. Using the entire train set would make no sense, since the best thing achieved could only be overfit or bias towards the more frequent class. I wanted to create a more universal model, even if it meant that I had to sacrifice accuracy (especially, since my model will be tested against data from train set…)

Mateusz monikowski
mateuszmonikowski@gmail.com

classify.py

In classify.py, I perform onehot encoding and standardization (with a scaler fit to entire train data, to preserve proper distribution) to achieve the same data format as I had while training the models. Then, I load the saved model from file, make the predictions, and append them to the csv file.