Luca Del Signore 2096442

Machine Learning Homework 1 A.Y. 2022/2023

Multi-UAV conflict risk analysis

1. Abstract

In this first homework we were asked to deal with two different tasks: one task of classification and one task of regression, both using two different methods. The dominion of application is five UAVs flying in a 2-D space. The goal for classification problem is to predict if there will be some collision (represented by 5 classes) knowing the starting point, the velocity and orientation and the arrival point of each UAVs. Instead, the regression problem is based on the prediction of the minimum closest point of approach (CPA) between these 5 UAVs.

In order to use the code it's necessary to load the dataset in the main directory of the drive.

I mainly used the libraries of scikit-learn and some of the code that professor gave to us.

2. Preprocessing Phase

The first part of this homework was to elaborate the data and to find in which way was better to normalize them. I have tried these 4 normalizations both for classification and regression:

1. Min-Max Scaler between 0 and 1
2. Unit Norm
3. Standardization with mean removal and variance scaling
4. Robust scaler (like standardization but stronger for outliers)

However, the variation of the behavior of the model adopted was very small. But the best that was working better was the Min-Max Scaler between 0 and 1, so we will consider that for each solutions and models, this normalization was adopted.

Moreover, not using no normalization at all brings to heavy computational problems, sometimes also blocking and crashing.

I divided the dataset ([X_all,y_all] in two splits: one used for training (X_test,y_test) and one for evaluating the solution obtained (X_train,y_train) using the function train_test_split and heuristically respecting the proportion of the original dataset. In fact, selecting a test size of 33% of all the dataset formed of 1000 samples, we obtained a Test set that respects the original proportion of number of collisions.

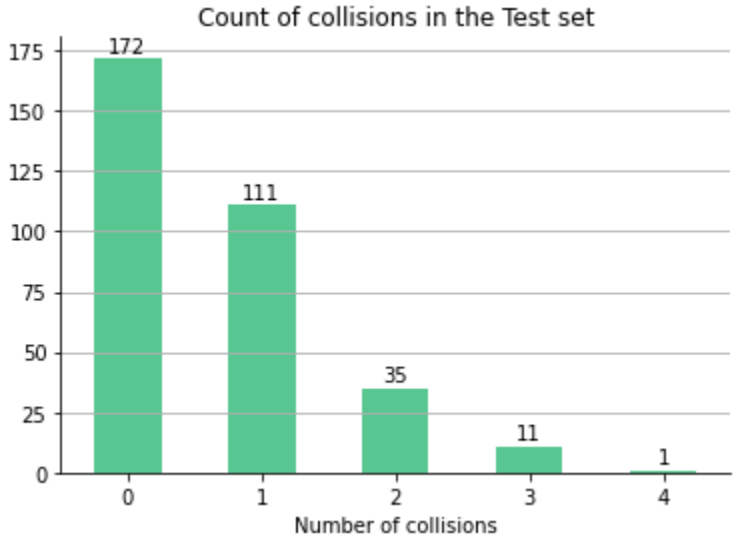


Fig. 2.1

3. Classification task

The two methods that I compared for this task are the Logistic Regression method and the Support Vector Machine for classification (SVC in the library). Considering the feature “num_collisions”, the dataset is unbalanced for this problem and in particular the models have problems in predicting classes that are not in 0 collisions due to the proportion of 53% as 0 collisions and literally few samples for the most important classes (with at least one collision).

For this motivation, I moved from optimizing the accuracy through the Grid_Search function of scikit library to the f1_weighted score. In fact, trying to optimize the accuracy (for each of the 3 models that I tried) we obtain a solution that classifies every vector in class 0 (that has 0 collision) with an accuracy of 53/54% following this normalized confusion matrix:

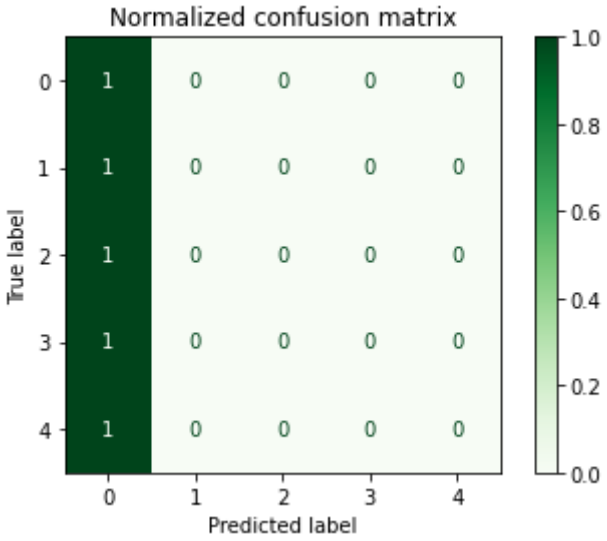


Fig. 3.1

This is reasonable because having a dataset that presents proportionally a 53% of vectors in 0 collision, in order to have the best accuracy on the Test set that follows the same proportion is like that having a constant solution h(x)=class_0. But logically this solution has no sense on real data and on the real field, and moreover the most important class to predict are not identified! So, trying to optimize f1_weighted metric and increasing weight for the most important classes, I obtained some solutions that were more realistic in approximating the problem.

3.1 SVC

I found that nearby of the 80/90% of the vectors belonging to the Train test were used by the method as support vectors. At first I thought that this problem was caused by the normalization between 0 and 1 that flattens the data and then they are no useful for margin constraints of SVC , but also varying the type of normalization and the type of the kernel it seems to very little. So, I started to think that the data are not linearly separable.

Following this article: https://maurygreen.medium.com/how-to-check-for-linear-separability-13c177ae5a6e which says that to see if some data are linearly separable it is possible to train a SVM with all the dataset using a linear kernel and configuring the hyperparameter C to a very high number (for example 2^32). Doing this, if we reach overfit over the dataset with an accuracy (measured on the same dataset used for the train part) of 100%, then the dataset is linearly separable. But what I

obtained is an accuracy of 36,1%. So, according to this article, it should be not linearly separable.

Starting fitting a SVC with a polynomial kernel of third-degree, I found that, as expecting, increasing the Hyperparameter C also the accuracy on the training set is increasing causing overfitting. For C=>2.2 the accuracy is 100%on Training set, instead the accuracy on the Test set seems to remain constant varying C (nearby 46/47%). The Normalized confusion matrix is like that:

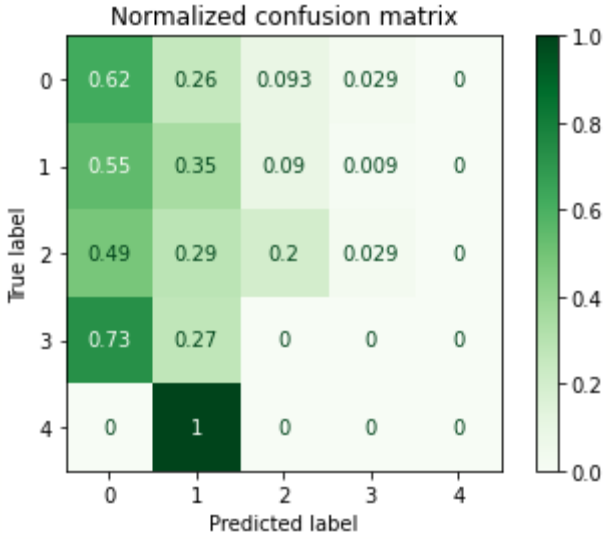


Fig. 3.2

If we use C= 0.01 we obtain the minimum accuracy over Train set that is 54%, but the best accuracy on the Test set of 52,1% that follow the confusion matrix of **Fig. 3.1** and then, as already discussed, it's not an interesting result.

Using GridSearch to optimize the accuracy in order to identify what kind of kernel is better and to tune other Hyperparameters like C and degree of the polynomial are better, we can find that we return to the situation of **Fig. 3.1**. So I evaluated Grid_Search for optimizing the f1_weighted. The best that we can obtain is using these parameters:

```
Best classification hyper-parameters: {'C': 1.9100000000000001, 'degree': 1,
'kernel': 'rbf'}
Cv=2
Best f1_weighted: 0.45
```

That brings to this confusion matrix:

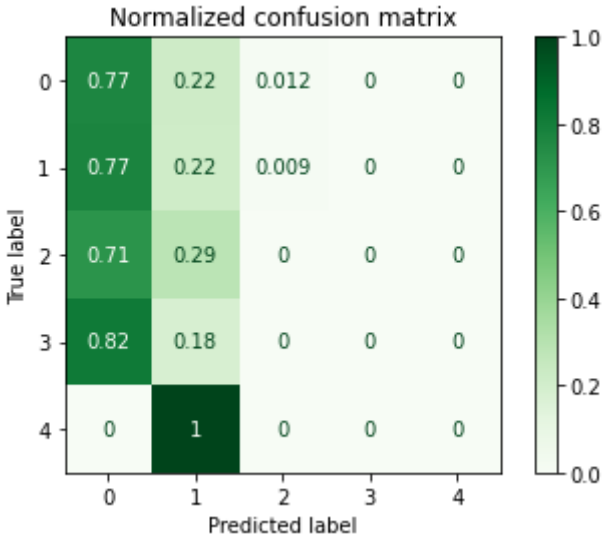
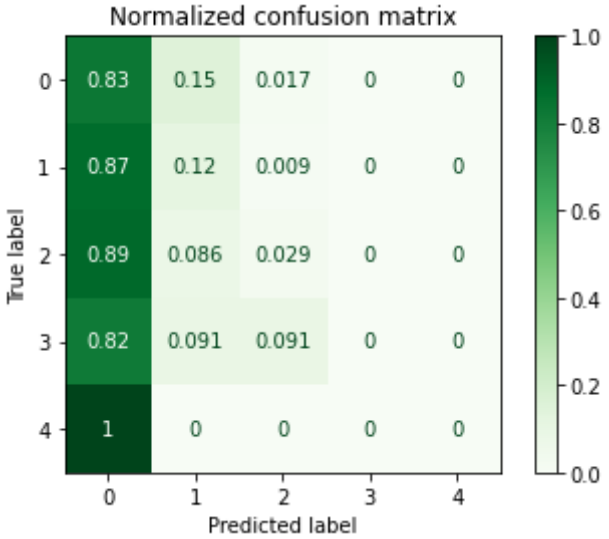


Fig.3.3

That maybe increment the f1 score, but confronting this figure to Fig.3.2, we see that the classes more important are now not predicted! So it's not convenient to go further with GridSearch and with this model because it won't bring to no better direction (and considering probably the data are not linearly separable).

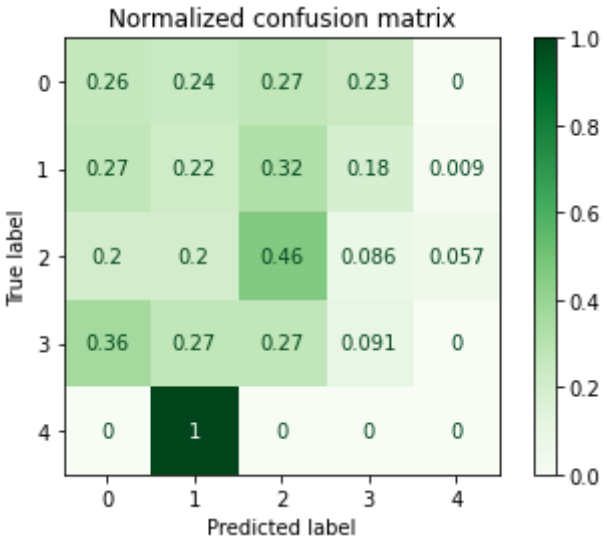
3.2 Logistic Regression

Using a model based on Logistic Regression with the solver 'newton-cg' with no tuning we can see that the behavior is worse than svc: the accuracy on the train set is now decreased to 55%, while the accuracy on the test set has not varied. Moreover, the f1 and precision are worse than before.



We don’t have a lot of to tune with Grid_Search (except for C); but Logistic regression implemented by sklearn has a feature that permits to adjust the weight of each class, in order to give much more importance to the class that we think to be more relevant (1,2,3,4 collisions) and based on the number of samples that we have.

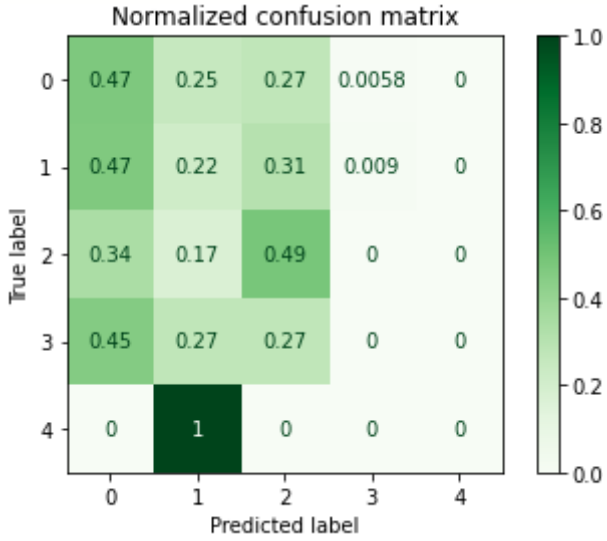
In a first try I tired with class_weight=’balanced’; the confusion matrix resulting is this:



As we can see, now we don’t have the class 0 being predominant in the confusion matrix, and we are going in the direction of most important classes (although the accuracy is decreased so much near 25%-30%)

Using Grid_Search with this solution, we can’t see so much differences (both in accuracy and f1_weighted)

Then I tried custom values of weight, giving more force to the last 2 classes obtaining some more stunning result:



with
```
class_weight={0:0.2,
              1:0.3,
              2:1,
              3:1,
              4:1}
```
Then I suppose that best way to follow is to try different values of weight, using also Grid_Search for finding the best combination.

Also the accuracy is increased (35/40%).

3.4 AdaboostClassifier

At the end I tried the Boost ensemble method because It is suitable when we have a bad classifier and we need to give more weight and relevance to the classes that have misclassified by the previous classifier (as described at the end of chapter 3.3).

The estimators used are of the types described in the solutions before, in order to see if it is possible to find more better solutions:

- SVC
- Logistic Regression

But sadly, both trying to optimize accuracy and f1_weighted we obtain always **Fig.3.1** as confusion matrix. Moreover, using GridSearch want to use only one estimator. This means that first estimator is working, while the others are not doing anything.

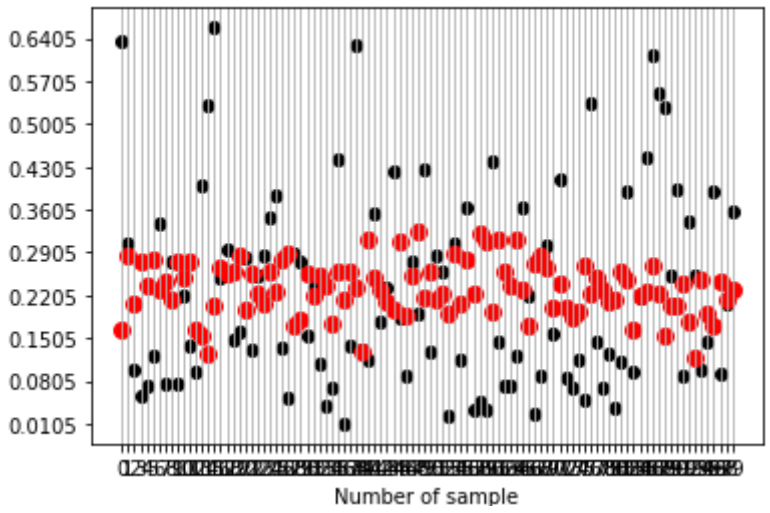
4 Regression Task

As I did for the Classification Task, I used 2 different models: SVR and Linear Regression.

At first I include the column “Number of collisions” in the X_all, and it was working good for mean squared error and regression score. Then I removed this feature as asked and I have started to have negative regression score and an enormous mean. Let’s try to analyze this particular situation.

4.1 Linear Regression

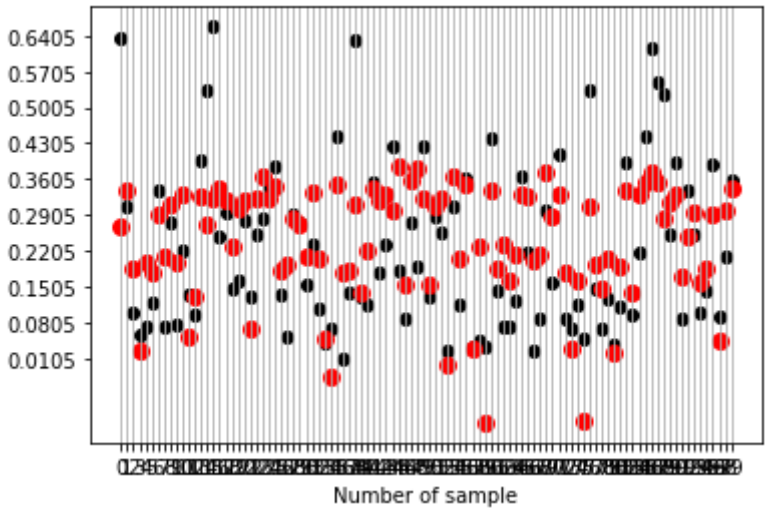
Excluding the “Number of collision” feature, we can find that the mean squared error is something good (like 0.03) but the regression score is negative. With the GridSearch we can optimize the value of the regression score but we can’t reach positive score.



In this Graphic are shown:

- The black point that are the real values (scaled between 0 and 1) of the minCPA of the samples belonging the test set
- The Red points that are the prediction of each blackpoints.
- On the left the value of minCPA.

Now, considering also the number of collisions we have this Graphic:



As we can see, just adding one feature the solution is now approximating better the black points (red points are less dense and nearer the black points)

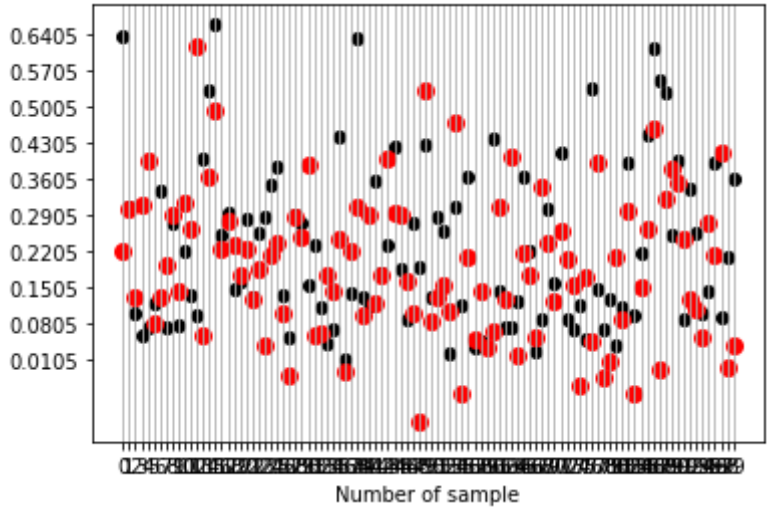
4.3 Support Vector Machine for Regression

Respect to Linear Regression, SVR has more parameters that we can tune and change in order to obtain a better solution (the most important is epsilon, through which we can change the interval of soft margin constraints).

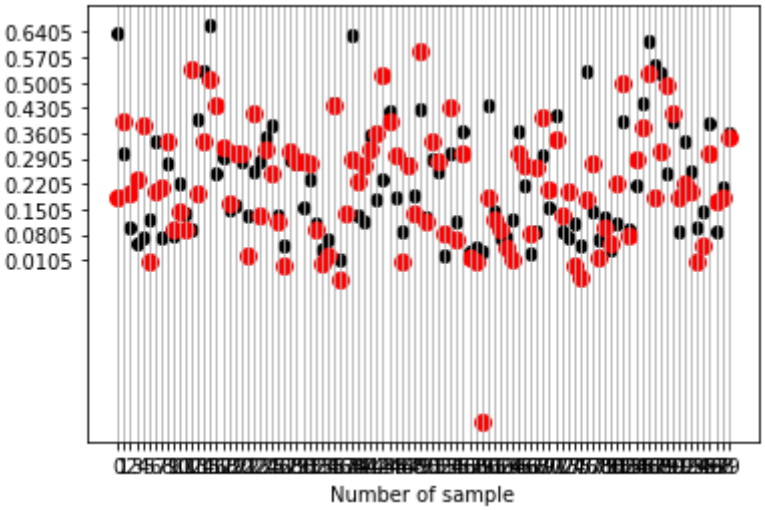
Moreover, using Grid_Search, it's possible to obtain a positive regression score on the test set (0.04) starting from a negative value using this parameter:

```
Best regression hyper-parameters: {'C': 0.2, 'degree': 1, 'epsilon': 0.1,
'kernel': 'rbf'}
```

This is the graphic:



And the same as linear regression, just adding the feature "num_collisions" we obtain this:



We can see that the Graphic is more agglomerate (so the red points are estimating well the black points) and moreover the support vector used are lower than before (30/40% versus 70/80%) and with this `Best regression hyper-parameters: {'C': 0.5, 'degree': 1, 'epsilon': 0.05, 'kernel': 'rbf'}` we reached `Mean squared error of 0.01` and `Regression score of 0.47`