CHAPTER 1: Introduction
1.1 Domain problem
1.2 Approach to the problems

CHAPTER 2: Classification
2.1 Setting up the environment
2.2 Random Forest Classifier
    2.2.1 Stratified K-Fold for RandomForest
    2.2.2 RandomForestClassifier
    2.2.3 Grid Search for RandomForestClassifier
2.3 K-Nearest Neighbors
    2.3.1 Pre-processing
    2.3.2 Stratified K-Fold
    2.3.3 K-Nearest Neighbors Algorithm
    2.3.4 Grid Search for K-NNClassifier
2.4 Result comparisons

CHAPTER 3: REGRESSION
3.1 Setting up the environment
3.2 Pre-processing: Normalization
3.3 Support Vector Regressor
    3.3.1 Grid Search for SVR
3.4 K-Nearest Neighbors Regressor
    3.4.1 Grid Search for K-NNRegressor
3.5 Result comparisons

CHAPTER 1: Introduction

The topic of this report is the multi-UAV conflict risk analysis, that is, the subject of the first homework.
In detail, the domain problem is related to two tasks: classification and regression

1.1 Domain problem:

Given the dataset, find a solution for the following problems:

- Classification problem: estimate and predict the number of collisions between the UAVs given the provided features.
- Regression problem: predict the minimum Closest Point of Approach among all the possible pairs of UAVs

The Closest Point of Approach (or shortly CPA) is an estimated point in which the distance between two objects, of which at least one is in motion, will reach its minimum value.

1.2 Approach to the problems

To accomplish these tasks, the work has been based on a try-and-compare approach: many models have been tested for both classification and regression, with different configurations of the same, and the results have been compared.
This report analyses in details only Random Forest and K-NN models for classification, and Support Vector Machine and K-NN models for regression.

CHAPTER 2: Classification

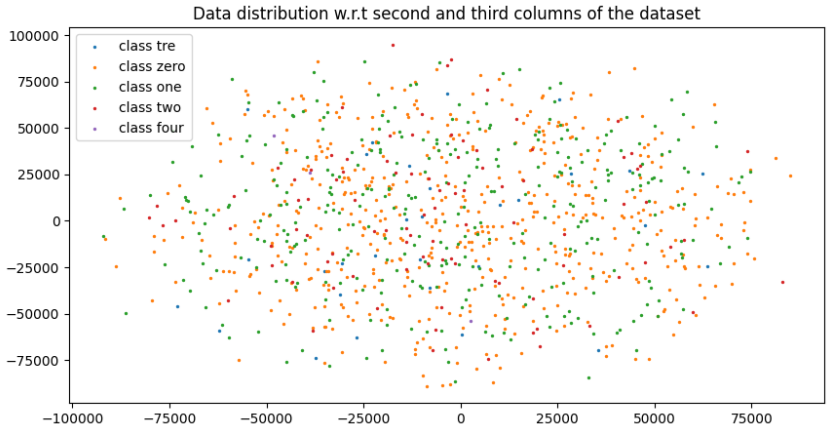
2.1 Setting up the environment

The first step, in order to set up a good working environment, is to manipulate the given dataset. In fact, at first look, the dataset has the following structure:

	UAV_1_TRACK	UAV_1_X	UAV_1_Y	UAV_1_VX	...	NUM_COLLISIONS	MIN_CPA
0	0.0270	-62300.5917	-59305.6820	6.7056	...	3	1673.7348
1	4.0231	-17220.6125	47439.5869	-167.6530	...	0	51230.5477
2	1.8419	-19900.3504	59030.8335	208.7166	...	0	18668.1777
3	3.6215	-48565.1265	-11986.4185	-113.5163	...	0	10159.6247
...	...	...	...	...	...	...	...
999	1.4646	-20386.3253	-69553.2805	247.6855	...	1	6115.5453

***T2.1 DESCRIPTION: a part of the dataset that shows some its elements

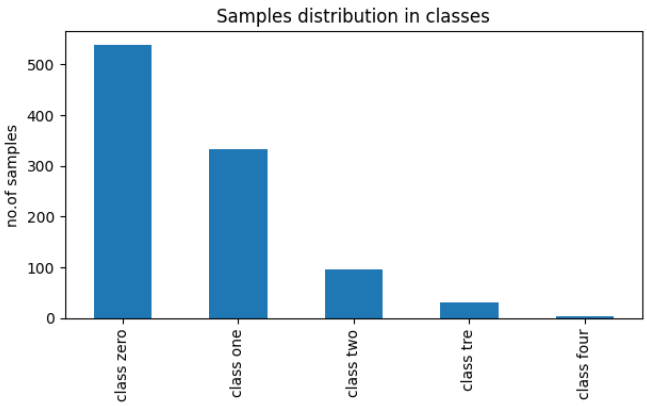
There are 35 column-features per 1000 row-samples. The last two columns represent the labels respectively for the classification and regression tasks for a total of 37 columns.
For the current task, classification, we need to reprocess the dataset to extract the useful and only the right ones. As said earlier, the dataset has 37 columns but for the classification we need all the feature columns and only the classification label. At this point we can have a preview of the dataset that will be used for the first task.



The image on the left represents the data distribution among the second and third columns of the dataset. This graph does not have any particular meaning: it is made only for visualisation purpose. But using a bit of domain knowledge (not allowed in this homework), the graph represents the different positions of the drones in the space.

A useful representation of the dataset is shown in the image on the right. The bar diagram represents the distribution of the samples in the target classes. For a total of 1000 samples:

- 538 samples belong to the class 0 (53.8%)
- 333 samples belong to the class 1 (33.3%)
- 96 samples belong to the class 2 (9.6%)
- 30 samples belong to the class 3 (3.0%)
- 3 samples belong to the class 4 (0.3%)



This plot gives an important information: the dataset is highly imbalanced.

2.2 Random Forest Classifier

The first model used to accomplish the task is the Random Forest classifier. To have a first look up at the skills of the model, it is used a cross-validation method. We know that the dataset is imbalanced, hence, a standard k-fold technique would give misrepresenting results. The method I used is the stratified k-fold cross-validation that splits the data in a stratified manner so to maintain the same class ratio throughout the K folds as the ratio in the original dataset.

This method has been applied systematically on every algorithm tested before going in depth to evaluate it. In this way I was able to perform a first selection among the most promising algorithms.

2.2.1 Stratified K-Fold for RandomForest

The Random Forest stratified k-fold is performed on the whole dataset. The following results are achieved using these values and these parameters: the RandomForestClassifier has been tuned with all the default parameters and a fixed random_state value; the StratifiedKFold is performed on 10 splits, with a fixed random_state and shuffle equals to true. Here the results:

Overall f1-score: 0.20017394045788178
Overall balanced accuracy: 0.2373571078898493

To evaluate the performance of the model is used the f1-score and balanced accuracy score: the dataset is highly unbalanced, so the standard precision metric is not reliable enough. F1-score allows to obtain a better summarization per each class using precision and recall. In particular, to evaluate the overall f1-score among all the 10 folds, the macro f1-score average has been calculated. Moreover, the balanced accuracy score has been calculated as a second metrics. It is used to deal with imbalanced dataset and it is calculated as average of recall obtained on each class.

The results obtained with all algorithms are not very good on this dataset, but in a general comparison of algorithms, the above results are quite valid and allow to deepen the classifier in question

2.2.2 RandomForestClassifier

The next step was to evaluate the classifier in different situations with different parameters. Two tests are made in parallel on the classifier: a first model has been run out on a balanced dataset and a second model has run on the initial dataset. Before the balancing, the dataset was split in train and test subsets, and only the training set has been balanced, in order to avoid to introduce biased data validation set.
The method used to balance the dataset is SMOTE (Synthetic Minority Oversampling TEchnique by the Imbalanced Learn Team) that balances the data by synthetizing new examples in classes with less examples. The results are the following:

BALANCED DATASET	**NON-BALANCED DATASET**
Accuracy score: 0.475	**Balanced Accuracy score: 0.212**

f1-score for class 0: 0.641667	**f1-score for class 0: 0.686131**
f1-score for class 1: 0.330579	**f1-score for class 1: 0.240000**
f1-score for class 2: 0.064516	**f1-score for class 2: 0.000000**
f1-score for class 3: 0.000000	**f1-score for class 3: 0.000000**
f1-score for class 4: 0.000000	**f1-score for class 4: 0.000000**

Macro avg. f1-score: 0.20735	**Macro avg. f1-score: 0.18523**

Results show how the model performs slightly better on a balanced dataset

2.2.3 Grid Search for RandomForestClassifier

To get the best hyper-parameters of the classifier, the GridSearch has been implemented. In particular, the parameters passed to the algorithm are:

- 'n_estimators': (10,100,200,500)
- 'criterion': ('gini', 'log_loss', "entropy")
- 'max_depth': (5, 9, None)

The search was based on 5 folds and on the best macro f1-score value.
The GridSearch algorithm has returned as best hyper-parameters: n_estimators=200, max_depth=None and criterion='gini'

The model tuned with these parameters has reached:

f1-score for class 0: 0.641667
f1-score for class 1: 0.330579
f1-score for class 2: 0.064516
f1-score for class 3: 0.000000
f1-score for class 4: 0.000000

Macro avg. f1-score: 0.20735

Performances are about the same of the initial default model

2.3 K-Nearest Neighbors

The second model used for accomplish the classification task is the K-NN. This model is one of the simpler models in machine learning and I found it interesting to see how it can work against a more complex model like random forest is.

2.3.1 Pre-processing

The K-NN algorithm is a distance-based classifier: it assumes that two points near each other are similar. With this assumption in mind, the first step to do is normalize the dataset. In this case, instead, a standardization helped to achieve better results. The RobustScaler technique has transformed the initial dataset as following:

	UAV_1_TRACK	UAV_1_X	UAV_1_Y	UAV_1_VX	…
0	-1.0134182	-1.024432	-1.0924547	0.0060524	…
1	0.2992834	-0.2366413	0.7929961	-0.5368735	…
2	-0.4172128	-0.2834708	0.9977333	0.6350834	…
3	0.1673673	-0.7843992	-0.2566505	-0.368300	…
…	…	…	…	…	…
999	-0.5411577	-0.2919634	-1.2734589	0.7564264	…

2.3.2 Stratified K-Fold

The stratified K-Fold cross-validation has been applied also for the K-NN classifier. The StratifiedKFold is performed on 10 splits, with a fixed random_state, a number of neighbors equals to two and shuffle equals to true. The results are:

Overall f1-score: 0.23599222589746313
Overall Balanced accuracy: 0.2511165297069848
The StratifiedKFold on K-NN has returned better values than StratifiedKFold on random forest.

2.3.3 K-Nearest Neighbors Algorithm

The workflow of evaluating the model followed the same structure of the previous model. The K-NN classifier has been tested on the same two datasets (in this case a standardized versions) of the Random Forest classifier.

In this case, the oversampling method used differs from the one of the RandomForest: firsty, only the class 4 has resampled up to 30 samples with RandomOverSampler; after, the new dataset has been resampled with the SMOTE technique. This method was tried aiming to give the possibility at SMOTE to use more samples in the class with less samples in its parameter k_neigbhors .

Before initializing the models, another standardization has been done on train and test subsets generated by the splitting.

The model has achieved for the two datasets the following results:

BALANCED DATASET	**NON-BALANCED DATASET**
Accuracy score: 0.330	**Balanced Accuracy score: 0.217**

f1-score for class 0: 0.397260	**f1-score for class 0: 0.650602**
f1-score for class 1: 0.400000	**f1-score for class 1: 0.247934**
f1-score for class 2: 0.162162	**f1-score for class 2: 0.173913**
f1-score for class 3: 0.157895	**f1-score for class 3: 0.000000**
f1-score for class 4: 0.000000	**f1-score for class 4: 0.000000**

Macro avg. f1-score: 0.22346	**Macro avg. f1-score: 0.21449**

The results are very different from the previous classifier: K-NN is able to predict samples of the class 3

2.3.4 Grid Search for K-NNClassifier

To get the best hyper-parameters of the classifier, the GridSearch has been implemented. In particular, the parameters passed to the algorithm are:

- 'n_neighbors': range(3, 100, 2),
- 'weights': ('uniform', 'distance'),
- 'algorithm': ('ball_tree', 'kd_tree', 'brute')

The search was based on 5 folds and on the best macro f1-score value.
The GridSearch algorithm has returned as best hyper-parameters: n_ neighbors'=3, 'weights'='uniform' and 'algorithm'= 'ball_tree'

The model tuned with these parameters has reached:

f1-score for class 0: 0.47204969
f1-score for class 1: 0.3880597
f1-score for class 2: 0.17647059
f1-score for class 3: 0.17142857

f1-score for class 4: 0.000000

Macro avg. f1-score: 0.242

Performances are slightly improved considering the initial default model.

2.4 Result comparisons

With all the results achieved, now is it possible to make a comparison between these two models. Since the first test with StratifiedKFold, the K-NN model achieves better results:

K-NN overall f1-score: 0.23599222589746313
RandomForest Overall f1-score: 0.20017394045788178

The f1 score evaluated on cross-validation gives a robust and good landscape about the performance of the models on the imbalanced dataset. Keeping the dataset highly skewed lowers the skills of the models in all cases.

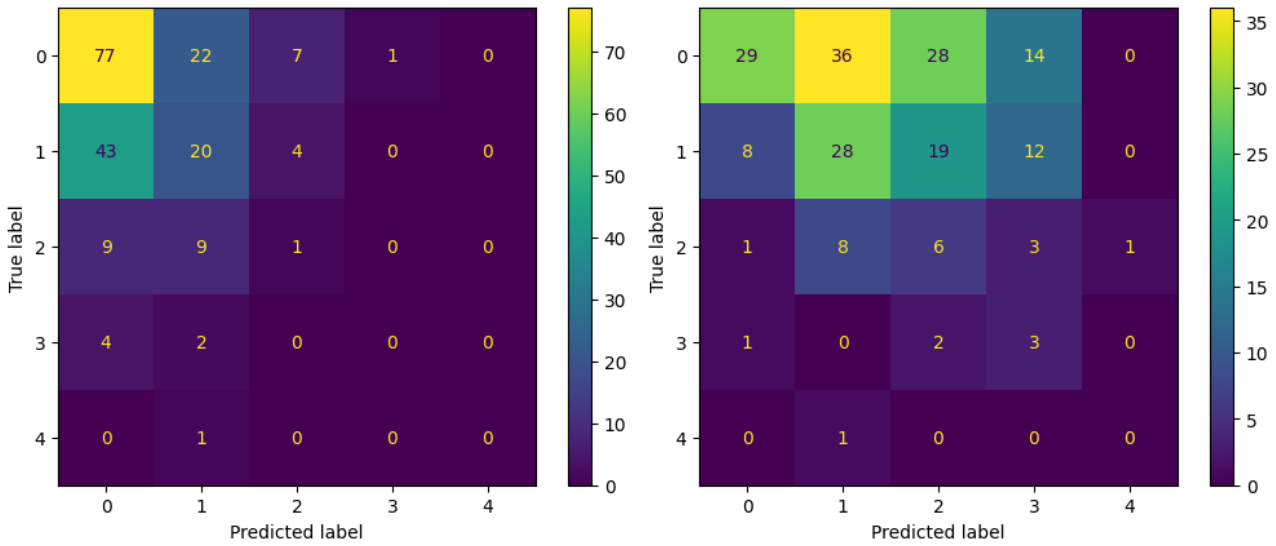
Balancing the dataset does not produce notable differences, however the score increases.

At this point, we can move the focus on the best models tuned with GridSearch. Also in this case, K-NN represents performs better:

K-NN with GridSearch	RandomForest with GridSearch
f1-score for class 0: 0.47204969	f1-score for class 0: 0.641667
f1-score for class 1: 0.3880597	f1-score for class 1: 0.330579
f1-score for class 2: 0.17647059	f1-score for class 2: 0.064516
f1-score for class 3: 0.17142857	f1-score for class 3: 0.000000
f1-score for class 4: 0.000000	f1-score for class 4: 0.000000

Both the models cannot predict the class 4, but a big difference is made on class 3: RandomForest cannot predict also that class, contrarily K-NN is able to predict and classify correctly samples of this class.

A better view can be obtained comparing the confusion matrix of the GridSearch models:



The left plot is the confusion matrix of the RandomForest and the loft plot represent the confusion matrix of K-NN. The first model works better on class 0 but is very imprecise on the other classes; in other hand K-

NN works better on the remaining classes except the fourth. In conclusion, both of models don’t work property well.

CHAPTER 3: REGRESSION

3.1 Setting up the environment

The first step, in order to set up a good working environment, is to manipulate the given dataset. In fact, at first look, the dataset has the following structure:

	UAV_1_TRACK	UAV_1_X	UAV_1_Y	UAV_1_VX	…	NUM_COLLISIONS	MIN_CPA
0	0.0270	-62300.5917	-59305.6820	6.7056	…	3	1673.7348
1	4.0231	-17220.6125	47439.5869	-167.6530	…	0	51230.5477
2	1.8419	-19900.3504	59030.8335	208.7166	…	0	18668.1777
3	3.6215	-48565.1265	-11986.4185	-113.5163	…	0	10159.6247
…	…	…	…	…	…	…	…
999	1.4646	-20386.3253	-69553.2805	247.6855	…	1	6115.5453

***T3.1 DESCRIPTION: a part of the dataset that shows some its elements

For the current task, classification, we need to reprocess the dataset to extract the useful and only the right ones. As said earlier, the dataset has 37 columns but for the regression we need all the feature columns and only the last column, the regression label. At this point we can have a preview of the dataset that will be used for the first task.

3.2 Pre-processing: Normalization

From the table T3.1 we can see that values of the dataset spread among a wide range. For some algorithms this can lead to weak performances in regression. Hence, the pre-processing technique used is the normalization of the dataset. To do this, it is used the *MinMaxScaler* class from the *sklearn* module "*preprocessing*" because this method applies by default a column-wise normalization: it normalizes the features independently from each other.
The mathematical formula behind the transformation is:

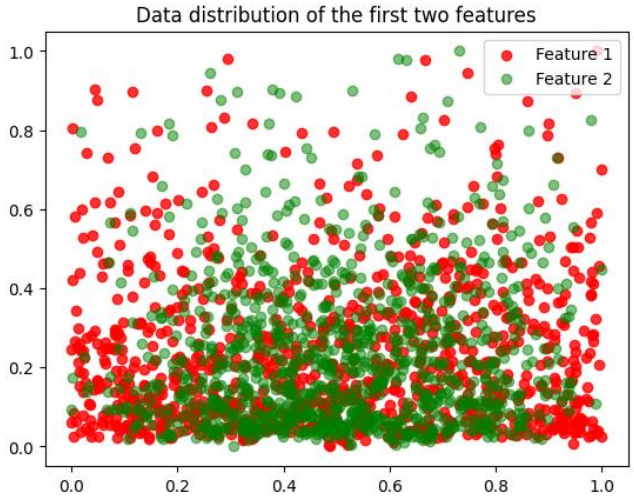
$$S_{scaled} = \left(\frac{X - X_{min}}{X_{max} - X_{min}}\right) * (\max - \min) + min$$

Thus now, the features values of the T2.1 table involved in regression task, have been transformed as following:

	UAV_1_TRACK	UAV_1_X	UAV_1_Y	UAV_1_VX	…
0	0.0035749	0.1678027	0.1614930	0.5088190	…
1	0.6403676	0.4226733	0.7417625	0.1650284	…
2	0.2927947	0.4075227	0.8047727	0.9071329	…
3	0.5763750	0.2454595	0.418721	0.2717721	…
…	…	…	…	…	…
999	0.2326689	0.404775	0.10578	0.9839695	…

Moreover, another method to normalize the dataset has been tried. Once again, from "preprocessing" module, I used the *normalize* class which performs a scaling of the individual samples to have unit norm according to L1 norm or better known as Manhattan Distance.
But, unfortunately, this second method gave the same accuracy results reached with the first *MinMaxScaler* technique on every algorithm that needs a normalized input.

A third method tried, is the *standardScaler:* removes the mean and scales the data to unit variance. But also, this method does not improve performances.



At the left is shown the data distribution of the first two features. The plot is built only for visualization purpose.

The next step is to split the dataset to get training and test subsets on which set up the models.

3.3 Support Vector Regressor

The first model tested to accomplish the regression task is Epsilon-Support Vector Regression. The test has made on the default regressor. Unfortunately, results are daunting:

Mean squared error: 0.04

Regression score: -0.03

The metrics used to evaluate models for this task are the Mean squared error: it corresponds to the expected value of the squared (quadratic) error or loss. The mathematical formulation behind this metric is the following:

$$\sum_{r=1}^{\min(K,M)} \frac{y_{(r)}}{\log(1+r)}$$

The second metrics used is the r2-score that computes the coefficient of determination denoted as R^2.

3.3.1 Grid Search for SVR

To get the best hyper-parameters of the regressor, the GridSearch has been implemented. In this particular case, two searches are made up: the first one GridSearch is for linear, rbf, poly or sigmoid kernels. The parameters passed to the algorithm are:

- 'C': np.arange(0.8,2,0.1),
- 'kernel': ('linear', 'rbf','poly','sigmoid'),
- 'gamma': ('scale','auto')

The search was based on 5 folds and on the best r2-score value.
The GridSearch algorithm has returned as best hyper-parameters: 'C'=0.8, gamma='scale', kernel='rbf'
The second GridSearch is for only poly kernel. The parameters passed to the algorithm are:

- 'C': np.arange(0.8,2,0.1),
- 'degree': np.arange(1,4,1)

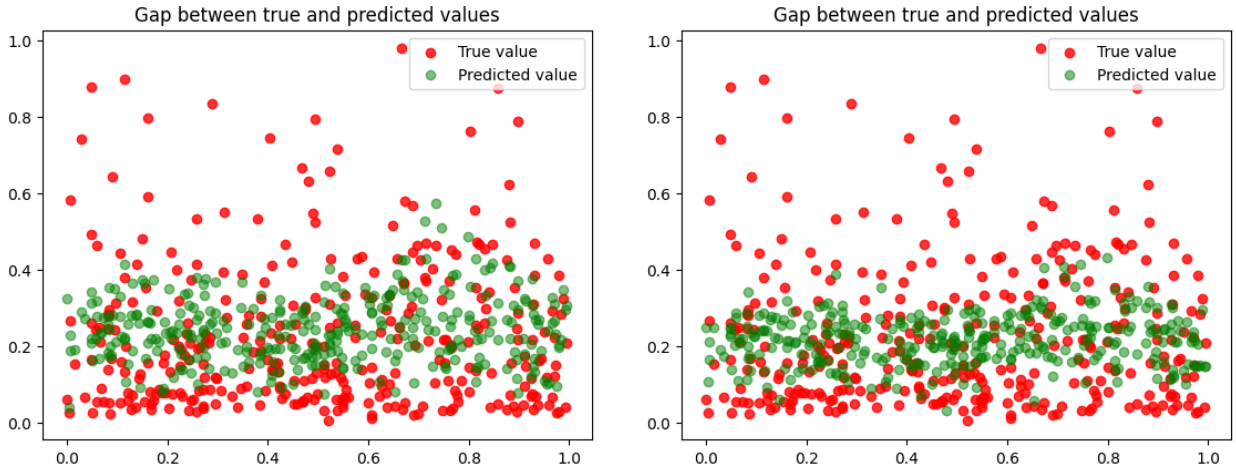
The search was based on 5 folds and on the best r2-score value.
Only the model tuned with second GridSearch increased perfomances.

The model tuned with these parameters has reached:

Mean squared error: 0.04

Regression score: 0.01

Graphically we can compare the two SVR models:



In the first plot we can see the SVR model initialized with standard parameters, the second plot represents also the sparsity of the data but of the SVR tuned with GridSearch.

3.4 K-Nearest Neighbors Regressor

The second model tested to accomplish the regression task is the K-NN regressor. The test has made on the regressor with n_neighbors parameter equals to 80. Fortunately, results are the same of the SVR with GridSearc:

Mean squared error: 0.04

Regression score: 0.01

This means that performing a good GridSearch can improve the overall results achieved.

3.4.1 Grid Search for K-NNRegressor

To get the best hyper-parameters of the regressor, the GridSearch has been implemented. In particular, the parameters passed to the algorithm are:

- 'n_neighbors': range(1, 200, 1),
- 'weights': ('uniform', 'distance'),
- 'algorithm': ('auto','ball_tree', 'kd_tree', 'brute')

The search was based on 5 folds and on the best macro f1-score value.
The GridSearch algorithm has returned as best hyper-parameters: n_ neighbors'=39, 'weights'='distance' and 'algorithm'= 'ball_tree'

The model tuned with these parameters has reached:

Mean squared error: 0.03

Regression score: 0.04

The GridSearch allowed to increase r2 score and decrease MSE value. These are the best values achieved.

3.5 Result comparisons

Every algorithm proposed in the regression task did not achieve any reasonable value. Among all of them, the K-NN tuned with GridSearch took the best values:

K-NN tuned with GridSearch	**SVR tuned with GridSearch**
Mean squared error: 0.03	*Mean squared error: 0.04*
Regression score: 0.04	*Regression score: 0.01*

The K-NN performed better than SVR