Andrea Morelli 1845525
First Homework Machine Learning
Multi Uav Conflict Predictions

Description of the homework

The homework assigned has as objective a classification problem and a regression problem. We are in an environment with 5 uavs and giiven the values of different sensors and other features we have to estimate in the classification problem the total number of conflicts between uavs and in the regression problem the minimum closest point of approach among all the possible pairs of uavs.

Dataset

The dataset is in the tsv format and is composed by the values of different sensors.
We have as input for each of the 5 uavs :
— the clockwise angle from north between the ith uav and its target (this value can assume values between o and 2 pi)
— the position components of the ith uavs in meteres
— the speed components of the ith uav in meters / seconds
— the position components of the ith uav in target in meters

Data preprocessing

I tried a lot of different thing in order to preprocess normalize and use the dataset.
I noticed firstly that the clockwise angle could be normalized dividing it by 2 times pi in order to have values between 0 and 1 , but another approach could be applying the cosine function to the angle. This would allow to have smoother changes when switching between 0 and 2 pi in angles and the resulting values between -1 and 1 could be normalized to 0 and 1.
For all the other features a normalization with the min max normalizer or the standard normalization could be applied.

Instead of using the cosine directly on the angle i preferred to add new features to the dataset.I added the cosine of the angle as a new feature and the distance between the target and the uavs for every uav both on the x and y direction.
Adding those new columns would not help

On those new dataset created i would perform different normalizations

I tried the MinMaxScaler $x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$ with different features ranges for example 0 - 1 or -1 - 1.

I tried the StandardScaler $z = \frac{x - \mu}{\sigma}$ $\mu = \text{Mean}$ $\sigma = \text{Standard Deviation}$ and this would normalize all the input values to have mean 0 and variance 1.

I tried the pca and the svd decomposition to get new features but found that they were not helpful

All those normalizations could be optimized with a grid search and a pipeline in order to see which performs better with the different models i did it for the support vector machine classifier.
I found that the standard scaler and the minmax scaler were performing better than the other methods in the classification task but also found that without normalization the performances of the different models were not that impacted.

Imbalanced dataset in classification
The problem of this dataset is that it's highly unbalanced and the elements of the latest class do not have many instances.
That was a major problem when dealing with the classification.
I tried to use the random upsampler from the imblearn package, the smote method to upsample the minority classes and the method random downsample to try to deal with the majority class.

I tried different combinations of upsample lower classes and downsample most popular classes but all would perform poorly.
Without the upsampling the performances of the models were good just on the majority class so i would receive a good overall f1 micro score but all the other classes would not have good results so f1 macro would be bad but the weighted f1 would be better.

Sampling the training set and using it for the training of the models i would see not good results but the performances on the other classes were better (the f1 macro would be better but the f1 micro would decrease a lot).

Sampling before the split of the dataset would perform a lot better and the results would be optimal but would be like cheating: the distribution of the test set would be modified and the results would not be reflecting the real world so would not be able to generalize even if the results were a lot better.

Imbalance dataset in regression

In the regression i found out that normalizing the dataset or not the performances would be pretty close but upsampling or downsampling would even worse than the results obtained.

Evaluation procedure classification

The dataset is splitted using the sklearn function in train and test set.
In the grid search i always used the k fold cross validation in the train set so i would keep the test set unwatched and the train set would be partitioned with k fold cross validation in train and validation set.

The metrics that were used or the classification were

$$Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

$$Precision = \frac{T_p}{T_p + F_p}$$

$$Recall = \frac{T_p}{T_p + T_n}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

the accuracy allows to see the number of correct predictions with respect to the total number of instances.

the precision allows to see how resistant the predictions are to the false positives.

the recall score allows to see how resistant the predictions are to the true negatives.

The f1 score is a combination of the above scores.
The f1 score has different version that have been used:

–the micro score calculates the f1 score on every predictions without taking in consideration the different classes (in this case it's the same of the accuracy)

– the macro score calculates the f1 score on every classes and then takes the average of those scores

–the average score calculates the f1 score on every classes and weights everything with the number of elements for each class.

evaluation procedure for the regression task

The metrics that were used for the regression task were:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

Where,

$\hat{y}$ – *predicted value of y*

$\bar{y}$ – *mean value of y*

– the mean absolute error sees how much the predictions differ from the true value in absolute value in average

– the mean squared error sees how much the predictions differ from the true values squared so its more sensible when we differ a lot from the true predictions.

–the root mean squared error is the root of the mean squared error

– the r2 score is a value with maximum of 1 and represents how good the predictions are

different evaluations

I divided the dataset in train and test set and i evaluated the performances on both of them. The train set has been used to train the models and would perform better than the test set because all the elements have been seen during the training phase.
The train set is divided in train and evaluation set when the hyperparameters are optimized with the grid search or the random hyperparameters search.

methods and algorithms used

I used different models both for the classification task and for the regression task.
Classification:
The models that i tried for the classification were:

– decision tree
builds a tree like model that allows to classify every input following the branches of the tree until the leaves

– random forest
builds a collection of decision trees and uses them to classify an input instance

– adaboost
uses boosting to classify inputs giving more importance on inputs misclassified

– logistic regression
uses the logistic regression to classify input instances

–support vector machine
finds the best hyperplane to divide classes

– k neighbors
classifies the inputs finding the classes of the closest inputs

–gaussian naive bayes
assumes the inputs can be described by gaussians distributions and finds the best gaussians to approximate the inputs and uses them to classify new instances

–bagging
uses baggings to aggregate different predictions and output a class

– balanced bagging
bagging with random undersampling and oversampling with decision trees

–balanced random forest
random forest with random undersample and oversample

– dummy classifier
simple classifier that simply returns the most frequent class

Regression:
the models that i tried for the regression were:

– random forest
uses collection of decision trees in order to find the values of the min cpa

– adaboost
uses boosting to give more importance to inputs that have a score really far from the true value of min cpa

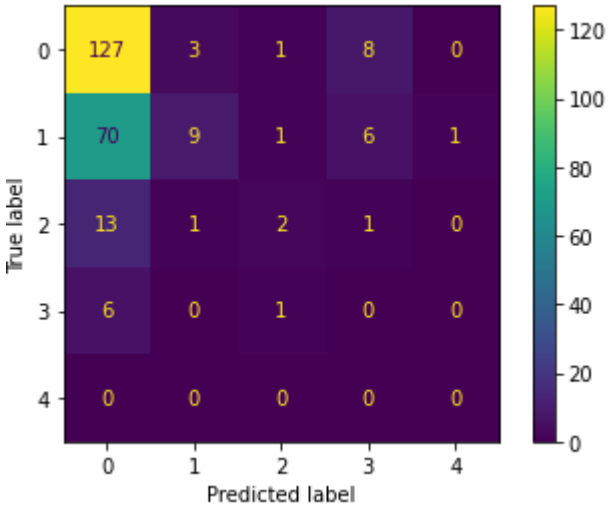
– linear regression
finds the best linear model to find the closest value to the min cpa

– support vector machine
uses svm regression with slack variables

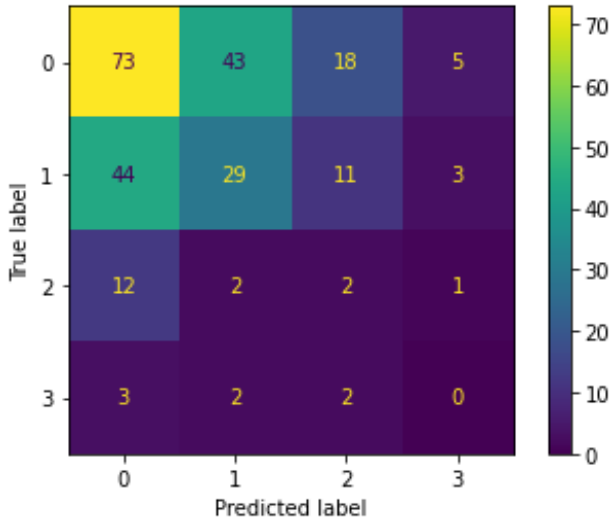
– k neighbors
returns the closest or an average min cpa of the neighbors

–decision tree
builds a tree with ranges of values and does something similar to a classifier to return a range where the regressed values is

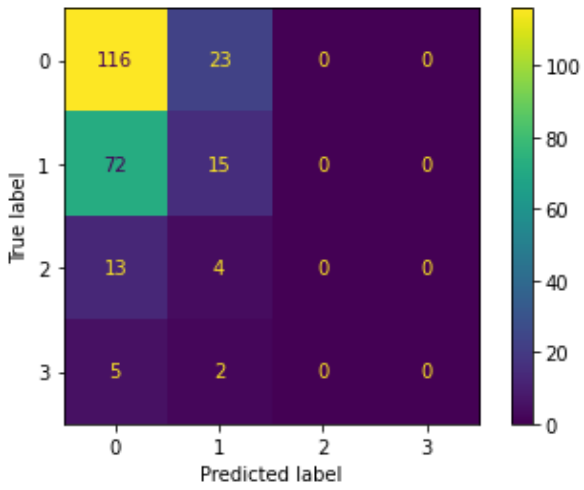
configurations tried and best results
Classification:
1)using the new features added(cosine of the angle and the distances from the target) ,normalizing the dataset without sampling we obtain the following results




```
the model is AdaBoostClassifier()
train report
              precision    recall  f1-score   support

         0.0       0.54      0.89      0.67       399
         1.0       0.37      0.05      0.09       246
         2.0       0.21      0.06      0.10        79
         3.0       0.11      0.17      0.14        23
         4.0       1.00      0.67      0.80         3

    accuracy                           0.51       750
   macro avg       0.45      0.37      0.36       750
weighted avg       0.44      0.51      0.41       750

test report
              precision    recall  f1-score   support

         0.0       0.59      0.91      0.72       139
         1.0       0.69      0.10      0.18        87
         2.0       0.40      0.12      0.18        17
         3.0       0.00      0.00      0.00         7
         4.0       0.00      0.00      0.00         0

    accuracy                           0.55       250
   macro avg       0.34      0.23      0.22       250
weighted avg       0.60      0.55      0.47       250
```




```
the model is DecisionTreeClassifier()
train report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       399
         1.0       1.00      1.00      1.00       246
         2.0       1.00      1.00      1.00        79
         3.0       1.00      1.00      1.00        23
         4.0       1.00      1.00      1.00         3

    accuracy                           1.00       750
   macro avg       1.00      1.00      1.00       750
weighted avg       1.00      1.00      1.00       750

test report
              precision    recall  f1-score   support

         0.0       0.55      0.53      0.54       139
         1.0       0.38      0.33      0.36        87
         2.0       0.06      0.12      0.08        17
         3.0       0.00      0.00      0.00         7

    accuracy                           0.42       250
   macro avg       0.25      0.24      0.24       250
weighted avg       0.44      0.42      0.43       250
```

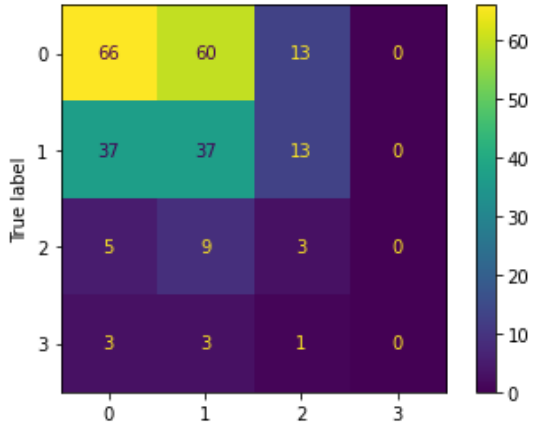



```
the model is RandomForestClassifier()
train report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       399
         1.0       1.00      1.00      1.00       246
         2.0       1.00      1.00      1.00        79
         3.0       1.00      1.00      1.00        23
         4.0       1.00      1.00      1.00         3

    accuracy                           1.00       750
   macro avg       1.00      1.00      1.00       750
weighted avg       1.00      1.00      1.00       750

test report
              precision    recall  f1-score   support

         0.0       0.56      0.83      0.67       139
         1.0       0.34      0.17      0.23        87
         2.0       0.00      0.00      0.00        17
         3.0       0.00      0.00      0.00         7

    accuracy                           0.52       250
   macro avg       0.23      0.25      0.23       250
weighted avg       0.43      0.52      0.45       250
```

We can see that in general the f1 micro is good and also the f1 weighted but the macro is not. Thas because the model try to simply return the first class that is the most populated. We can see also that the random forest and decision tree are overfitting a lot but pruning or building smaller trees or augmenting the number of trees in the forest would not help we would just see worse results both in the training and the testing set(i tried it in grid search)

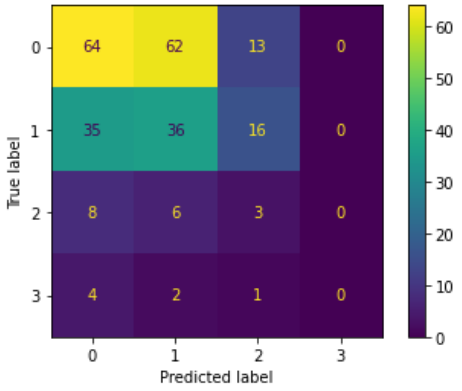
2)Using downsampling and upsampling with both random and smote on the training set and minmax normalization

we balance elements ignoring the last 2 classes and having 200 elements for the first 3 classes using smote but the performances of the previous models were not good at all.Other models would perform in this way:




```
the model is SVC()
train report
              precision    recall  f1-score   support

         0.0       0.82      0.86      0.84       200
         1.0       0.80      0.83      0.82       200
         2.0       0.93      0.96      0.95       200
         3.0       0.00      0.00      0.00        23
         4.0       0.00      0.00      0.00         3

    accuracy                           0.85       626
   macro avg       0.51      0.53      0.52       626
weighted avg       0.81      0.85      0.83       626

test report
              precision    recall  f1-score   support

         0.0       0.59      0.47      0.53       139
         1.0       0.34      0.43      0.38        87
         2.0       0.10      0.18      0.13        17
         3.0       0.00      0.00      0.00         7

    accuracy                           0.42       250
   macro avg       0.26      0.27      0.26       250
weighted avg       0.46      0.42      0.43       250
```

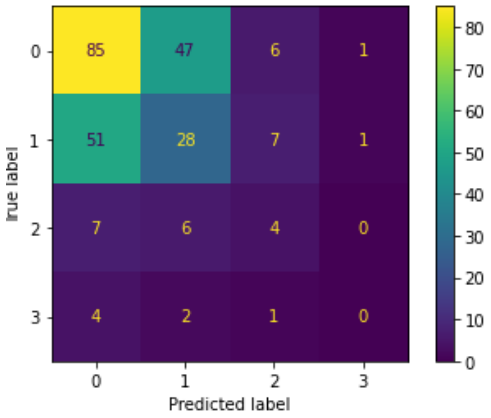
i show also the random forest for example to see the difference




```
the model is RandomForestClassifier()
train report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       200
         1.0       1.00      1.00      1.00       200
         2.0       1.00      1.00      1.00       200
         3.0       1.00      1.00      1.00        23
         4.0       1.00      1.00      1.00         3

    accuracy                           1.00       626
   macro avg       1.00      1.00      1.00       626
weighted avg       1.00      1.00      1.00       626

test report
              precision    recall  f1-score   support

         0.0       0.58      0.46      0.51       139
         1.0       0.34      0.41      0.37        87
         2.0       0.09      0.18      0.12        17
         3.0       0.00      0.00      0.00         7

    accuracy                           0.41       250
   macro avg       0.25      0.26      0.25       250
weighted avg       0.44      0.41      0.42       250
```

we improve the performances on the other classes but the 0 class is not as accurate anymore(the micro is lower)

3) Balancing all the classes instead
we can see an increased f1 macro with the svc



the model is SVC()
train report

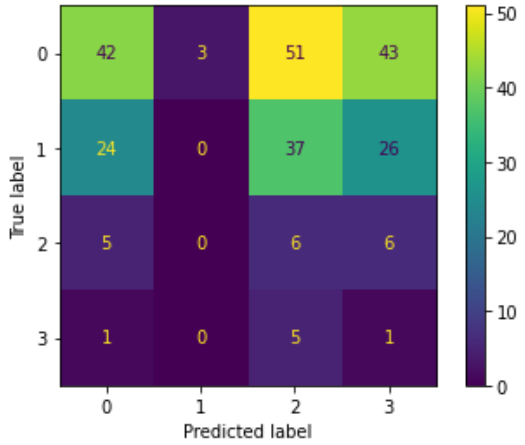
	precision	recall	f1-score	support
0.0	0.89	0.90	0.90	399
1.0	0.91	0.88	0.90	399
2.0	0.97	0.98	0.98	399
3.0	1.00	1.00	1.00	399
4.0	1.00	1.00	1.00	399
accuracy			0.95	1995
macro avg	0.95	0.95	0.95	1995
weighted avg	0.95	0.95	0.95	1995

test report

	precision	recall	f1-score	support
0.0	0.58	0.61	0.59	139
1.0	0.34	0.32	0.33	87
2.0	0.22	0.24	0.23	17
3.0	0.00	0.00	0.00	7
accuracy			0.47	250
macro avg	0.28	0.29	0.29	250
weighted avg	0.45	0.47	0.46	250

Using different normalizations would not improve results

4)Using svd or pca features would worse the performances of the models by a lot for example the adaboost with svd



the model is AdaBoostClassifier()
train report

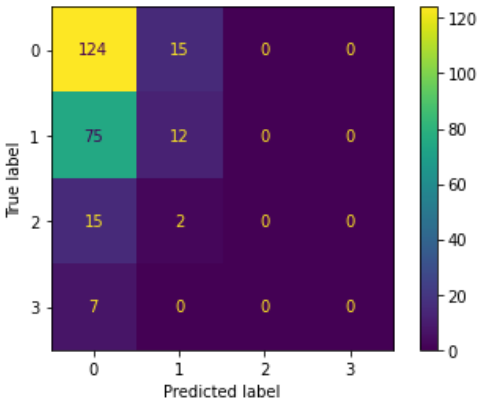
	precision	recall	f1-score	support
0.0	0.58	0.35	0.44	399
1.0	0.36	0.02	0.03	246
2.0	0.13	0.43	0.19	79
3.0	0.03	0.26	0.05	23
4.0	0.00	0.00	0.00	3
accuracy			0.25	750
macro avg	0.22	0.21	0.14	750
weighted avg	0.44	0.25	0.27	750

test report

	precision	recall	f1-score	support
0.0	0.58	0.30	0.40	139
1.0	0.00	0.00	0.00	87
2.0	0.06	0.35	0.10	17
3.0	0.01	0.14	0.02	7
accuracy			0.20	250
macro avg	0.16	0.20	0.13	250
weighted avg	0.33	0.20	0.23	250

adding my new features (the cosine of the angle and the distance between uavs and their target) would not improve the results

5) no normalization or upsampling would obtain worse results
for example the random forest would perform like this




```
the model is RandomForestClassifier()
train report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       399
         1.0       1.00      1.00      1.00       246
         2.0       1.00      1.00      1.00        79
         3.0       1.00      1.00      1.00        23
         4.0       1.00      1.00      1.00         3

    accuracy                           1.00       750
   macro avg       1.00      1.00      1.00       750
weighted avg       1.00      1.00      1.00       750

test report
              precision    recall  f1-score   support

         0.0       0.56      0.89      0.69       139
         1.0       0.41      0.14      0.21        87
         2.0       0.00      0.00      0.00        17
         3.0       0.00      0.00      0.00         7

    accuracy                           0.54       250
   macro avg       0.24      0.26      0.22       250
weighted avg       0.46      0.54      0.46       250
```

some models in general could be improved by reducing their “power” since they are overfitting completely the input but also doing that the performances would approximately be the same

Regression:
for the regression task no matter how many different types of normalizations or hyperparameters for the models i would always obtain not good results
1)Not normalizing

```
the model is RandomForestRegressor()
Train MSE:  17386826.268296175
Train MAE:  3267.947588822419
Train RMSE:  4169.751343700986
Train R2:  0.8580595107035709
Test MSE:  116156752.29178847
Test MAE:  8389.279670519927
Test RMSE:  10777.604199996791
Test R2:  0.03384394256352241
```

```
the model is AdaBoostRegressor()
Train MSE:  107394675.54151371
Train MAE:  9154.652743422314
Train RMSE:  10363.140235542203
Train R2:  0.12326421401072019
Test MSE:  132113322.59078059
Test MAE:  9590.971498314693
Test RMSE:  11494.055967793987
Test R2:  -0.09887789018499937
```

```
the model is LinearRegression()
Train MSE:  117191237.1387065
Train MAE:  8519.849854826964
Train RMSE:  10825.490156972408
Train R2:  0.04328821810031658
Test MSE:  120388728.55763303
Test MAE:  8568.653019172345
Test RMSE:  10972.179754161569
Test R2:  -0.0013563314067939203
```

```
the model is SVR()
Train MSE:  132969623.69578992
Train MAE:  8389.857952654891
Train RMSE:  11531.245539653984
Train R2:  -0.0855214837774998
Test MSE:  133625502.31268317
Test MAE:  8353.394187329353
Test RMSE:  11559.649748702734
Test R2:  -0.1114557349458336
```

```
the model is Ridge()
Train MSE:  117191238.2921265
Train MAE:  8519.843153356356
Train RMSE:  10825.490210245747
Train R2:  0.04328820868416461
Test MSE:  120387777.00078265
Test MAE:  8568.62753806748
Test RMSE:  10972.13639182373
Test R2:  -0.0013484166502528705
```

```
the model is BayesianRidge()
Train MSE:  121651104.14918318
Train MAE:  8665.13784378053
Train RMSE:  11029.55593617364
Train R2:  0.006879290105311298
Test MSE:  119999153.25279002
Test MAE:  8571.493167323515
Test RMSE:  10954.412501489525
Test R2:  0.0018840358828856596
```

```
the model is DecisionTreeRegressor()
Train MSE:  0.0
Train MAE:  0.0
Train RMSE:  0.0
Train R2:  1.0
Test MSE:  199545854.24598745
Test MAE:  10774.785513298662
Test RMSE:  14126.07002127582
Test R2:  -0.6597609007851577
```

2)Normalizing with minmax

```
the model is RandomForestRegressor()
Train MSE:  16943714.843741722
Train MAE:  3215.163074265454
Train RMSE:  4116.27438878189
Train R2:  0.8616769306653027
Test MSE:  109534309.03111169
Test MAE:  8071.407499384688
Test RMSE:  10465.863988754663
Test R2:  0.08892738407761991
```

```
the model is AdaBoostRegressor()
Train MSE:  92478099.91320793
Train MAE:  8465.733665584541
Train RMSE:  9616.553432140225
Train R2:  0.24503836707565407
Test MSE:  116420020.50799048
Test MAE:  8925.524384689197
Test RMSE:  10789.810957935755
Test R2:  0.03165415869994537
```

```
the model is LinearRegression()
Train MSE:  116858925.9171547
Train MAE:  8522.030283664817
Train RMSE:  10810.130707681323
Train R2:  0.04600110063896401
Test MSE:  119692665.52198151
Test MAE:  8572.995672725041
Test RMSE:  10940.414321312584
Test R2:  0.004433306345499544
```

```
the model is SVR()
Train MSE:  132970076.73546122
Train MAE:  8389.917748000846
Train RMSE:  11531.265183641439
Train R2:  -0.08552518224849304
Test MSE:  133627202.99700406
Test MAE:  8353.376652279707
Test RMSE:  11559.723309707895
Test R2:  -0.11146988071373776
```

```
the model is Ridge()
Train MSE:  116989648.52480944
Train MAE:  8518.22553465681
Train RMSE:  10816.175318697891
Train R2:  0.04493392307554345
Test MSE:  119850622.37801562
Test MAE:  8562.607477242474
Test RMSE:  10947.630902529352
Test R2:  0.00311946991102674
```

```
the model is BayesianRidge()
Train MSE:  121882971.47477654
Train MAE:  8675.301635713766
Train RMSE:  11040.062113719132
Train R2:  0.004986399411016751
Test MSE:  119938642.38386412
Test MAE:  8557.484335414574
Test RMSE:  10951.650212815606
Test R2:  0.002387346636673282
```

```
the model is DecisionTreeRegressor()
Train MSE:  0.0
Train MAE:  0.0
Train RMSE:  0.0
Train R2:  1.0
Test MSE:  211096543.86011377
Test MAE:  10701.482137981699
Test RMSE:  14529.161843000917
Test R2:  -0.755835976216185
```

the results would be slightly better buts still not good at all
and we can see that the models with the exception of the decision tree would underfit the data so more powerful models would have to be used

its obviously not useful to upsample and downsample the dataset to solve the regression task i tried it and obtained really poor results.

i also tried to normalize the min cpa value with minmax scaler but even in this case the performances were more or less the same.

hyperparameters tuning

In order to find models with better results i started to do hyperparameters tuning.
The techniques applied were grid search and randomized search and those would find better hyperparameter for the models.

Grid search would have a systematic approach on finding hyperparameters for the models while the randomized search would try different hyperparameters randomly.

Classification hyperparameters tuning:
I applied different types of searches on different models
Tuning the hyperparameters without sampling would be possible with a simple grid search for example meanwhile tuning hyperparameters when applying sampling would require to use the imblearn package otherwise it will test the model just on the train set upsampled and the results would be biased.

Tuning without sampling:
I tuned different models one of them was the support vector machine classifier:
i tried different normalization methods like the minmax scaler and the standard scaler and different type of kernels (the rbf, the linear, the poly and the sigmoid optimizing the f1_micro.
With standard scaler and sigmoid kernel i would obtain the best results using also k fold cross validation with 5 different partitions on the train set with a f1_micro of 0.532 , applying the same model on the test set would produce an f1_micro of 0.556.
Adding more parameters to tune would make the search a lot more time consuming.

Tuning with sampling:
The tuning when sampling has to be done with the imblearn pipeline .
This allows to not bias our results since we do not test on on the part of the dataset that has been upsampled so the results are not compromised.
The results are tho not really good
I tested the svc with parameters c kernel and class_weight with different values and obtained a max f1_macro score with c equal to 10 classweight to none and kernel rbf a value of 0.23 that is really low but its the correct way of doing hyperparmaeters search.
.
Using randomized search would not improve the performances but would be usually faster to evaluate

I also tried to reduce the power of decision trees and random forest since they were overfitting the dataset , pruning with a ccp_alpah parameter max depth min_ samples and min_samples leaf but ht performances were not improving.
Regression hyperparameters tuning:
In the regression task i tried to do hyperparameters tuning but the performances would not improve
I tried ot do hyperparameter search on the random forest regressor.
This time there wasn't the need to use imblearn and the pipeline since the sampling was not useful so i simply performed a grid search on n_estimators and max_depth .

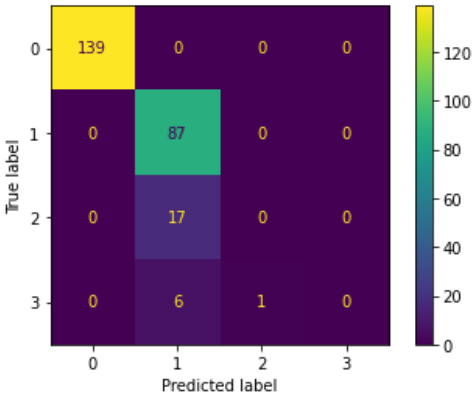
ways to improve

We could use a more time consuming hyperparameters search but i don't think the performances would improve a lot.
We could use different preprocessing for the data but also in this case i don't think the performances would improve since i tried to use pytorch with neural network and saw that also in this case the performances were not improving so the features that we are provided with are ok and could not be improved by non linearities.
I think the main problem of this dataset is that it is too small and trying to balance it or making it bigger with simple sampling would not be good and have valuable information added.
The thing that maybe could be working is the use of models that try to perform regression and classification together for example using neural network in order to have a composed loss and have one task regularize the other and viceversa.
If that doesn't improve the performances i thing the only thing that could work is have a bigger dataset or extensive hyperparameters search.

things found and what could be done

The thing i found out is that as one could expect min cpa and number of collisions to be related so solving one task could allow to solve the other.
So i used the min cpa column in the training set to solve the classification task with incredible results and the viceversa using the number of collisions to solve the regression optimally.

random forest classiication




```
the model is RandomForestClassifier()
train report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       399
         1.0       1.00      1.00      1.00       246
         2.0       1.00      1.00      1.00        79
         3.0       1.00      1.00      1.00        23
         4.0       1.00      1.00      1.00         3

    accuracy                           1.00       750
   macro avg       1.00      1.00      1.00       750
weighted avg       1.00      1.00      1.00       750

test report
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       139
         1.0       0.79      1.00      0.88        87
         2.0       0.00      0.00      0.00        17
         3.0       0.00      0.00      0.00         7

    accuracy                           0.90       250
   macro avg       0.45      0.50      0.47       250
weighted avg       0.83      0.90      0.86       250
```

random forest regression

```
the model is RandomForestRegressor()
Train MSE:  8238487.688841959
Train MAE:  1917.9131545576854
Train RMSE:  2870.276587515907
Train R2:  0.9327436212007755
Test MSE:  61004209.15257081
Test MAE:  5346.546062191267
Test RMSE:  7810.519134639567
Test R2:  0.4925857942909726
```

Obviously this is not something that could be done to solve this homework

If we could use the other column of the y set to solve the tasks i would use it on the training set and then on the test set i would inset the value of the closest element in the train set for example using cosine similarity or the output of another model.

With neural network we could solve the tasks together and using a combined loss to regularize the model (restricts the values the parameters can have so its like we have less parameters so its regularizing) and i think we could obtain better results.