INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

MESTRADO EM ENGENHARIA INFORMÁTICA

Análise e previsão de acidentes rodoviários usando data mining

BRUNO MIGUEL FERREIRA TEIXEIRA

Outubro de 2019

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Análise e previsão de acidentes rodoviários usando data mining

Bruno Miguel Ferreira Teixeira

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Elsa Ferreira Gomes

Porto, Outubro 2019

Abstract

Road traffic crashes is an impactful problem in nowadays society, causing significant life and property losses. Due to the urbanization process across the world and the population's growth, the number of crashes is also increasing. Predicting a crash severity and cost is an important step to better understand which causative variables have more influence and therefore, implement prevention measures that can reduce the number of crashes. Road traffic crashes predictions is a complex problem due to the high number of independent causative variables that contribute to the event.

The used dataset contains crashes occurred in the State of Iowa in the recent years. Feature selection and data cleaning techniques are applied to improve the data quality and enhance the learning process.

Previous research on the road safety field applied approaches that led to unsatisfactory results. Recent studies based on more complex approaches like neural networks had better results. This document's work is based on deep learning, studying how the usage of deep neural networks can enhance previous results on road traffic crashes predictions taking causative variables as input. Various models are built using different optimization and activation functions. The evaluation is based on the comparison of these models.

Keywords: Road traffic crashes, Crashes prediction model, Machine learning, Artificial Neural Networks, Iowa.

Resumo

Os acidentes rodoviários representam um dos maiores problemas da comunidade atual, tendo um grande impacto social e económico. Além da enorme quantidade de feridos e mortos resultantes deste tipo de eventos (sendo mesmo considerada uma das maiores causas de morte a nível global, a maior em jovens adultos), a prevenção e consequentes custos de um acidente rodoviário representam também uma parte respeitável dos orçamentos de estado. Existe, um conjunto de variáveis envolvidas neste tipo de eventos que os tornam possíveis de prever e evitar, como por exemplo a existência de álcool, luminosidade no local e estado da estrada. Entender o impacto destas variáveis permite criar relações lógicas entre os seus valores e a gravidade e custos inerentes a um acidente, tornando possível a implementação de medidas de prevenção mais eficientes. Contudo e devido ao elevado número de variáveis a considerar, este é um problema complexo.

Apesar de ser um problema global, este documento foca-se num contexto mais específico, o do estado de Iowa nos Estados Unidos da América. O conjunto de dados utilizados foi recolhido pelo departamento de transportes do estado de Iowa e contém variáveis ambiente, gravidade e custo dos acidentes rodoviários ocorridos nos últimos anos. O número de registos é elevado, o que permite a existência de diversificados cenários. No entanto, estes dados contêm algumas falhas (valores não recolhidos) e, em alguns cenários, não se encontram balanceados. Diversas técnicas de pré-processamento de dados como limpeza e transformação destes são aplicadas de forma a ultrapassar este problema. A partir da análise dos dados é possível ainda identificar quais os campos que não representam interesse no contexto deste problema, procedendo-se com a sua remoção e consequente redução do tamanho do conjunto de dados.

A área de prevenção e previsão de acidentes rodoviários utilizando técnicas de *data mining* já foi explorada anteriormente. A aplicação de modelos mais clássicos (como modelos probabilísticos e baseados em procura) não obteve resultados totalmente satisfatórios. Nos estudos mais recentes, onde técnicas com maior poder computacional foram aplicadas (métodos baseados em otimização), os resultados foram melhores. Desta forma e tendo em consideração as conclusões dos estudos referidos na literatura, este documento pretende abordar como a utilização de *deep learning*, uma técnica de redes neuronais profundas e de elevado poder computacional, pode melhorar os resultados previamente obtidos. Para tal, são implementados diversos modelos para prever a gravidade e custo de um acidente com recurso a redes neuronais. A configuração dos modelos varia, sendo utlizados diferentes funções de custo e de ativação, de forma a explorar quais são as melhores abordagens a estes problemas.

De forma a otimizar o processo de desenvolvimento é também utilizada uma *framework* de *deep learning*, o Tensorflow. Esta *framework*, além de primar pela flexibilidade e capacidade de implementação de arquiteturas variadas, permite uma elevada abstração do processo de treino das redes neuronais, calculando dinamicamente qual a profundidade e largura da rede mais indicada. A sua utilização teve também por base a comunidade *open-source*, que garante a manutenção e otimização desta *framework* no futuro. Os resultados da utilização de *frameworks* no processo de treino de redes neuronais no contexto de acidentes rodoviários não são ainda conclusivos, sendo este um fator a ter em conta no desenvolvimento do projeto.

Os modelos desenvolvidos são depois comparados, utilizando métricas como Exatidão e AUC (*Area Under the Curve*), e com recurso a validação do tipo Holdout de forma a perceber se os resultados obtidos são válidos. São utilizados dois conjuntos de dados, um de treino e um outro de teste, para a avaliação da solução.

Palavras-chave: Acidentes rodoviários, Modelos de previsão de acidentes rodoviários, Machine learning, Redes neuronais, Iowa.

Table of Contents

1 Introduction 19

1.1 Context 19

1.2 Problem 20

1.3 Objectives 20

1.4 Value analysis 20

1.5 Approach 21

2 State of the art 23

2.1 Problem 23

2.1.1 Iowa’s context 24

2.1.2 Dataset 24

2.2 Value analysis 27

2.2.1 New concept development model 27

2.2.1.1 Opportunity identification 27

2.2.1.2 Opportunity analysis 27

2.2.1.3 Idea creation 28

2.2.1.4 Idea selection 28

2.2.1.5 Concept definition 29

2.2.2 Value 29

2.2.2.1 Value for customer 29

2.2.2.2 Perceived value 30

2.2.3 Value proposition 31

2.2.4 Canvas model 32

2.2.5 Value evaluation 32

2.3 Machine learning 34

2.3.1 Unsupervised learning 34

2.3.2 Supervised learning 35

2.3.2.1 Classification 35

2.3.2.1.1 Support Vector Machine (SVM) 36

2.3.2.1.2 K Nearest Neighbors (kNN) 36

2.3.2.1.3 Decision Trees 36

2.3.2.1.4 Naive Bayes 37

2.3.2.1.5 Artificial Neural Network 37

2.3.2.2 Regression 39

2.3.2.2.1 Logistic Regression 39

2.3.2.2.2 Linear Regression 40
2.3.2.2.3 Other methods 40
2.3.2.3 Deep learning 40
2.3.2.3.1 Optimization functions 41
2.3.2.3.1.1 GradientDescent 41
2.3.2.3.1.2 Adam 42
2.3.2.3.1.3 Adagrad 42
2.3.2.3.2 Activation functions 43
2.3.2.3.2.1 Softplus 43
2.3.2.3.2.2 Rectified Linear Unit (ReLU) 43
2.3.2.3.2.3 Sigmoid 44
2.3.2.3.2.4 Tanh 44
2.3.2.3.3 Deep learning frameworks 45
2.3.2.3.3.1 Tensorflow 45
2.3.2.3.3.2 Theano 45
2.3.2.3.3.3 Caffe 46
2.3.2.3.3.4 Torch 46
2.3.3 Data preprocessing 46
2.3.3.1 Data Splitting 47
2.3.3.2 Data Cleaning 47
2.3.4 Evaluation metrics 48
2.3.4.1 Classification 48
2.3.4.1.1 Precision 49
2.3.4.1.2 Recall 49
2.3.4.1.3 Specificity 49
2.3.4.1.4 Accuracy 50
2.3.4.1.5 Area Under the Curve (AUC) 50
2.3.4.1.6 F-measure 51
2.3.4.2 Regression 51
2.3.4.2.1 Metrics 51
2.3.4.2.2 Correlation Coefficient 52
2.3.4.3 Evaluation capacity 52
3 Analysis 55

3.1 Proposed hypothesis 55

3.2 Literature review 55
3.2.1 Related research 56
3.2.2 Deep learning frameworks comparison 58
3.2.3 Data analysis & preparation 58
3.2.4 Algorithms analysis 59
3.2.5 Evaluation metrics analysis 60

4 Design 63

4.1 Development environment 63

4.2 Proposed methodologies 66
4.2.1 Dataset cleaning 66
4.2.2 Model implementation 68

5 Implementation 71

5.1 Development environment construction 71

5.2 Data handling 72
5.2.1 Data acquisition 72
5.2.2 Data analysis 73
5.2.2.1 Severity analysis 73
5.2.2.2 Cost analysis 74
5.2.3 Data cleaning 76
5.2.3.1 Feature selection 76
5.2.3.2 Round crash date time 78
5.2.3.3 Removal of missing data 79
5.2.3.4 Removal of low impact data 81
5.2.3.5 Removal of outliers 81
5.2.4 Feature columns key to value transformation 82
5.2.5 RTC severity classification data definition 84
5.2.6 Training and test sample datasets 84

5.3 Model definition 84
5.3.1 Feature columns definition 85
5.3.2 Models' hyperparameter definition 86
5.3.2.1 RTC severity using classification setup 86
5.3.2.2 RTC cost using regression setup 88

5.4 Train and evaluate 89
5.4.1 Classification 89
5.4.2 Regression 91

6 Evaluation 93

6.1 Evaluation metrics, methodologies and hypothesis 93

6.2 RTC severity classification results 94
6.2.1 Results analysis 98
6.2.1.1 Other considerations 99

6.3 RTC cost regression results 100
6.3.1 Results analysis 102
6.3.1.1 Other considerations 103

6.4 Comparison with other studies 105

7 Conclusion 107
7.1.1 Future work 108

List of Figures

Figure 1 - World motor vehicle production (“World Production | ACEA - European Automobile Manufacturers’ Association,” 2016) 28
Figure 2 - EU road deaths reduction and target for injuries and deaths reduction (Presidency & Union, 2015) 30
Figure 3 - Canvas model 32
Figure 4 - Evaluation hierarchy 33
Figure 5 - Three-Layer Network by (Hagan et al., 1995) 38
Figure 6 - Representation of simple deep neural network (DZone, n.d.) 41
Figure 7 - GradientDescent action (Goodfellow et al., 2015) 42
Figure 8 - Softplus activation function (Goodfellow et al., 2015) 43
Figure 9 - ReLU activation function (Goodfellow et al., 2015) 44
Figure 10 - Sigmoid activation function (Goodfellow et al., 2015) 44
Figure 11 - Tanh activation function (“Neural activation functions,” n.d.) 45
Figure 12 - ROC curve for two different algorithms and the random classifier line 50
Figure 13 - Evaluation capacity scenarios (Goodfellow et al., 2015) 53
Figure 14 - Neural network width test (Kovalev et al., 2016) 64
Figure 15 - Neural network deepness test (Kovalev et al., 2016) 65
Figure 16 - Neural Network model – inspired by (Yuan et al., 2017) and (Çodur & Tortum, 2015) models 68
Figure 17 - Proposed implementation state diagram 70
Figure 18 – Data acquisition code snippet in Jupyter Notebook 73
Figure 19 - Human injuries and deaths data distribution 74
Figure 20 - Crash major cause distribution 74
Figure 21 - RTC cost distribution per dataset instance 75
Figure 22 - Boxplot to identify cost outliers 75
Figure 23 - Code to delete redundant and irrelevant features from dataset 78
Figure 24 - Optimized RTC crash time transformation....78
Figure 25 - Not optimized RTC crash time transformation 79
Figure 26 - Code snippet corresponding to the removal of missing data 80
Figure 27 - Code snippet corresponding to the removal of not supported data entries 81
Figure 28 - Feature keys' occurrences counter 81
Figure 29 - Boxplot after outliers removal 82
Figure 30 - Feature key to value correspondence example 83
Figure 31 - Feature key to description string transformation example 83
Figure 32 - Data handling to define RTC severity classification 84
Figure 33 - Dataset split into training and test samples following holdout method....84
Figure 34 - Model's categorical feature columns definition 85
Figure 35 - Model's numeric feature columns definition....85
Figure 36 - Tensorflow creation of a DNNClassifier model instance....87
Figure 37 - Tensorflow creation of a Random Forest model instance 88

Figure 38 - Tensorflow creation of a DNNRegressor model instance 89
Figure 39 - Definition of training and accuracy operation in Random Forest instance............ 89
Figure 40 - Random Forest model’s training process.. 90
Figure 41 – DNN model’s training process 90
Figure 42 - Creation of predictions.. 91
Figure 43 - Calculation of MSE and MedAE 92
Figure 44 - Accuracy fluctuation comparison on top classification models for RTC alternative 1 99
Figure 45 - AUC fluctuation on top classification models for RTC alternative 1..................... 100
Figure 46 - Regression model predictions comparison with actual values 103
Figure 47 - MedAD fluctuation between most performant regression models’ instances..... 104
Figure 48 - MedAD fluctuation on regression models with fixed parameters and different DNN structures 104

List of Tables

Table 1 - Dataset structure 25
Table 2 - CSEV values description 26
Table 3 - Longitudinal perspective of value 31
Table 4 - AHP evaluation table...33
Table 5 - Normalized AHP matrix 34
Table 6 - RTC prevention existing solutions 56
Table 7 - Programming languages comparison from (TIOBE, 2018) 63
Table 8 - Deep learning frameworks community size in Github by 15 September 2019...........65
Table 9 - Alternative 1 for RTC severity classification 67
Table 10 – Alternative 2 for RTC severity classification 67
Table 11 - Features identification as irrelevant, redundant and relevant 76
Table 12 - Relevant discrete fields' missing data identification 80
Table 13 – RTC severity neural network parameters variations 86
Table 14 - RTC severity neural network structure alternatives...87
Table 15 - Random forest parameter variations 87
Table 16 - RTC cost neural network parameters variations 88
Table 17 - RTC cost neural network structure variations..88
Table 18 - DNN results for alternative 1 of RTC 95
Table 19 - DNN results for alternative 2 of RTC 96
Table 20 - DNN results for alternative 1 of RTC with low impact data removed 97
Table 21 – DNN results for alternative 2 of RTC with low impact data removed 97
Table 22 - Random Forest results for alternative 1 of RTC 97
Table 23 - Random Forest results for alternative 2 of RTC 98
Table 24 - Regression results for variation without outliners and with all features 101
Table 25 - Regression results for variation without outliners and with reduced features 102
Table 26 - Regression results containing outliers 102

Acronyms and Symbols

Acronyms list

RTC	Road Traffic Crashes
EU	European Union
USA	United States of America
ETSC	European Transport Safety Council
IDT	Iowa's Department of Transportation
WHO	World Health Organization
CBT	Compulsory Breath Test
ANN	Artificial Neural Network
DNN	Deep Neural Network
NN	Neural Network
ReLU	Rectified Linear Unit
AHP	Analytic hierarchy process
MedAE	Median Absolute Error
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
ROC	Receiving Operating Characteristics
API	Application Program Interface
AUC	Area Under the Curve

Symbols list

∞	Infinite
∇	Del
$\Re$	All real numbers

1 Introduction

In this chapter it is possible to find a brief description of the project's problem, context, objectives and value. Is also referred the technological approaches to the project's development.

1.1 Context

This project's focus is the road crashes' causative variables analysis, having in consideration its influence on human life (injuries and fatalities) and related costs.

Road crashes have a meaningful impact on nowadays society. According to the World Health Organization (WHO) more than 1.2 million people die on the world's roads per year (World Health Organization, 2015a), being one of the major cause of death globally, and the main cause of death at age 15 to 29 years (World Health Organization, 2015b). Each of these crashes results not only in human injuries (or even fatalities) but also in financial costs. The United Nations and Road Safety estimated that global losses due to road traffic injuries total $518 billion and cost Governments between 1 and 3 per cent of their gross national product ("The United Nations and Road Safety," 2011).

With the population growth, vehicles' production is growing, increasing the risk of more traffic crashes and compromising road safety. Prevention measures can be implemented to reduce the number of accidents. Governments are aware of it and multiple global causes are currently taking place (e.g. the Decade of Action for Road Safety 2011-2020).

Even though traffic crashes can be considered accidents (a random event), the crashes are predictable and preventable (World Health Organization, 2015a). To prevent these crashes from happening a deeper understanding of its leading causes is required.

1.2 Problem

Factors like weather, road degradation and human behavior (presence of alcohol or drugs) have impact on the severity of a crash. While an accident is a random event, road crashes could be predictable and also preventable (World Health Organization, 2015a) and its severity is affected by the surrounding environment. The causative variables of a crash influences its severity and cost (Kononov & Janson, 2002).

Iowa's Department of Transportation (IDT) has been tracking the crashes occurring in the state, including the causative variables to understand how to reduce the number of fatalities. Besides the human injuries and deaths that collisions can cause, IDT also tracks the estimated cost and the date of a crash.

This research hypothesizes that the causative variables of a crash influence its consequences. This document's analysis and development are based on a dataset provided by Iowa's Department of Transportation.

1.3 Objectives

The impact of environmental factors (weather, road condition and driver's sobriety) on crash severity and cost modeling problem has been already studied and deeply analyzed in the past. Most of these studies take in consideration causative variables (Chong, Abraham, & Paprzycki, 2005) (Kumar & Toshniwal, 2015) or local conditions (Yuan, Zhou, Yang, Tamerius, & Mantilla, 2017).

Taking into consideration past studies on this topic, the primary objective of this work is to understand the impact of the crash factors in its severity and cost, using Iowa's crash data and data mining techniques and to use deep learning frameworks to predict severity and cost impacts and compare with other approaches. The development is focused in producing various models, and comparing the models using evaluation metrics like AUC to understand which model better performs on this context.

This objective can be achieved by analyzing Iowa's crash variables (prioritizing the variables that past investigations already identified as being the most impactful), resulting in the application of multiple data mining approaches later described in this document.

1.4 Value analysis

Road traffic crashes (RTC) are responsible for a high amount of injuries and fatalities. Also, governmental entities have costs associated with road safety. Drivers nowadays are worried about their safety (Shinar, 2017). The population growth is also contributing to the vehicles number growth, increasing the risk of road traffic crashes (WHO, 2014).

For these reasons, it is essential to identify the factors that influence crashes' severity and cost. This knowledge allows the implementation of prevention measures, which reduce the road traffic crashes number, severity and cost, thus increasing road safety and decreasing the associated costs.

1.5 Approach

This work focuses on the development of a model to predict road traffic crash severity and cost. The dataset obtained from Iowa’s Department of Transportation and supervised learning techniques like artificial neural networks were used for predicting the possible cause of the RTC.

Deep learning is used to overcome the problem’s complexity, as the number of input features is high, and to understand how this approach improves the RTC prevention result comparing with other researches. TensorFlow is used as the deep learning framework due to its flexibility, performance and open-source community.

Data processing is an important step in data mining. The used dataset contains missing data and potentially unbalanced data. For the missing data various strategies can be applied like removing the input or replacing the missing values by the most probable value. Some input columns are removed from the dataset using feature selection as these inputs are irrelevant to the problem’s context.

It is expected that the classification model proposed, using classification and regression approaches, is able to perform road traffic crash severity predictions and road traffic crash cost prediction.

2 State of the art

This chapter is meant to explain the problem approached in this document, its context and the value analysis. The used dataset is also contextualized in this problem's environment.

2.1 Problem

Traffic crashes are one of the most impactful issues humanity has to deal with as of the time of this writing, creating significant amounts of deaths, injuries, and costs over the last decades.

Considering World Health Organization reports, more than 1.2 million people die per year on the world's roads, making road traffic injuries a leading cause of death (World Health Organization, 2015a). The principal cause of death among aged teenagers and young adults (15 to 29 years) is traffic crashes (World Health Organization, 2015b). Road safety is a global issue that affects millions of people, transcending the transport sector, being a health, social, and economic problem (Leon, Cal, & Sigua, 2005) affecting multiple sectors and the economy as well. Besides human injuries and fatalities, it is important to consider the property and administration costs. It is estimated that global losses due to road traffic injuries total $518 billion. It represents between 1 and 3 per cent of governments gross national product ("The United Nations and Road Safety," 2011).

Considering this problem's impact, a global initiative named Decade of Action for Road Safety ("WHO | Decade of Action for Road Safety 2011-2020," 2017) was created and is underway by the time of this writing. Governments are making significant efforts to reduce the number of traffic crashes. These efforts are focused on prevention, therefore trying to reduce the number of traffic crashes.

In the traffic safety's domain, there is a need to distinguish between a crash and an accident. While an accident is a random event, crashes are influences by the environment (Shinar, 2017). There are various factors involved in a traffic crash that can be studied and identified. These factors, also mentioned in this document as causative variables, are directly related to the crashes frequency and severity (Kononov & Janson, 2002).

According to WHO, although road traffic injuries have been a leading cause of mortality in recent years, these crashes are predictable and also preventable (World Health Organization, 2015a). Predicting traffic crash causes and understanding the patterns associated with it is a crucial step to improve the society's safety.

2.1.1 Iowa's context

Although traffic crashes are a global problem, it is not plausible to focus on such a large scale.

For this reason, the scope of this document is restricted to the United States of America (USA) and more precisely to Iowa state context. Iowa is one of the USA's most concerned states about road safety as some programs and prevention measures have been implemented in the recent years.

One of the primary road safety programs is the "Zero Fatalities Program" ("Iowa Zero Fatalities | Homepage," n.d.). The Zero Fatalities program is a united effort from the Iowa Departments of Transportation, Public Safety and Public Health to bring attention to the factors that causes crashes, which can cause road fatalities ("Zero Fatalities," n.d.).

The program's leading focus is prevention. Since the implementation the number of fatalities has been reducing progressively over time, hitting an all-time lowest in 2013 with 317 fatalities.

2.1.2 Dataset

Iowa open data is a governmental initiative which consists in a portal to provides data about the state information divided by multiple topics as economy, education, transportation, and others. The provided data can be retrieved as spreadsheets, JSON files or by an API. Therefore, it can be used on local initiatives, open source projects, and others.

This document's dataset source is provided by the Iowa's open data portal. The dataset is named "Iowa crash data" and contains detailed information about crash conditions, including injuries, fatalities and causative variables from 2008 until 2016 occurrences. Iowa's Department of Transportation also keeps track of the crash property cost and date[1]. The dataset structure is described in Table 1[2].

[1] Dataset official source: http://data.iowadot.gov/datasets/crash-data-1

[2] The dataset structure description is not accessible by Portugal based addresses. Dataset structure provided by open-source community in the following snippet: https://pastebin.com/ui7dex3A

Table 1 - Dataset structure

Key	**Description**	**Value type**
CRASH_KEY	Crash id in Iowa's database	Id
CASENUMBER	Case number	Id
LECASENUM	Law enforcement case number	Id
CRASH_DATE	Date of crash	Complete date
CRASH_MONTH	Month of crash	Discrete data
CRASH_DAY	Day of week	Discrete data
TIMESTR	Time of crash	Hour and minutes
DISTRICT	DOT District	Discrete data
COUNTY_NUMBER	County	Discrete data
CITYNAME	City	Discrete data
SYSTEMSTR	Route with system	Reference
LITERAL	Derived literal description	Description
FRSTHARM	Frist harmful event	Discrete data
LOCFSTHRM	Location of first harmful event	Discrete data
CRCOMNNR	Manner of crash / collision	Discrete data
MAJCSE	Major cause	Discrete data
DRUGALC	Drug or alcohol related	Discrete data
ECNTCRC	Contributing circumstances	Discrete data
LIGHT	Light conditions	Discrete data
CSRFCND	Surface conditions	Discrete data
WEATHER	Weather conditions	Discrete data
RCNTCRC	Contributing circumstances - Roadway	Discrete data
RDTYP	Type of roadway junction / feature	Discrete data
PAVED	Paved or not	Discrete data
WZRELATED	Work zone related	Discrete data
CSEV	Crash severity	Discrete data
FATALITIES	Number of fatalities	Number
INJURIES	Number of injuries	Number
MAJINJURY	Number of major injuries	Number

Key	**Description**	**Value type**
MININJURY	Number of minor injuries	Number
POSSINJURY	Number of possible injuries	Number
UNKINJURY	Number of unknown injuries	Number
PROPDMG	Amount of property damage ($)	Number
VEHICLES	Number of vehicles involved	Number
TOUCCUPANTS	Total number of occupants	Number
REPORT	Report type	Discrete data
XCOORD	X Coordinate (UTM NAD 83 Zone 15)	Coordinate
YCOORD	Y Coordinate (UTM NAD 83 Zone 15)	Coordinate
OBJECTID	Object ID	Id
SHAPE	Shape	Geometry field

Most of the fields related with the causative variables are discrete data. This means that instead of having a description, it is attributed a class that represents a state in the dataset (e.g. for the *LIGHT* field, the value “1” represents “daylight” and value “2” represents “dusk”). The *CSEV* is one of the most important attributes, representing the crash severity. In Table 2 it is possible to find the *CSEV* values and corresponding descriptions.

The discrete fields also contain an empty / not reported state, which is also represented by a class (e.g. value “77” represents a “Not Reported” state for the *LIGHT* field). This dataset contains around 65530 RTC occurrences considering a timeframe of around 10 years until 2017.

Table 2 - CSEV values description

CSEV value	**Description**
1	Fatal
2	Major injury
3	Minor injury
4	Possible / Unknown
5	Property damage only

2.2 Value analysis

In this section it is possible to find the value analysis of the project, which follows Peter Koen new concept development model (NCD) (Koen et al., 2001). A detailed study of this project's development opportunities, ideas, value and concepts can be found in the sections below.

2.2.1 New concept development model

"The New Concept Development Model (NCD) provides a common language and definition of the key components of the Front End of Innovation" (Koen et al., 2001). The NCD is represented by the following five elements: opportunity identification, opportunity analysis, idea creation, idea selection and concept definition. The description for each of these elements on this project's context is described in the chapters below.

2.2.1.1 Opportunity identification

As referred earlier in the Problem section (see 2.1), traffic crashes are responsible for a significant rate of society's mortality rate and imply public costs. The world population is growing, and as cars become a more commonplace, the number of accidents tend to increase (Shinar, 2017). With more profound knowledge about the factors involved in traffic crashes, it is possible to invest in prevention, increasing people's safety. Multiple governments, mainly in developed countries, are implementing various prevention measures to reduce the number of RTC.

This opportunity identification resulted from a personal study to better understand the implications of specific factors on road safety.

2.2.1.2 Opportunity analysis

According to the European Automobile Manufacturers Association, the number of vehicles produced across the world is growing as seen in Figure 1. The number of cards growth on the roads, and considering that traffic safety is one of the great interest to most drivers today (Shinar, 2017), requires more effective prevention measures to be implemented soon.

Having in consideration the efforts that governments are placing worldwide to guarantee road safety, understanding the primary causes of traffic crash is one of the essential needs to prevent more crashes from happening.

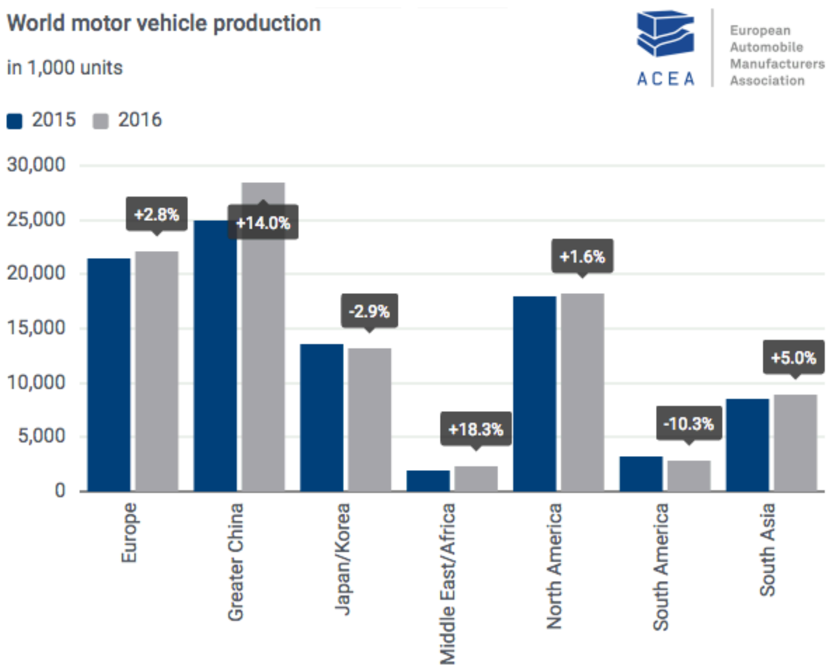


Figure 1 - World motor vehicle production ("World Production | ACEA - European Automobile Manufacturers' Association," 2016)

2.2.1.3 Idea creation

RTC events were already the target of many studies (as more precisely referenced in the 3.2.1 section). On these studies were applied different methods and data mining techniques to prevent more crashes from happening. These studies discussed this problem based on the crashes' causative variables and local conditions, showing reasonable to good results.

The Iowa's dataset contains the crashes' locations, causative variables, dates and costs. It's possible to apply various technological solutions (data mining techniques, new frameworks) to study each of these fields.

2.2.1.4 Idea selection

The selected idea is to analyze the crashes' causative variables and cost, using data mining techniques, supported in a deep learning framework (which are explained in more detail in the 2.3.2.3.3 section) and neural network usage.

Previous studies using Iowa's crash dataset already given particular attention to the urban data (Yuan et al., 2017). The use of artificial neural networks, which are indicated for complex problems (as explained in the 2.3.2.1.5 section) can reveal the relations that exist between roadway conditions, environmental characteristics and a crash severity and correspondent cost.

This research is mostly focused on the causative variables of Iowa's dataset to understand how these variables influence the crash severity and costs. Using Tensorflow as the chosen deep learning framework (explained in detail in 2.3.2.3.3.1 and compared with other frameworks in the 3.2.2 section) and therefore using deep neural networks, it is possible to explore how a deep learning framework's usage impacts the results.

2.2.1.5 Concept definition

Develop an algorithm capable of predicting a road traffic crash severity (possibility of injuries and fatalities) and cost using machine learning and different data mining approaches.

If the predictions are accurate, it is possible to have a more comprehensive understanding of how causative variables influence RTC. This knowledge allows the implementation of more precise prevention measures to increase road safety.

2.2.2 Value

Value can be defined as “different theoretical contexts as need, desire, interest, standard / criteria, beliefs, attitudes, and preferences” (Nicola, Ferreira, & Ferreira, 2012).

In the context of RTC, the principal value of this research is the road safety improvement, reducing costs and improving people’s health (preventing injuries or deaths). From the consumer’s concerns, traffic safety is of great interest to most drivers today (Shinar, 2017).

2.2.2.1 Value for customer

Drivers recognize the RTC dangers and impact on their safety. Following an american survey in the US in 2005, safety is the most important feature that americans value in their car (Shinar, 2017). Crashes harm public health, one of the most important fields people value. These events cause massive costs to often overburdened health care systems, consuming a high amount of resources (e.g. hospital rooms and beds) and resulting in significant losses of productivity and prosperity due to its costs. Also, these events causes deep social and economic repercussions (WHO, 2014).

Governmental entities are also interested in RTC prevention. As referenced on the sections above, according to the United Nations and Road Safety, global losses due to road traffic injuries are about $518 billion representing between 1 and 3 per cent of governments’ gross national product (“The United Nations and Road Safety,” 2011). Governments are also responsible to improve people’s safety and as an example, in the European Union (EU) various milestones were set to reduce the number of RTC over time (as seen in Figure 2).

European Transport Safety Council (ETSC) estimated that “if the EU countries had moved towards the 2020 road safety target through constant progress, the greater reductions in deaths in the years 2011-2013 would have raised the benefit to society by 4.6 billion Euro to about 23 billion Euro over those years” (Presidency & Union, 2015).

Due to the magnitude of these numbers, the reduction of RTC from prevention measures are interesting from a public and governmental perspective.

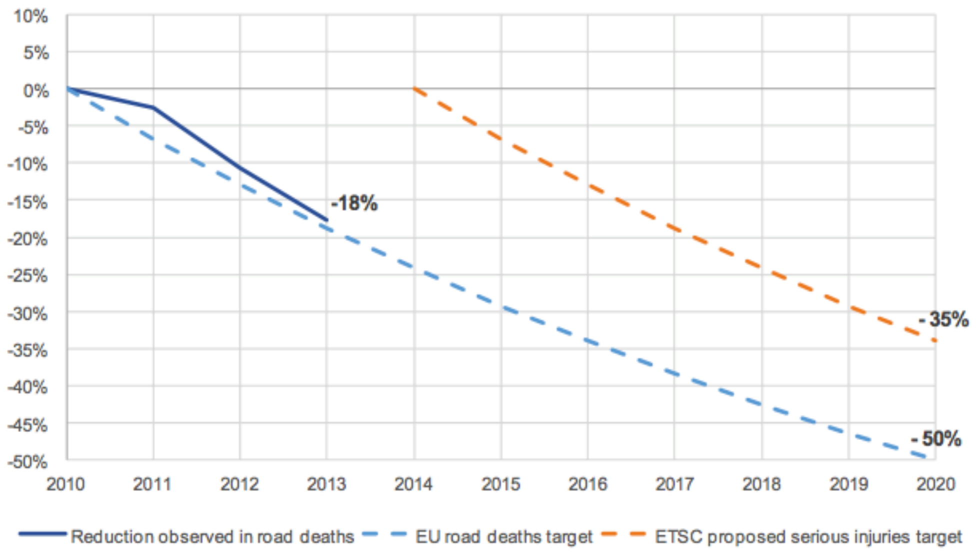


Figure 2 - EU road deaths reduction and target for injuries and deaths reduction (Presidency & Union, 2015)

2.2.2.2 Perceived value

In this section, it's referenced which sacrifices and benefits this research brings to the customer. "Perceived value is the consumer's overall assessment of the utility of a product based on perceptions of what is received and what is given" (Walker, Lutz, Park, & Schmalensee, 1988).

Road safety comes with a cost resulting from the prevention measures implementation, structural changes, campaigns and others that can arise.

As an example, alcohol is an of the most impactful causative variables on RTC severity. One of the most known prevention measures that resulted in a fatalities reduction was the compulsory breath tests (CBT). CBT intention is to reduce the amount of alcohol ingested by drivers, therefore reducing the risk of serious crashes. The implementation of this measure implies at least:

- Social costs;
- Infrastructural costs (more police agents on the road, protocol creation, …);
- Acquisition of equipment to analyze alcohol in blood;
- Campaign creation for drivers' sensitization.

Ireland was one of the countries which introduced random CBT to increase road safety. The legislation changed with the introduction of random breath testing. Were introduced tougher penalties for drunk driving and the disqualification periods for drunk driving range increased from 1 to 6 years. Since 2004 the number of full-time police officers in the Traffic Corps has increased from 500 to 1,200. The new legislation also brought more campaigns to public television and radios (Jost, Popolizio, Allsop, & Eksler, 2008). After CBT were implemented in

2007 (included in Ireland's "Road Safety Strategic Plan"), fatalities resulting from RTC reduced up to 7% (Jost et al., 2008).

Random breath tests (RBT) implementation has high costs as previously mentioned but the benefits are considerable, overlapping the initial costs. Other studies found that RBT operations are effective in reducing overall numbers of crashes. In New Zealand, the implementation of this prevention measure resulted in benefit cost ratios in the order of 26:1 (Wundersitz & Woolley, 2008).

Considering the costs of the program's implementation, it was still effective. The governmental entities probably saved more than it spent on the program's implementation and furthermore increased road safety for its citizens. In Table 3 it is possible to find the RTC prevention's longitudinal perspective of value.

Table 3 - Longitudinal perspective of value

	Benefits	**Sacrifices**
Before purchase	—	Research cost; Data acquisition cost.
Transaction	—	Development cost.

After purchase	Public sensitization;
	Support.
		Social costs;
		Infrastructural costs;
		Campaign creation costs;
		Maintenance costs.

After implementation	RTC prevention and reduction;
	Increase of road safety;
	Public health;
	RTC severity drop;
	Saving money (from RTC consequences costs).
		Infrastructural costs;
		Maintenance costs.

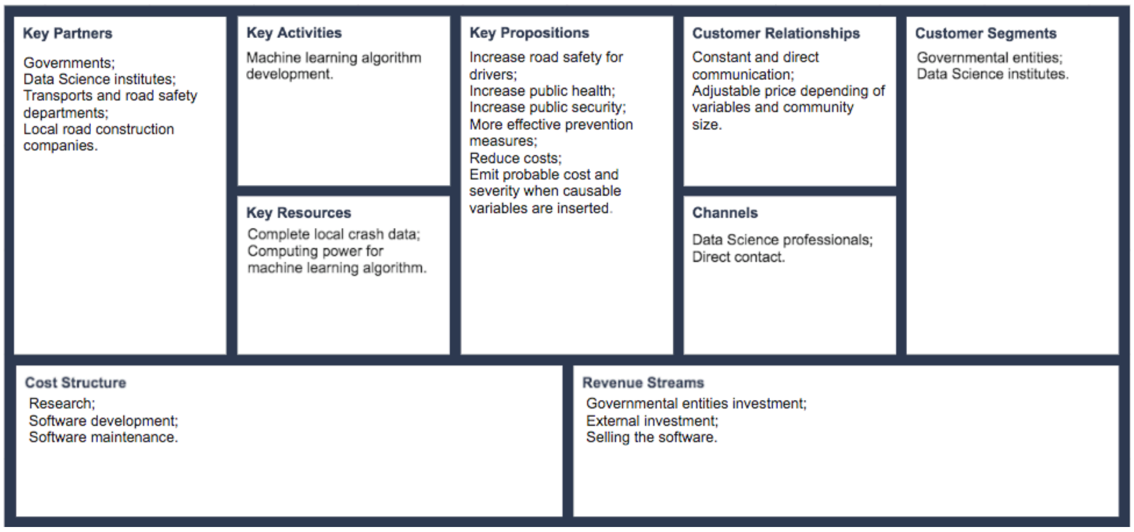
2.2.3 Value proposition

The value proposition of this research is to create an algorithm that given a combination of causative variables, can determine an RTC severity and cost with high accuracy and precision.

This way it is possible to provide more tools for governmental entities and the society to understand the significant crash factors, allowing the implementation of more accurate prevention measures. In consequence, it is expected to improve road safety, public health, legal and administrative costs with RTC.

The usage of a different development methodology comparing with previous studies is also a compelling characteristic of this research.

2.2.4 Canvas model



Key Partners
Governments;
Data Science institutes;
Transports and road safety departments;
Local road construction companies.

Key Activities
Machine learning algorithm development.

Key Resources
Complete local crash data;
Computing power for machine learning algorithm.

Key Propositions
Increase road safety for drivers;
Increase public health;
Increase public security;
More effective prevention measures;
Reduce costs;
Emit probable cost and severity when causable variables are inserted.

Customer Relationships
Constant and direct communication;
Adjustable price depending of variables and community size.

Channels
Data Science professionals;
Direct contact.

Customer Segments
Governmental entities;
Data Science institutes.

Cost Structure
Research;
Software development;
Software maintenance.

Revenue Streams
Governmental entities investment;
External investment;
Selling the software.

Figure 3 - Canvas model

As observed on the canvas model (Figure 3), using crash data and computing power for machine learning technology it's expected to significantly increase public safety and decrease current governmental costs with RTC.

The revenue is expected to come from external investments (from entities which are interested in improved local road prevention measures) and the software selling. Attending to the used technologies, it is possible to, in the future, upgrade the software and sell it as a set of services (with custom configuration depending on the input data).

The key partners are all related to data collection and road constructions to understand how the causative variables vary comparing with the data obtained.

2.2.5 Value evaluation

As referenced in the Canvas model chapter (see 2.2.4), one of the leading key propositions of this work is to allow more effective prevention measures implementation. The implementation of these measures enables road safety to increase, leading to the public health and security increase and cost reduction.

The analytic hierarchy process (AHP) is used for the value evaluation. This method structures the problem as a hierarchy and, following the Saaty's pairwise comparison scale, defines weights for the criteria (Mu & Pereyra-Rojas, 2017).

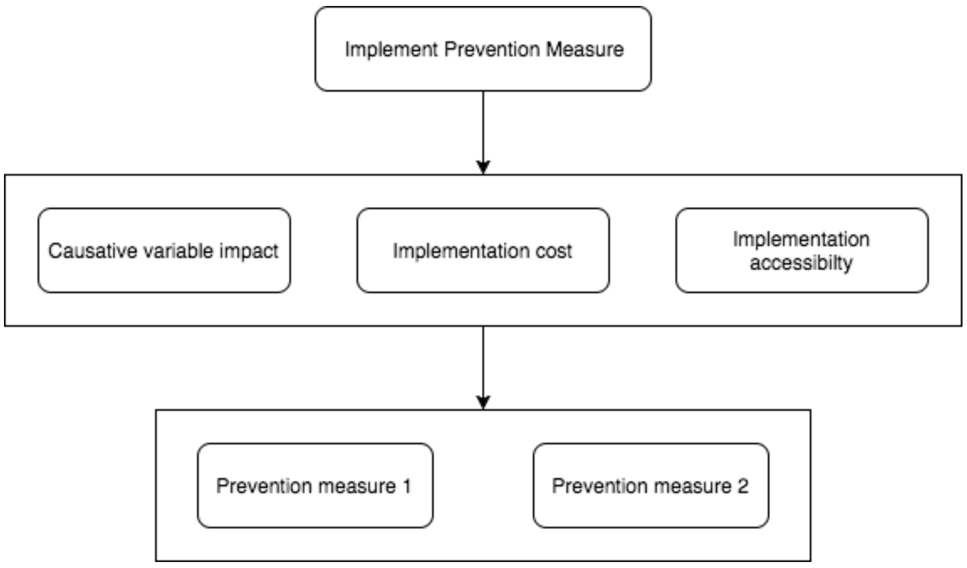

Implement Prevention Measure
Causative variable impact
Implementation cost
Implementation accessibilty
Prevention measure 1
Prevention measure 2

Figure 4 - Evaluation hierarchy

As observed in Figure 4, the first layer matches the primary objective, implementing a prevention measure. The involved criteria are:

- Causative variable impact – Understand which causative variables affects public health and costs the most;
- Implementation cost – A prevention measure has associated charges as stated in the Perceived value chapter;
- Implementation accessibility – If the means required for the prevention measure's implementation are accessible.

Based on this criterion it is possible to evaluate the best prevention measure to implement. Table 4 includes the attribution of weights for the criteria. Table 5 describes the normalized matrix with the final weights for each the criteria.

Table 4 - AHP evaluation table

	Causative Variable	**Cost**	**Accessibility**
Causative Variable	1	7	4
Cost	1/7	1	1/4
Accessibility	1/4	4	1
SUM	1.39	12	5,25

The causative variable impact is the most valuable feature since it directly influences public health and associated costs. The implementation cost is the less relevant feature as the investment in the prevention measures can be recovered on long-term cost reduction from RTC. The accessibility has a medium weight due to its direct impact on costs and structural changes required for the prevention measures implementation (e.g. the introduction of compulsory breath tests requires additional costs for equipment acquisition as already referred in Perceived value section).

Table 5 - Normalized AHP matrix

	Causative Variable	Cost	Accessibility	Weights
Causative Variable	0,7179	0,5833	0,7619	0,6877
Cost	0,1026	0,0833	0,0476	0,0778
Accessibility	0,1795	0,3333	0,1905	0,2344

2.3 Machine learning

Machine learning is a technique that allows computer programs to improve with experience automatically. This process happens due to the automated detection of meaningful patterns in data which allows computer programs to understand and balance past knowledge to classify new cases.

Learning is the process of converting experience into expertise or knowledge. The input to a learning algorithm is training data, which represents experience, and the output is some expertise. The expertise can then be used to solve other problems with unseen input (Ben-David & Shalev-Shwartz, 2014).

Machine learning problems are usually divided into two major fields (Gama, Ponce de Leon Carvalho, Carolina Lorena, Oliveira, & Faceli, 2017):

- Descriptive tasks – **Unsupervised learning**;
- Predictive tasks – **Supervised learning**.

Unsupervised learning is briefly mentioned on the 2.3.1 section while supervised learning is described in detail in 2.3.2 section as this document's work is related with prediction tasks instead of descriptive tasks.

2.3.1 Unsupervised learning

Descriptive tasks on machine learning are associated with the identification of structures related with dataset properties, allowing the algorithm's decision making enhancement and learning discovery (Gama et al., 2017).

In contrast to the supervised learning, where the desired output for a given input is known beforehand and the algorithm's performance changes to accomplish the expected result, unsupervised learning algorithm adapts its behavior based on the surroundings observations without being told to, which can be analogically compared to a “learning without a teacher” concept (Barlow, 1989).

The most known problem of unsupervised learning is clustering, where the output labels are not predicted and instead data is grouped in a meaningful way (Ben-David & Shalev-Shwartz, 2014). As there are no explicit output targets, unsupervised learning is mainly useful to find patterns in the input data, therefore used to better understand the dataset (Gama et al., 2017).

2.3.2 Supervised learning

In prediction tasks of machine learning, it is expected to have a dataset where the input and output values are known. In the supervised learning paradigm of machine learning (Gama et al., 2017), an external supervisor that knows the output label for the set of input values is simulated. Based on the previous knowledge learned from the training set is expected for the supervisor to evaluate and predict the outputs for unseen input data. The idea is the machine to “learn” from experience (the labelled values of the dataset), therefore being able to identify unlabeled examples with high accuracy (Erik G., 2014).

The training set consists (Erik G., 2014) of n pairs (x_i, y_i) where x represents the set of input values (a vector) which correspond to a single output label, the y value. The training set is processed by the learning algorithm which then creates a model to predict the output labels of the unseen cases (test set).

The test set is (Erik G., 2014) structurally similar to the training set. To evaluate the learning algorithm, its outputs labels y_i are hidden and compared with the predictions $\widehat{y_i}$ made on the test data.

Supervised learning tasks distinguish themselves depending of the data output label type (Gama et al., 2017):

- Discrete data (e.g. existence, or not, of fatalities in the road crash) – **Classification**;
- Continuous data (e.g. the road crash’s cost is 6300$) – **Regression**.

Both these techniques are explained in detail in the 2.3.2.1 and 2.3.2.2 chapters.

One of the most widely used and accepted methods based on learning data representations is Deep Learning, which takes advantage of deep structures to solve the proposed problems. This method can use supervised, semi-supervised and unsupervised techniques (Goodfellow, Bengio, & Courville, 2015), although in the context of this work it is approached mainly as supervised learning as specified in 2.3.2.3 section.

2.3.2.1 Classification

Classification is a supervised learning technique that assigns categories to cases. The algorithm learns from a dataset $\{x_1, \ldots, x_n\}$ where the input values have classes $\{c_1, \ldots, c_n\}$ already assigned and produces a function f, that predicts to which categories the unseen data belongs to. This technique is applied in situations where $y_i = f(x_i) \epsilon \{c_1, \dots, c_k\}$. In other words, $f(x_i)$ is related to discrete, non-ordered values (Gama et al., 2017).

The sections below expose some of the most relevant classification algorithms.

2.3.2.1.1 Support Vector Machine (SVM)

Support Vector Machine (SVM) is one of the most influential approaches to supervised learning (Boser, Guyon, & N. Vapnik, 1996) as outputs a class identity instead of calculating probabilities (like other techniques like logistic regression).

Linear SVM uses a score function that is linear and parametric in order to clarify the concept of margin maximization in a simplified context. Afterwards, SVM can also be referenced as non-linear and non-parametric due to the kernel introduction (Auria & Moro, 2008).

SVM looks for the hyperplane that best separates solvent from insolvent values (where the distance of the hyperplane from the closest points is the largest) according to a criterion. The criterion used by SVM is based on margin maximization (Auria & Moro, 2008).

During the training process, all the points have associated coefficients, which represent the strength of these points in the final decision function. The points that are closer to the hyperplane have coefficients greater than 0, while the rest of the points have coefficients equal to zero (Dominik Wisniewski & Wawezyniak, 2014).

The SVM usage provides a good generalization (which makes SVM robust even when training samples has some bias) and, instead of Artificial Neural Networks (ANN), delivers a unique solution (as the optimality problem is convex). A disadvantage of this technique is the lack of transparency of results (Auria & Moro, 2008).

2.3.2.1.2 K Nearest Neighbors (kNN)

K-nearest neighbor (kNN) is another non-probabilistic supervised learning algorithm that can be used either for classification or regression problem. Although it doesn't have a fixed number of parameters (being a non-parametric learning algorithm), kNN usually works better when applied to smaller features' vectors (Goodfellow et al., 2015).

This algorithm uses the nearest values of the input x in the dataset X to produce the y output, the class assigned to the unseen example. One of the biggest disadvantages of this algorithm is that it cannot learn that one feature is more discriminative than another (Goodfellow et al., 2015). kNN is simple to implement and applicable in complex problems (Gama et al., 2017) which can be considered as a major advantage.

2.3.2.1.3 Decision Trees

In decision trees algorithms, each node of the tree is associated with a region in the input space (Goodfellow et al., 2015). The tree is then divided in smaller regions, dividing a complex problem in simpler problems until it is ready to define the output. Decision trees can be used in classification and regression problems.

Each node of the decision tree is a function and has a conditional test (Goodfellow et al., 2015). In classification problems, decision trees algorithm uses the decision rules to assign a class as output.

This technique is usually robust, flexible and efficient (taking a top-down approach) but is not appropriated to deal with missing data (as missing values will leave empty nodes in the tree hierarchy) and is not very stable (Gama et al., 2017).

CHAID is a decision tree variation, where more than two states can be represented (which represents non-binary trees). This variation is well suited for larger datasets ("IBM Knowledge Center - CHAID classification tree," n.d.).

2.3.2.1.4 Naive Bayes

Naive Bayes is a probabilistic method that output a class based in prior knowledge (calculating the probability to each of the possible outputs). The class with a higher probability for the unseen data is the output (Gama et al., 2017). It is based in a conditional independency principle between variables to achieve the equilibrium (considering that the variables are discrete values). This technique is known to perform well in specific contexts like mail spam (Smola & Vishwanathan, 2010).

2.3.2.1.5 Artificial Neural Network

Artificial neural networks (ANN), commonly mentioned to as "neural networks" (NN), is a learning system inspired by human brain computation. It holds several neurons (the processing unit), a structure that helps to perform highly complex, nonlinear and parallel operations to developed knowledge.

The neurons have the capability of storing experiential knowledge. In an ANN it is possible to find various layers of neurons, connected between themselves. The relations between neurons are associated with weights (also named as "synaptic weights), which change during the learning process. Is possible to understand that (Haykin, 1999):

- In ANN, knowledge is acquired through a learning process. The learning process uses the various layers of neurons to enhance the knowledge during the learning iterations;
- Synaptic weights (or interneuron connection strengths) are used to store the acquired knowledge (Haykin, 1999).

The total entrance value u of a neuron can be calculated by (Gama et al., 2017):

$$u = \sum_{j=1}^{d} x_j \, w_j \tag{1}$$

Considering a neuron with d entrances, which weights are defined by w. The x value represents the vector with the input values.

The neuron output is obtained taking into consideration the connections' weights, a bias (a constant input weight), the input values and an activation function. Therefore, it is possible to understand the single input neuron model (Hagan, Demuth, & Beale, 1995):

$$a = f(wp + b) \tag{2}$$

In the neuron model, a represents the output and p the input of the neuron. The synapse weight is represented by w and the bias represented by b. To calculate the neuron output, an activation function, f, is used. This function represents the single input neuron (Hagan et al., 1995) as only one input is considered.

In an ANN, as various layers of neurons exist, one neuron has more than one input and output. Each of the neuron's inputs are weighted by the corresponding elements. The multiple input neuron (Hagan et al., 1995) is represented by:

$$a = f(Wp + b) \tag{3}$$

In which W represents the matrix of weighted inputs of a neuron. Each neuron calculates its output, which is received as input to the next layer of the ANN. The first layer of the ANN is named input layer and receives the initial data (da Silva, Hernane Spatti, Andrade Flauzino, Liboni, & dos Reis Alves, 2017).

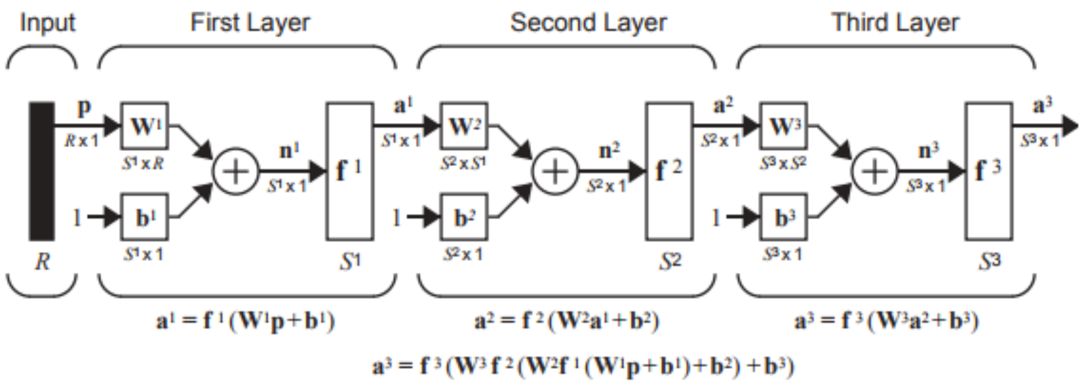


Figure 5 - Three-Layer Network by (Hagan et al., 1995)

The neural network net is also built using hidden layers, which output is not the network output but an input for the next layer (Gama et al., 2017). The network output, represented by a^3 in Figure 5, is calculated based on the ANN inputs and hidden layers outputs (Hagan et al., 1995) and is presented by the output layer.

During the learning process, the ANN also optimizes its predictions using loss functions. These functions measures the difference between the real output and the ANN's prediction (Smola & Vishwanathan, 2010).

The number of neurons composing the network's layers varies depending on the complexity of the problem being mapped by the network and on the quantity and quality of the available data. Usually, the number of neurons of the first hidden layer is immediately different from the number of neurons of the input layer (da Silva et al., 2017).

The ANN's architecture have multiple advantages (Haykin, 1999):

- **Input-Output Mapping** – Related with Supervised learning, the modification of the ANN weights occurs during the learning process, applying a set of labeled data (also named as training samples). For each input set, the neural network will adapt the relations' weight to minimize the difference between the desired response (the output label in the training samples) and the ANN response. The training is repeated during multiple iterations over the training samples until the ANN reaches a stable state where it is not detected significant changes in the neurons' relations weights;
- **Adaptability** – Neural networks adapt synaptic weights to surrounding environment, the data. Even if the neural network is already trained, it can be easily retrained to adjust the weights to unseen data. This allows the neural network to support a high level of feature complexity without disrupting the system performance and accuracy;
- **Contextual Information** – "Knowledge is represented by the very structure and activation state of a neural network. Every neuron in the network is potentially affected by the global activity of all other neurons in the networks. Consequently, contextual information is dealt with naturally by a neural network" (Haykin, 1999).

ANN benefit from considerable amounts of initial data (too much data can be a problem as later approached in this document) as it can learn from different, illustrative samples of the dataset. These advantages allow ANN to solve large-scale, complex problems as it can be applied in classification and regression problems.

2.3.2.2 Regression

Regression is a supervised learning technique used to predict continuous values. Considering a dataset where $f(x_i) \epsilon \Re$, $f(x_i)$ only assumes real and ordered values inside an infinite set (Gama et al., 2017).

In this chapter it is explained in detail some regression techniques, with emphasis on the linear regression technique.

2.3.2.2.1 Logistic Regression

Logistic regression is an algorithm that helps to select the class of a given input and uses the logistic sigmoid function. The logistic regression function can be represented by (Storkey, n.d.):

$$\sigma(b + x^T w) \tag{5}$$

Where b is a constant scalar and w is a constant vector. The classified value depends on the argument value. The greater the value, the higher is the probability of the classified value to belong the class 1. On the other hand, the lower is the argument value, the more probable is the classified value to belong to the class 0 (Storkey, n.d.).

2.3.2.2.2 Linear Regression

Linear regression defines the output as a linear function of the input (Goodfellow et al., 2015). The input parameters control the behavior of the system, as the weights of each feature affects the way the system outputs the prediction.

The feature's weight has proportional effect on the predict. If a feature's weight is negative, then increasing the value of that feature, decreases the prediction's value. When a feature's weight is zero, it has no effect on the predict (Goodfellow et al., 2015). The objective is to minimize the error and have the best prediction possible. Linear regression is a simple but limited learning algorithm that can work efficiently in continuous values prediction.

2.3.2.2.3 Other methods

SVM were initially developed to classification problems but were adapt to be used in regression cases by the introduction of an alternative loss function (Brereton & Lloyd, 2010).

The kNN algorithm can also be used to regression problems but it usually reveals problems to fit the data and give accurate predictions.

Decision Trees' output is usually an estimative (Gama et al., 2017) when applied to regression. The trees' leaf predicts a real number and not a class, looking to minimize the prediction square error through the training process. Each leaf prediction is based on the weighted mean of that node.

ANN, due to its architecture flexibility, can also be applied to regression. Is also possible to train ANN to support other algorithms like linear regression.

2.3.2.3 Deep learning

Deep learning is a machine learning technique to train deep ANN and its popularity grown up in the last years (Goodfellow et al., 2015) and is based on deep neural networks, which have two or more hidden layers as seen in Figure 6 (Gama et al., 2017).

Deep neural networks are more efficacious as the highest number of layers allows a better pattern recognition. The learning process consists on the first layers extracting high complexity characteristics from the input. The following layers then focus on more specific attributes from the first layers' output. This process allows the first layers to create a more abstract representation of the input, avoiding irrelevant data and enhancing the prediction model (Gama et al., 2017).

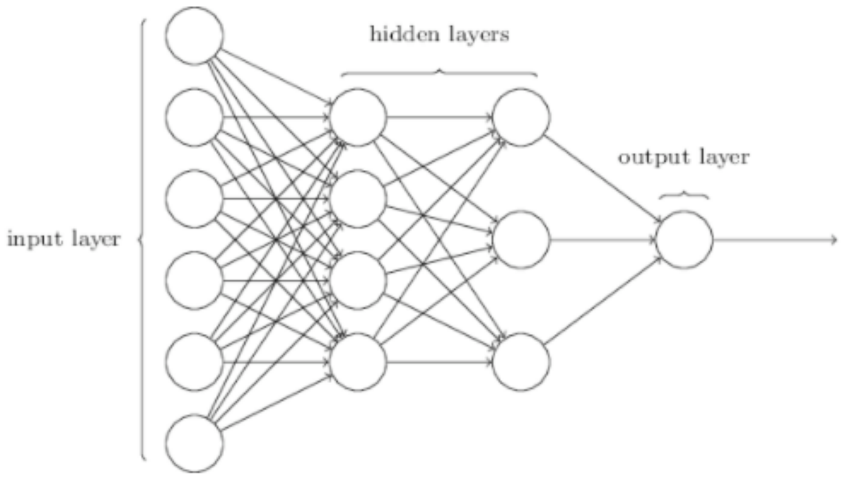


Figure 6 - Representation of simple deep neural network (DZone, n.d.)

This approach is known for reliable results solving problems related with image processing due to deep ANN's power and versatility, but is also indicated for complex problems with a high number of features as input (Goodfellow et al., 2015).

The usage of the previously mentioned activation functions (see 2.3.2.3.2 section, Activation functions) is highly recommended in deep learning, as it allows the learning process to be more precise and faster.

Deep learning approaches usually divide the dataset in a small number of sets, allowing the definition of a validation environment against which the trained algorithm is proved and where the output is already defined, following the supervised learning guidelines.

2.3.2.3.1 Optimization functions

Also known as training function, optimization functions are used to enhance the weights update and biases of neurons in the ANN in order to reduce the prediction's error. It is named as optimization as it helps to minimize the loss by the network's training process. The usage of the correct optimizer also enhances the learning process, helping to achieve more accurate models (Goodfellow et al., 2015).

2.3.2.3.1.1 GradientDescent

GradientDescent remains as the most common optimization algorithm for deep learning models today (Goodfellow et al., 2015). Is used for minimizing a convex function and analyze its convergence properties (Singer, 2016) as represented in Figure 7.

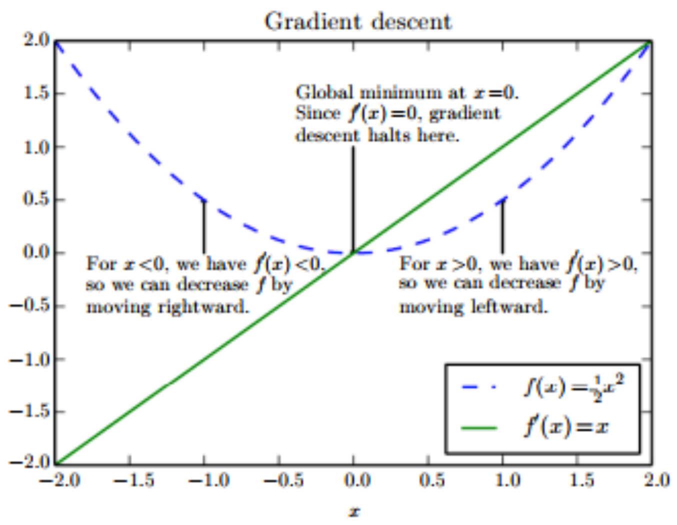


Figure 7 - GradientDescent action (Goodfellow et al., 2015)

GradientDescent searches for a stationary point, that represents the minimum loss possible to be applied in the learning process. The objective is to maximize or minimize a function, the objective function (Goodfellow et al., 2015).

2.3.2.3.1.2 Adam

Adaptive Moment Estimation (Adam) updates the learning rate for each parameter. It involves processes from others optimization algorithm like storing an exponentially decaying average of past squared gradients (usually associated with other less used optimizers like Adadelta and RMSprop) and keeping an exponentially decaying average of past gradients (usually associated with Momentum) (Ruder, 2016). This function usually has good results when compared with to other adaptively learning-method algorithms (Ruder, 2016).

2.3.2.3.1.3 Adagrad

Adagrad is a function that adapts the learning rate to the parameters, performing larger updates for infrequent parameters. Performs better when dealing with sparse data. (Ruder, 2016).

One of the main benefits of this optimization function is that it eliminates the need to manually tune the learning rate. Adagrad will update the learning rate for every parameter θ_i at every time step t. Being $g_{t,i}$ the gradient of the objective function with respect to the parameter θ_i at time step t (Ruder, 2016) and where J represents the error:

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}) \tag{4}$$

Adagrad's main weakness is its accumulation of the squared gradient in the denominator, causing the learning rate to shrink and become too small (Ruder, 2016), eventually causing the algorithm to stop learning during the training process.

2.3.2.3.2 Activation functions

The activation function associated with a neuron determines its output for a given input. Neurons in ANN behave like switches, either activating or not upon the learning process. The learning process is faster using activation functions (Gama et al., 2017).

These functions play a major role in the ANN training. In this section are described the considered activation function for this project's development.

2.3.2.3.2.1 Softplus

The softplus function is a commonly found function in ANN problems due to its range being $[0, \infty]$ (Goodfellow et al., 2015). Its function its represented by $\log(1 + \exp(x))$ as seen in Figure 8.

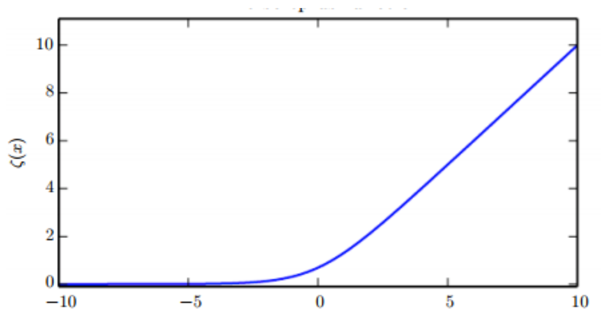


Figure 8 - Softplus activation function (Goodfellow et al., 2015)

2.3.2.3.2.2 Rectified Linear Unit (ReLU)

ReLU is the default activation function for most feedforward neural networks (Goodfellow et al., 2015). The ReLU range is $[0, \infty]$, being defined by the $f(x) = \max(0, x)$ formula as seen in Figure 9.

The output of a linear transformation when this function is applied results in a nonlinear transformation. The function still remains very close to linear, as is a piecewise linear function with two linear pieces. Rectified linear units are nearly linear and preserve many of the properties that make linear models easy to optimize (using gradient based methods) (Goodfellow et al., 2015).

Due to the function's characteristics where weights are not adjusted during descent, neurons can stop responding. This problem is called dying ReLU problem ("Neural activation functions," n.d.).

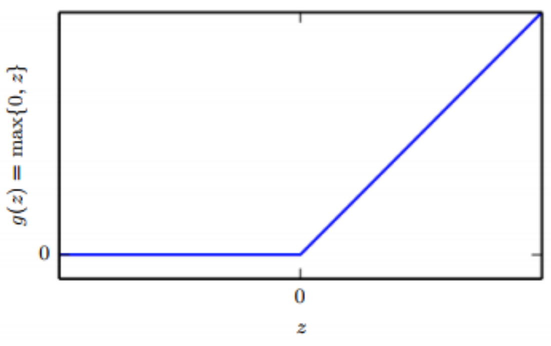


Figure 9 - ReLU activation function (Goodfellow et al., 2015)

2.3.2.3.2.3 Sigmoid

The logistic sigmoid activation function is a non-linear function, as represented in Figure 10. It saturates to a high value when the input is very high, saturates to a low value when the input is very low, and is strongly sensitive to near 0 input values (Goodfellow et al., 2015). The usage of an appropriate cost function can reduce the impact of the high saturation caused by the sigmoid function.

This function can be mathematically represented by $\sigma(x) = 1/(1 + \exp(-x))$, transforming a real value into a value between 0 and 1 (Goodfellow et al., 2015).

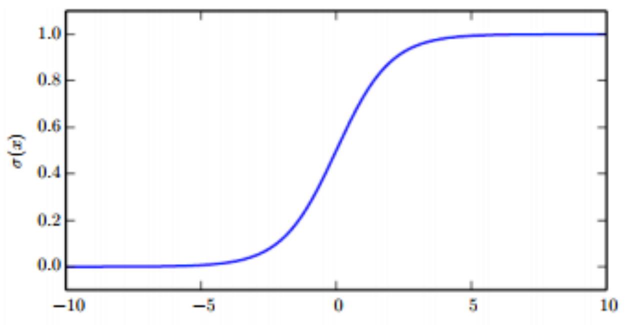


Figure 10 - Sigmoid activation function (Goodfellow et al., 2015)

2.3.2.3.2.4 Tanh

The tanh function is mathematically represented by the $\tanh(x) = 2\sigma(2x) - 1$, where σ represents the sigmoid function (mentioned earlier in section 2.3.2.3.2.3). Figure 11 holds this function's representation.

While the sigmoid function consisted in transforming a real value in a value between 0 and 1, tanh function transforms a real value in a value between -1 and 1. For this reason, training a ANN with tanh is usually easier and, in most scenarios, performs better than the logistic sigmoid (Goodfellow et al., 2015).

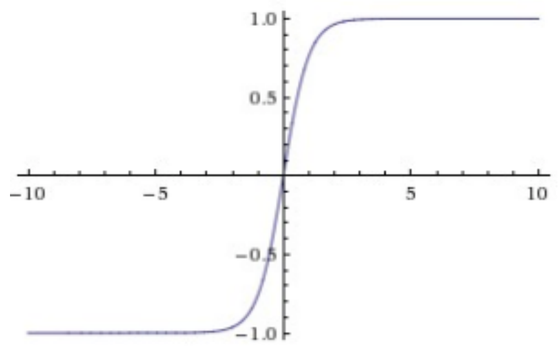


Figure 11 - Tanh activation function ("Neural activation functions," n.d.)

2.3.2.3.3 Deep learning frameworks

Deep learning implementation have changed several application domains, being important to areas like computer vision, speech recognition and solving complex problems (Bahrampour, Ramakrishnan, Schott, & Shah, 2015). Due to the importance of this method, the number deep learning frameworks growth.

The frameworks usually create a wrapper around the ANN implementation, abstracting the low-level implementation. Using a deep learning framework allows to (Li, Johnson, & Yeung, 2017):

- Easily build big computational graphs;
- Easily compute gradients in computational graphs;
- Run it all efficiently on GPU.

This section describes each of the most used deep learning frameworks.

2.3.2.3.3.1 Tensorflow

Tensorflow is an open-source project, developed and maintained by Google. Its development is C++ based alongside with Python APIs (when developing in Tensorflow, most of the interaction is with Python).

Tensorflow is known for being extremely flexible (Bahrampour et al., 2015) as it has automatic differentiation and parameter sharing capabilities. This property allows the implementation of multiple architectures. It is one of the most popular open-source projects in Github[3].

2.3.2.3.3.2 Theano

Theano[4] is a symbolic manipulation library used for machine learning algorithms development. It uses a high-level description language similar to a functional language which reduces the implementation time. The programming language used is Python.

This deep learning framework benefits from a large community. Theano is a general mathematical expression library and may have a relatively steep learning curve for writing

[3] Github repository: https://github.com/tensorflow/tensorflow
[4] Github repository: https://github.com/Theano/Theano

efficient code and debugging (due to the low-level implementation of it). Other libraries like Keras and Lasagne have been developed on top of Theano, which are specifically tailored for deep learning algorithm, allowing fast experimentation of the well-known methods. (Bahrampour et al., 2015).

2.3.2.3.3.3 Caffe

Caffe[5] is a deep learning framework developed by Berkeley AI Research and community contributors in C++. It is built with expression, speed and modularity in mind. It separates the network architecture from its implementation, allowing multiple architectures implementation and flexibility (Bahrampour et al., 2015).

Caffe is used not only in large industrial and multimedia applications but also in speech and computer vision fields and in research projects (Pittaras, Markatopoulou, Mezaris, & Patras, 2017).

2.3.2.3.3.4 Torch

Torch[6] is built using Lua and contains mature machine learning and optimization packages (Bahrampour et al., 2015). To optimize the ANN training, Torch contains multi-GPU support and parallelizing packages.

It is used in machine learning, computer vision, signal processing, parallel processing and multimedia projects. Torch allows different ANN topologies implementation with great flexibility and optimization.

2.3.3 Data preprocessing

There are many factors which can influence a machine learning algorithm accuracy. One of which is the data quality (Kotsiantis & Kanellopoulos, 2006). Data should be reliable, complete and consistent (Han, Kamber, & Pei, 2012). Inaccurate, incomplete (missing fields) or inconsistent data can reduce the learning process performance.

Data preprocessing is a crucial step which takes a significant time in a machine learning solution development. During preprocessing process, data should be treated to guarantee that the training set is made up by relevant information.

The following steps are standard in the preprocessing process (Han et al., 2012):

- Data Splitting – Obtaining a reduced representation of the dataset to feed the algorithm. Described in the 2.3.3.1 section;
- Data Cleaning – Data analysis, missing data and feature selection. Explained in detail in the 2.3.3.2 chapter.

[5] Github repository: https://github.com/BVLC/caffe
[6] https://github.com/torch/torch7

A machine learning algorithm's performance can have a significant variation depending on the data preprocessing process (Kotsiantis & Kanellopoulos, 2006) and it is a crucial step for an accurate result.

2.3.3.1 Data Splitting

Datasets can have a high number of examples, turning the learning process less efficient (as it takes more time to train the algorithm without significant advantages). Applying data reduction techniques allows the input data to have less volume but still maintain its integrity, producing the same analytical results (Han et al., 2012). Some of this strategies are (Han et al., 2012):

- Dimensionality reduction – Remove irrelevant attributes from the dataset. The objective is to find the minimum set of attributes such the final output is similar or better than the original result with all the attributes;
- Numerosity reduction – Reduce the number of examples provided to the learning algorithm by clustering and sampling approaches.

Data splitting allows the process to speed up, while maintaining the integrity and reliability of the final output.

2.3.3.2 Data Cleaning

Data cleaning is the process which enhances the input data quality turning incomplete, noisy and inconsistent data into reliable data for the learning process of an algorithm.

Data cleaning's first step is the discrepancy detection (Kotsiantis & Kanellopoulos, 2006). Several factors can affect the data quality like data decay, human error in data entry or data representation inconsistency. Also, irrelevant data can compromise the data quality. For this reason, one of the crucial steps of data cleaning is the **feature selection** step. During this stage, features should be identified as (Han et al., 2012):

- Relevant – Influence the output and have high importance for the learning process;
- Irrelevant – Doesn't influence the output;
- Redundant – When one feature can take the role of another.

This analysis depends directly on the problem's context. New features can also be created using the original features using a process called **feature construction** (Han et al., 2012). This approach is applied primarily to supervised learning problems.

In classification contexts, it may be relevant to guarantee that both output classes are balanced using **data balance** techniques (e.g. the number of examples of a class is not considered significant compared with the other class). Several methods might be applied to overcome data balance issues (Gama et al., 2017):

- Resize the dataset (ignoring some examples of the class which has bigger size);
- Use different costs to classify each of the classes;
- Create a class model.

Unbalanced data can influence the learning algorithm to output with greater tendency the class with most examples in the dataset (Gama et al., 2017). In classification approaches, the accuracy of the minority class is harmed by unbalanced data. For this reason, it is important not to assume that data is evenly distributed (Ali, Shamsuddin, & Ralescu, 2015), even though the most robust models should consider the initial data distribution.

The input data can also contain missing values (either missing fields or fields with a default value or state). **Missing data** influence negatively the learning process. Some approaches to complete the data (Gama et al., 2017) (Han et al., 2012):

- Delete the tuples where missing values are present – Usually not very effective as the deleted data can be useful for the learning process;
- Fill the missing values manually – When the dataset is large, this is a time-consuming task;
- Automatically fill the missing values with a constant – Replace missing values by a default value. Can negatively influence the learning process;
- Automatically fill the missing values with the most probable value – Using the mean or median (depending on the field data type) to recognize which value is more probable.

Not always missing data implies a dataset error (Han et al., 2012). In some contexts, missing values can be intentional and should be considered by the learning algorithm (e.g. a user, while completing the form, refused to insert private data).

2.3.4 Evaluation metrics

In this section it is described the evaluation metrics for machine learning problems, mainly related with classification and regression scenarios.

Evaluation metrics are used to understand if the machine learning algorithm is precise and accurate in its predictions. Usually, the metrics are applied in test sets (unseen data) and not in the training set (Baranauskas & Monard, 2003).

2.3.4.1 Classification

The error rate in classification algorithms corresponds to the wrong class prediction rate (Gama et al., 2017). In a test set of n objects, the error rate of $\hat{f}$ is calculated comparing the known class y_i for the x_i object with the predict $\hat{f}(x_i)$ class, which can be mathematically represented by (Gama et al., 2017):

$$err(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{f}(x_i)\right) \qquad (5)$$

The error rate values vary between 0 and 1, where values closer to 0 means a better prediction algorithm.

In classification problems involving two classes, it is possible to measure the algorithm's performance using the following metrics (Gama et al., 2017):

- True Positive (TP) – Matches the number of examples of positive class which were correctly predicted;
- True Negative (TN) – The number of negative class elements correctly predicted;
- False Positive (FP) – Corresponds to the number of false positives (i.e. the number of elements predicted with positive class but having negative class);
- False Negative (FN) – Matches the number of false negatives (i.e. the number of elements predicted with negative class but having positive class).

The measures described in the following chapters use these metrics to evaluate the algorithm's performance.

2.3.4.1.1 Precision

Precision refers to the proportion of true positives in the total predictions. Calculated using the following expression (Gama et al., 2017):

$$\frac{TP}{TP + FP} \qquad (6.1)$$

2.3.4.1.2 Recall

Recall corresponds to the prediction success rate of the positive class. Represented by (Gama et al., 2017):

$$\frac{TP}{TP + FN} \qquad (6.2)$$

2.3.4.1.3 Specificity

The specificity evaluation measure corresponds to the prediction's success rate of the negative class and is represented by the following mathematical expression:

$$\frac{TN}{TN + FP} \qquad (6.3)$$

2.3.4.1.4 Accuracy

Also named as success rate, the algorithm's predictions accuracy is calculated by diving the success predictions by the total amount of predictions, which are represent by n in the following equation:

$$\frac{TN + TP}{n} \qquad (6.4)$$

2.3.4.1.5 Area Under the Curve (AUC)

It is possible to evaluate a classification algorithm performance (two-class classification problem) using a receiving operating characteristics (ROC) analysis. The ROC is a bi-dimensional graph where are represented the false positive rate (in the x axis) and false negative rate (in the y axis) (Gama et al., 2017).

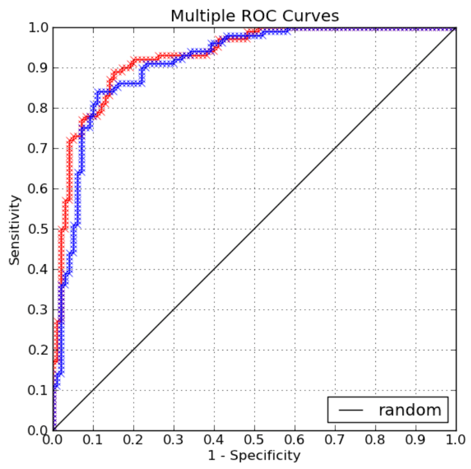


Figure 12 - ROC curve for two different algorithms and the random classifier line[7]

The random classifier is defined by a line connecting the $(0,0)$ and $(1,1)$ points. The $(0,1)$ point is referenced as the perfect classification (e.g. performance is perfect) while the $(1,0)$ represents the worse performance. The ROC points are defined by the values of the false positive rate and false negative rate, creating lines as seen in Figure 12. The AUC is then determined by the area under the ROC curve (Fawcett, 2005).

[7] Image source: https://stackoverflow.com/questions/10304109/library-in-python-for-neural-networks-to-plot-roc-auc-det

2.3.4.1.6 F-measure

F-measure, also known as F-score or F1-score, shows the trade-off between the recall and precision regarding the positive class (Goodfellow et al., 2015). Can be represented by:

$$F_{\beta}measure = \frac{(1+\ \beta^2)\ \times\ Recall\ \times\ Precision}{\beta^2 \times Recall + Precision} \tag{6.5}$$

Recall and precision are evenly weight if the F-measure β has a value of 1 (Goodfellow et al., 2015). It is usually applied in scenarios in which both recall and precision are used.

2.3.4.2 Regression

In regression problems, the error is calculated based on the distance between the known value from the dataset and the algorithm's prediction value (Baranauskas & Monard, 2003).

2.3.4.2.1 Metrics

The two most known regression evaluation metrics are the Median Absolute Error (MedAE) and Mean Squared Error (MSE) (Gama et al., 2017). In both metrics, lower values correspond to better predictions.

These metrics can be mathematically represented by (Gama et al., 2017)(Dosualdo, Daniel Gomes; Rezende, 2003):

$$MSE(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(x_i)\right)^2 \tag{7.1}$$

$$MedAE\left(\hat{f}\right) = median|y_i - \hat{f}(x_i)| \tag{7.2}$$

Where $\hat{f}$ represents the error hypothesis, y_i the dataset known value and $\hat{f}(x_i)$ the algorithm's predicted value. MedAE is the median absolute error between real values and predicted values for a meta-attribute[8]. MSE consists in the squared mean difference between real values and predicted values for a meta-attribute[9].

MSE has some variations like the normalized mean square error (NMSE), which generally shows the most striking differences among models. In some scenarios the root mean square error (RMSE) is also used and is mathematically represented as:

---

[8] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.median_absolute_error.html

[9] Free translation from (Dosualdo, Daniel Gomes; Rezende, 2003). Original text: "A medida MSE (Mean Squared Error) consiste na média da diferença ao quadrado entre os valores reais e preditos para um atributo-meta"

$$RMSE(\hat{f}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(x_i)\right)^2} \tag{7.3}$$

2.3.4.2.2 Correlation Coefficient

The correlation coefficient is always scaled between -1 and 1 and is a summary measure that describes the statistical relationship between two interval variables. When the correlation coefficient, r, is closer to 0, it means that the relationship between variables is weak. On the other hand, if it's far from 0 (either in the positive or negative direction), the relationship is stronger (Gingrich, 1992).

Assuming that there are two variables, X and Y (each having n values), and considering that the mean of X is $\bar{X}$ and the mean of Y is $\bar{Y}$, the correlation coefficient r is calculated using the following equation (Gingrich, 1992):

$$r = \frac{\sum(X_i - \bar{X})\sum(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \sum(Y_i - \bar{Y})^2}} \tag{7.4}$$

2.3.4.3 Evaluation capacity

One of the principal challenges of a machine learning algorithm is to perform well when facing unseen data. The factors determining the machine algorithm success are (Goodfellow et al., 2015):

- Make the training error small;
- Make the gap between training and test error small.

Cross-validation is a machine learning statistical method of evaluating and comparing learning algorithms (Refaeilzadeh, Tang, & Liu, 2006). In cross-validation, the data set is divided into k folds and at each iteration, a different fold is reserved for testing while all the others are used for training the classifier (Japkowicz & Shah, 2011). On the other hand, the **holdout** method defines that the initial dataset should be divided into two parts (Raschka, 2018), one for training (usually around 66% to 70% of the initial dataset) and the last one for testing (usually around 30% to 33% of the initial dataset, depending on the size of the training set).

Depending on the used metric, and the results from the training set and test set, the evaluation capacity can vary as observed in Figure 13. Some studies also use a third set, usually called validation set. While the training set and the test set results can influence the algorithm development and possible enhancements, the validation set works like a last validation before deploying the algorithm to the production environment.

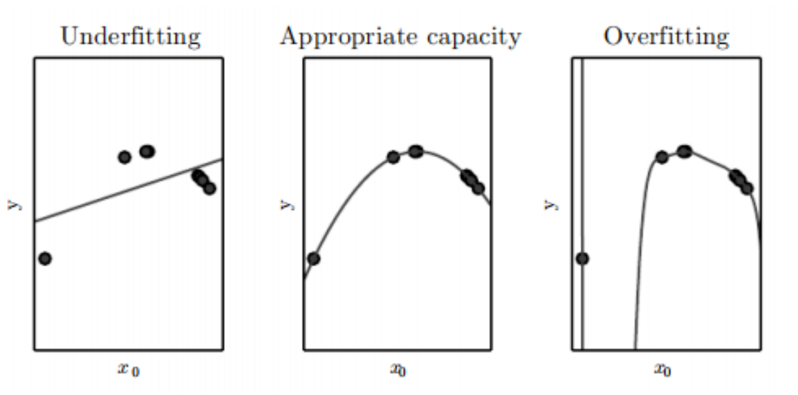


Figure 13 - Evaluation capacity scenarios (Goodfellow et al., 2015)

Overfitting occurs when the difference between the training set error and the test set is too large (Goodfellow et al., 2015). **Underfitting** represents the scenario in which the algorithm isn't able to obtain a sufficiently low error value on the training set (Goodfellow et al., 2015).

3 Analysis

This chapter describes this document's proposed objectives, approaches and guidelines. It is also presented the literature review, comparing the existing solutions to justify the decisions taken. The available evaluation metrics are also analyzed.

3.1 Proposed hypothesis

This document's proposal is focused on the usage of a deep learning framework to enhance previous studies results in the RTC prevention and prediction field. The main objective of this work is to predict an RTC severity and cost using a deep learning framework and data mining techniques.

Comparing the models' performance, it is expected to understand which classifier performs better. It is intended to disprove a null hypothesis, where all the classifiers perform equally.

Deep learning frameworks are dynamic and allow the algorithm's training using various techniques. Approaches like feature selection and missing data treatment are also considered for data preprocessing. Several evaluation metrics are acknowledged to evaluate the solution (see section 2.3.4), using holdout method.

From the literature review mentioned below, it is possible to interpret which approaches and algorithms are the best for this context. In the Design section is possible to find the final decisions of this chapter study.

3.2 Literature review

Some researches in the RTC prevention field are significant for this document's work analysis due to the approaches taken and results. This chapter is meant to highlight the most important studies, the used methodologies and results, and how those are considered for this document's solution.

3.2.1 Related research

The RTC prevention field was already explored using machine learning approaches. Table 6 shows the studies mentioned in the literature in a summarized manner.

Table 6 - RTC prevention existing solutions

Reference	**Dataset**	**Evaluation measures**	**Approaches**

(Beshah & Hill, 2010)	18288 crashes with no missing data, Ethiopia.	AUC;
		Accuracy.
			Decision tree;
			Naive bayes;
			K-nearest neighbor.

(Chong et al., 2005)	417610 sample crashes from USA.	Accuracy.	Decision tree;
			Hybrid Decision tree–ANN;
			ANN;
			SVM.

(Kumar & Toshniwal, 2015)	11574 road accidents for 6 years in Dehradun District of Uttarakhand State.	Support; Accuracy.	Clustering; Association rule mining algorithm (clustering variation).

(Vasavi, 2018)	Major national highways that pass-through Krishna district for the year 2013 crashes.	Precision;
		Recall;
		F-measure.
			Clustering;
			K-medoids.

(Nurkkala, Kalermo, & Jarvilehto, 2014)	RTC data collected from the police department in Dubai, United Arab Emirates.	Precision;
		Recall;
		F1-measure.
			SVM.

(Shanthi & Ramani, 2011)	37259 cases in USA.	Accuracy.	Random Tree;
			Decision List;
			Naive Bayes;
			Other less relevant techniques.

Reference	**Dataset**	**Evaluation measures**	**Approaches**
(Rakha, Arafeh, Abdel-Salam, Guo, & Flintsch, 2010)	Data from 186 access road sections in the state of Virginia.	None referenced.	Linear regression;

(Zhong-Xiang, Shi-Sheng, Wei-Hua, & Nan-Nan, 2014)	Road traffic accidents in China from 2002 to 2011.	Relative error;
		Comprehensive error;
			Multiple linear regression;

(Moradkhani, Ebrahimkhani, & Sadeghi Begham, 2014)	4382 records of accident in Hampshire, England.	None referenced;	Association rules.

(Santos, 2015)	RTC data from São Paulo, Brazil.	Mean;
		MSE;
		MAD.
			Decision Tree;
			Classification & Regression Tree;
			CHAID;
			Multiple linear regression;

(Yuan et al., 2017)	Iowa's RTC dataset with 65530 RTC.	Accuracy;
		Precision;
		Recall;
		F-Score;
		AUC.
			SVM;
			Deep Neural Network (DNN);
			Decision Tree;
			Random Forest.

(Çodur & Tortum, 2015)	7,780 RTC from Erzurum, Turkey.	MSE; Root mean square error (RMSE); Coefficient of determination.	ANN.

(Moghaddam, Afandizadeh, & Ziyadi, 2011)	52447 crash cases from Iran.	MSE;
		Normalized Mean Square Error;
		Correlation coefficient.
			ANN.

As observed in Table 6, the applied approaches to RTC prevention are varied. Depending on the approach, the evaluation measures also changes. The most used evaluation measures in classification algorithms are AUC, precision, recall and accuracy while in regression algorithms MSE is the most used metric.

The most impactful research found for this document's work is the (Yuan et al., 2017) study. It is based in the same dataset (explained in detail in the Dataset section) and uses deep learning. Although this study's objectives are slightly difference (map matching), it takes in consideration the same causative variables and achieved the best result using Deep Neural Networks (DNN) comparing with other approaches.

Also important to mention the (Chong et al., 2005) study which consider similar RTC severity classes (non-injury, minor injury, major injury, fatality) and used a hybrid approach with ANN and Decision Trees. Considering the achieved accuracy, results were reasonable. (Moghaddam et al., 2011) also uses ANN to predict an RTC severity.

This document's development and evaluation are influenced by these studies' analysis and approaches as the objectives are sometimes similar.

3.2.2 Deep learning frameworks comparison

As previously mentioned in the Deep learning frameworks chapter, the use of frameworks enhances machine learning algorithms development by abstracting most of the heavier processes (like weights calculation). To compare which framework is the best for this document's work, consider the following metrics without any particular order:

- Flexibility – to allow different algorithms implementation with ease;
- Multi-GPU and Multi-threaded CPU – to enhance the training process;
- Programming language;
- Number of lines of code – to reduce the development effort;
- Community size.

(Kovalev, Kalinovsky, & Kovalev, 2016) and (Bahrampour et al., 2015) studies contribute to the deep learning frameworks comparison and, its results are used for the decision mentioned in the Design chapter.

3.2.3 Data analysis & preparation

For **data analysis**, (Chong et al., 2005) explored each dataset's field distribution. This approach identifies clear outliers (if a class only has a 0.5% distribution, its influence on the prediction algorithm is almost irrelevant), which can be removed to reduce the dataset size and irrelevant data (using techniques mentioned in Data Cleaning).

Other researches like (Kumar & Toshniwal, 2015) and (Vasavi, 2018) explored the clustering and association rules approaches. As the Iowa's dataset contains many variables, which relation is not anteriorly evaluated, these approaches seem not to fit the current objectives.

This document's dataset has around 65530 input values containing empty fields (a considerable amount comparing with other studies in the 3.2.1 section) and random factors might cause some crashes. As this document's work is focused in the causative variables and not in mapping the RTC, (Shanthi & Ramani, 2011) approach to remove unnecessary feature columns like coordinates (using **feature selection** approaches like previously mention in 2.3.3.2) seems appropriate. This approach allows the dimensionality reduction of the dataset (as explained in Data Cleaning and Data Splitting section). Other features might be considered as irrelevant depending on the problem context.

For **missing data** treatment, (Yuan et al., 2017) research (which uses the same dataset of this document's work) considered the removal of the missing data registries and the missing data interpolation. Removing inputs from the dataset due to its large size is an alternative to consider if the total amount of missing data entries is not significant. In (Yuan et al., 2017) study, the missing data is calculated based on the nearest stations. Filling the missing values manually is not appropriated to this context due to the dataset size.

To **divide data into classes**, (Moghaddam et al., 2011) segregated the crash severity into two major classes during the data preprocessing stage. One represents the existence injuries or fatalities (i.e. human damage), the other represents human damage absence. Due to the multiple classes of this document's dataset (as presented in Table 2, this approach seems appropriate and contributes for the data balance (as the same class holds the inputs with injuries and fatalities).

(Yuan et al., 2017) rounded the crash time to the nearest hour before the crash (e.g. 12:36 becomes 12:00), to reduce the number of time classes as input. This approach is valuable when applied to time and dates as features to reduce the number of classes, enhancing the algorithm's performance.

3.2.4 Algorithms analysis

Clustering techniques don't match this document's work as it is associated with unsupervised learning rather than supervised learning.

(Shanthi & Ramani, 2011) used **Naive Bayes** and **Random Decision Trees** (a combination of individual decision trees) to classify vehicle collision patterns in road accidents. Although Naive Bayes is indicated for problems with a large dataset due to its solid performance, it is usually less accurate when compared with other classifiers (Jadhav & Channe, 2013) as happened in (Shanthi & Ramani, 2011) studies. On the other hand, Decision Trees has good results dealing with noisy data but can create over-complex trees, which can result in overfitting problems, and does not provide incremental learning (Brijain, Patel, Kushik, & Rana, 2014).

(Chong et al., 2005) used **Support Vector Machine (SVM)**, **Artificial Neural Network** and a hybrid solution which combined **Decision Trees** with ANN to predict traffic accidents. In this work's conclusions, it is possible to observe that SVM fail to model the complexity of the different injury classes besides SVM being indicated for problems with many features as input (Tomar, Nagpal, & Abdul, 2016). The NN approach and the hybrid approach resulted in the best results of (Chong et al., 2005) research.

Although not very used in RTC classification solutions, (Beshah & Hill, 2010) used **K Nearest Neighbors (kNN)** alongside Naive Bayes and Decision Trees. The kNN approach produced the best results and is indicated for a dataset with a large number of inputs and behaves robustly with noisy data.

(Moghaddam et al., 2011) and (Çodur & Tortum, 2015) used **ANN** in their studies, obtaining good results. Also, (Moghaddam et al., 2011) and other studies used various **Optimization functions** and **Activation functions** to compare its results. The variations are significant, indicating that changing these functions can influence the performance of an algorithm.

(Yuan et al., 2017) used Decision Trees, SVM, DNN and Decision Random Forest to predict RTC. The best results were achieved using DNN, using a complete evaluation composed by AUC, Recall, Precision, F1 and Accuracy. **DNN** seems to be appropriate for this context, with **Decision Random Forest** also showing good results.

3.2.5 Evaluation metrics analysis

There are many relevant evaluation metrics (as previously mentioned in Evaluation metrics). In the 3.2.1 section is possible to observe that the most used evaluation metrics for classification purposes are **Accuracy, Precision, AUC** and **Recall**.

Accuracy reveals several weaknesses, which can manipulate the final algorithm's conclusions if not combined with other measures like precision or recall. With accuracy, the classifier is measured based on the total number of correct predictions. Due to its simplicity, this metric can lead to inadequate solutions when dealing with unbalanced, which can be the this document's work scenario, and produces less discriminable values (Hossin & Sulaiman, 2015). For these reasons, precision and recall are usually more significant.

AUC's high evaluation value and recent popularity also make this metric an option. AUC reveals problems when dealing with multiclass dilemmas due to its limitation (high computational cost) but performs well and, it is a very valuable metric when applied to two classes problems (Hossin & Sulaiman, 2015).

(Yuan et al., 2017), study that uses the same dataset as this document's proposed dataset, also uses AUC and Accuracy to evaluate the model, using holdout method.

For regression problems, **MSE** seems the most popular metric. MSE is highly dependent on the weight initialization process (Hossin & Sulaiman, 2015), but the usage of a deep learning framework (which calculates the weights dynamically) can counter this potential weakness. **MedAE** can also be used if the error dispersal is high and is more robust against outliers (the mean is sensitive to outliers).

The same metrics are used to compare different solutions. The highest the metric value, the best (e.g. if an algorithm has 90% precision and 95% recall, it is more efficient than an algorithm with 70% precision and 75% recall). Most of the previous studies, like (Chong et al., 2005) and (Moghaddam et al., 2011) used cross-validation between a test set and the training set to understand the algorithm's performance.

4 Design

In this chapter is possible to find design alternatives for this document's work implementation. The main decisions and guidelines for the project's development are also mentioned, alongside the proposed methodologies for the evaluation. The analysis of the Literature review influenced the decisions taken in the sections below.

4.1 Development environment

This chapter is meant to compare and present the reasons that lead to the technological decisions of this document's work. In the Deep learning frameworks comparison chapter is possible to find the comparison terms that lead to most of these decisions.

One of the most influent factors to determine the development environment is the used programming language. Tensorflow, Theano and Caffe development uses Python (even though Tensorflow and Caffe core is built with C++) while Torch uses Lua.

Table 7 - Programming languages comparison from (TIOBE, 2018)

	Current position	**Variation**	**Market share**
Python	#4	+1.12%	5.2%
Lua	#31	-	0.52%

Table 7 presents the programming languages ranking of Python and Lua. As observed, Python is more popular than Lua, being the forth more used programming language in the market as of February 2018. Python is also recognized for having a small learning curve and for being widely accepted by the deep learning community (as most frameworks are built or have interfaces to work with Python).

The (Kovalev et al., 2016) and (Bahrampour et al., 2015) studies were used to better understand the frameworks performance and technical constraints of Tensorflow, Theano, Caffe and Torch.

From a features perspective, (Bahrampour et al., 2015) study is relevant to understand the frameworks' flexibility, extensibility and robustness. In the study's conclusions is mentioned that (Bahrampour et al., 2015):

- Theano and Torch are the most extensible frameworks due to the supported libraries;
- Theano and Torch are the most performant frameworks (either CPU-based or GPU-based training and deployment);
- Tensorflow is a very flexible framework (e.g. allows the explicit configuration of a ANN deepness and complexitiy), however, it's performance with a single GPU isn't as competitive compared to other frameworks.

The (Kovalev et al., 2016) tests consisted in comparing how neural networks behave on each of these frameworks by:

- Vary the neuron number of a fixed number of layers, width test (Figure 14);
- Vary the layers number with a fixed neuron number, deepness test (Figure 15).

The neural networks were tested with the Rectified Linear Unit (ReLU) and Tanh activation functions to obtain the classification accuracy, training time, prediction time and code complexity (lines of code).

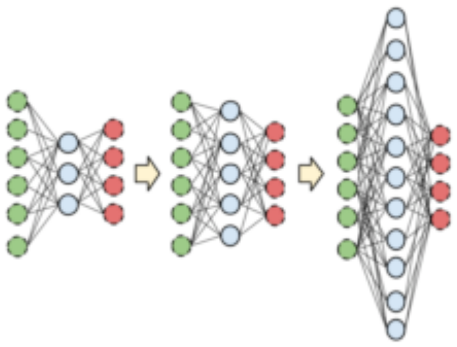


Figure 14 - Neural network width test (Kovalev et al., 2016)

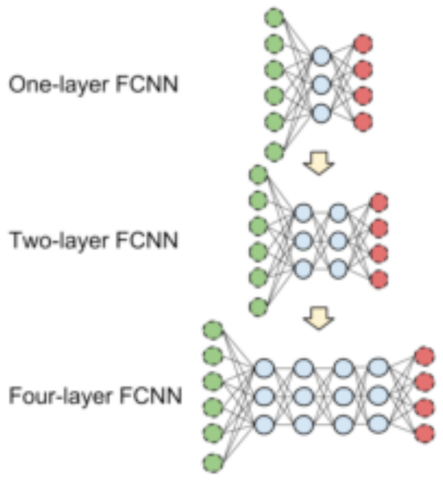


Figure 15 - Neural network deepness test (Kovalev et al., 2016)

The results shown that, considering the metrics previously mentioned, the frameworks behavioral ranking is as follows: Theano (with Keras – a NN library), Tensorflow, Caffe and Torch.

Although (Bahrampour et al., 2015) studies reveal that Torch is highly performant, when tested against deeper neural networks like in (Kovalev et al., 2016) study Torch performance drops in opposition to Tensorflow where the performance remains reliable (Kovalev et al., 2016).

In terms of community size (see Table 8 results as of February 2018), Tensorflow has a clear advantage in every field comparing with the other frameworks. Also, the releases number of Tensorflow in the last year incremented, which suggests that the framework performance, feature's number and support is increasing.

Table 8 - Deep learning frameworks community size in Github by 15 September 2019

	Stars	**Forks**	**Contributors**
Tensorflow	134058	77291	2184
Theano	8910	2504	332
Torch	8392	2342	130
Caffe	29034	17546	264

Due to (Bahrampour et al., 2015) and (Kovalev et al., 2016) studies outcomes and taking into account the community sizes, documentation and recent growth, **Tensorflow** was chosen as deep learning framework to this document's work.

As Tensorflow programming is Python based, a development environment is used to guarantee the project's integrity over the iterations, control the libraries versions and to respect engineering good practices.

Anaconda[10] (Python's most popular data science platform) is one of the requirements and Jupyter Notebook editor is used for the project's development. The environment holds the following libraries:

- Python v3.5.4;
- Tensorflow v1.14.0;
- Jupyter v1.0;
- Pandas v0.20.3;
- Numpy, v1.13;
- Matplot v2.0.2;
- Scikit-learn v.0.19;
- Other less relevant util libraries.

4.2 Proposed methodologies

This chapter is meant to compare machine learning techniques and show alternatives to achieve this project's objectives taking into consideration the data and algorithm analysis presented in the Literature review chapter and the proposed development environment presented in the Development environment chapter.

4.2.1 Dataset cleaning

Numpy and Pandas libraries are used to analyze the dataset. The objective of this analysis is to find clear outliers, noisy data, which can harm the algorithm's predictions. An outlier is considered a sample which value falls a long way outside of the other observations. It usually represents a small distribution in the dataset.

Currently the missing data values on the dataset are filled with a constant (see the Data Cleaning section for more detailed information). This approach considers the missing field as a class, as the default value will be interpreted as a real value by the algorithm. Other valid strategies are:

- **Remove the missing data** inputs – Reduces the dataset size and removes the possibility of unclear predictions, but the removed inputs can be precious as it may have RTC examples that are not replicated in the dataset;
- **Replace missing fields by the most probable value** – Following (Yuan et al., 2017) approach. No inputs are removed contributing to the dataset diversity, but the newly calculated values can be false, introducing false data in the dataset.

Remaining with the missing data is not the best approach. Both removing the missing data inputs and replacing the missing fields by the most likely value approaches seems to have advantages and disadvantages. Considering the risk related with the introduction of false data

[10] https://www.anaconda.com/

in the dataset and the required time and complexity to determine the impact of this operation, the removal of missing data was the chosen approach.

For simplification and performance purposes, (Yuan et al., 2017) rounded the crash time to the nearest hour before the crash (e.g. 12:36 to 12:00). This approach considers the time as a feature without overloading the algorithm with too many different classes, degrading the algorithm's prediction performance. This approach is followed as the daytime can be relevant for this problem's context as an input feature.

The removal of low impact data records, which may be considered as noisy data, (e.g. a feature value which number of occurrences is very reduced) is also considered as being a possible approach during the data cleaning process.

The unnecessary features are removed from the dataset using feature selection and the (Shanthi & Ramani, 2011) study approach. In the Implementation chapter is possible to find which features were removed.

Multi-class (more than two classes) classification problems are also harder to interpret and evaluate. Using feature construction (see 2.3.3.2), the dataset is modified to hold a two classes field, which is used when classifying an RTC severity reducing the development constraints. Although this decision removes some complexity from the problem (as the number of labels is reduced), the main purpose of this work is not affected as the labels defined on the tables below provide adequate information to a proper analysis of the results. The idea is when interpreting the RTC severity, to understand if there is human damage, either injuries or fatalities. To create this additional field, two alternative approaches are considered as represented in the tables below, which are based on Table 2 values.

Table 9 - Alternative 1 for RTC severity classification

Class	Description
0	Fatal, Major injury, Minor injury
1	Property cost only

Table 10 – Alternative 2 for RTC severity classification

Class	Description
0	Fatal, Major injury
1	Minor injury, Property cost only

Both approaches have some disadvantages. Alternative 1 (Table 9) classifies any human damage under the same category, being harder to understand what are the RTC causative variables that have more impact on major injuries and fatalities. On the other hand, is expected that alternative 2 (Table 10) causes unbalanced data issues as the number of scenarios where only minor injuries or property cost are registered is probably substantially higher than the situations where major injuries or fatalities occur. With this said, both approaches are tested in the Implementation chapter.

4.2.2 Model implementation

As previously mentioned in the 3.2.4 Algorithms analysis chapter, K Nearest Neighbors (kNN) and neural networks / deep neural networks are the algorithms that seem more appropriate for this problem. Tensorflow's flexibility allows both implementations. It is expected that neural networks behavior in Tensorflow is more performant as shown in the studies exposed in this document. This framework is also the one which supplies more documentation, therefore used for this document's implementation.

The (Beshah & Hill, 2010) study revealed more performant results using kNN but didn't consider the NN usage. The (Moghaddam et al., 2011), (Çodur & Tortum, 2015) and (Yuan et al., 2017) studies used NN, having good results, but didn't mention kNN. There are no direct comparison terms between both approaches in previous studies about RTC.

Although kNN is efficient with large datasets, it usually works better when applied to smaller features' vectors (as previously mentioned in the 2.3.2.1.2 chapter). Deep neural networks are indicated for complex problems where the number of features and inputs being substantial high. Tensorflow can also automatically control the number of layers and neurons per layer when using NN if required. For this reason, **deep neural networks** approach was chosen for implementation. Figure 16 shows the proposed model. **Decision random forest** is also considered as a design alternative based on (Yuan et al., 2017) reasonable results with the same dataset and as initially explained in 3.2.4.

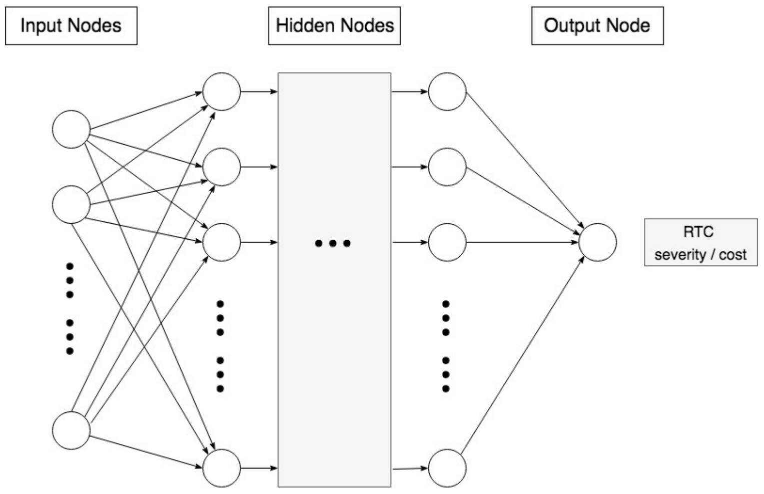


Figure 16 - Neural Network model – inspired by (Yuan et al., 2017) and (Çodur & Tortum, 2015) models

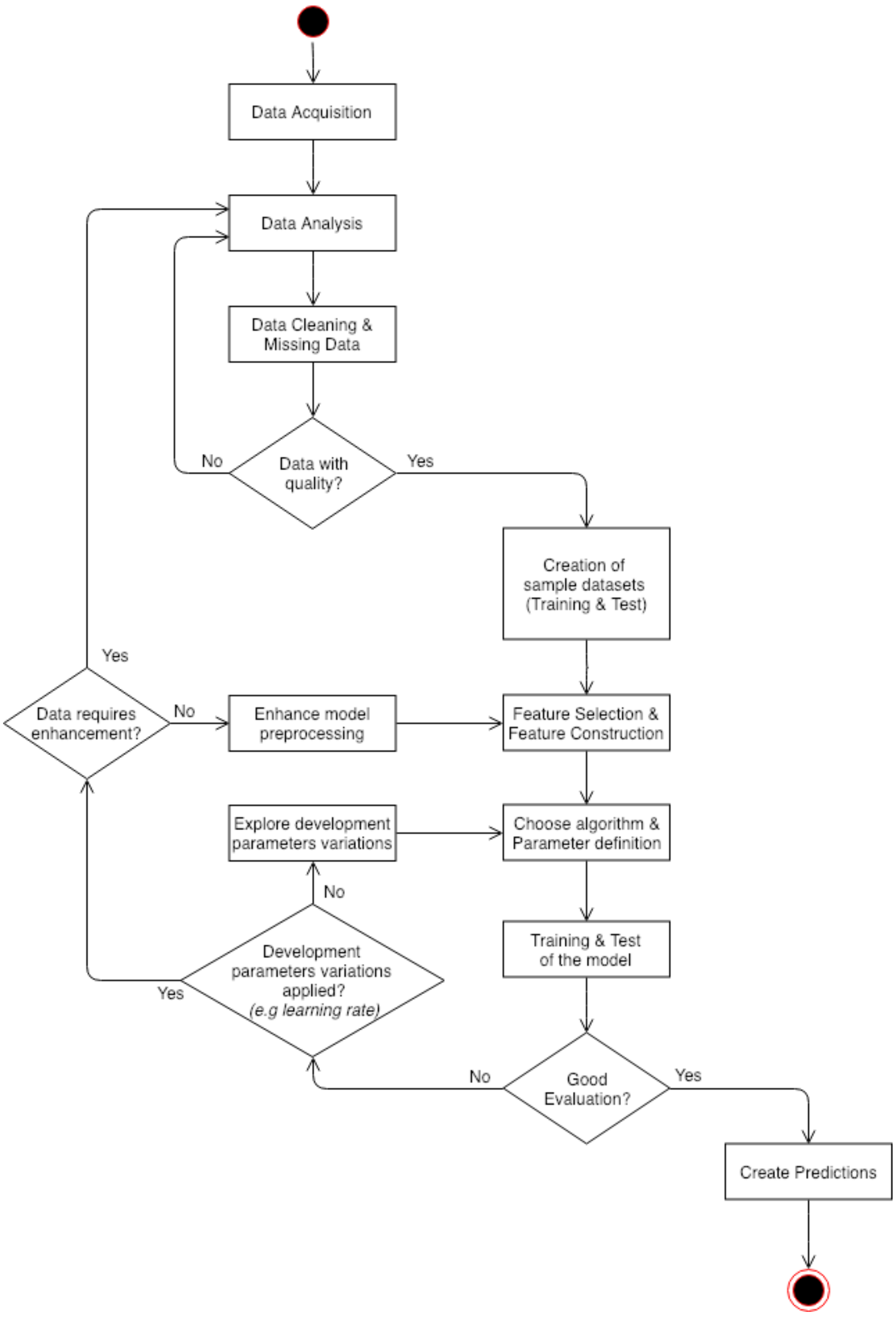
An analysis of the data balance and feature's value distribution is made after performing the data acquisition. As a consequence of this process, it's possible to identify and apply the best Data Cleaning and Missing Data processes. If the dataset' quality is sufficient, meaning that there's no missing data and the data has minimum balance, it's possible to proceed.

The dataset is then divided into training and test sets, allowing the model training and corresponding evaluation of the solution. Following the industry's best practices, the training set represents 67% while the test set represents 33% of the initial of the dataset. Other approaches suggest a validation set, which works like a final verification before deploying to production and doesn't influence the algorithm. This approach doesn't seem appropriate for considering this document's context and corresponding complexity. Also, this approach is not recommended in the literature review. Therefore, only the training set and test set are considered.

During the learning process, ANN updates the bias and weights to reduce the error with optimization functions. Some alternatives are shown in the Optimization functions chapter. GradientDescent is the most used function, and Adam is referenced in some of the RTC studies like in (Yuan et al., 2017). Is expected that Adagrad reveals some problems during the learning process due to its weaknesses related to the learning rate adaptation. For this reason, both **GradientDescent and Adam are considered for the algorithm's implementation**, but not Adagrad.

Activation functions can have unexpected behavior depending on the data representation. As the effort to change activation functions is reduced when developing in Tensorflow, **Softplus, Rectified Linear Unit (ReLU), Sigmoid, Tanh are considered for development**. Having in mind other studies in the RTC field, is expected that ReLU and Tanh behave better than Sigmoid and Softplus in classification. The results' comparison is presented in the Evaluation chapter.

The model can be iteratively refined depending on the evaluation process. **Holdout** is used instead of cross-validation considering other relevant studies in the RTC field like (Yuan et al., 2017) and the recurrent usage of this method in solutions using DNN. If the final evaluation is not satisfactory and depending on the methods already applied during the implementation stage, the model's preprocessing or the data cleaning process can be refined. Also, some variations can be applied to the development parameters (e.g. learning rate, activation functions) in order to achieved better results. Figure 17 represents this process.



Data Acquisition
Data Analysis
Data Cleaning & Missing Data
No
Data with quality?
Yes
Creation of sample datasets (Training & Test)
Yes
Data requires enhancement?
No
Enhance model preprocessing
Feature Selection & Feature Construction
Explore development parameters variations
Choose algorithm & Parameter definition
No
Development parameters variations applied? (e.g learning rate)
Yes
Training & Test of the model
No
Good Evaluation?
Yes
Create Predictions

Figure 17 - Proposed implementation state diagram

5 Implementation

This chapter's content provides visibility on how the implementation was handled considering the approach defined earlier in this document, mainly in the Design chapter. It also provides technical details about decisions taken during the execution phase, which were previously mapped in chapter 4.

5.1 Development environment construction

A stable development environment that provides robustness and immutability is important to achieve reliable results. As mentioned earlier in the section 4.1 above4.1, a Python based virtual environment was considered to achieve this document's goals. The following requirements were considered to build the environment:

- Ensure the environment immutability and stability over time;
- Easy configuration and environment setup;
- Capability to quickly add an external library or to upgrade / downgrade an existing one.

Conda[11], a package dependency and environment management tool, was used to establish the development environment. Conda is mostly controlled by command line and provides a set features that accommodate the needs stated above. A Python v3.5 environment was created using Conda following the steps mentioned in the Code 1 bash example.

A *tfdl_env.yml* file was added to the project's folder containing a set of dependencies and libraries to be installed using pip command. Each dependency and third-party library have an associated version, therefore allowing the control, immutability and easy configuration of the

[11] Conda documentation: https://docs.conda.io/en

whole setup following Conda's good practices (see Code 1 example). Some of the most important libraries' versions are described in the section 4.1.

```
conda create -n tfdeeplearning python=3.5
conda install jupyter
conda install numpy
conda install pandas
conda install scikit-learn
conda install matplotlib
pip install --upgrade tensorflow
source activate tfdeeplearning
jupyter notebook
```

Code 1 - Instructions to setup and access Jupyter Notebook development environment

A Git repository[12] was also created to hold this project's implementation versioning for the sake of engineering best practices. This approach also allows this project to become open-source in the near future.

5.2 Data handling

In this chapter it is possible to find the required steps to acquire and analyze the content of the dataset file. It also contains a description of the data cleaning operations and how the feature selection and construction was handed.

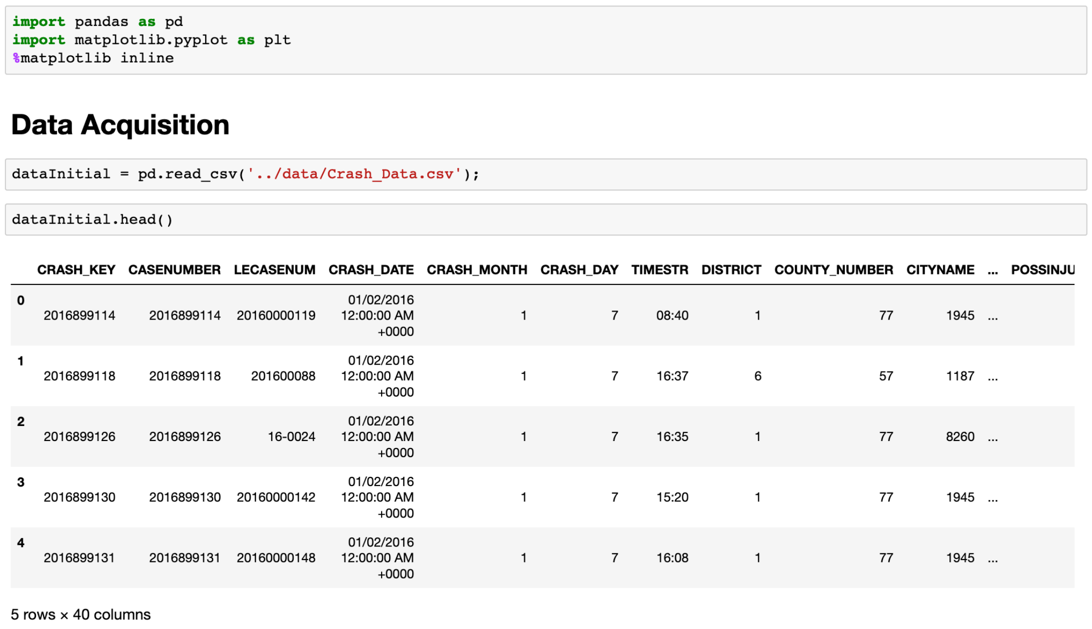
5.2.1 Data acquisition

The data used in this project was acquired in the Iowa open data portal[13], a governmental initiative that provides open-source datasets as mentioned earlier in the 2.1.2 section.

The dataset was downloaded in a spreadsheet format and added to the project's folder. This approach allows the usage of relative paths instead of absolute paths, which are more error prone. Pandas, Python's most used data analysis library, was used to get the spreadsheet content into a local variable. Figure 18 demonstrates this process.

[12] Repository source: https://github.com/bmfteixeira/thesis-crash-data-ml

[13] Iowa datasets portal: http://data.iowadot.gov/datasets



```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Data Acquisition

```
dataInitial = pd.read_csv('../data/Crash_Data.csv');
```

```
dataInitial.head()
```

	CRASH_KEY	CASENUMBER	LECASENUM	CRASH_DATE	CRASH_MONTH	CRASH_DAY	TIMESTR	DISTRICT	COUNTY_NUMBER	CITYNAME	...	POSSINJU
0	2016899114	2016899114	20160000119	01/02/2016 12:00:00 AM +0000	1	7	08:40	1	77	1945	...	
1	2016899118	2016899118	201600088	01/02/2016 12:00:00 AM +0000	1	7	16:37	6	57	1187	...	
2	2016899126	2016899126	16-0024	01/02/2016 12:00:00 AM +0000	1	7	16:35	1	77	8260	...	
3	2016899130	2016899130	20160000142	01/02/2016 12:00:00 AM +0000	1	7	15:20	1	77	1945	...	
4	2016899131	2016899131	20160000148	01/02/2016 12:00:00 AM +0000	1	7	16:08	1	77	1945	...	

5 rows × 40 columns

Figure 18 – Data acquisition code snippet in Jupyter Notebook

5.2.2 Data analysis

Data distribution and related characteristics could influence the technical approach of the cleaning process. The sections below demonstrate the analysis process for the severity and cost areas.

5.2.2.1 Severity analysis

The following two plots were considered as highly important to a correct analysis:

- RTC severity outcome distribution – Allows the visualization of the amount of RTC that cause injuries or deaths. This is represented in Figure 19;
- Major cause of RTC distribution – Permits the identification of a small set of values that are significantly predominant in the dataset entries. Represented in Figure 20.

From the analysis of plots mentioned above it is possible to observe that most of the RTC do not create any injuries nor deaths and that there are some crash major causes which have a low impact in the dataset. Unbalanced data is also noticeable when comparing RTC with human damage and RTC without human damage, even though for the purpose of this document's work, the original data distribution is preferred.

```
n_bins = 20
fig, axs_human_danger = plt.subplots(1, 4, sharey=True)

axs_human_danger[0].hist(dataInitial['INJURIES'], bins=n_bins)
axs_human_danger[1].hist(dataInitial['MININJURY'], bins=n_bins)
axs_human_danger[2].hist(dataInitial['MAJINJURY'], bins=n_bins)
axs_human_danger[3].hist(dataInitial['FATALITIES'], bins=n_bins)
```

```
(array([5.69969e+05, 0.00000e+00, 3.24800e+03, 0.00000e+00, 0.00000e+00,
        2.68000e+02, 0.00000e+00, 0.00000e+00, 4.50000e+01, 0.00000e+00,
        0.00000e+00, 1.30000e+01, 0.00000e+00, 0.00000e+00, 4.00000e+00,
        0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 1.00000e+00]),
 array([0.  , 0.35, 0.7 , 1.05, 1.4 , 1.75, 2.1 , 2.45, 2.8 , 3.15, 3.5 ,
        3.85, 4.2 , 4.55, 4.9 , 5.25, 5.6 , 5.95, 6.3 , 6.65, 7.  ]),
 <a list of 20 Patch objects>)
```

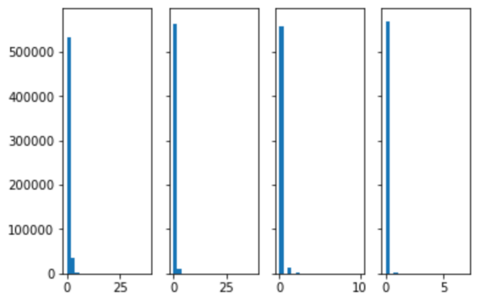


Figure 19 - Human injuries and deaths data distribution

```
dataInitial['MAJCSE'].hist(bins=74)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x110e54438>
```

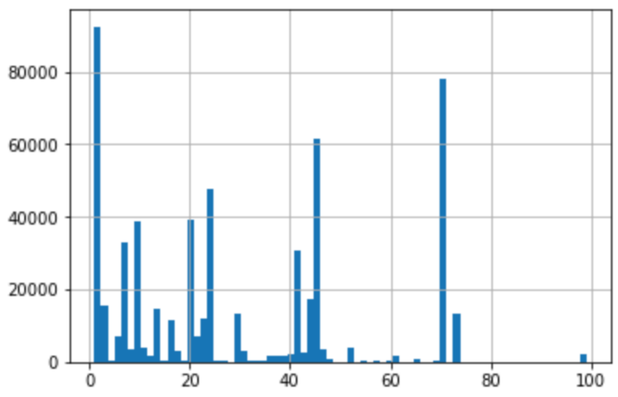


Figure 20 - Crash major cause distribution

5.2.2.2 Cost analysis

A couple plots were built to allow a precise RTC cost analysis:

- RTC cost distribution – High-level visualization of the cost per dataset instance in order to identify patterns and understand data balance. Represented in Figure 21;
- Boxplot – Clear identification of cost outliers that could negatively influence models' performance. Shown in Figure 22.

As it is possible to observe in the plots shown below, and as expected having in mind the analysis already made in 5.2.2.1 section, the data is unbalanced. There's a significantly higher proportion of RTC with lower cost. Higher cost RTC may vary from values close to 15000 up until near to 500000, which is a big interval considering the most predominant cost values.

This being said and as stated on Figure 22, the amount of outliers in the dataset is significant and should require a data cleaning process to reduce the risks of data balance negative impact in the models. During this process it was also identified that the initial property cost mean of all dataset records is around 5885 US dollars. As an example of outliers, there are crashes in the dataset which property cost goes up to 4851387 or 400000 US dollars (highest values represented in Figure 21 and Figure 22), which compared to the mean value is extremely high.

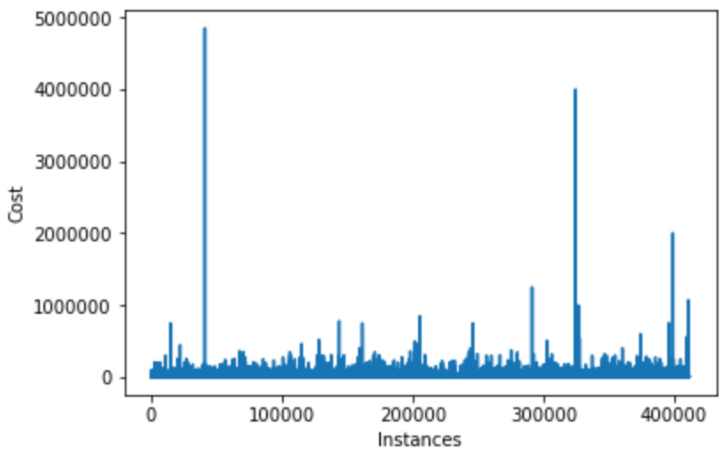


Figure 21 - RTC cost distribution per dataset instance

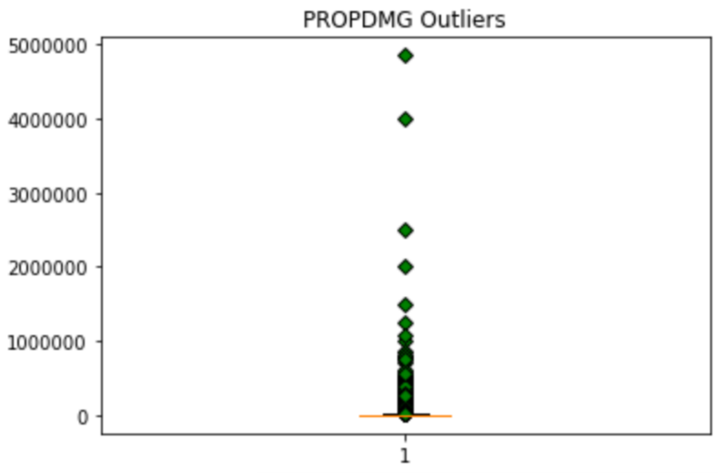


Figure 22 - Boxplot to identify cost outliers

5.2.3 Data cleaning

During the data cleaning process, and considering the decisions explained on the 4.2.1 Dataset cleaning section, the main goals were:

- Identify and remove irrelevant or redundant features in the dataset;
- Follow the approach of (Yuan et al., 2017) to round the crash time for simplification purposes;
- Identify and remove missing data from the dataset.

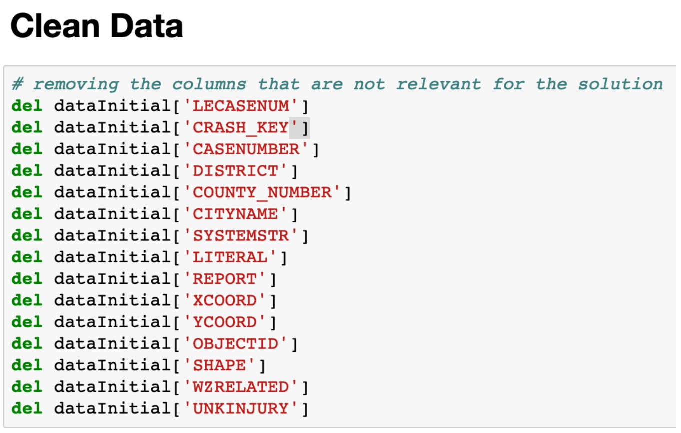
5.2.3.1 Feature selection

The study of (Shanthi & Ramani, 2011) was used to identify which features are the most relevant, irrelevant and redundant following the procedure explained in the 2.3.3.2 section. Some features were considered as irrelevant, not because their impact in RTC is low, but because the context of this document's work does not consider their usage. As an example of this statement, localization related features are not used as coordinates are ordinary values, therefore meaningless without a mapping process as represented in the (Yuan et al., 2017) study. Irrelevant and redundant features, which identification is demonstrated in Table 11, were removed from the dataset as seen in Figure 23.

Table 11 - Features identification as irrelevant, redundant and relevant

Key	**Description**	**Importance Classification**
CRASH_KEY	Crash id in Iowa's database	Irrelevant
CASENUMBER	Case number	Irrelevant
LECASENUM	Law enforcement case number	Irrelevant
CRASH_DATE	Date of crash	Relevant
CRASH_MONTH	Month of crash	Relevant
CRASH_DAY	Day of week	Relevant
TIMESTR	Time of crash	Relevant
DISTRICT	DOT District	Irrelevant
COUNTY_NUMBER	County	Irrelevant
CITYNAME	City	Irrelevant
SYSTEMSTR	Route with system	Irrelevant
LITERAL	Derived literal description	Irrelevant
FRSTHARM	Frist harmful event	Relevant
LOCFSTHRM	Location of first harmful event	Relevant

Key	**Description**	**Importance Classification**
CRCOMNNR	Manner of crash / collision	Relevant
MAJCSE	Major cause	Relevant
DRUGALC	Drug or alcohol related	Relevant
ECNTCRC	Contributing circumstances	Relevant
LIGHT	Light conditions	Relevant
CSRFCND	Surface conditions	Relevant
WEATHER	Weather conditions	Relevant
RCNTCRC	Contributing circumstances - Roadway	Relevant
RDTYP	Type of roadway junction / feature	Relevant
PAVED	Paved or not	Relevant
WZRELATED	Work zone related	Irrelevant
CSEV	Crash severity	Relevant
FATALITIES	Number of fatalities	Relevant
INJURIES	Number of injuries	Relevant
MAJINJURY	Number of major injuries	Relevant
MININJURY	Number of minor injuries	Relevant
POSSINJURY	Number of possible injuries	Relevant
UNKINJURY	Number of unknown injuries	Irrelevant
PROPDMG	Amount of property damage ($)	Relevant
VEHICLES	Number of vehicles involved	Relevant
TOUCCUPANTS	Total number of occupants	Relevant
REPORT	Report type	Irrelevant
XCOORD	X Coordinate (UTM NAD 83 Zone 15)	Irrelevant
YCOORD	Y Coordinate (UTM NAD 83 Zone 15)	Irrelevant
OBJECTID	Object ID	Irrelevant
SHAPE	Shape	Irrelevant



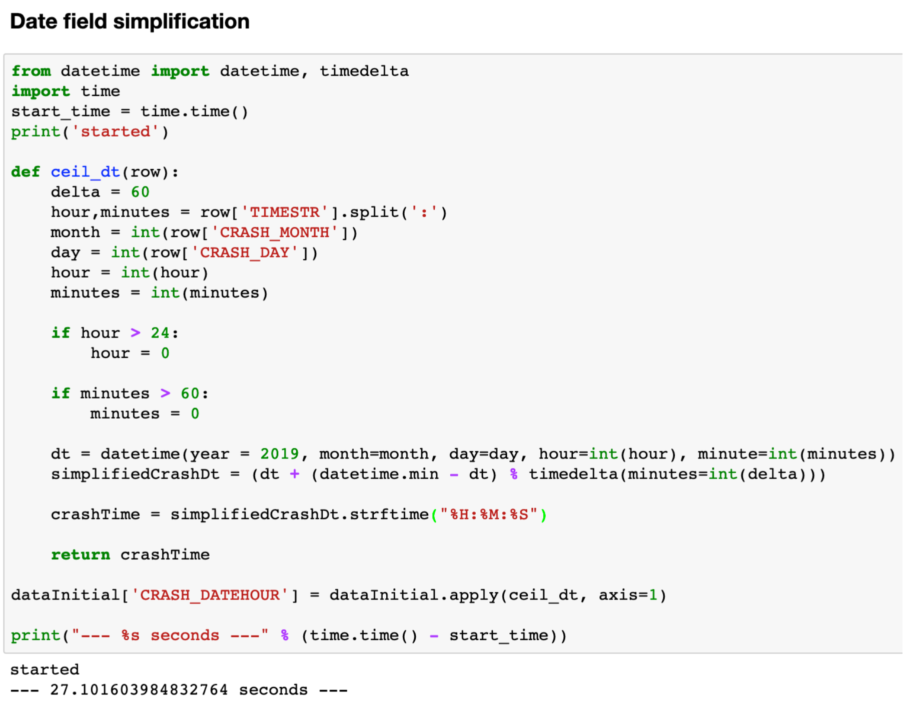
Clean Data

```
# removing the columns that are not relevant for the solution
del dataInitial['LECASENUM']
del dataInitial['CRASH_KEY']
del dataInitial['CASENUMBER']
del dataInitial['DISTRICT']
del dataInitial['COUNTY_NUMBER']
del dataInitial['CITYNAME']
del dataInitial['SYSTEMSTR']
del dataInitial['LITERAL']
del dataInitial['REPORT']
del dataInitial['XCOORD']
del dataInitial['YCOORD']
del dataInitial['OBJECTID']
del dataInitial['SHAPE']
del dataInitial['WZRELATED']
del dataInitial['UNKINJURY']
```

Figure 23 - Code to delete redundant and irrelevant features from dataset

5.2.3.2 Round crash date time

Following the approach of (Yuan et al., 2017), the RTC crash time should be rounded in order to consider a controlled set of values which correspond to full hours of the day (i.e. 24 different values). This approach was implemented following what is displayed in Figure 24. The *strftime* operation was preferred due to performance reasons as stated in Figure 25 (27.1 seconds instead of 1691.7 seconds).

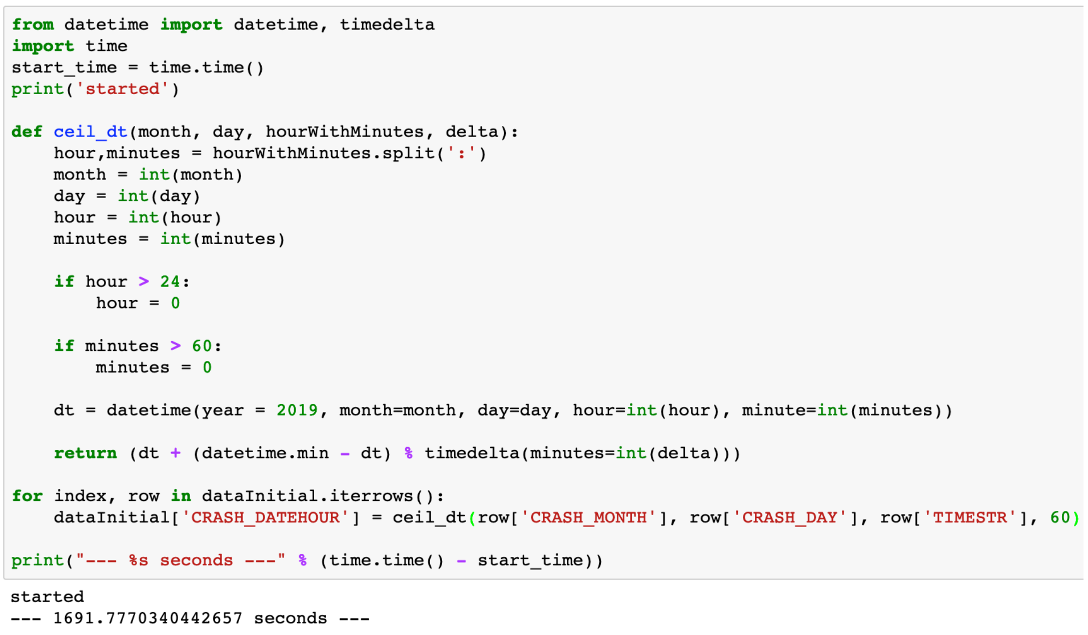


Date field simplification

```
from datetime import datetime, timedelta
import time
start_time = time.time()
print('started')

def ceil_dt(row):
    delta = 60
    hour,minutes = row['TIMESTR'].split(':')
    month = int(row['CRASH_MONTH'])
    day = int(row['CRASH_DAY'])
    hour = int(hour)
    minutes = int(minutes)

    if hour > 24:
        hour = 0

    if minutes > 60:
        minutes = 0

    dt = datetime(year = 2019, month=month, day=day, hour=int(hour), minute=int(minutes))
    simplifiedCrashDt = (dt + (datetime.min - dt) % timedelta(minutes=int(delta)))

    crashTime = simplifiedCrashDt.strftime("%H:%M:%S")

    return crashTime

dataInitial['CRASH_DATEHOUR'] = dataInitial.apply(ceil_dt, axis=1)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
started
--- 27.101603984832764 seconds ---
```

Figure 24 - Optimized RTC crash time transformation



```python
from datetime import datetime, timedelta
import time
start_time = time.time()
print('started')

def ceil_dt(month, day, hourWithMinutes, delta):
    hour,minutes = hourWithMinutes.split(':')
    month = int(month)
    day = int(day)
    hour = int(hour)
    minutes = int(minutes)

    if hour > 24:
        hour = 0

    if minutes > 60:
        minutes = 0

    dt = datetime(year = 2019, month=month, day=day, hour=int(hour), minute=int(minutes))

    return (dt + (datetime.min - dt) % timedelta(minutes=int(delta)))

for index, row in dataInitial.iterrows():
    dataInitial['CRASH_DATEHOUR'] = ceil_dt(row['CRASH_MONTH'], row['CRASH_DAY'], row['TIMESTR'], 60)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
started
--- 1691.7770340442657 seconds ---
```

Figure 25 - Not optimized RTC crash time transformation

The *TIMESTR*, *CRASH_DATE*, *CRASH_MONTH* and *CRASH_DAY* features became redundant and were removed during this process as these features were replaced by a new feature named *CRASH_DATEHOUR*.

5.2.3.3 Removal of missing data

There are various variations of missing data depending on the dataset being used. Following the dataset structure explanation and characteristics presented in the 2.1.2 Dataset section of this document, the following was considered as missing data:

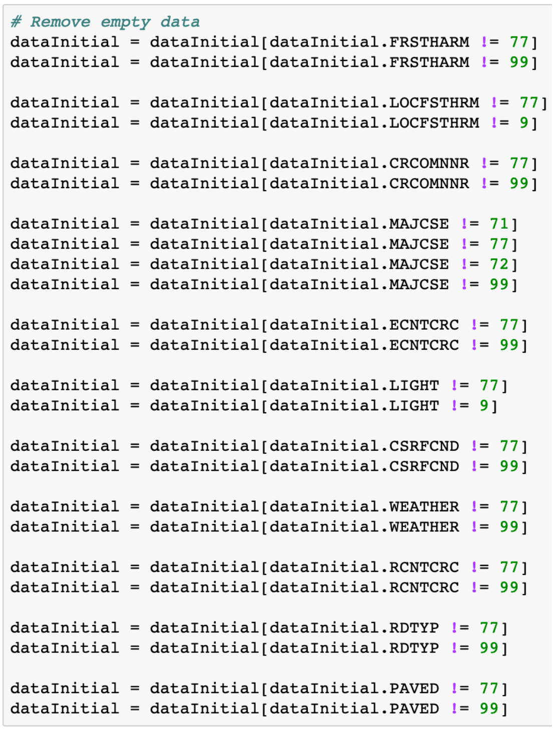
- Dataset features' values which keys correspond to "Not Reported" or "Unknown";
- Dataset features' values which keys do not correspond to any of the dataset possible key values[14];
- Empty cells.

It was possible to conclude that all dataset's cells had associated values during the analysis, meaning there's no empty registries to remove. The identification of unreported, unknown and not supported key values is presented in Table 12 considering the remaining relevant features after the actions mentioned in the 5.2.3.1 and 5.2.3.2 sections and having in mind that this procedure is only valid on discrete fields as specified in the 2.1.2 section.

[14] JSON file with dataset keys' description and allowed values: https://pastebin.com/ui7dex3A

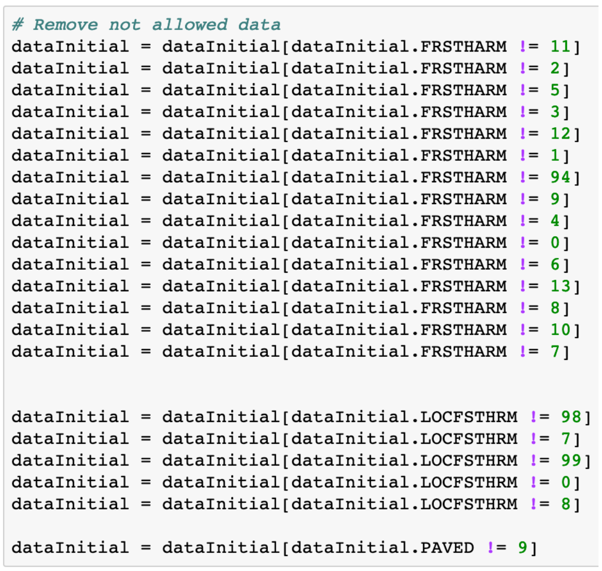
Table 12 - Relevant discrete fields' missing data identification

Key	Not supported key values	Missing data values
FRSTHARM	11, 2, 5, 3, 12, 1, 94, 9, 5, 0, 6, 13, 8, 10, 7.	77, 99.
LOCFSTHRM	98, 7, 99, 0, 8.	77, 99.
CRCOMNNR	Nothing to report.	77, 99.
MAJCSE	Nothing to report.	9, 71, 72.
DRUGALC	Nothing to report.	Nothing to report.
ECNTCRC	Nothing to report.	77, 99.
LIGHT	Nothing to report.	77, 99.
CSRFCND	Nothing to report.	77, 99.
WEATHER	Nothing to report.	77, 99.
RCNTCRC	Nothing to report.	77, 99.
RDTYP	Nothing to report.	77, 99.
PAVED	9.	77, 99.
WZRELATED	Nothing to report.	77, 99.
CSEV	Nothing to report.	Nothing to report.



```
# Remove empty data
dataInitial = dataInitial[dataInitial.FRSTHARM != 77]
dataInitial = dataInitial[dataInitial.FRSTHARM != 99]

dataInitial = dataInitial[dataInitial.LOCFSTHRM != 77]
dataInitial = dataInitial[dataInitial.LOCFSTHRM != 9]

dataInitial = dataInitial[dataInitial.CRCOMNNR != 77]
dataInitial = dataInitial[dataInitial.CRCOMNNR != 99]

dataInitial = dataInitial[dataInitial.MAJCSE != 71]
dataInitial = dataInitial[dataInitial.MAJCSE != 77]
dataInitial = dataInitial[dataInitial.MAJCSE != 72]
dataInitial = dataInitial[dataInitial.MAJCSE != 99]

dataInitial = dataInitial[dataInitial.ECNTCRC != 77]
dataInitial = dataInitial[dataInitial.ECNTCRC != 99]

dataInitial = dataInitial[dataInitial.LIGHT != 77]
dataInitial = dataInitial[dataInitial.LIGHT != 9]

dataInitial = dataInitial[dataInitial.CSRFCND != 77]
dataInitial = dataInitial[dataInitial.CSRFCND != 99]

dataInitial = dataInitial[dataInitial.WEATHER != 77]
dataInitial = dataInitial[dataInitial.WEATHER != 99]

dataInitial = dataInitial[dataInitial.RCNTCRC != 77]
dataInitial = dataInitial[dataInitial.RCNTCRC != 99]

dataInitial = dataInitial[dataInitial.RDTYP != 77]
dataInitial = dataInitial[dataInitial.RDTYP != 99]

dataInitial = dataInitial[dataInitial.PAVED != 77]
dataInitial = dataInitial[dataInitial.PAVED != 99]
```

Figure 26 - Code snippet corresponding to the removal of missing data

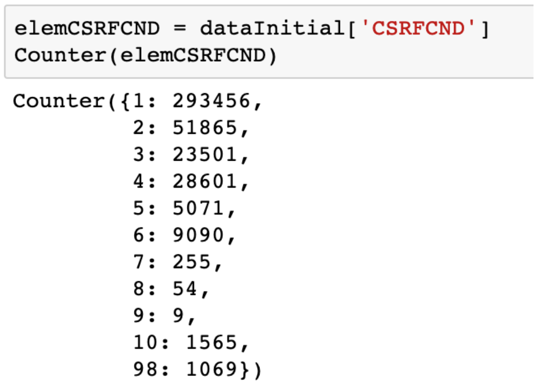


```
# Remove not allowed data
dataInitial = dataInitial[dataInitial.FRSTHARM != 11]
dataInitial = dataInitial[dataInitial.FRSTHARM != 2]
dataInitial = dataInitial[dataInitial.FRSTHARM != 5]
dataInitial = dataInitial[dataInitial.FRSTHARM != 3]
dataInitial = dataInitial[dataInitial.FRSTHARM != 12]
dataInitial = dataInitial[dataInitial.FRSTHARM != 1]
dataInitial = dataInitial[dataInitial.FRSTHARM != 94]
dataInitial = dataInitial[dataInitial.FRSTHARM != 9]
dataInitial = dataInitial[dataInitial.FRSTHARM != 4]
dataInitial = dataInitial[dataInitial.FRSTHARM != 0]
dataInitial = dataInitial[dataInitial.FRSTHARM != 6]
dataInitial = dataInitial[dataInitial.FRSTHARM != 13]
dataInitial = dataInitial[dataInitial.FRSTHARM != 8]
dataInitial = dataInitial[dataInitial.FRSTHARM != 10]
dataInitial = dataInitial[dataInitial.FRSTHARM != 7]


dataInitial = dataInitial[dataInitial.LOCFSTHRM != 98]
dataInitial = dataInitial[dataInitial.LOCFSTHRM != 7]
dataInitial = dataInitial[dataInitial.LOCFSTHRM != 99]
dataInitial = dataInitial[dataInitial.LOCFSTHRM != 0]
dataInitial = dataInitial[dataInitial.LOCFSTHRM != 8]

dataInitial = dataInitial[dataInitial.PAVED != 9]
```

Figure 27 - Code snippet corresponding to the removal of not supported data entries

5.2.3.4 Removal of low impact data

Data records which number of occurrences is extremely low when compared with the total amount of records in a dataset can be considered as noisy data depending on the problem's context following (Chong et al., 2005) study, previously explained in 3.2.3 section. Therefore, and considering the initial analysis made in 5.2.2 section, a new data handling variation was created with the low impact feature keys being removed from the dataset. With the analysis represented in Figure 28 it was possible to calculate which keys could be removed using the approach stated in Figure 27.



```
elemCSRFCND = dataInitial['CSRFCND']
Counter(elemCSRFCND)
```

```
Counter({1: 293456,
         2: 51865,
         3: 23501,
         4: 28601,
         5: 5071,
         6: 9090,
         7: 255,
         8: 54,
         9: 9,
         10: 1565,
         98: 1069})
```

Figure 28 - Feature keys' occurrences counter

5.2.3.5 Removal of outliers

As mentioned in 5.2.2.2 and for the regression models, a new variation where outliers would be removed was considered in order to prevent models' performance of being harmed. A deeper analysis allowed the classification of instances with cost above 12000 as outliers. **Error!**

Reference source not found. represents a new boxplot that was generated after the outliers' removal process.

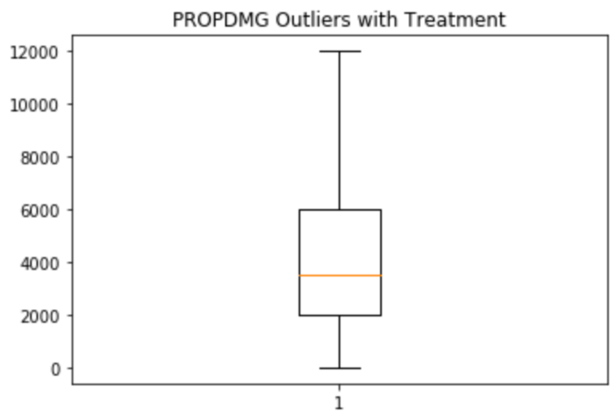


Figure 29 - Boxplot after outliers removal

An amount of 40352 records were removed from the dataset, per consequence reducing the initial dataset size to a total of 371011 instances to be used in regression models. The mean RTC cost moved from the initial 5885 US dollars mentioned in 5.2.2.2 to 4267 US dollars.

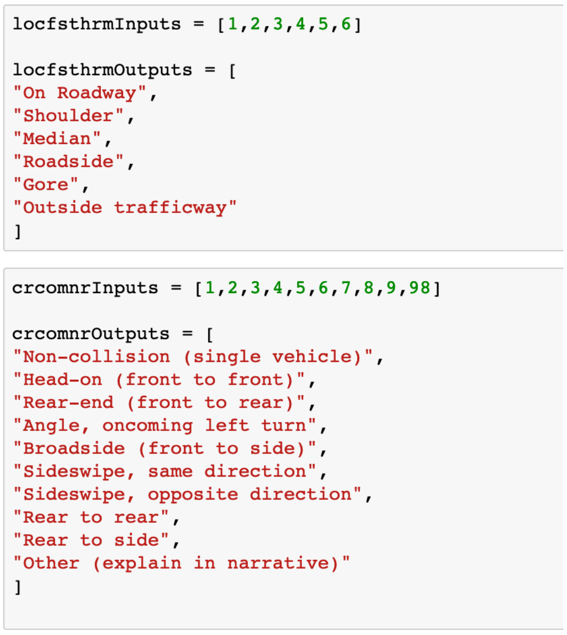
5.2.4 Feature columns key to value transformation

The dataset initial composition, in which each feature is represented by a set of keys corresponding to description fields as explained in 2.1.2 section, didn't matched Tensorflow preferred feature columns definitions for a neural network construction.

Following the official documentation[15], Tensorflow feature definition mechanism transforms feature columns values into lower-dimensional vector containing integers using a mapping function. These values are therefore used as inputs in the neural network. The preferred input values' typology is string so Tensorflow can map them into integers.

With this in mind, a key to description value transformation was applied as the initial dataset feature column values were integers. This process was based in a mapping function, which created a relation between each key and the corresponding description[16] as seen in Figure 30. Each feature column key values were therefore transformed into the corresponding description strings using Python's array replace function as stated in Figure 31.

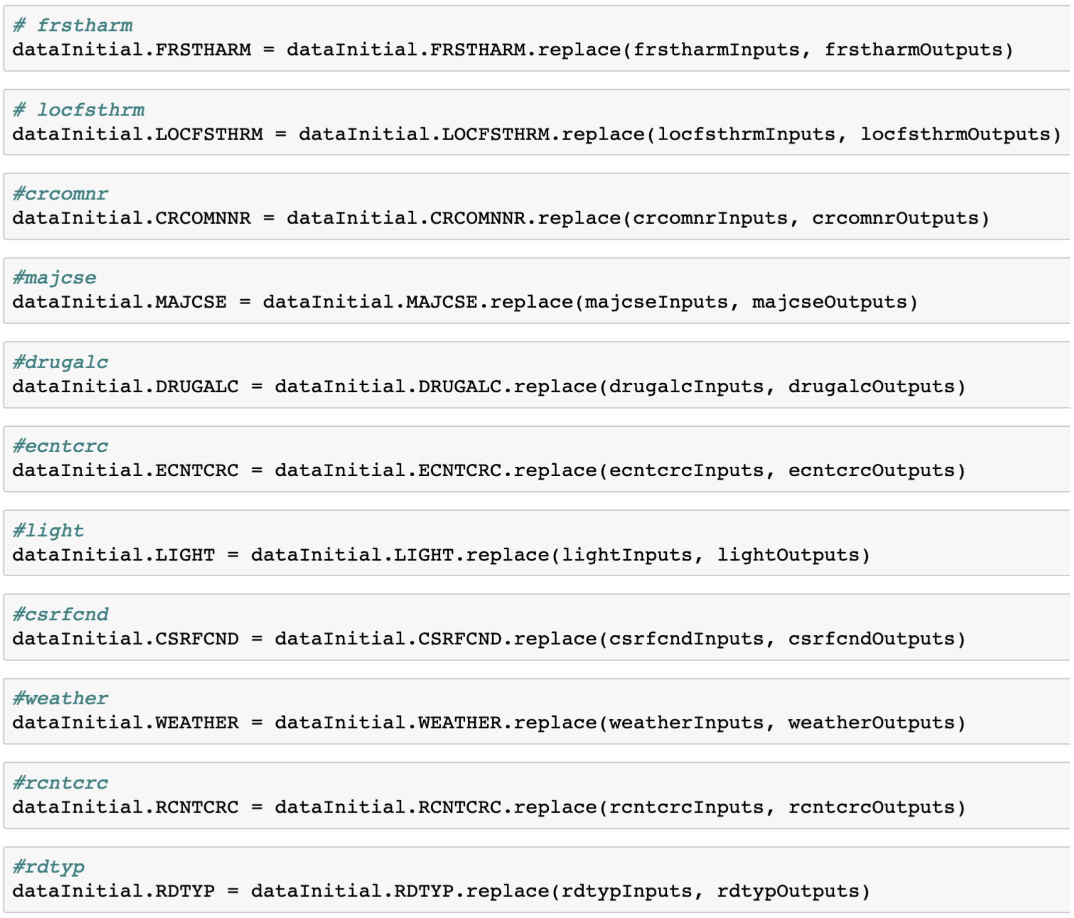
[15] Tensorflow feature columns documentation: https://www.tensorflow.org/guide/feature_columns
[16] JSON with key to description correspondence: https://pastebin.com/ui7dex3A



```
locfsthrmInputs = [1,2,3,4,5,6]

locfsthrmOutputs = [
"On Roadway",
"Shoulder",
"Median",
"Roadside",
"Gore",
"Outside trafficway"
]
```

```
crcomnrInputs = [1,2,3,4,5,6,7,8,9,98]

crcomnrOutputs = [
"Non-collision (single vehicle)",
"Head-on (front to front)",
"Rear-end (front to rear)",
"Angle, oncoming left turn",
"Broadside (front to side)",
"Sideswipe, same direction",
"Sideswipe, opposite direction",
"Rear to rear",
"Rear to side",
"Other (explain in narrative)"
]
```

Figure 30 - Feature key to value correspondence example



```
# frstharm
dataInitial.FRSTHARM = dataInitial.FRSTHARM.replace(frstharmInputs, frstharmOutputs)
```

```
# locfsthrm
dataInitial.LOCFSTHRM = dataInitial.LOCFSTHRM.replace(locfsthrmInputs, locfsthrmOutputs)
```

```
#crcomnr
dataInitial.CRCOMNNR = dataInitial.CRCOMNNR.replace(crcomnrInputs, crcomnrOutputs)
```

```
#majcse
dataInitial.MAJCSE = dataInitial.MAJCSE.replace(majcseInputs, majcseOutputs)
```

```
#drugalc
dataInitial.DRUGALC = dataInitial.DRUGALC.replace(drugalcInputs, drugalcOutputs)
```

```
#ecntcrc
dataInitial.ECNTCRC = dataInitial.ECNTCRC.replace(ecntcrcInputs, ecntcrcOutputs)
```

```
#light
dataInitial.LIGHT = dataInitial.LIGHT.replace(lightInputs, lightOutputs)
```

```
#csrfcnd
dataInitial.CSRFCND = dataInitial.CSRFCND.replace(csrfcndInputs, csrfcndOutputs)
```

```
#weather
dataInitial.WEATHER = dataInitial.WEATHER.replace(weatherInputs, weatherOutputs)
```

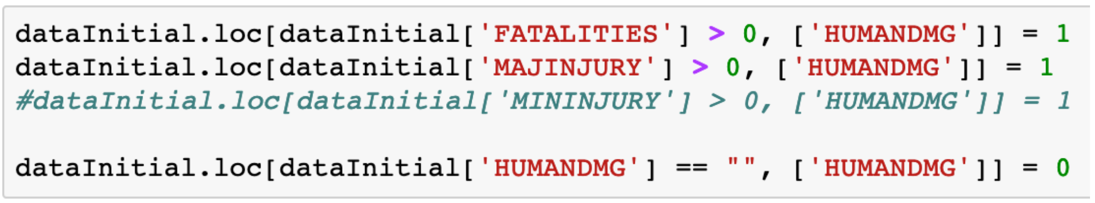
```
#rcntcrc
dataInitial.RCNTCRC = dataInitial.RCNTCRC.replace(rcntcrcInputs, rcntcrcOutputs)
```

```
#rdtyp
dataInitial.RDTYP = dataInitial.RDTYP.replace(rdtypInputs, rdtypOutputs)
```

Figure 31 - Feature key to description string transformation example

5.2.5 RTC severity classification data definition

RTC severity classification is the expected outcome of this document's work and two alternatives were considered as described in Table 9 and Table 10. The data transformation mentioned in Figure 32 was used to accomplish the proposal's goals using a new column in the dataset, named *HUMANDMG*, to store the outcome.

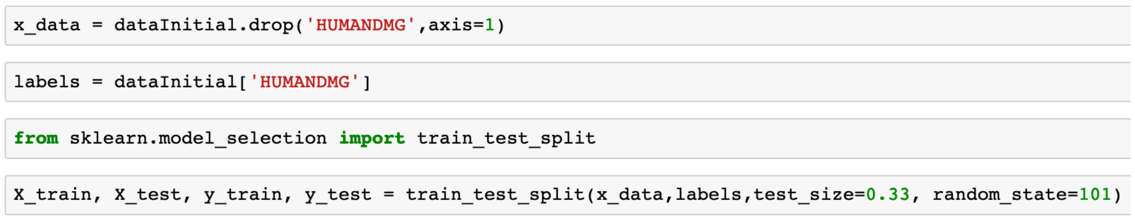


```
dataInitial.loc[dataInitial['FATALITIES'] > 0, ['HUMANDMG']] = 1
dataInitial.loc[dataInitial['MAJINJURY'] > 0, ['HUMANDMG']] = 1
#dataInitial.loc[dataInitial['MININJURY'] > 0, ['HUMANDMG']] = 1

dataInitial.loc[dataInitial['HUMANDMG'] == "", ['HUMANDMG']] = 0
```

Figure 32 - Data handling to define RTC severity classification

5.2.6 Training and test sample datasets

One of the necessary steps to train and evaluate a model is to split the initial dataset into training and test sample datasets, which are correspondently used to train the model and to evaluate it. According to what is explained in the 4.2.2 section, 33% of the original dataset was defined as a test set. Following the decision stated in the 5.2.5 section above, *HUMANDMG* is used as label, therefore representing the expected outcome. This process can be seen in the Figure 33.



```
x_data = dataInitial.drop('HUMANDMG',axis=1)
```

```
labels = dataInitial['HUMANDMG']
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(x_data,labels,test_size=0.33, random_state=101)
```

Figure 33 - Dataset split into training and test samples following holdout method

5.3 Model definition

This chapter is meant to explain the technical approach to build the model according to the proposed methodology defined in the 4.2.2 section. The parameter definition of the model as well as various design alternatives are also approached.

5.3.1 Feature columns definition

Following Tensorflow's approach to feature columns configuration, there are two types of feature columns to be considered:

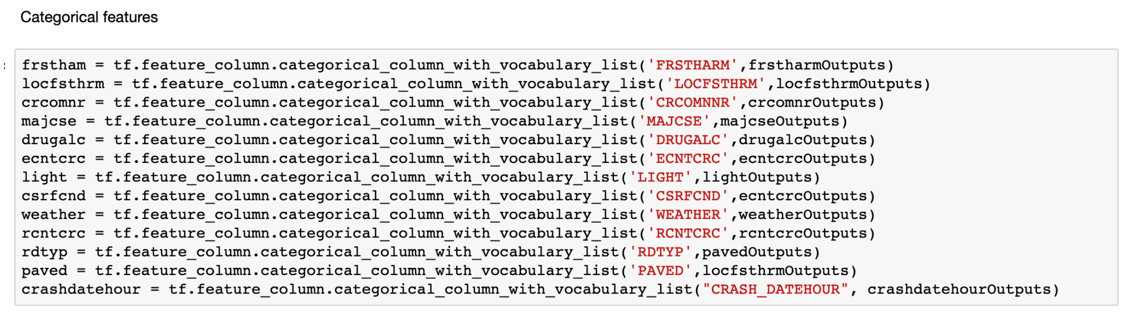
- Categorical – Each entry represents a category. Discrete data;
- Numeric – Each entry represents a continuous value. Numeric data.

Regarding categorical feature columns, two sub-types are considered:

- With vocabulary list – Input values should obey to an already defined set of values. Any introduced value outside the range of this set will trigger an error, providing full control of the allowed values;
- With hash bucket – Maximum amount of input values is defined beforehand. Consumes less memory and provides more flexibility as it is not necessary to specify all required values.

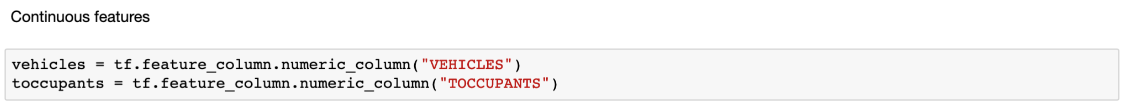
The dataset features' identification into categorical or numeric was done with the support of the data placed in Table 1.

Reliability and control were prioritized over flexibility and performance as this document's work is related with health. Therefore, the decision was to proceed with vocabulary lists instead of hash buckets. The outputs definition created in the 5.2.4 section was used to specify the set of possible values to be introduced in the vocabulary list. The creation of the model's feature columns can be seen in Figure 34 and Figure 35.


Categorical features

```
frstham = tf.feature_column.categorical_column_with_vocabulary_list('FRSTHARM',frstharmOutputs)
locfsthrm = tf.feature_column.categorical_column_with_vocabulary_list('LOCFSTHRM',locfsthrmOutputs)
crcomnr = tf.feature_column.categorical_column_with_vocabulary_list('CRCOMNNR',crcomnrOutputs)
majcse = tf.feature_column.categorical_column_with_vocabulary_list('MAJCSE',majcseOutputs)
drugalc = tf.feature_column.categorical_column_with_vocabulary_list('DRUGALC',drugalcOutputs)
ecntcrc = tf.feature_column.categorical_column_with_vocabulary_list('ECNTCRC',ecntcrcOutputs)
light = tf.feature_column.categorical_column_with_vocabulary_list('LIGHT',lightOutputs)
csrfcnd = tf.feature_column.categorical_column_with_vocabulary_list('CSRFCND',ecntcrcOutputs)
weather = tf.feature_column.categorical_column_with_vocabulary_list('WEATHER',weatherOutputs)
rcntcrc = tf.feature_column.categorical_column_with_vocabulary_list('RCNTCRC',rcntcrcOutputs)
rdtyp = tf.feature_column.categorical_column_with_vocabulary_list('RDTYP',pavedOutputs)
paved = tf.feature_column.categorical_column_with_vocabulary_list('PAVED',locfsthrmOutputs)
crashdatehour = tf.feature_column.categorical_column_with_vocabulary_list("CRASH_DATEHOUR", crashdatehourOutputs)
```

Figure 34 - Model's categorical feature columns definition


Continuous features

```
vehicles = tf.feature_column.numeric_column("VEHICLES")
toccupants = tf.feature_column.numeric_column("TOCCUPANTS")
```

Figure 35 - Model's numeric feature columns definition

5.3.2 Models' hyperparameter definition

Various variations of the model were elaborated having in mind the decisions exposed in the 4.2.2 section based in other studies in the RTC field. A set of parameters, which content can vary, was defined to accommodate various design alternatives.

5.3.2.1 RTC severity using classification setup

Tensorflow permits the usage of a DNNClassifier for classification problems using deep neural networks. The input function creation in Tensorflow allows the customization of **batch size** (i.e. number of samples to be propagated in the DNN) and **number of epochs** (i.e. maximum iterations for the algorithm to run on top of the entire dataset during training). During Tensorflow's training optimizer definition it's possible to customize the **learning rate** and **optimizer**. The **activation function** parameter is also manually introduced during the model instantiation. Even though Tensorflow automatically defines a default **neural network structure**, and in order to test different architectural approaches, different DNN structures were used. Variations are based on complexity and deepness.

Random Forest implementation is still recent in Tensorflow's framework, therefore requiring an upgrade for the 1.13 version during this algorithm's implementation. Even though some features for this algorithm's implementation are lacking, it is possible to customize the **maximum number of nodes** and **number of trees**. The number of Random Forest variations is smaller than DNN variations as it is expected for DNN to perform better according to what is mentioned in 4.2.2.

The DNN parameters variations are presented in Table 13, which values are separated by commas, and structural alternatives are described in Table 14. Table 15 provides information about the used random forest's parameter variations.

Table 13 – RTC severity neural network parameters variations

Parameter	**Set of values**
Batch size	10, 32, 128, 256
Number of epochs	1000, 5000, 10000
Learning rate	0.01, 0.02, 0.03, 0.04, 0.05, 0.3
Optimizer	GradientDescent, Adam
Activation function	Softplus, Rectified Linear Unit (ReLU), Sigmoid, Tanh

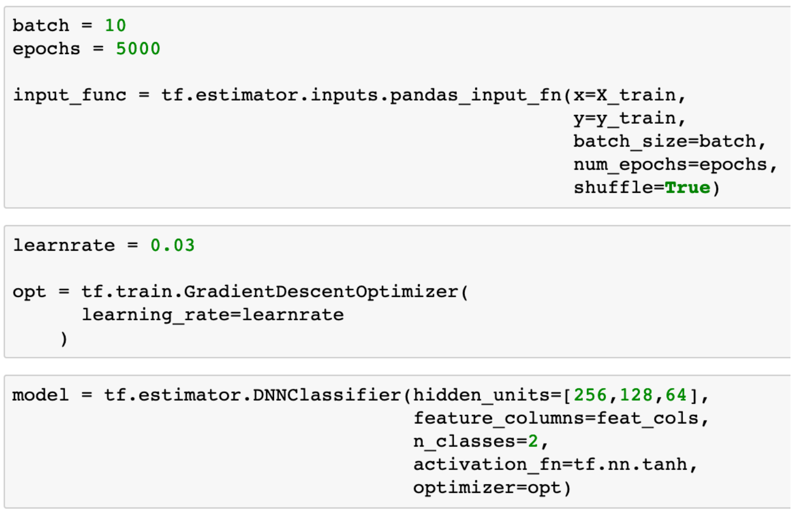
Table 14 - RTC severity neural network structure alternatives

Alternative	Value
A1	[256, 128, 64]
A2	[1024, 512, 256, 128, 64]
A3	[4096, 2048, 1024, 512, 256, 128, 64]
A4	[8192, 4096, 2048, 1024, 512, 256, 128, 64]
A5	[16384, 8192, 4096, 2048, 1024, 512, 256, 128, 64]
A6	[16, 16, 16, 16, 16, 16]
A7	[64, 64, 64, 64, 64, 64, 64, 64, 64]
A8	[256, 256, 256, 256, 256, 256, 256, 256, 256]

Table 15 - Random forest parameter variations

Parameter	Set of values
Max number of nodes	1000, 3000, 5000, 10000
Number of trees	10

A new instance of a model is then created using Tensorflow *estimator* package. Random forest requires *tensor_forest* package to create new instances. These processes are represented in Figure 36 and Figure 37.

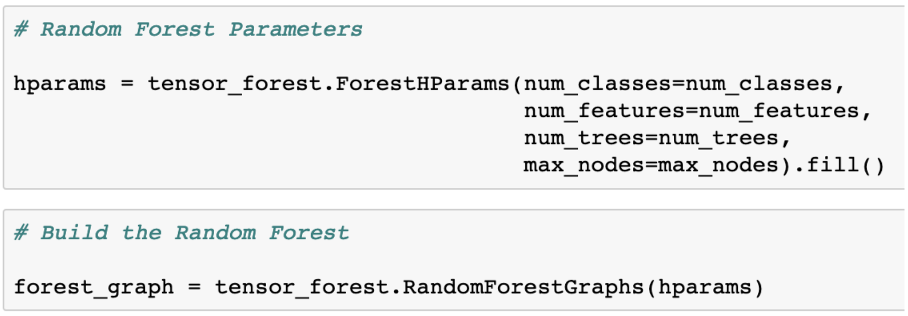


```
batch = 10
epochs = 5000

input_func = tf.estimator.inputs.pandas_input_fn(x=X_train,
                                                 y=y_train,
                                                 batch_size=batch,
                                                 num_epochs=epochs,
                                                 shuffle=True)
```

```
learnrate = 0.03

opt = tf.train.GradientDescentOptimizer(
      learning_rate=learnrate
    )
```

```
model = tf.estimator.DNNClassifier(hidden_units=[256,128,64],
                                   feature_columns=feat_cols,
                                   n_classes=2,
                                   activation_fn=tf.nn.tanh,
                                   optimizer=opt)
```

Figure 36 - Tensorflow creation of a DNNClassifier model instance



```
# Random Forest Parameters

hparams = tensor_forest.ForestHParams(num_classes=num_classes,
                                      num_features=num_features,
                                      num_trees=num_trees,
                                      max_nodes=max_nodes).fill()
```

```
# Build the Random Forest

forest_graph = tensor_forest.RandomForestGraphs(hparams)
```

Figure 37 - Tensorflow creation of a Random Forest model instance

5.3.2.2 RTC cost using regression setup

To implement regression models in Tensorflow it's necessary to use a DNNRegressor classifier. Utilizing this classifier and taking into account some of the properties already explained in 5.3.2.1, it's possible to customize the **optimizer**, **learning rate**, **batch size** and **number of epochs**, which variations are represented in Table 16. Tensorflow also allows the manual setup of the **neural network structure** as represented in Table 17.

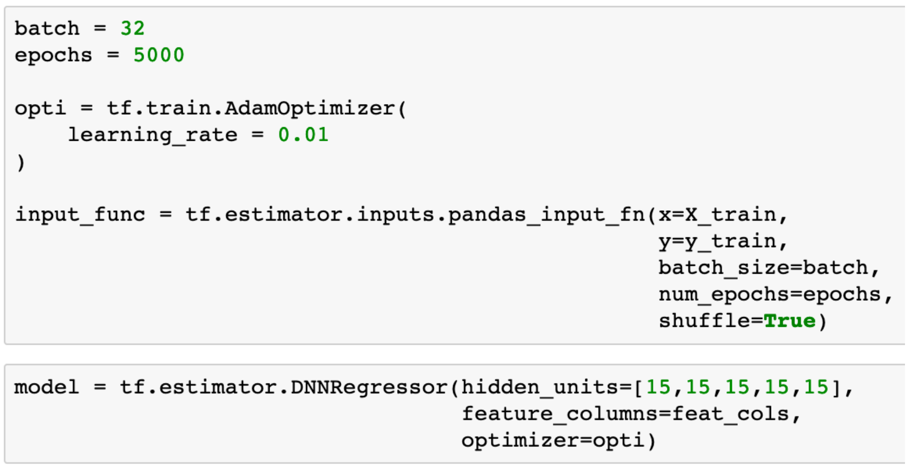
Table 16 - RTC cost neural network parameters variations

Parameter	**Set of values**
Batch size	10, 32
Number of epochs	5000, 10000
Learning rate	0.01, 0.03
Optimizer	GradientDescent, Adam

Table 17 - RTC cost neural network structure variations

Alternative	**Value**
B1	[15, 15, 15]
B2	[60, 30, 15]
B3	[120, 60, 30, 15]
B4	[240, 120, 60, 30, 15]
B5	[15, 15, 15, 15, 15]

With the above in mind, various variations were built in order to combine different design alternatives in the regression models. The DNN model instance creation can be seen in Figure 38, which process is similar to the one used for deep neural networks mentioned in the 5.3.2.1 section, therefore allowing a consistent implementation across both contexts.



```
batch = 32
epochs = 5000

opti = tf.train.AdamOptimizer(
    learning_rate = 0.01
)

input_func = tf.estimator.inputs.pandas_input_fn(x=X_train,
                                                 y=y_train,
                                                 batch_size=batch,
                                                 num_epochs=epochs,
                                                 shuffle=True)
```

```
model = tf.estimator.DNNRegressor(hidden_units=[15,15,15,15,15],
                                  feature_columns=feat_cols,
                                  optimizer=opti)
```

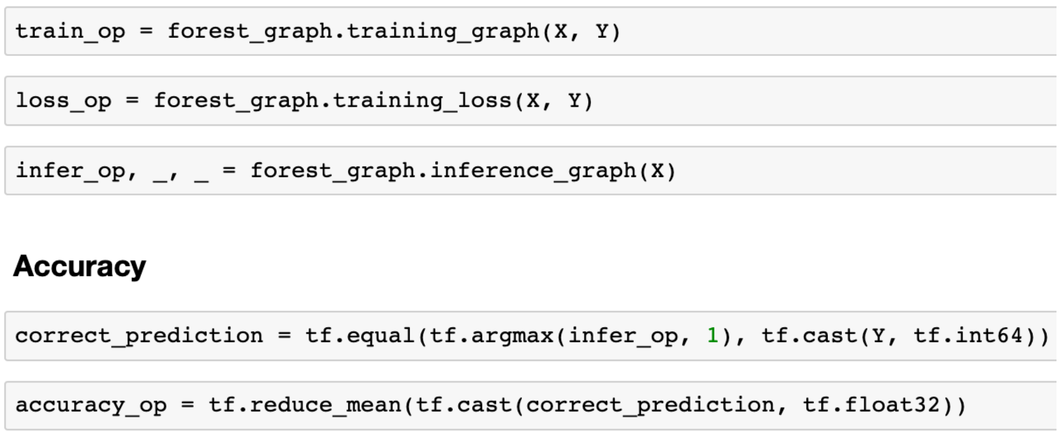
Figure 38 - Tensorflow creation of a DNNRegressor model instance

5.4 Train and evaluate

This section is meant to expose the training and evaluation process for the classification and regression models which variations are mentioned in 5.3.2.1 and 5.3.2.2.

5.4.1 Classification

Tensorflow's estimator package provides tools for the neural network models' training and subsequent evaluation. As seen in Figure 39, for Random Forest instances it is necessary to manually define the training operation and accuracy calculation.



```
train_op = forest_graph.training_graph(X, Y)
```

```
loss_op = forest_graph.training_loss(X, Y)
```

```
infer_op, _, _ = forest_graph.inference_graph(X)
```

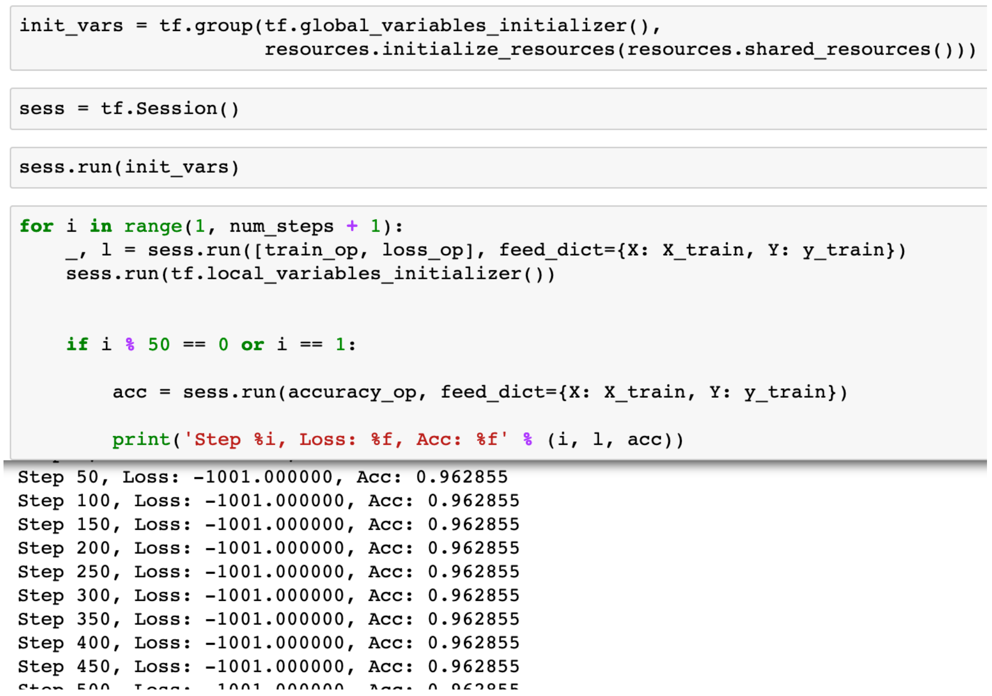
Accuracy

```
correct_prediction = tf.equal(tf.argmax(infer_op, 1), tf.cast(Y, tf.int64))
```

```
accuracy_op = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Figure 39 - Definition of training and accuracy operation in Random Forest instance

The DNN training function provides some real-time data information during the process by default. The number of steps during training was set to 5000 following community standards. In some scenarios it was also tested 10000 steps as alternative. Training process example is shown in Figure 41. The training process in the Random Forest is executed on top of a Tensorflow's session and as seen in Figure 40, it uses the previously defined accuracy and training functions. The number of steps in Random Forest varies between 100, 500, 1000, 2000, and 10000.



```
init_vars = tf.group(tf.global_variables_initializer(),
                     resources.initialize_resources(resources.shared_resources()))
```

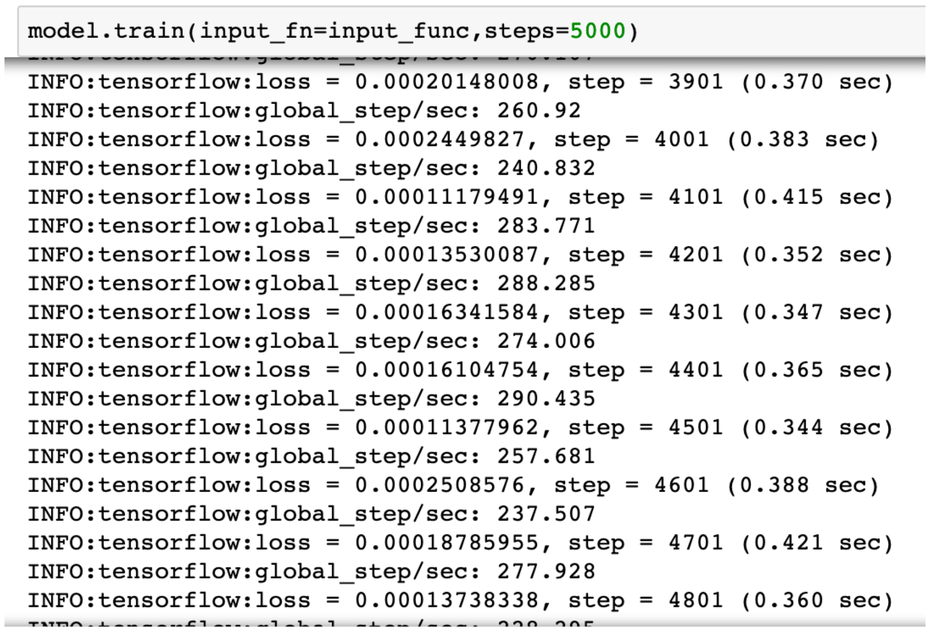
```
sess = tf.Session()
```

```
sess.run(init_vars)
```

```
for i in range(1, num_steps + 1):
    _, l = sess.run([train_op, loss_op], feed_dict={X: X_train, Y: y_train})
    sess.run(tf.local_variables_initializer())


    if i % 50 == 0 or i == 1:

        acc = sess.run(accuracy_op, feed_dict={X: X_train, Y: y_train})

        print('Step %i, Loss: %f, Acc: %f' % (i, l, acc))
```

```
Step 50, Loss: -1001.000000, Acc: 0.962855
Step 100, Loss: -1001.000000, Acc: 0.962855
Step 150, Loss: -1001.000000, Acc: 0.962855
Step 200, Loss: -1001.000000, Acc: 0.962855
Step 250, Loss: -1001.000000, Acc: 0.962855
Step 300, Loss: -1001.000000, Acc: 0.962855
Step 350, Loss: -1001.000000, Acc: 0.962855
Step 400, Loss: -1001.000000, Acc: 0.962855
Step 450, Loss: -1001.000000, Acc: 0.962855
```

Figure 40 - Random Forest model's training process



```
model.train(input_fn=input_func,steps=5000)
```

```
INFO:tensorflow:loss = 0.00020148008, step = 3901 (0.370 sec)
INFO:tensorflow:global_step/sec: 260.92
INFO:tensorflow:loss = 0.0002449827, step = 4001 (0.383 sec)
INFO:tensorflow:global_step/sec: 240.832
INFO:tensorflow:loss = 0.00011179491, step = 4101 (0.415 sec)
INFO:tensorflow:global_step/sec: 283.771
INFO:tensorflow:loss = 0.00013530087, step = 4201 (0.352 sec)
INFO:tensorflow:global_step/sec: 288.285
INFO:tensorflow:loss = 0.00016341584, step = 4301 (0.347 sec)
INFO:tensorflow:global_step/sec: 274.006
INFO:tensorflow:loss = 0.00016104754, step = 4401 (0.365 sec)
INFO:tensorflow:global_step/sec: 290.435
INFO:tensorflow:loss = 0.00011377962, step = 4501 (0.344 sec)
INFO:tensorflow:global_step/sec: 257.681
INFO:tensorflow:loss = 0.0002508576, step = 4601 (0.388 sec)
INFO:tensorflow:global_step/sec: 237.507
INFO:tensorflow:loss = 0.00018785955, step = 4701 (0.421 sec)
INFO:tensorflow:global_step/sec: 277.928
INFO:tensorflow:loss = 0.00013738338, step = 4801 (0.360 sec)
```

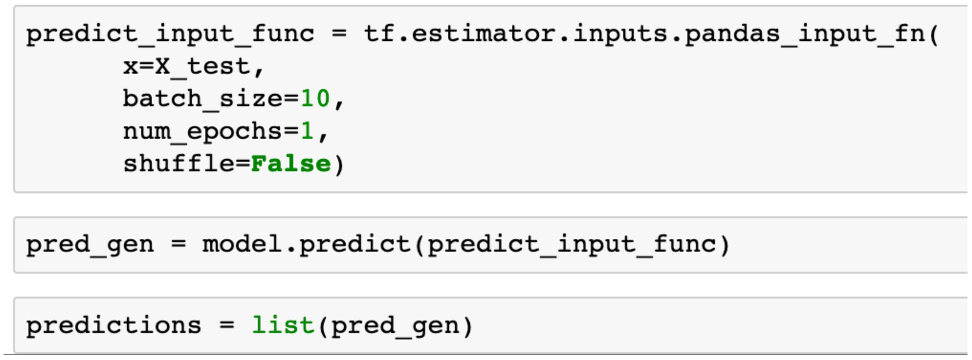
Figure 41 – DNN model's training process

The model's evaluation process runs on top of the test data samples defined in Figure 33. Tensorflow's *estimator* evaluate function retrieves an object with evaluation metrics (e.g. Accuracy and AUC), which are used in the next chapter to discuss the model's predictive ability. Random Forest evaluation metric is limited to Accuracy due to technical constraints on the calculation of AUC as this metric is not provided by default on Tensorflow's implementation of this algorithm. Some variations of DNN models using GradientDescent combined with ReLU resulted in a *NaN error*, which is a known issue in the community[17] due to unbounded outputs, which make the neural network especially prone to exploding gradient issues.

5.4.2 Regression

Using a DNNRegressor classifier there's no need to manually configure the training and evaluation process as Tensorflow provides methods for both purposes. The training process is similar to the one represented above in Figure 41. During the training process it was detected some issues with GradientDescent optimizer, which were already felt in the classification training process as mentioned in 5.4.1. The number of steps during training was set to 5000 by default.

Predictions were created for the evaluation process as seen in Figure 42. These predictions were then compared with the real data from the test dataset defined in Figure 33, therefore allowing the calculation of MSE and MedAE as shown in Figure 43. During the evaluation process analysis, some feature variations were made for testing purses. Therefore, a variation without the *LIGHT*, *CSRFCND* and *WEATHER* features was considered for testing purposes.



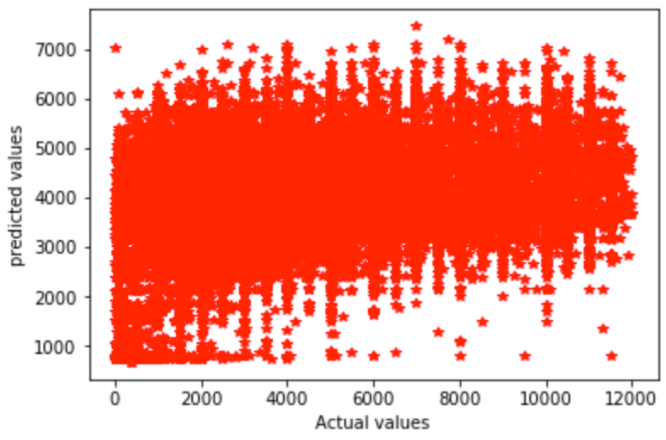
```
predict_input_func = tf.estimator.inputs.pandas_input_fn(
      x=X_test,
      batch_size=10,
      num_epochs=1,
      shuffle=False)
```

```
pred_gen = model.predict(predict_input_func)
```

```
predictions = list(pred_gen)
```

Figure 42 - Creation of predictions

[17] https://stackoverflow.com/questions/37232782/nan-loss-when-training-regression-network



```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import median_absolute_error
```

```
mean_squared_error(y_test,final_preds)**0.5
```

```
2526.559332135075
```

```
median_absolute_error(y_test,final_preds)
```

```
1683.11669921875
```

Figure 43 - Calculation of MSE and MedAE

6 Evaluation

In this chapter is possible to find the used metrics, evaluation methodologies and the final results. It is also explained how the solution was evaluated considering all the variations that were considered.

6.1 Evaluation metrics, methodologies and hypothesis

Typically, the metrics to be considered for the solution's **classification** evaluation are Accuracy, Precision, Recall, F-measure and AUC. Tensorflow's default evaluation function metrics do not include Precision, Recall and F-measure (as shown in 5.4.1 section). Even though Tensorflow allows the construction of custom metrics and subsequent addition to the evaluation process, this was a time-consuming process and due to time and technical constraints it was not possible to consider these metrics. Accuracy is the most referenced metric in the 3.2 chapter among studies in the RTC field. Some of these studies also mention AUC. Therefore, AUC and Accuracy are the used metrics.

For **regression**, the most typical metric is MSE as mentioned earlier on 3.2.5 section. As also mentioned in that section, MedAE is also used mainly in scenario where the error dispersal is high. Following the plots analysis on 5.2.2.2, which indicate a high probably of significant error dispersal, MedAE was also included in the set of used regression metrics.

As previously explained in the section 4.2 and demonstrated on chapter 5 of this document, various models' variations were built. Classification models vary on:

- **Algorithm alternatives using DNN and Random Forest** – Proposed in 4.2.2, implementation described in 5.3.2.1 and 5.4.1;
- **Various data cleaning approaches** – Initially explained in 2.3.3.2 and 4.2.1, implementation described in 5.2.3 section;

- **Different RTC severity classification classes definition** – As seen in Table 9 and Table 10 , concept concretization shown in 5.2.5;
- **DNN parameter variations** – Explained in 4.2.2, set of possible values shown in 5.3.2.1 and Table 13;
- **DNN structure alternatives** – Approached in 3.2.2 and 4.2.2 sections, configuration described in detail in 5.3.2.1 and more precisely in Table 14.

For regression, the following variations are considered:

- **Various data cleaning approaches** – Approached in 2.3.3.2 and 4.2.1, implementation described in 5.2.3 and more specially in 5.2.3.5 section;
- **DNN parameter variations** – Explained in 4.2.2, variations described in detail on 5.3.2.2 section and Table 16;
- **DNN structure alternatives** – Approached in 3.2.2 and 4.2.2 sections, regression models variations shown in 5.3.2.2 and more precisely in Table 17;
- **Features variations** – Initially approached in 3.2.2 section, considered after analysis explained in 5.4.2.

Considering the variations mentioned above, a comparison between the models' results is performed to understand which model fits this problem's context the best, both for classification and regression.

Holdout (referenced in the 2.3.4.3 section and implementation mentioned in 5.2.6) is used to evaluate and understand if the algorithm is under an overfitting or underfitting situation, where the algorithm's performance isn't reliable.

As the intention is to compare classifiers, the null hypothesis is that all the classifiers perform equally. The rejection of that null hypothesis means that there exists at least one pair of classifiers with significantly different performances. It is possible to use Friedman, a non-parametric test. In case of rejection of this null hypothesis, post-hoc test to identify the significantly different pairs of classifiers. It is possible to use the Nemenyi test (post-hoc test).

6.2 RTC severity classification results

Specific analysis and implementation of RTC severity are mentioned in the 5.2.2.1, 5.2.5, 5.3.2.1 and 5.4.1 sections of this document. **Accuracy** and **AUC** are used for the classification evaluation.

The below tables display the models' results. Due to the amount of information to be displayed, the DNN and Random Forest results were split into different tables due to having different parameter definition. The best results from more than 250 variations were selected to be shown in the tables, in which the best records from each RTC severity alternative (see Table 9 and Table 10) are marked as bold.

The DNN classifier results table structure is the following:

- Optimizer;
- Activation function, column named as *AF*;
- Learning rate, column named as *LR*;
- Neural network structure, following the Table 14 alternative acronyms, column named as *NNS*;
- Number of epochs, column named as *NE*;
- Batch size, column named as *BS*;
- Accuracy;
- AUC.

For Random Forest results table the following structure was used:

- Max number of nodes;
- Number of trees;
- Steps;
- Accuracy.

On Table 18 it's possible to find the best results using DNN models for alternative 1 of RTC. The best results were achieved by combining Adam and ReLU with a simple DNN structure, which allowed a result containing an accuracy of 0.862 and AUC of 0.73. The table also shows that other combinations using GradientDescent and Tanh also perform close to the results achieved using Adam and ReLU. The model that performed with the highest AUC was also combination of Adam and ReLU but with a higher batch size, resulting in an inferior accuracy.

Table 18 - DNN results for alternative 1 of RTC

Optimizer	**AF**	**LR**	**NNS**	**NE**	**BS**	**Accuracy**	**AUC**
Adam	**ReLU**	**0.01**	**A1**	**5000**	**128**	**0.862**	**0.73**
GradientDescent	**Tanh**	**0.02**	**A3**	**5000**	**10**	**0.862**	**0.72**
GradientDescent	**Tanh**	**0.01**	**A3**	**5000**	**10**	**0.862**	**0.718**
Adam	Tanh	0.01	A1	5000	10	0.862	0.53
Adam	ReLU	0.01	A1	1000	128	0.861	0.73
GradientDescent	Tanh	0.03	A1	5000	10	0.861	0.71
GradientDescent	Tanh	0.02	A3	10000	10	0.860	0.7
GradientDescent	Tanh	0.01	A2	5000	10	0.859	0.723
GradientDescent	Tanh	0.01	A2	1000	10	0.859	0.718
GradientDescent	Tanh	0.03	A8	5000	10	0.859	0.5
Adam	ReLU	0.01	A1	5000	256	0.858	**0.737**
Adam	ReLU	0.01	A3	5000	128	0.858	0.727

Optimizer	AF	LR	NNS	NE	BS	Accuracy	AUC
Adam	ReLU	0.01	A2	5000	128	0.858	0.727
Adam	ReLU	0.01	A1	1000	32	0.858	0.717
Adam	ReLU	0.01	A1	5000	32	0.858	0.713

Table 19 displays the best results for DNN models for alternative 2 of RTC. Similarly to what is explained above on Table 18 results' analysis, the best results for this alternative were also achieved using a combination of Adam and ReLU and a simple DNN structure. The result was an accuracy of 0.963 and AUC of 0.777. The best accuracy values are consistent throughout the multiple models' variations in which AUC varies more.

Table 19 - DNN results for alternative 2 of RTC

Optimizer	AF	LR	NNS	NE	BS	Accuracy	AUC
Adam	**ReLU**	**0.01**	**A1**	**5000**	**256**	**0.963**	**0.777**
GradientDescent	**ReLU**	**0.01**	**A1**	**1000**	**128**	**0.963**	**0.771**
GradientDescent	**ReLU**	**0.01**	**A1**	**5000**	**128**	**0.963**	**0.771**
Adam	ReLU	0.01	A1	5000	128	0.963	0.769
Adam	ReLU	0.01	A3	1000	128	0.963	0.765
GradientDescent	ReLU	0.03	A1	1000	32	0.963	0.762
GradientDescent	ReLU	0.01	A1	5000	32	0.963	0.761
GradientDescent	Tanh	0.03	A1	5000	10	0.963	0.756
GradientDescent	ReLU	0.01	A1	1000	32	0.963	0.755
Adam	ReLU	0.01	A1	5000	32	0.963	0.754
GradientDescent	Tanh	0.03	A1	10000	10	0.963	0.752
GradientDescent	Tanh	0.02	A3	5000	10	0.963	0.751
GradientDescent	Tanh	0.01	A2	1000	10	0.963	0.749
GradientDescent	Tanh	0.03	A6	5000	10	0.963	0.744
GradientDescent	Tanh	0.02	A4	5000	10	0.963	0.742

On Table 20 is possible to find the best results for alternative 1 of RTC with low impact data removed using DNN models. As it possible to state when looking at the results, the experimented models perform worse than the ones mentioned in Table 18. Best result achieved with GradientDescent and Tanh under a simple DNN structure.

Table 20 - DNN results for alternative 1 of RTC with low impact data removed

Optimizer	**AF**	**LR**	**NNS**	**NE**	**BS**	**Accuracy**	**AUC**
GradientDescent	Tanh	0.03	A1	1000	10	0.855	0.719
GradientDescent	Tanh	0.03	A3	1000	10	0.855	0.713
Adam	ReLU	0.03	A3	5000	128	0.855	0.687
GradientDescent	Tanh	0.03	A2	1000	10	0.855	0.649
Adam	ReLU	0.01	A3	5000	32	0.855	0.5

Similarly to Table 20, Table 21 displays the best results for DNN models with low impact data removed but for alternative 2 of RTC. When compared to Table 19 results, the models described below also perform worse even though the performance difference isn't as noticeable as RTC alternative 1 results. GradientDescent and Tanh combination with a simple DNN structure provided reasonable AUC results.

Table 21 – DNN results for alternative 2 of RTC with low impact data removed

Optimizer	**AF**	**LR**	**NNS**	**NE**	**BS**	**Accuracy**	**AUC**
GradientDescent	Softplus	0.03	A3	5000	10	0.963	0.5
GradientDescent	Tanh	0.03	A1	5000	10	0.962	**0.756**
GradientDescent	Tanh	0.03	A1	1000	10	0.962	0.752
GradientDescent	Softplus	0.03	A1	5000	10	0.962	0.732
GradientDescent	Tanh	0.03	A3	1000	10	0.962	0.731

Table 22 demonstrates the results of Random Forest models for alternative 1 of RTC. The best results were found using a max number of nodes of 5000 and 1000 in which the accuracy achieved 0.86. Still, these models' accuracy is still not as good as the DNN variations shown in Table 18.

Table 22 - Random Forest results for alternative 1 of RTC

Max number of nodes	**Number of trees**	**Steps**	**Accuracy**
5000	10	500	0.86
1000	10	1000	0.86
10000	10	2000	0.859
5000	10	2000	0.859
3000	10	2000	0.859
10000	10	1000	0.859
10000	10	500	0.859
10000	10	100	0.859

Max number of nodes	Number of trees	Steps	Accuracy
5000	10	100	0.859
3000	10	500	0.859

In Table 23 it's possible to find Random Forest results for alternative 2 of RTC. Comparably to what was observable in Table 19 (DNN results for alternative 2 of RTC), the accuracy values are also very consistent independently of the models' parameters.

Table 23 - Random Forest results for alternative 2 of RTC

Max number of nodes	Number of trees	Steps	Accuracy
5000	10	2000	0.963
3000	10	2000	0.963
10000	10	2000	0.963
10000	10	1000	0.963
5000	10	1000	0.963
1000	10	2000	0.963
1000	10	1000	0.963
10000	10	500	0.963
3000	10	1000	0.963
5000	10	500	0.963

6.2.1 Results analysis

Having in mind that the initial dataset contained a total of 573548 samples, the following was considered for evaluation purposes:

- **Alternative 1** (results in Table 18 and Table 22) – 411633 remained after data cleaning process, from which 59069 correspond to the positive class. Therefore, **0.856** is the minimum accuracy value for a positive evaluation;
- **Alternative 1 with low impact data removed** (results in Table 20) – 408686 remained after data cleaning process, from which 58870 correspond the positive class. Therefore, **0.855** is the minimum accuracy value for a positive evaluation;
- **Alternative 2** (results in Table 19 and Table 23) – 411633 remained after data cleaning process, from which 15241 correspond to the positive class. Therefore, **0.962** is the minimum accuracy value for a positive evaluation;
- **Alternative 2 with low impact data removed** (results in Table 21) – 408686 remained after data cleaning process, from which 15198 correspond the positive class. Therefore, **0.962** is the minimum accuracy value for a positive evaluation.

As it is possible to state from the tables shown in section 6.2, it was possible to achieve **marginal positive evaluations** (i.e. where the model's accuracy is bigger than the minimum values defined above but not significantly) to both RTC severity alternatives.

The best results for RTC severity alternative 1 are held by a combination of Adam optimizer with ReLU which resulted in an accuracy of **0.862** and AUC of **0.73**. For the alternative 2, the best results were also achieved with Adam optimizer and ReLU combining in an accuracy of **0.963** and AUC of **0.777**. During this process, other variations with a smaller set of features were experimented but all results were significantly worst, therefore not considered for this document's evaluation purpose.

6.2.1.1 Other considerations

Adam and GradientDescent perform in a similar way as there's not a significant discrepancy between both optimizers' results. This outcome was expected considering what was explained in 4.2.2 section and can be stated by the top results representation on Figure 44 and Figure 45.

Figure 44 demonstrates the accuracy fluctuation per model variation. The top model variations are represented in the X axis, where the 0 corresponds to the best result, 1 to the second best and continuously like this until the fifth best result. As it is possible to observe, accuracy fluctuation across variations is not high as the results are similar between the various approaches even though DNN with Adam and GradientDescent containing all data perform slightly better.

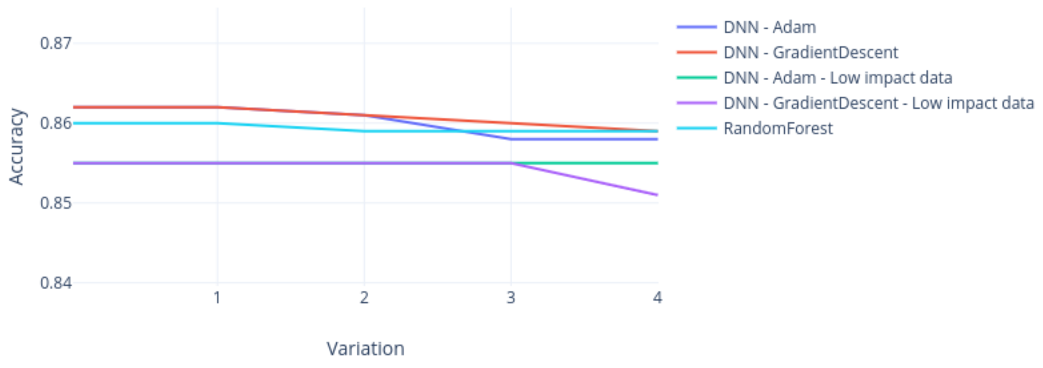


Figure 44 - Accuracy fluctuation comparison on top classification models for RTC alternative 1

The same logic from Figure 44 is applied to Figure 45 but focused in the AUC results. Similarly to what is possible to observe on Figure 44, DNN using Adam and GradientDescent will all data perform similarly, with the AUC fluctuation between both models' variations being consistent and following a observable tendency. Variations with removed low impact data have a higher discrepancy in the AUC results.

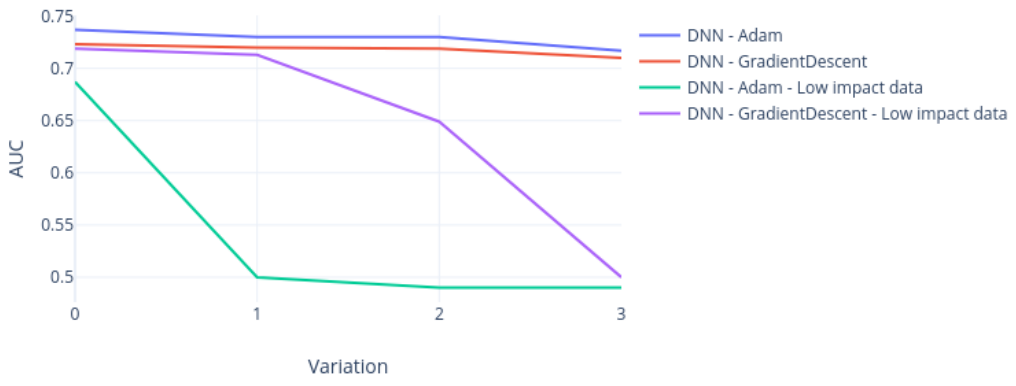


Figure 45 - AUC fluctuation on top classification models for RTC alternative 1

The tables presented in section 6.2 show that the best results are usually achieved by using either ReLU or Tanh under low learning rates. Significantly deeper or complex DNN structure alternatives, as A7 and A5 from Table 14, don't perform as well as other simpler structures, as A1 and A3. The best results' batch size and number of epochs vary more.

Considering the technical restrictions found to calculate AUC for Random Forest models, it is not possible to conclude if RTC severity alternative 2 algorithm variation using Random Forest (Table 23) performs better than DNN (Table 19) as the accuracy value it's the same and it is not possible to compare the AUC values. Even though these technical restrictions are also noticeable comparing RTC severity alternative 1 models, as the best accuracy outcome for DNN (Table 18) is higher than the best accuracy outcome for Random Forest (Table 22), it is possible to consider that DNN may perform better under this context.

The results achieved with low impact data being removed (Table 20 and Table 21) are worse when compared with the variations that use more data (Table 18, Table 19, Table 22, and Table 23). This can be related with the fact that the values removed from dataset, even though are the ones with the fewer occurrences, are important values to calibrate the model. Variations with fewer features performed significantly worse and therefore are not considered for this document's evaluation process.

6.3 RTC cost regression results

Besides the relevant information presented in prior chapters of this document, the explicit cost analysis and implementation approach are described on sections 5.2.2.2, 5.2.3.5, 5.3.2.2 and 5.4.2. **MSE** and **MedAD** are used for the classification evaluation. Having in mind the errors found while using GradientDescent in the regression models, which are mentioned in 5.4.2, the models' outcomes presented below only use Adam optimizer.

Best results are marked as bold. The tables consider the following structure:

- Learning rate, column named as *LR*;
- Neural network structure, following the Table 17 alternative acronyms, column named as *NNS*;
- Number of epochs, column named as *NE*;
- Batch size, column named as *BS*;
- MSE;
- MedAD.

Table 24 provides information about the results for regression model which data had no outliers and contains all features following the procedure explained in detail on 5.2.3.5. The instance with the lowest MedAE record, therefore the best result, was **1548.24** in which MSE recorded **2593.48**. The instance with the lowest MSE achieved **2510.13**.

Lower learning rates combined with a batch size of 32 and number of epochs around 10000 seem to perform better as several models with these characteristics appear on the top results seen in Table 24. DNN structure is the less consistent parameter as all five possible variations are represented in the results below.

Table 24 - Regression results for variation without outliners and with all features

LR	**NNS**	**NE**	**BS**	**MSE**	**MedAE**
0.03	**B3**	**10000**	**32**	**2593.48**	**1548.24**
0.01	**B2**	**10000**	**32**	**2534.39**	**1612.56**
0.01	**B1**	**5000**	**32**	**2545.93**	**1617.02**
0.03	B4	10000	32	2546.75	1618.12
0.03	B3	10000	32	2540.78	1634.80
0.01	B1	10000	32	2535.07	1661.50
0.01	B5	10000	32	2520.56	1726.53
0.01	B3	10000	10	**2516.69**	1726.78
0.01	B5	10000	10	2525.93	1729.79
0.01	B4	5000	32	**2519.59**	1748.34

On Table 25 it is possible to find the results for regression models without outliners but not containing the some features, which were removed during the process explained in 5.4.2 section (*LIGHT*, *CSRFCND* and *WEATHER*). This table contains records which correspond to the top results present in Table 24 to allow a direct comparison between both approaches.

Table 25 - Regression results for variation without outliners and with reduced features

LR	NNS	NE	BS	MSE	MedAE
0.03	B3	10000	32	2524.17	1755.99
0.01	B2	10000	32	2520.45	1856.55
0.01	B1	5000	32	2519.48	1722.67
0.03	B4	10000	32	2604.52	2053.46
0.03	B3	10000	32	2520.90	1706.68

The regression results for models which data contains outliers (i.e. where the process described in 5.2.3.5 section was not applied) are presented in Table 26. As expected, the models' performance is affected by the presence of discrepant values, negatively influencing MSE and MedAE records.

Similarly to what happened in Table 24, lower learning rates combined with a batch size of 32 and number of epochs around 10000 seems to be the most performant combination with the DNN structure varying between the possible set of values.

Table 26 - Regression results containing outliers

LR	NNS	NE	BS	MSE	MedAE
0.01	B4	10000	32	10156.83	2049.90
0.01	B3	10000	32	10167.38	2187.01
0.01	B4	5000	32	10153.84	2200.30
0.03	B3	5000	32	10108.65	2244.35
0.01	B5	10000	32	10175.36	2271.72
0.03	B3	10000	32	10115.61	2364.01
0.01	B2	10000	32	10174.91	2425.38
0.03	B5	10000	32	10126.58	2453.58
0.01	B5	10000	10	10141.31	2488.15
0.01	B1	10000	32	10135.11	2525.39

6.3.1 Results analysis

Considering the 573548 records in the initial dataset, which mean cost was around 5469 US dollars, the following notes were considered:

- **Alternative without outliers** (results in Table 24) – Data was treated using all procedures mentioned in 5.2.3. 371011 samples after cleaning process with mean cost of 4267 US dollars;

- **Alternative without outliers and with less features** (results in Table 25) – Same data cleaning process as point above. Same number of samples and mean cost. Does not contain the *LIGHT*, *CSRFCND* and *WEATHER* features;
- **Alternative with outliers** (results in Table 26) – A total of 411363 samples with a mean cost of around 5885 US dollars. Data cleaning processes mentioned in 5.2.3 were applied with the exception of the outliers removal represented in the 5.2.3.5 section.

The best results were achieved in the alternative without outliers and with all features where MedAE was **1548.24** and MSE recorded **2593.48** under a neural network with medium complexity and deepness when compared with the other alternatives. In this instance, batch size was set to 32, number of epochs to 10000 and learning rate set to 0.03. The lowest MSE result was **2510.13** under a simpler neural network structure (B1 alternative).

Considering a mean cost of 4267 US dollars in the alternative without outliers, an MSE of 2593.48 is still considered a significant value, therefore not allowing an extremely precise prediction as the error dispersal is reasonable as seen in Figure 46. Considering the lowest MedAE value of 1548.24, it possible to conclude that data is also very unbalanced. Still, these values are significantly better than other approaches with a smaller set of features and with a larger set of values which were considered outliers after an analysis made in the 5.2.2.2 and 5.2.3.5 sections.

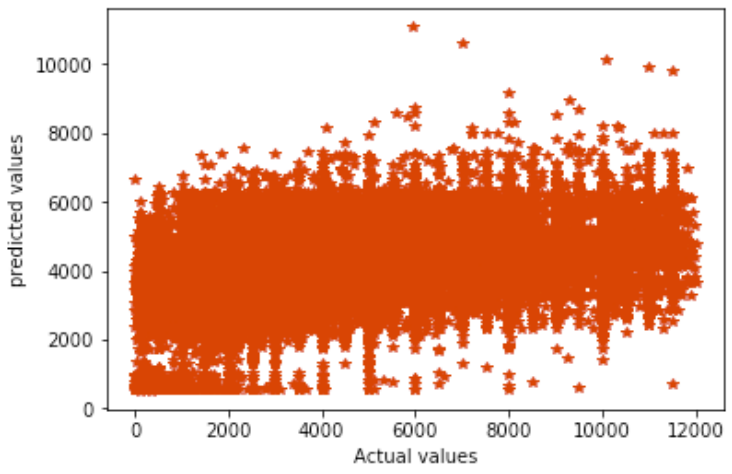


Figure 46 - Regression model predictions comparison with actual values

6.3.1.1 Other considerations

The alternatives without outliers and containing all features were the ones performing better in this context. As expected and explained in 5.2.2.2 section, the existence of outliers harmed the models' performance. Having additional features benefiting the algorithms outcome was something already stated during the classification models evaluation as mentioned in 6.2.1.1 section, in which the impact was significant.

Figure 47 demonstrates the MedAD fluctuation across the most performant regression models for both variations without outliners. Similarly to Figure 44 and Figure 45, the top instances are displayed in the X axis. It's possible to observe that there's a reasonable difference between models with the same characteristics. The presence of the *LIGHT*, *CSRFCND* and *WEATHER* features reveled to be important for the model's outcome, allowing lower MedAD values.

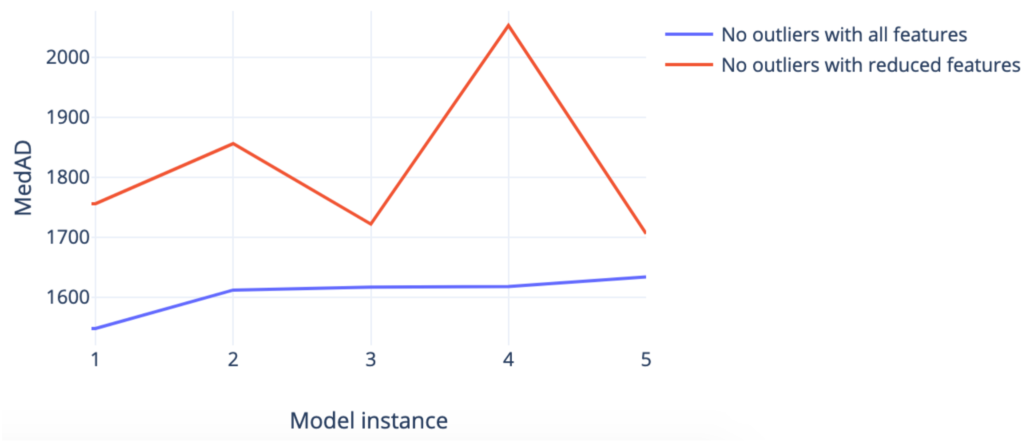


Figure 47 - MedAD fluctuation between most performant regression models' instances

In Figure 48 it's possible to observe the MedAD fluctuation across models which characteristics are similar but where different DNN structure were applied. The X axis represents the DNN structure variations' values as 1 corresponds to B1, 2 corresponds to B2, and so on. The models represented in this figure had the learning rate set to 0.01, number epochs to 10000 and batch size to 32. Even though a better MedAD tendency analysis across DNN structures requires a deeper analysis with other parameters configuration, it seems like models which data contained a smaller set of features react better to complex DNN structures as B1 and B5 are the less performant variations. For models which data contains all features the fluctuation seems less implactful.

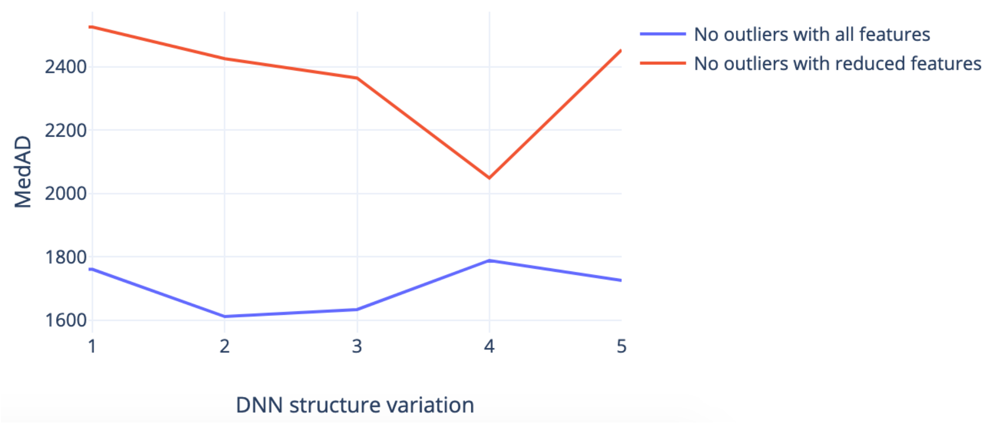


Figure 48 - MedAD fluctuation on regression models with fixed parameters and different DNN structures

The results containing outliers are considered bad results as the MSE outcomes, which best result was 10108.65, significantly surpassed the mean cost of 5885 US dollars even though MedAE achieved a top result of 2244.35.

Due to technical constraints explained in the 5.4.2 section, it's not possible to determine what optimizer from Adam and GradientDescent performs better as GradientDescent variations triggered an error related with *NaN* loss during the training process.

6.4 Comparison with other studies

Even though (Yuan et al., 2017) study uses the same dataset, the study's goal and general approach is different from this document's work. (Yuan et al., 2017) study aims to predict if in a given road, crashes will occur considering each hourly slot. Demographic features from the dataset are also used, something that was not considered in this document. With this in mind, it is not possible to do a direct comparison between both studies' results as the problem is not the same. Still, it is possible to observe that on both studies the best results were achieved using DNN, which is an expected outcome considering the problem's complexity and recent studies outcomes.

Other studies in the RTC field using regression like (Sailaja & Raju, 2015) and (Minesh Desai, n.d.) use these models to predict the number of occurrences and not their property cost. Therefore, a direct results comparison is not possible.

Comparisons with other studies in the literature were not considered as the dataset structure, data and studies' goals were significantly different.

7 Conclusion

This document investigated and analyzed the road traffic crashes problem, a major problem in today’s society, aiming to build models capable of predicting crashes’ outcomes using past occurrences data. For this purpose, Iowa state crash data from 2008 to 2016 was used. Besides the high complexity of the dataset, which contained several features that could impact the models’ outcome as road traffic crashes relate to human, vehicle and road environment factors, this problem is also complex due to imbalanced classes and data that had no previous treatment.

Most of the prior work on this field used classic data mining techniques and approaches combined with limited amount of data. Recent studies already approached the road traffic crashes analysis and prediction topic using DNN combined with mapping strategies. Imbalanced data was a constant factor on these studies.

In this document, the problem was formulated as a binary classification problem using two different approaches, separating major injuries, minor injuries and fatalities from non-injuries and separating major injuries and fatalities from minor injuries and non-injuries. To analyze and predict a crash monetary property damage cost, regression was used. The methodology consisted in using a deep learning framework, Tensorflow, which usage is uncommon in past studies, to build DNN and Random Forest models to achieve the proposed classification goals while using environmental factors from the dataset. DNN were used for the regression problem.

Having in mind this document’s context, classification results were positive, even though marginally, and show that DNN usually perform better in the RTC field problems, something also detected by past studies in this area. Regression results were not as good but already provide leeway to perform predictions. Some technical constraints harm a direct comparison with some of the Random Forest classification models’ outcomes or with GradientDescent regression models. The results could have been improved if some other features were made available in the dataset. Crash speed could highly influence the models’ outcomes as mentioned in (Moghaddam et al., 2011) study. Cost evaluation also lack a better specification of surroundings value (e.g. vehicles value, what kind of property items were harmed by the crash).

7.1.1 Future work

Considering the work done, followed approaches and the restrictions found, the following next steps are recommended:

- Apply Random Forest custom calculation of AUC, therefore allowing a better comparison with the obtained DNN results;
- Use the most recent datasets versions provided by Iowa[18], containing additional data and features, and apply them in the developed models comparing the results with the ones obtained during this document's work;
- Combine the developed models and data treatment approaches with mapping strategies;
- Explore resampling techniques due to huge unbalanced of data. As undersampling was tried in this document's work, in future studies the application of oversampling techniques like SMOTE (Synthetic Minority Over-sampling Technique) as mentioned in (Chawla, Bowyer, & Hall, 2002) may be considered a possibility;
- Deploy the developed models using Docker containers and Kubernets, following Tensorflow's official documentation[19], therefore allowing the model predictions to be served as a service and used by external software (e.g. using an API integration).

[18] Datasets new version source: https://public-iowadot.opendata.arcgis.com/datasets
[19] https://www.tensorflow.org/tfx/serving/serving_kubernetes

Bibliographic references

Ali, A., Shamsuddin, S. M., & Ralescu, A. L. (2015). Classification with class imbalance problem: A Review. *Int. J. Advance Soft Compu. Appl*, *7*(3). Retrieved from http://home.ijasca.com/data/documents/13IJASCA-070301_Pg176-204_Classification-with-class-imbalance-problem_A-Review.pdf

Auria, L., & Moro, R. A. (2008). Support Vector Machines (SVM) as a Technique for Solvency Analysis. Retrieved from www.diw.de

Bahrampour, S., Ramakrishnan, N., Schott, L., & Shah, M. (2015). Comparative Study of Deep Learning Software Frameworks. https://doi.org/10.1227/01.NEU.0000297044.82035.57

Baranauskas, J. A., & Monard, M. C. (2003). Combining symbolic classifiers from multiple inducers. *Knowledge-Based Systems*, *16*(3), 129–136. https://doi.org/10.1016/S0950-7051(02)00021-7

Barlow, H. B. (1989). Unsupervised Learning. *Neural Computation*, *1*(3), 295–311. https://doi.org/10.1162/neco.1989.1.3.295

Ben-David, S., & Shalev-Shwartz, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. *Understanding Machine Learning: From Theory to Algorithms*. https://doi.org/10.1017/CBO9781107298019

Beshah, T., & Hill, S. (2010). Mining road traffic accident data to improve safety: Role of road-related factors on accident severity in Ethiopia. *AAAI Spring Symposium - Technical Report*, *SS-10-01*(1997), 14–19. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-77957956575&partnerID=40&md5=a06a85740a7d21166dadbb737ccfd110

Boser, B., Guyon, I., & N. Vapnik, V. (1996). *A Training Algorithm for Optimal Margin Classifier*. *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory* (Vol. 5). https://doi.org/10.1145/130385.130401

Brereton, R. G., & Lloyd, G. R. (2010). Support Vector Machines for classification and regression. *The Analyst*, *135*(2), 230–267. https://doi.org/10.1039/B918972F

Brijain, M., Patel, R., Kushik, M., & Rana, K. (2014). A Survey on Decision Tree Algorithm For Classification. *International Journal of Engineering Development and Research*, *2*(1), 2321–9939. Retrieved from https://www.ijedr.org/papers/IJEDR1401001.pdf

Chawla, N. V, Bowyer, K. W., & Hall, L. O. (2002). SMOTE : Synthetic Minority Over-sampling Technique, *16*, 321–357.

Chong, M., Abraham, A., & Paprzycki, M. (2005). Traffic Accident Data Mining Using Machine Learning Paradigms. *Informatica*, *29*(1), 88–98.

Çodur, M. Y., & Tortum, A. (2015). An Artificial Neural Network Model for Highway Accident Prediction: A Case Study of Erzurum, Turkey. *PROMET - Traffic&Transportation*, *27*(3), 217–225. https://doi.org/10.7307/ptt.v27i3.1551 da Silva, I. N., Hernane Spatti, D., Andrade Flauzino, R., Liboni, L. H. B., & dos Reis Alves, S. F. (2017). Artificial Neural Networks, 21–29. https://doi.org/10.1007/978-3-319-43162-8

Dominik Wisniewski, & Wawezyniak, W. (2014). Support Vector Machines. Retrieved from http://cs.joensuu.fi/pages/whamalai/expert/svm.pdf

Dosualdo, Daniel Gomes; Rezende, S. O. (2003). *Análise da Precisão de Métodos de Regressão*. Retrieved from http://conteudo.icmc.usp.br/CMS/Arquivos/arquivos_enviados/BIBLIOTECA_113_RT_197.pdf

DZone. (n.d.). Deep Learning via Multilayer Perceptron Classifier - DZone Big Data. Retrieved February 7, 2018, from https://dzone.com/articles/deep-learning-via-multilayer-perceptron-classifier

Erik G. (2014). Introduction to Supervised Learning, 1–5. Retrieved from https://people.cs.umass.edu/~elm/Teaching/Docs/supervised2014a.pdf

Fawcett, T. (2005). An introduction to ROC analysis. https://doi.org/10.1016/j.patrec.2005.10.010

Gama, J., Ponce de Leon Carvalho, A., Carolina Lorena, A., Oliveira, M., & Faceli, K. (2017). *Extração de Conhecimento de Dados*. (E. Sílabo, Ed.) (Third). Lisboa: Edições Sílabo.

Gingrich, P. (University of R. (1992). Association Between Variables. In *Introductory statistics for the social sciences* (pp. 795–835). Retrieved from http://uregina.ca/~gingrich/corr.pdf

Goodfellow, I., Bengio, Y., & Courville, A. (2015). Deep Learning. *Nature Methods*, *13*(1), 35–35. https://doi.org/10.1038/nmeth.3707

Hagan, M. T., Demuth, H. B., & Beale, M. H. (1995). *Neural Network Design*. *Boston Massachusetts PWS* (Second Edi, Vol. 2). https://doi.org/10.1007/1-84628-303-5

Han, J., Kamber, M., & Pei, J. (2012). 3 - Data Preprocessing. In *The Morgan Kaufmann Series in Data Management Systems* (pp. 83–124). https://doi.org/http://dx.doi.org/10.1016/B978-0-12-381479-1.00003-4

Haykin, S. (1999). *Neural networks - A comprehensive foundation* (Second Edi). Person Education. Retrieved from https://cdn.preterhuman.net/texts/science_and_technology/artificial_intelligence/Neural Networks - A Comprehensive Foundation - Simon Haykin.pdf

Hossin, M., & Sulaiman, M. N. (2015). a Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process (IJDKP)*, *5*(2), 1–11. https://doi.org/10.5121/ijdkp.2015.5201

IBM Knowledge Center - CHAID classification tree. (n.d.). Retrieved February 23, 2018, from https://www.ibm.com/support/knowledgecenter/en/SS4QC9/com.ibm.solutions.wa_an_overview.2.0.0.doc/chaid_classification_tree.html

Iowa Zero Fatalities | Homepage. (n.d.). Retrieved January 20, 2018, from http://ia.zerofatalities.com/

Japkowicz, N., & Shah, M. (2011). *Evaluating Learning Algorithms: A Classification Perspective [Hardcover]*. University of Cambridge. Retrieved from http://www.amazon.ca/Evaluating-Learning-Algorithms-Classification-Perspective/dp/0521196000

Jost, G., Popolizio, M., Allsop, R., & Eksler, V. (2008). 2nd PIN Annual Report Countdown to 2010 – Only two more years to act. *European Transport Safaety Council (ETSC)*, 83. Retrieved from www.etsc.be

Koen, P., Ajamian, G., Burkart, R., Clamen, A., Davidson, J., D'Amore, R., … Wagner, K. (2001). Providing Clarity and a Common Language To the "Fuzzy Front End." *Research Technology Management*, *44*(2), 46–55. https://doi.org/Article

Kononov, J., & Janson, B. (2002). Diagnostic Methodology for the Detection of Safety Problems at Intersections. *Transportation Research Record*, *1784*(1), 51–56. https://doi.org/10.3141/1784-07

Kotsiantis, S. B., & Kanellopoulos, D. (2006). Data preprocessing for supervised leaning. *International Journal of …*, *1*(2), 1–7. https://doi.org/10.1080/02331931003692557

Kovalev, V., Kalinovsky, A., & Kovalev, S. (2016). Deep Learning with Theano , Torch , Caffe , TensorFlow , and Deeplearning4J : Which One Is the Best in Speed and Accuracy ?, (October), 99–103. Retrieved from http://imlab.grid.by/

Kumar, S., & Toshniwal, D. (2015). A data mining framework to analyze road accident data. *Journal of Big Data*, *2*(1), 26. https://doi.org/10.1186/s40537-015-0035-y

Leon, M. R. M. D. E., Cal, C. P., & Sigua, R. G. (2005). Estimation of Socio-Economic Cost of Road Accidents in Metro Manila. *Journal of the Eastern Asian Society for Transportation Studies*, *6*, 3183–3198. Retrieved from https://www.jstage.jst.go.jp/article/easts/6/0/6_0_3183/_pdf/-char/en

Li, F.-F., Johnson, J., & Yeung, S. (2017). Lecture 8 : Deep Learning Software, 1–159. Retrieved from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture8.pdf

Minesh Desai, M. (n.d.). Road Accidents Study Based On Regression Model: A Case Study of Ahmedabad City. Retrieved from http://bvmengineering.ac.in/misc/docs/published-20papers/civilstruct/Civil/101027.pdf

Moghaddam, F. R., Afandizadeh, S., & Ziyadi, M. (2011). Prediction of accident severity using artificial neural networks. *International Journal of Civil Engineering*, *9*(1), 41–49. Retrieved from http://ijce.iust.ac.ir/article-1-544-en.pdf

Moradkhani, F., Ebrahimkhani, S., & Sadeghi Begham, B. (2014). Road Accident Data Analysis: A Data Mining Approach. *Indian Journal of Scientific Research*, *3*(3), 437–443. Retrieved from https://www.researchgate.net/publication/262186265

Mu, E., & Pereyra-Rojas, M. (2017). Practical Decision Making, (2012). https://doi.org/10.1007/978-3-319-33861-3

Neural activation functions. (n.d.). Retrieved February 4, 2018, from https://staff.fnwi.uva.nl/l.dorst/math/sigma.pdf

Nicola, S., Ferreira, E. P., & Ferreira, J. J. P. (2012). A NOVEL FRAMEWORK FOR MODELING VALUE FOR THE CUSTOMER, AN ESSAY ON NEGOTIATION. *International Journal of Information Technology & Decision Making*, *11*(03), 661–703. https://doi.org/10.1142/S0219622012500162

Nurkkala, V., Kalermo, J., & Jarvilehto, T. (2014). Implementation of Kalman Filter on PSoC-5 Microcontroller Microcontroller for Mobile Robot Localization Garth, *11*, 403–411. https://doi.org/10.17265/1548-7709/2014.05

Pittaras, N., Markatopoulou, F., Mezaris, V., & Patras, I. (2017). Comparison of fine-tuning and extension strategies for deep convolutional neural networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 10132 LNCS, pp. 102–114). https://doi.org/10.1007/978-3-319-51811-4_9

Presidency, L., & Union, E. (2015). Road safety priorities for the EU in, (December 2014). Retrieved from http://etsc.eu/wp-content/uploads/2015_lux_pres_briefing_final.pdf

Rakha, H., Arafeh, M., Abdel-Salam, A. G., Guo, F., & Flintsch, A. M. (2010). LINEAR REGRESSION CRASH PREDICTION MODELS: ISSUES AND PROPOSED SOLUTIONS Virginia Transportation Research Council. Retrieved from https://vtechworks.lib.vt.edu/bitstream/handle/10919/55103/VT-2008-02.pdf;sequence=1

Raschka, S. (2018). *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*.

Refaeilzadeh, P., Tang, L., & Liu, H. (2006). Cross-Validation. Retrieved from http://leitang.net/papers/ency-cross-validation.pdf

Ruder, S. (2016). An overview of gradient descent optimization algorithms. Retrieved from http://arxiv.org/abs/1609.04747

Sailaja, V., & Raju, S. S. (2015). Accident Analysis on NH-18 by Using Regression Model and its Preventive Measures. *International Journal of Science and Research (IJSR) ISSN (Online Index Copernicus Value Impact Factor*, *4*(4), 2467–2470. Retrieved from https://www.ijsr.net/archive/v4i4/SUB153631.pdf

Santos, M. C. (2015). *Explorando técnicas para modelagem de dados agregados de óbitos provenientes de acidentes por automóvel*. Escola de Engenharia de São Carlos Departamento.

Shanthi, S., & Ramani, R. G. (2011). Classification of Vehicle Collision Patterns in Road Accidents using Data Mining Algorithms. *International Journal of Computer Applications*, *35*(12), 30–37. Retrieved from https://pdfs.semanticscholar.org/fc7b/9cc0f99e7b3a38857f91dcbb0a6e5a90eec0.pdf

Shinar, D. (2017). *Traffic Safety and Human Behavior Second Edition.* (Emerald Group Publishing, Ed.) (Second). Emerald Group Publishing. Retrieved from https://books.google.pt/books?id=1iUpDwAAQBAJ

Singer, Y. (2016). *Advanced Optimization*. Retrieved from https://people.seas.harvard.edu/~yaron/AM221-

S16/lecture_notes/AM221_lecture9.pdf

Smola, A., & Vishwanathan, S. V. . (2010). *Introduction to Machine Learning*. 2008: Press syndicate of the university of cambridge. Retrieved from http://alex.smola.org/drafts/thebook.pdf

Storkey, A. J. (n.d.). Machine Learning and Pattern Recognition Logistic Regression. *University of Edinburgh: Course Lecturer*. Retrieved from http://www.inf.ed.ac.uk/teaching/courses/mlpr/lectures/mlpr-logreg.pdf

The United Nations and Road Safety. (2011). Retrieved January 23, 2018, from http://www.un.org/en/roadsafety/

TIOBE. (2018). TIOBE the software quality company. Retrieved February 17, 2018, from https://www.tiobe.com/tiobe-index/

Tomar, A., Nagpal, A., & Abdul, A. P. J. (2016). Comparing Accuracy of K-Nearest-Neighbor and Support-Vector-Machines for Age Estimation. *International Journal of Engineering Trends and Technology*, *38*(6). Retrieved from http://www.ijettjournal.org

Vasavi, S. (2018). Extracting Hidden Patterns Within Road Accident Data Using Machine Learning Techniques.

Walker, O. C., Lutz, R., Park, C. W., & Schmalensee, D. (1988). Consumer Perceptions of Price, Quality, and Value: A Means-End Model and Synthesis of Evidence. *Journal of Marketing*, *52*, 2–22. Retrieved from https://hec.unil.ch/docs/files/123/997/zeithaml88-1.pdf

WHO. (2014). Why are road traffic injuries a public health issue? Retrieved from http://www.who.int/violence_injury_prevention/publications/road_traffic/Road_safety_media_brief_full_document.pdf

WHO | Decade of Action for Road Safety 2011-2020. (2017). Retrieved January 20, 2018, from http://www.who.int/roadsafety/decade_of_action/en/

World Health Organization. (2015a). *GLOBAL STATUS REPORT ON ROAD SAFETY, 2015, Executive Summary*. Retrieved from http://www.who.int/violence_injury_prevention/road_safety_status/2015/Executive_summary_GSRRS2015.pdf

World Health Organization. (2015b). GLOBAL STATUS REPORT ON ROAD SAFETY 2015 SUMMARY, 14. Retrieved from www.who.int/about/licensing/copyright_form/en/index.html

World Production | ACEA - European Automobile Manufacturers' Association. (2016). Retrieved January 21, 2018, from http://www.acea.be/statistics/tag/category/world-production

Wundersitz, L. N., & Woolley, J. E. (2008). *Best practice review of drink driving enforcement in South Australia*.

Yuan, Z., Zhou, X., Yang, T., Tamerius, J., & Mantilla, R. (2017). Predicting Traffic Accidents Through Heterogeneous Urban Data: A Case Study. Retrieved from https://doi.org/10.475/123_4

Zero Fatalities. (n.d.). Retrieved January 20, 2018, from http://www.kcci.com/article/zero-fatalities-1/6895497

Zhong-Xiang, F., Shi-Sheng, L., Wei-Hua, Z., & Nan-Nan, Z. (2014). Combined prediction model of death toll for road traffic accidents based on independent and dependent variables. *Computational Intelligence and Neuroscience*, *2014*, 1–7. https://doi.org/10.1155/2014/103196