Project Report

GitHub URL

https://github.com/schmly/UCDPA_hlaing

Abstract

We use a real-world dataset derived from police reports on traffic collisions to attempt to predict the number of injured persons based on information about the vehicle types involved and the contributing factors leading to the collisions. To do this, we use supervised learning methods for multiclass classification. As a motivation, we propose that if an algorithm of sufficient quality can be created, it could prove useful in designing strategies or policies to improve safety for persons in urban traffic.

Estimating the number of injured persons based on rather rudimentary information from a police report may seem a daunting task. However, as we are dealing with machine learning, we have no reservations about trying to push a machine to its limits.

In order to find the best algorithm, we compare different classification algorithms and apply hyper parameter tuning and ensemble methods.

This project was created using Visual Studio Code.

Please note that the dataset used to create this report was too big to be exported and submitted. To access the data, please import it by running the jupyter notebook. The data is free to access for any user without any keys. If the data source receives an update between the time of writing and the time the process is rerun, the results may be slightly different than what is presented here.

Introduction

From the machine learning perspective, we are motivated by two examples from supervised learning. The first one is tumour classification. Typically, this is a class-inbalanced problem where the number of malignant tumours is small. Similarly, most collisions in our dataset do not result in a cyclist injury. This is called a class-inbalanced dataset. Class inbalance hower is not our focus here. More basically, both are classification problems of supervised learning with labelled data. While tumour classification is binary, we view our problem as multiclass, where we predict the number of injured persons. However, this could easily be aligned by considering instead the question “was at least one cyclist injured or not?”.

Another example from supervised learning which we consider structurally comparable is the prediction of app-ratings, which is also a multiclass classification problem.

One difficulty of choosing a topic is that a significant amount of work might be required to realise that the initial idea is not as interesting as desired. If that is the case, then we at least hope that methods shown here could be tweaked to a slightly different, but more interesting case.

Dataset

I consider the dataset “Motor Vehicle Collisions – Crashes”, published by the City of New York. It is available here:

https://catalog.data.gov/dataset/motor-vehicle-collisions-crashes

or also here:

https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95

The source describes the dataset as follows:

The Motor Vehicle Collisions crash table contains details on the crash event. Each row represents a crash event. The Motor Vehicle Collisions data tables contain information from all police reported motor vehicle collisions in NYC. The police report (MV104-AN) is required to be filled out for collisions where someone is injured or killed, or where there is at least $1000 worth of damage.

There are several reasons why I chose that dataset. For once, it is a real-world dataset rather than a dataset which was specially selected for data science purposes. Moreover, it is available via an API, which is a feature that was covered in some detail in the course and is very suitable to be included in this project. Furthermore, it is free for everyone to download. Finally, it appears to be one of the most extensive real-world datasets concerned with the very relevant und current topic of accidents and injuries in urban traffic.

Implementation Process

Data Collection

The source provides a JSON API Endpoint under the URL

https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95

With this URL, the data can be collected via the pandas read_json function. The access is covered by the Socrata Open Data API (SODA), which provides a specialist syntax. In particular, data can be requested by specifying a “limit” referring to the total number of rows to be downloaded and an “offset” indicating the index of the first row. Using this, we define helper functions to permit the download of the dataset in chunks

Due to its large size, not the entire dataset is downloaded. Instead, we provide a number of lines to be read, which will be read starting with the most recent entry.

Data Preparation

Cleaning

Some initial cleaning is required to rectify inconsistent column names.

We set the column collision_id as the index of the dataframe, we therefore have to ensure that it is unique by removing duplicate collision ids.

Feature Engineering

The information contained in the dataset can roughly be grouped as follows:

- Time, location, address of collision
- Information on involved vehicles: vehicle type and contributing factor (of the collision)
- Number and type (as traffic participants) of persons injured or killed.

In our analysis, we want to focus on the vehicle type and contributing factor as input for the machine learning algorithms. These two features come in 5 columns each, covering up to 5 involved vehicles.

To make them usable for machine learning algorithms, we would like to transfer the information into binary dummy columns. This is achieved with the following steps:

- Merge the information into a single column and add a second index referring to the number of the vehicle (in other words: making the dataframe narrower and longer)
- Apply string extraction and manipulation methods to align the strings syntactically.
- Create the dummies
- Group by the first index (collision id) and aggregating the dummy columns over the vehicle index

As a result, we obtain two sets of columns:

Prefix	Description
.cf	Contributing factor dummies
.vt	Vehicle type dummies

As mentioned, both of these have been aggregated over the collision id, which means that in one row, multiple dummies for a contributing factor or a vehicle type may be true if there were several vehicles involved.

Note that during the string manipulation, the contributing factor comments have been abbreviated using a mapping. The mapping is export to allow the recovery of the original comment.

Class Imbalance

Our dataset has a degree of class imbalance as most collisions do not result in injuries. Resampling techniques such as SMOTE from the imbalanced-learn module can be used to balance the dataset before algorithms are trained. We chose not to do that, as our dataset is rather large, and we felt that the effects of applying a balancing algorithm would be a bit unpredictable. We prefer a more basic approach.

Algorithm Evaluation

Initial Comparison

First, we split the downloaded data into a train and test set. Thereby, we follow the scikit-learn guide on cross-validation: an initial split is carried out to hold out a test set for final evaluation, while the cross validation to initially measure algorithm performance is carried out on the training set.

As mentioned above, our training data is the set of dummy columns we have created, while the target data is the number of injured persons.

As mentioned above, we are faced with a multiclass classification problem. We pick three algorithms labelled as inherently multiclass by the scikit-learn guide:

Module	Estimator
linear_model	RidgeClassifier
tree	DecisionTreeClassifier
naive_bayes	BernoulliNB

Then, we fix a Kfold split of the training data and determine the cross-validation score for the three chosen models. The result is displayed in a chart (see the next section for a discussion). Note that the

score returned by cross_val_score is the ratio of correctly predicted outcomes versus total number of outcomes.

To estimate a potential issue with overfitting, we also compute the score on the actual test set as well as the training set and compare that to the cross-validation scores.

Comparison of tree-based Models

We select three tree-based models:

- DecisionTreeClassifier
- RandomForest
- BaggingClassifier (with a decision tree as default estimator)

For these, we execute the same steps as above.

In addition, we perform a direct comparison of the decision tree and the bagging classifier on the isolated test set.

Focus on Decision Tree

We drill down into the prediction of the isolated test set with a decision tree by looking at the confusion matrix.

Boosting

We chose an AdaBoostClassifier, which also uses a decision tree as default meta estimator. We use 100 copies of a decision tree estimator with max depth set to 2. We perform a cross-validation as before and compare the performance to the tree-based models previously considered.

Hyper-Parameter Tuning

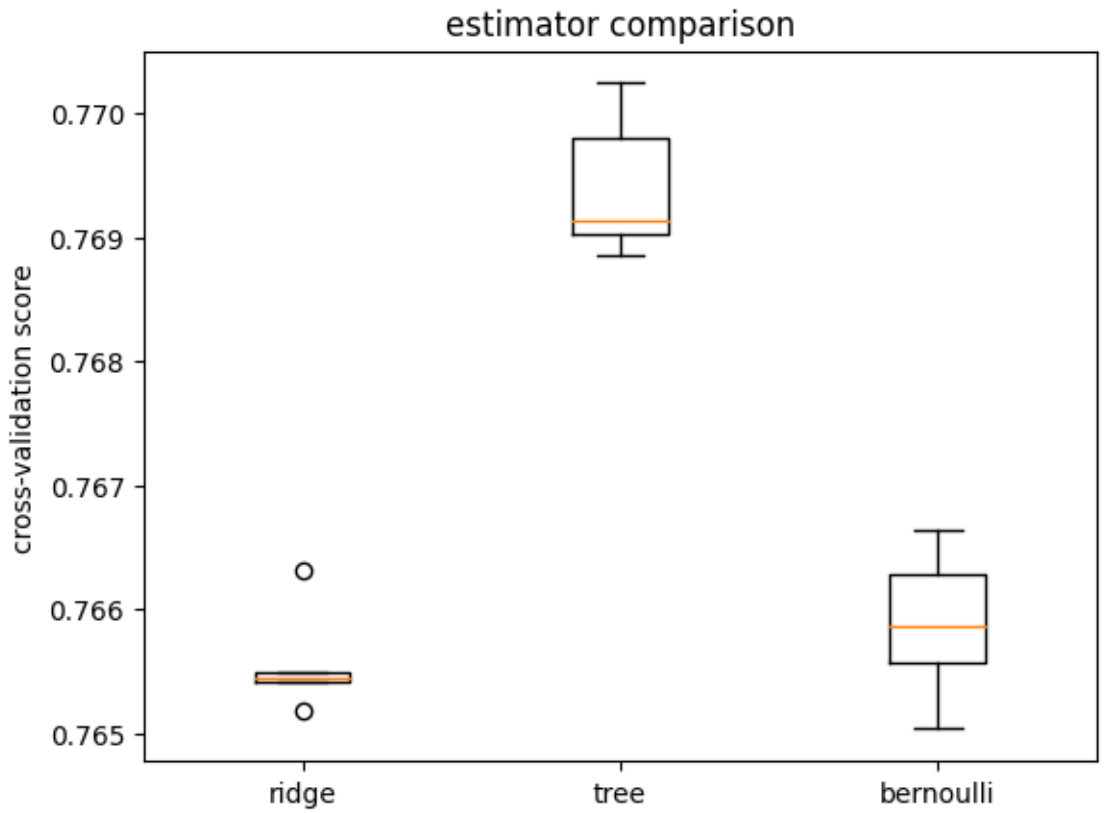
At last, we focus on a decision tree classifier to perform the tuning of the parameter min_samples_leaf, which controls the minimum number of samples required to be at a leaf node in a decision tree. To do this, we use the GridSearchCV utility from the model_selection module.

Results

Comparison of Estimators

Initial Comparison

For the initial comparison of classification estimators, we pick the three multiclass classifiers RidgeClassifier, DecisionTreeClassifier and BernoulliNB. Note that, as mentioned in the previous section, these are supervised-learning classifiers suitable for multiclass prediction.



As can be seen in the boxplot, the DecisionTreeClassifier yields the best result. This motivates a deeper analysis of tree-based models[1].

Before we come do that, we want to also compute the score for the above models when used on the actual test set that we have isolated earlier. This is helpful to detect issues such as over-fitting, which in particular a decision tree classifier may be prone to. We compare the cross-validation scores and the test score in a table:

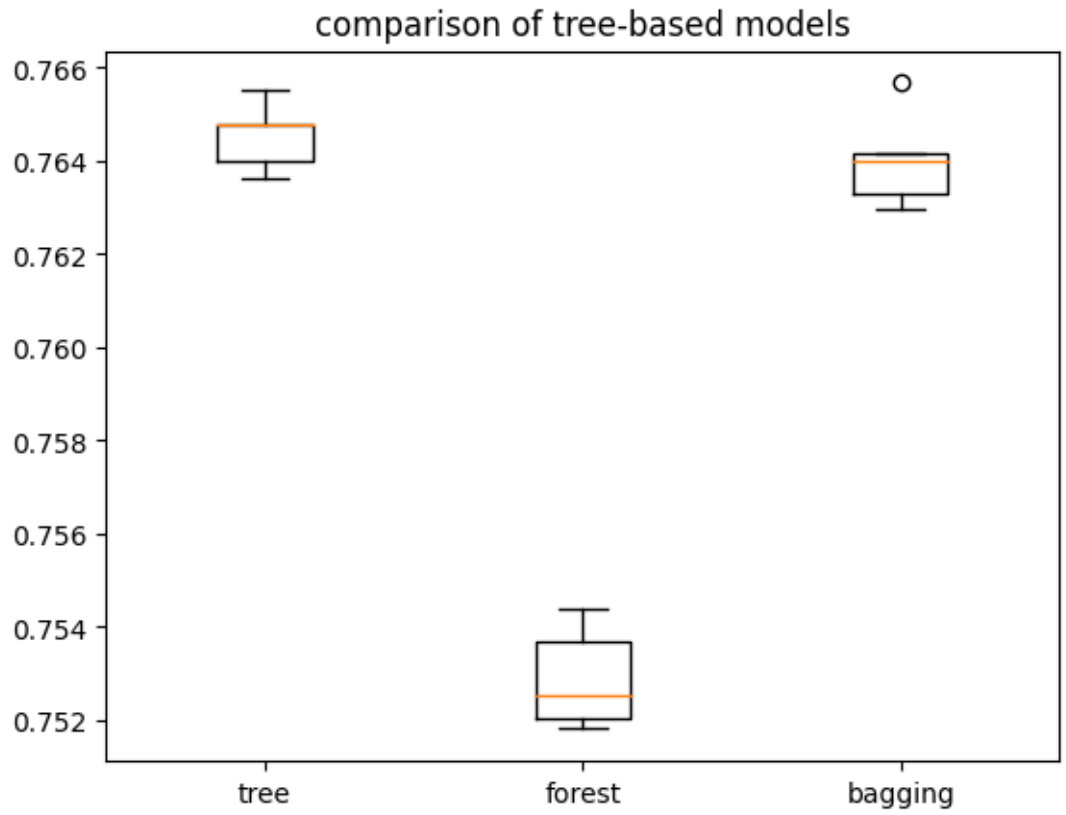
	test_score	training_score	cv_mean_score	cv_vs_test	train_vs_test
ridge	0.764913	0.765623	0.765566	0.065229	0.071007
tree	0.7692	0.77777	0.769412	0.021231	0.857012
bernoulli	0.765107	0.765843	0.765877	0.077007	0.073674

In the column train_vs_test, we can see that the decision tree model performs 0.86% better on the training set than on the test set. This indicates that some over-fitting may be present. Nevertheless, the decision tree is still the best performer on the test set, too. These findings are another reason to further investigate tree-based models.

Comparison of tree-based Models

In addition to the DecisionTreeClassifier, we consider an instance of RandomForestClassifier as well as a BaggingClassifier based on a decision tree estimator. The latter is an example of an ensemble method from the scikit module of the same name and aims to combine the predictions a meta-estimator, in this case a decision tree.

[1] Note that the creation of dummy variables during the data preparation was necessary for a RidgeClassifier or BernoulliNB, but it is not strictly necessary for a DecisionTree estimator.



We observe that a decision tree still performs best, however, a bagging classifier is almost as good.

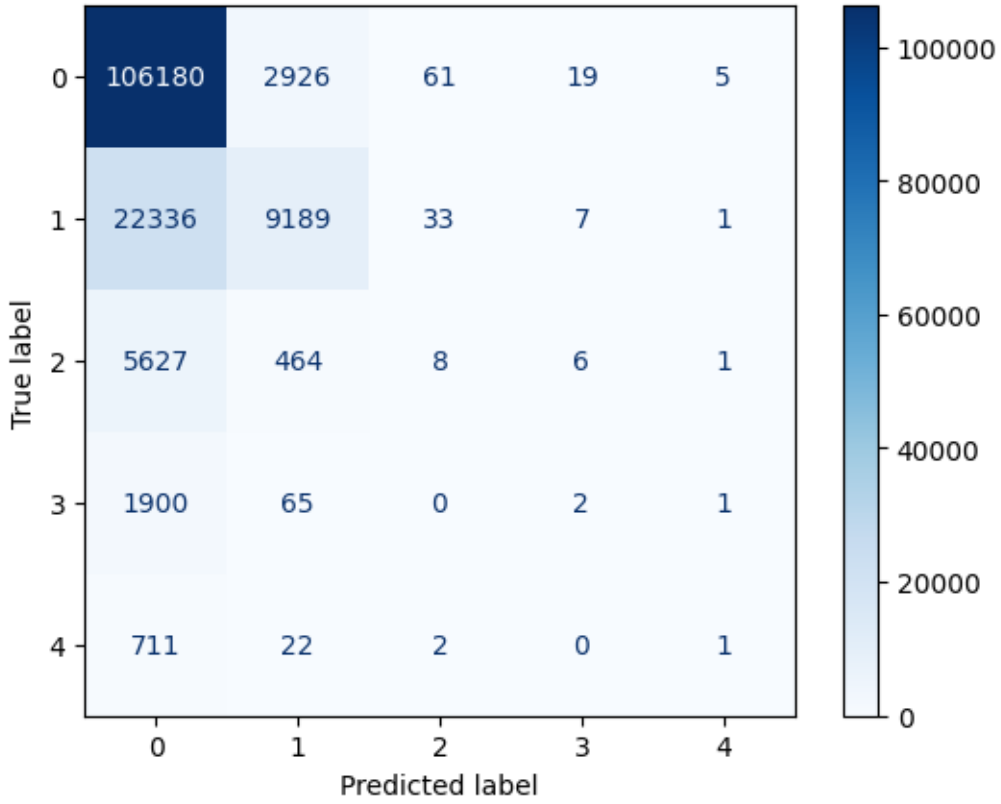
We therefore use these two estimators again to perform a prediction on the isolated test set. As a result, we obtain:

- The decision tree achieves a score of 0.7692 on the test set.
- The bagging classifier achieves a score of 0.76906 on the test set.

Hence, a decision tree performs slightly better on the test set as well.

Focus on Decision Tree

We display below an extract from the confusion matrix from a decision tree prediction on the test set.

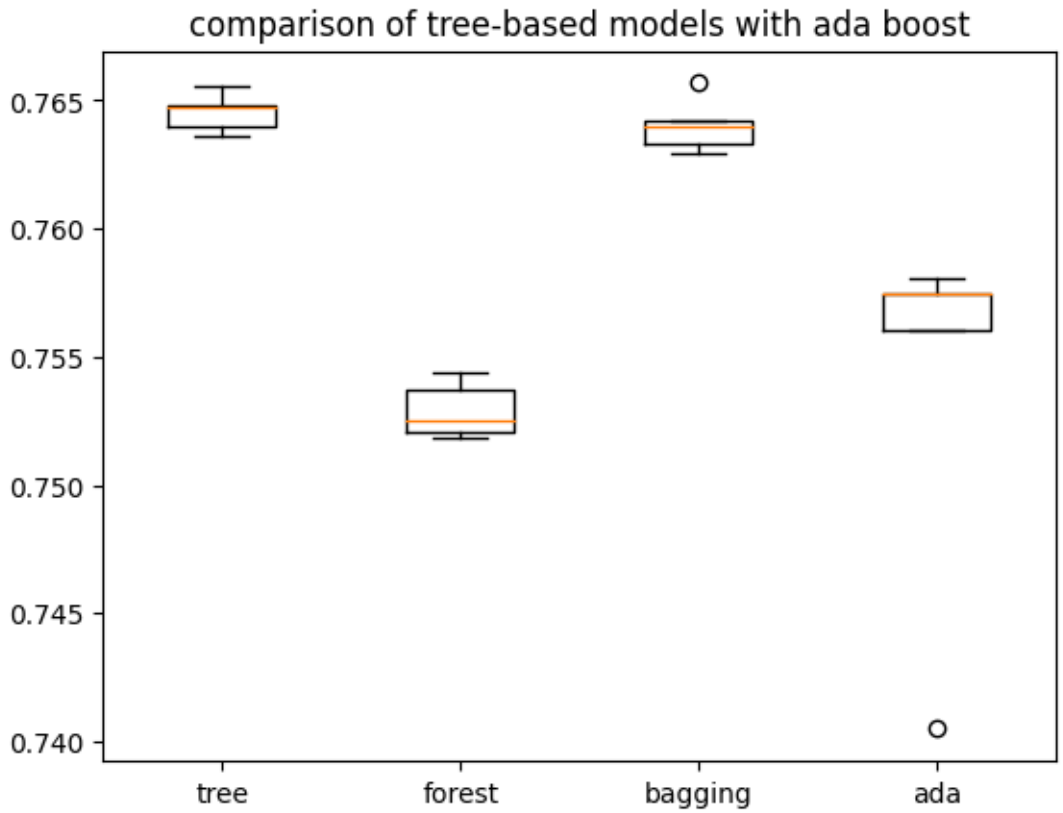


One may conclude that the accuracy is rather low. We must consider, though, that we are dealing with an immensely difficult, virtually impossible task of predicting the number of injured persons based on very rudimentary information about the vehicle type and the nature of the collision. With that in mind, a correct prediction of 9189 cases out of 31566 of the cases where 1 person got injured (row with index 1), for example, may seem not so bad after all. The prediction of a high number of injured persons, however, is significantly worse.

To achieve a better performance, it might make be sensible to consider the binary problem of whether any person was injured or not. This would render the problem structurally closer to the motivation of breast cancer prediction, for example.

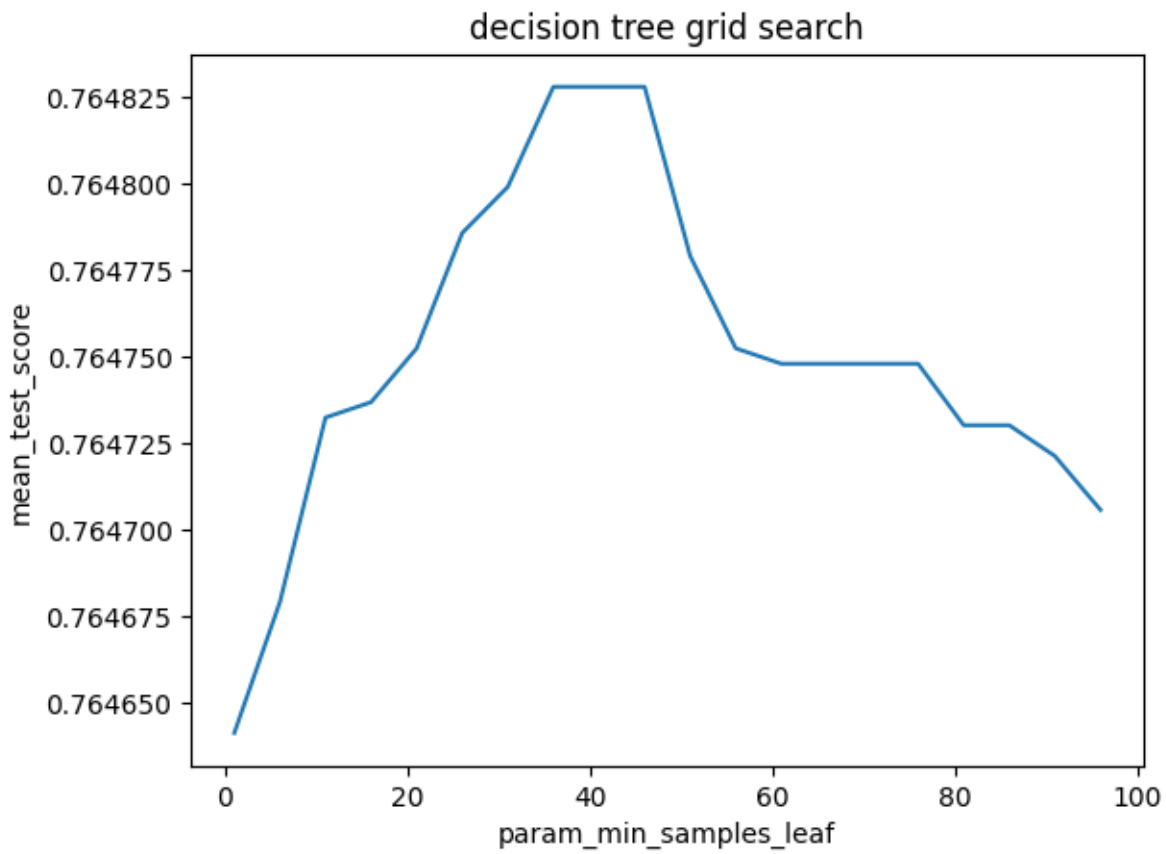
Boosting

We perform cross-validation with an AdaBoostClassifier using 100 decision trees with max depth set to 2. According to the documentation, an AdaBoost "begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases". In our case, however, we were unable to reach a satisfying performance when compared to the previously considered tree-based models:



Hyper-Parameter Tuning

At last, we return to an ordinary decision tree estimator to perform hyper parameter tuning. As mentioned before, one risk when using a decision tree is overfitting. This risk can be mitigated by the choice of certain parameters. We select the parameter min_samples_leaf, which controls the minimum number of samples required to be at a leaf node. Increasing this parameter may smoothen the model and in fact improve the performance with unknown data.



In order to set up this model in a robust way for any size of input data, it would be advisable to set the min_samples_leaf parameter as a float representing the ration of the min_sample_leaf relative to the overall volume of data, rather than setting it to a fixed integer as we did in this example.

Insights

- A decision tree is the best performer for the chosen machine-learning setting
- More advanced decision tree-based models, such as the ensemble models RandomForestClassifier or BaggingClassifier have not performed better than an ordinary decision tree. It might be possible to improve these models by increasing parameters such as the number of estimators, however, the run-time is already significantly longer for our chosen parameters, so any potential improvement appears to be expensive.
- Boosting methods such as an AdaBoost have also not succeeded in outperforming an ordinary decision tree. A GradientBoostingClassifier has been tried as well, however, we found it requires a prohibitively long runtime.
- Aiming for a multiclass prediction appears to be too demanding for the available information. A suggestion for further analysis would be to focus on the simplified binary problem: have persons been injured or not?
- Even though, in theory, a decision tree with a low number of minimum sample leaves should be more accurate, we find that a higher number is more accurate in our setting. This may be due to the mitigation of over-fitting with a higher number of minimum sample leaves.

References

We mostly relied on the scikit portal and in particular the user guide.

Moreover, we got inspiration from the datacamp courses referenced in the course, in particular the course Machine Learning with Tree-Based Models in Python.