



UNIVERSITÀ DI PISA

DOCUMENTATION

ACCIDENT CLASSIFIER

Lorenzo Mazzei

GitHub Repository: <https://github.com/Loremazze/DATA-MINING-PROGETTO>

Index

1 INTRODUCTION	3
2 SPECIFICATIONS	3
2.1 Dataset.....	3
2.2 Target Feature	3
2.3 Papers.....	3
2.4 Feature Description.....	4
3 PREPROCESSING	6
3.1 Feature Drop.....	6
3.2 Rows Drop	6
3.3 Feature Extraction.....	7
3.4 Handling Missing Values	7
3.5 Result	7
3.6 Categorical Values.....	7
4 Cross Validation	8
5 Training the Model	9
6 Feature Selection.....	10
7 Application.....	11

1 INTRODUCTION

The goal of the project is to create a classifier of road accidents. The classification is done after preprocessing steps and with the help of Cross-Validation to ensure the quality of the result. Users will be able to interact with an Application for this matter and choose the parameters of an accident to check if it would be '**lethal**' or '**non lethal**'.

The service also allows users not to pick some of the parameters in case they don't know what value to choose. In this case the application will use the mean of the value or its most frequent value or its mode, depending on the type of parameter.

2 SPECIFICATIONS

2.1 Dataset

The dataset was retrieved from Kaggle.com:

<https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents>

Number of records: about 2.8 million.

Number of features: 47.

2.2 Target Feature

The target feature for this problem is the column '**Severity**' which represent the possible risk of an accident if it would occur. This index varies in range depending on the country it is used in, in the case of this dataset it ranges from 1 to 4.

2.3 Papers

There are 2 papers regarding this dataset:

- The first one it's focused more on a general analysis of the dataset, specifying how the data were collected through API and the description of the features, as well as statistical results about the different cities of the US that are present in the records.
- The second one focuses on the classification problem, explaining the preprocessing steps that were done as well as data transformations. Among the models that were more efficient Logistic Regression and Deep Neural Networks stood out. The models were tested on the records regarding 6 different cities of the US and the **weighted average fscore** was reported to show the accuracy reached (around 85% with both of the methods).

<https://arxiv.org/pdf/1906.05409.pdf>

<https://arxiv.org/pdf/1909.09638.pdf>

2.4 Feature Description

ID = This is a unique identifier of the accident record

Severity = Shows the severity of the accident, a number between 1 and 4, where 1 indicates the least impact on traffic (i.e., short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e., long delay).

Start_Time = Shows start time of the accident in local time zone.

End_Time = Shows end time of the accident in local time zone. End time here refers to when the impact of accident on traffic flow was dismissed.

Start_Lat = Shows latitude in GPS coordinate of the start point.

Start_Lng = Shows longitude in GPS coordinate of the start point.

End_Lat = Shows longitude in GPS coordinate of the end point.

Distance(mi) = The length of the road extent affected by the accident.

Description = Shows a human provided description of the accident.

Number = Shows the street number in address field.

Street = Shows the street name in address field.

Side = Shows the relative side of the street (Right/Left) in address field.

City = Shows the city in address field.

County = Shows the county in address field.

State = Shows the state in address field.

Zipcode = Shows the zipcode in address field.

Country = Shows the country in address field.

Timezone = Shows timezone based on the location of the accident (eastern, central, etc.).

Airport_Code = Denotes an airport-based weather station which is the closest one to location of the accident.

Weather_Timestamp = Shows the time-stamp of weather observation record (in local time).

Temperature(F) = Shows the temperature (in Fahrenheit).

Wind_Chill(F) = Shows the wind chill (in Fahrenheit).

Humidity(%) = Shows the humidity (in percentage).

Pressure(in) = Shows the air pressure (in inches).

Visibility(mi) = Shows visibility (in miles).

Wind_Direction = Shows wind direction.

Wind_Speed(mph) = Shows wind speed (in miles per hour).

Precipitation(in) = Shows precipitation amount in inches, if there is any.

Weather_Condition = Shows the weather condition (rain, snow, thunderstorm, fog, etc.)

Sunrise_Sunset = Shows the period of day (i.e. day or night) based on sunrise/sunset.

Civil_Twilight = Shows the period of day (i.e. day or night) based on civil twilight.

Nautical_Twilight = Shows the period of day (i.e. day or night) based on nautical twilight.

Astronomical Twilight = Shows the period of day (i.e. day or night) based on astronomical twilight.

Type	Description
Amenity	Refers to particular places such as restaurant, library, college, bar, etc.
Bump	Refers to speed bump or hump to reduce the speed.
Crossing	Refers to any crossing across roads for pedestrians, cyclists, etc.
Give-way	A sign on road which shows priority of passing.
Junction	Refers to any highway ramp, exit, or entrance.
No-exit	Indicates there is no possibility to travel further by any transport mode along a formal path or route.
Railway	Indicates the presence of railways.
Roundabout	Refers to a circular road junction.
Station	Refers to public transportation station (bus, metro, etc.).
Stop	Refers to stop sign.
Traffic Calming	Refers to any means for slowing down traffic speed.
Traffic Signal	Refers to traffic signal on intersections.
Turning Loop	Indicates a widened area of a highway with a non-traversable island for turning around.

3 PREPROCESSING

3.1 Feature Drop

Starting from 2.8 million records, the first step was analyzing the features to understand if there were irrelevant ones. Upon inspection, these were the results:

- **Civil_Twilight,Astronomical_Twilight,Nautical_Twilight** = useless, they are just a different way to identify what already is expressed in the Sunrise_Sunset feature
- **ID** = the id of the accident
- **Zipcode** = useless in terms of accident severity
- **County,State,Country,City,Street** = just the places where the data were collected
- **Number** = number of the street
- **End Time** = we are just interested in the Start Time
- **Description,Timezone,Airport_Code,Weather_Timestamp** = useless in this classification problem
- **Wind_Direction** = This feature was not consistent: some of the values are indeed direction of the wind but others refers to the power of the wind
- **Bump,Give_Way,No_Exit,Roundabout,Traffic_Calming,Turning_Loop** = these features contain basically only one value (False) so they don't contribute to classification

3.2 Rows Drop

Data were collected in the arc of 6 years, I decided to focus only on 4 of these years to cut down the number of records in order to be able to compute the classification without too much computational time: 2016,2017,2018,2019.

Also upon inspection of the target class, it stood out that the classes were imbalanced, basically divided in two groups:

- Severity = 1 and Severity = 4: present in very few records, both of them (e.g 26000 records for the index = 4).
- Severity = 2 and Severity = 3: present for 90% of the records, the former for about 70% of them and the latter for about 20%.

Consulting the papers, the best decision resulted in dropping all the rows regarding the Severity Index equal to 1 and 4 and transforming the problem into a binary classification, between lethal and non-lethal accidents.

3.3 Feature Extraction

As said in the feature description, one of the features represent the timestamp of the accident. The format of the timestamp makes it unusable for classification purposes, so some transformations were carried out on it to retrieve the week of the day the accident happened:

- 1) Some of the timestamps contain also the milliseconds. The `split()` method was used to retrieve the part without milliseconds
- 2) transform all the timestamps according to the format `%Y-%m-%d %H:%M:%S`
- 3) Use the `weekday()` method on the timestamps to retrieve the day of the week the accident happened
- 4) Transform the integer values in the correspondent labels by using a simple mapping function

3.4 Handling Missing Values

Upon the inspection of the features, this is the result:

Severity	0
Start_Time	0
Start_Lat	0
Start_Lng	0
End_Lat	0
End_Lng	0
Distance(mi)	0
Number	505269
Side	0
Weather_Timestamp	9231
Temperature(F)	15420
Wind_Chill(F)	351617
Humidity(%)	16059
Pressure(in)	13027
Visibility(mi)	16547
Wind_Speed(mph)	82551
Precipitation(in)	385856
Weather_Condition	16736
Amenity	0
Crossing	0
Junction	0
Railway	0
Station	0
Stop	0
Traffic_Signal	0
Sunrise_Sunset	24

As said before, the features “**Number**” and “**Weather_Timestamp**” will be dropped, so the focus is on the other features.

Temperature, Humidity and Sunrise_Sunset = The rows with Nan values were dropped, this way the percentage of tuples lost only amount to 1% (the size was already reduced a lot by previous drops).

Wind Chill, Pressure, Visibility, Wind Speed, Precipitation = Mean of the feature values (these are all numerical attributes).

Weather_Condition = Mode of the feature (this is a categorical attribute)

3.5 Result

After all the preprocessing steps mentioned above, the dataset was reduced to:

- Number of **records** = 606668
- Number of **features** = 24

```
accidents_final_df.shape  
  
(606668, 24)
```

3.6 Categorical Values

At this point the only features remained to handle were **Start_Time** and **Weather_Condition**: using a simple **LabelEncoder** wouldn't be the optimal solution because these values are relevant on their own and not in an ordinal way. So the possible alternatives to deal with them were **OneHotEncoder** or the **get_dummies()** method, the outcome wouldn't have been much different in both cases anyway. The option chosen was the **get_dummies()** method.

4 Cross Validation

Since we have to build a model for the prediction of the samples, cross-validation was used in order to test different classifiers and choose one of the most optimal in terms of accuracy. The cross-validation was carried out using the **cross_validate** method in combination with a Pipeline. The Pipeline was used to prevent Data Leaking and also to organize the code in the best way possible.

Pipeline estimators:

- **scaling** -> **StandardScaler()**
- **clf** -> **classifier given to the function as a parameter**

The cross-validation was done using **StratifiedKfold** where **k** was chosen equal to **10**.

```
def execute_classifier(classifier):  
  
    y = preprocessed_accidents_df['Severity']  
    X = preprocessed_accidents_df.drop('Severity',axis=1)  
  
    pipeline_estimators = [('scaling',StandardScaler()),('clf',classifier)]  
    pipe = Pipeline(pipeline_estimators)  
    skf = StratifiedKFold(10, shuffle = True, random_state = 21)  
  
    results_validation = cross_validate(pipe, |  
    |X,  
    |y,  
    |scoring = {'f1score': make_scorer(f1_score),  
    |          'accuracy': make_scorer(accuracy_score)},  
    |error_score= 'raise',  
    |return_estimator = True,  
    |cv = skf,  
    |n_jobs = -1)|  
  
    metrics = results_validation['test_f1score']  
  
    print(results_validation)  
    print("Mean f1score: ",mean(metrics))
```

As it can be seen in the picture above, the **f1score** metric was used to check the accuracy of the different classifiers: being them tested on 10 subsets of data, the mean of 10 f1score that were obtained was done for each classifier.

Among the classifiers that were tested, here below the most succesful ones are reported:

DecisionTree:

- **Gini index -> 79%**
- **Entropy index -> 80%**

RandomForest:

- **Gini index -> 85,8%**
- **Entropy index -> 86%**

LogisticRegression:

- **Class_Weight = None -> 80%**
- **Class_Weight = 'balanced' -> 80%**

Adaboost:

- **n_estimators = 50 -> 81,9%**
- **n_estimators = 60 -> 82%**
- **n_estimators = 70 -> 82,3%**
- **n_estimators = 80 -> 82,7%**

K-NN:

- **k=10 -> 81%**
- **k=9 -> 82%**
- **k=8 -> 82%**
- **k=7 -> 82%**
- **k=6 -> 82%**
- **k=5 -> 82%**

5 Training the Model

After understanding which classifier was the best one, it was used and trained on the dataset to get the final model that would allow us to predict future samples. In particular, the **Pickle** module was used to save the **X_train,y_train,X_test** and **y_test** after the preprocessing steps. After this operation, the training of the model was carried out in another file where another Pickle module was used in order to retrieve the aforementioned set of data which were used to complete the training set.

```
import pickle

class MyClass():
    def __init__(self, param):
        self.param = param

def save_object(obj,filename):
    try:
        with open(filename, "wb") as f:
            pickle.dump(obj, f, protocol=pickle.HIGHEST_PROTOCOL)
    except Exception as ex:
        print("Error during pickling object (Possibly unsupported):", ex)

save_object(X_train,"X_train.pickle")
save_object(X_test,"X_test.pickle")
save_object(y_train,"y_train.pickle")
save_object(y_test,"y_test.pickle")
```

```
import pickle

class MyClass():
    def __init__(self, param):
        self.param = param

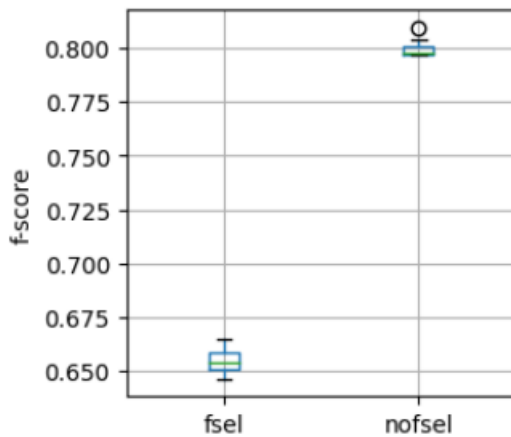
def load_object(filename):
    try:
        with open(filename, "rb") as f:
            return pickle.load(f)
    except Exception as ex:
        print("Error during unpickling object (Possibly unsupported):", ex)

X_train = load_object("X_train.pickle")
X_test = load_object("X_test.pickle")
y_train = load_object("y_train.pickle")
y_test = load_object("y_test.pickle")
```

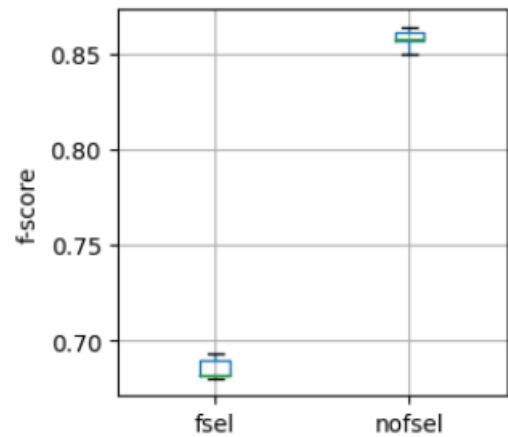
6 Feature Selection

Feature Selection was tried as a pipeline estimator to see if the accuracy of the classifiers could be improved. Here below the results are reported:

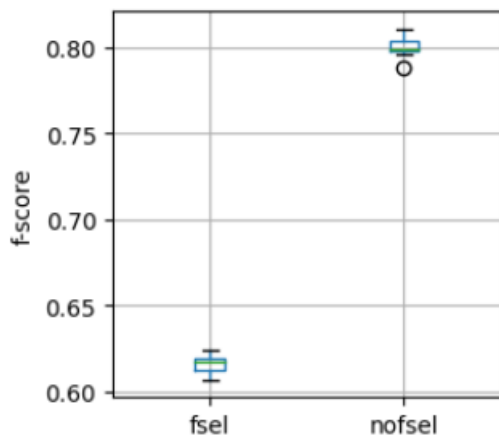
DecisionTree



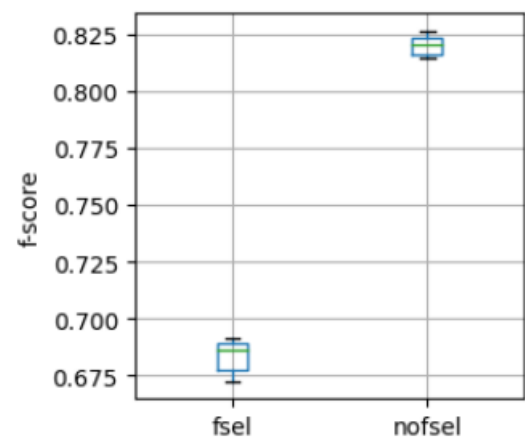
RandomForest



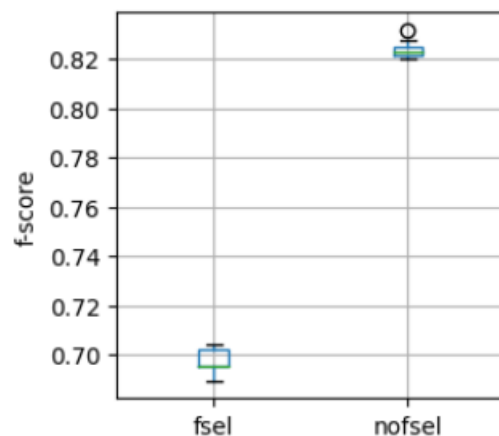
LogisticRegression



K-NN



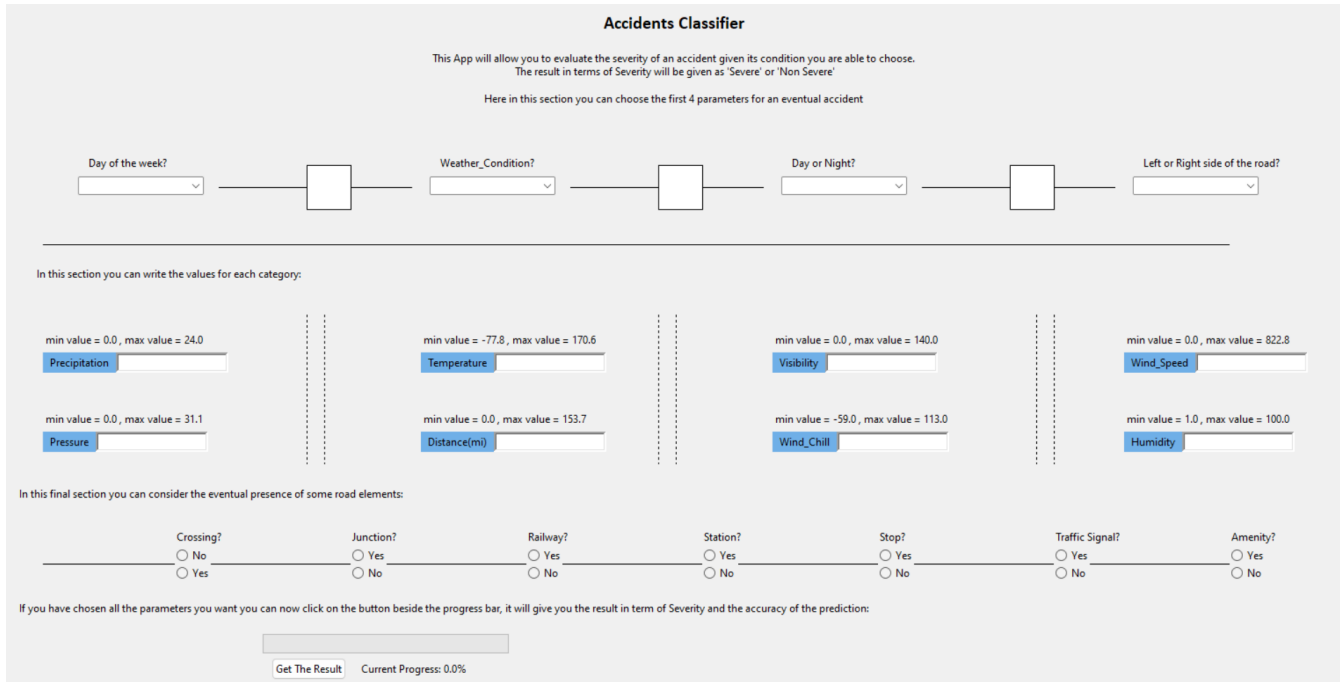
Adaboost



7 Application

Once the model was completed, it could be used to predict the outcome of new samples (therefore the severity in our case). An application was created to give to the users the possibility to test how risky an accident could be given the parameters' values chosen by them.

The application was realized with **Tkinter**. Here below the window is shown:



The application window is composed of **4 sections**:

- 1) first 4 parameters, created as **comboboxes** giving the user a set of possible option to choose. The fields were set to 'readonly' to prevent the user to write in them.
- 2) This part contains all the **Entry** fields where the user can write the value of the related parameters. Above each field the minimum and a conventional maximum value are shown to give context of the different ranges to the user.
- 3) In this section the remaining parameters are assessed with **radio buttons**, as it can be noted these features are the one that were binary, meaning they can only assume the 'true' or 'false' value.
- 4) This last part is composed by a **progress bar** and a **button**. Once the user has chosen all the parameters they want to check, they can press the button and the prediction will be done. Once the prediction process is completed, the progress bar will reach the 100% progress and a message will be shown to the user telling them the result.

The user can also decide not to choose the value of some of the parameters. In this case the **mean** value of that attribute will be assigned to it for the final prediction.