

Instituto Tecnológico y de Estudios Superiores de Occidente

Department of Electronics, Systems, and Informatics

Computing Systems Engineering

Machine Learning Course

Teacher: Edna L. Guevara Rivera

Bird strike fatality prediction on Airplane crashes

PROJECT

Presented by:

Marco Ricardo Cordero Hernández, 727272

Carlos Eduardo Rodríguez Castro, 727366

December 1st, 2022

AUTUMN, 2022

Tlaquepaque, México

# Index

[Introduction 1](#_Toc120772874)

[Problem to solve 2](#_Toc120772875)

[Data collection 3](#_Toc120772876)

[Learning type to use 4](#_Toc120772877)

[Cleaning process introduction 5](#_Toc120772878)

[Cleaning process walkthrough 6](#_Toc120772879)

[Model implementation introduction 12](#_Toc120772880)

[Model selection and motivation(s) 13](#_Toc120772881)

[Implementation 14](#_Toc120772882)

[Linear Regression 15](#_Toc120772883)

[Neural Network 19](#_Toc120772884)

[Decision tree 26](#_Toc120772885)

[Results description 29](#_Toc120772886)

[Performance comparative 31](#_Toc120772887)

[Conclusions and pending work 32](#_Toc120772888)

[References 33](#_Toc120772889)

# Introduction

It is often said that one is more likely to die in a car crash than in an airplane accident. This isn’t an exaggeration, even being backed up by the United State National Safety Council [1]. This fact it’s usually accompanied by the contrast of the highly likeliness of having an automobile accident in the way to an airport rather than in the plane itself. These aspects are backed up by the common knowledge of what is required in order to get a pilot’s license versus the minimum aspects needed for a driver’s license. Setting aside the economic resource needed for wings to fly, not anyone can become an airplane pilot, not even a private one, and those who do, they need to be in constant training [2]. For this sole reason, the probability of being in an aerial incident, fatal or not, it’s very low. But, what about when there isindeed an accident? It can’t be denied that human factor plays a big role in the final outcome of an aerial sinister, whether it’s from land by air traffic controllers or by pilots stunned by unusual conditions [3]. Although it might seem contradictory to the first lines of this paragraph, the reality is that even by staying extremely calm, the most prepared and experienced cabin crew can’t deal with a motor failure or complete loss in its entirety. This can be aided by analytics.

Machine learning (from now on referred as ML), as revised by Brown [4], may be seen as “the capability of a machine to imitate *intelligent human behavior*”. Given this short but meaningful definition, the problem that this project will try to tackle can be seen as this: humans cannot think fast enough in a matter of life and death, whereas computers could certainly do.

By giving a proof by counterexample, Gupta [5] details two scenarios in which ML should be avoided thoughtfully: fairly ease or complexity lacking problems, and lack of labeled data. To put in someone’s hands the life of several people it’s not something to be taken lightly. The beautiful field of applied math conjoined with computer science usage could potentially save hundreds if not thousands of lives; just by applying simple algebra concepts such as matrices and dot products [6] great things can be achieved, solutions can be made and existing methods of avoiding fatalities can be drastically improved, in this case, through the application of ML. Although this it’s just the introductory part of this work, it can be assured that poorly classified data or niche information won’t be a problem in the becoming development.

Furthermore, and getting into a deeper level of detail, it’s almost immediately recognized that the problem found can be addressed by applying supervised learning algorithms; as defined by Richards & Jia [7], these classifying algorithms make quantitative analysis over a dataset to decide whether an entry or set of entries correspond to some type of classification. This type of classification it’s called like so because in order for it to work, desired outputs have to be given.

With the previous being said, it is not without reminding that ML it’s just as strongest as its weakest link. ML it’s a powerful tool, but it won’t do miracles. In any case, the following sections will explore specific portions of the whole project.

# Problem to solve

Ever since it happened, the US Airways flight 1549, or the “Miracle on the Hudson” as its often referred to, has become the flagship of aircraft incidents that turned out well in terms of fatalities. [8] [9] On January 15, 2009, said flight suffered a *bird strike* which led to a successful water landing, in which only injured passengers were reported, this meaning that no deaths were suffered on the incident. This is extremely rare, as the odds of surviving a plane crash versus those of an aquatic emergency landing are completely different [10]. At the moment of the incident, Chesley Sullenberger, the pilot that made the maneuvers for the successful landing, had over 40 years of experience or *training*, key factor in the fortunate outcome of the situation. With this in mind, does it really take a flight veteran to make or predict a favorable result in terms of lives lost?

Perhaps it might seem harmless at first glance, but when organic material such as birds’ corpses get stuck into complex and carefully engineered machinery such as airplane turbines or helicopter rotors, disastrous events take place. The broken components of these aircrafts can be easily diagnosed with modern on-board systems, a detail of vastly interest, because with this piece of information, severity can be predicted ipso facto.

As a form of summarization, this project seeks to predict the fatality of a bird crash incident over type of aircraft, having such outcomes as *fatal* (0) and *non-fatal* (1).

# Data collection

A dataset containing 25558 registers and 26 features has been retrieved from a data science platform [11].

The description of said set states that the values contained within the dataset comes directly from the Federal Aviation Administration (FAA), who provided the number and details of incidents where birds have struck a plane over a period of ten years, this being from 2000 to 2011 (two years after the Hudson incident).

With aid from pandas (a popular python data analysis library [12]), a quick analysis was made in order to determine the absence of values, which, in this case, was indeed found.

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

Fig. 1 Overview of dataset and analysis of values

The previous results show that the features number of engines, origin state and remarks not only have null values, but also that they all are objects, most likely strings.

# Learning type to use

A supervised categorical algorithm has been chosen for this type of problem because there needs to be determined if a flight accident will or will not be fatal and our output will always be between "Fatal" and "Not fatal", ideally. In this case, the type of algorithm that’ll be developed is categorical because it needs to classify all of the results under one of these two categories that are set.

One of the greatest advantages that this type of algorithm will bring to the project is that the result will be easily readable and no further processing is needed to extract real value from the output. Despite this, the algorithm has one disadvantage, debugging a categorical algorithm can be harder since there cannot be explicitly seen that an issue exists. The issue can only be detected when tests are made from the predicted results; since there is no complete control over specific operations the algorithm is doing, the debugging process can be quite time consuming.

When doing the comparison between the two main algorithm contenders (regression and categorical), discovers were made: even though regression can be considered a more precise algorithm, it lacks the output simplicity that the categorical algorithm is known for. All of the research points to use the categorical algorithm to predict whether a flight accident is fatal or not, the pros outweigh the cons for this specific application.

# Cleaning process introduction

Through the last segments, the first stone has been set for the incoming steps that would encapsulate the knowledge gathered along the course for which this text has been written.

The most prevalent piece of work that needs to be made it’s the transformation of the dataset as demonstrated in previous sections. Text or string fields were present, and, although this could be seem as problematic, the reality is that this information needs to undergo over a transformation and cleaning process in which these categorical data would be transformed into numerical values.

This process may vary depending on dataset structure, having multiple types of data crammed into several columns (referred as *features* from now on). For this particular project, additional measures had to be taken in order to transform string to numerical data.

The steps to clean the project’s dataset are described below. Just as a reminder, the dataset comes from an external source [11].

# Cleaning process walkthrough

First, libraries have to be imported in order to use their methods for data loading, manipulation, and visualization.

*# Libraries*

*import* pandas *as* pd

*import* numpy *as* np

*import* seaborn *as* sns

*import* matplotlib.pyplot *as* plt

Next, the file containing the data itself has to be loaded. The file name it’s *bird\_strikes.csv*.

*# Read dataset*

ds\_bst = pd.**read\_csv**('bird\_strikes.csv')

After the previous step, the dataset can be visualized just by invoking the store valuable.

Relevant information related to data types for each feature has to be displayed to determine which columns could be kept.

*# Dataset info*

ds\_bst.**info**()

Texto

Descripción generada automáticamente

Fig. 2 Dataset original features

Although output has been trimmed by the method containing library, critical information it’s displayed at the bottom, indicating that 16 object type features (most likely strings) are present. Also, several boolean features are contained within other features, and although they could work in their original state, it’s better to transform them into pure dichotomic values.

Before transforming present values, presence of null fields has to be taken into consideration.

*# Null values identification*

ds\_bst.**isna**().**sum**(*axis* = 0)

Texto

Descripción generada automáticamente

Fig. 3 Null presence in features

Through preliminary analysis, only one of the three null value containing features will have to be transformed into full data feature, this being *aircraft\_number\_of\_engines*, as the other ones will be suppressed later on.

*# Null values replacement*

ds\_bst['aircraft\_number\_of\_engines'].**fillna**(*value* = int(ds\_bst['aircraft\_number\_of\_engines'].**mean**()), *inplace* = True)

**print**(f"Null qty remaining: {ds\_bst.**isna**().**sum**(*axis* = 0)['aircraft\_number\_of\_engines']}")



Fig. 4 Null absence verification

Just as the previous step it’s completed, dropping the irrelevant features will take place. These droppable features are selected by looking at the information their data holds. Remaining features are shown below the step code.

*# Non-relevant columns dropping*

nrc = [

    'record\_id',

    'airport\_name',

    'wildlife\_number\_struck',

    'flightdate',

    'aircraft\_airline\_operator',

    'origin\_state',

    'remains\_of\_wildlife\_sent\_to\_smithsonian',

    'remarks',

    'wildlife\_species',

    'cost\_total'

]

'''

    Although 'cost\_total' could be used, the information related to that feature

    it's only obtained after the accident has occurred

'''

ds\_bst.**drop**(nrc, *inplace* = True, *axis* = 1)

ds\_bst.**info**()

Texto

Descripción generada automáticamente

Fig. 5 Remaining features after deletion

Now, just as stated before, object/string and bool features have to be transformed to numerical values. Two functions were made, one to transform categorical values, and another to turn boolean values to their binary representation.

def **categorize**(*dataset*, *feature*):

    holder = {}

    index = 0

*for* row *in* *dataset*[*feature*]:

*if* (row *not* *in* holder):

            holder[row] = index

            index += 1

*for* val *in* holder:

*dataset*[*feature*] = *dataset*[*feature*].replace([f'{val}'], holder[val])

def **to\_binary**(*dataset*, *feature*):

*dataset*[*feature*] = *dataset*[*feature*].apply(lambda *x* : 1 *if* *x* *else* 0)

With the aid of [Fig. 4](#Fig4), indexes of each feature and their corresponding transformation can be done easily.

features = ds\_bst.columns.values

to\_modify = (0, 1, 2, 4, 5, 7, 8, 10, 11)

to\_bin = (9, 12, 15)

*# Implementation not recommended for long features lenght (<50)*

*for* i *in* range(16):

*if* (i *in* to\_modify):

**categorize**(ds\_bst, features[i])

*elif* (i *in* to\_bin):

**to\_binary**(ds\_bst, features[i])

ds\_bst.**info**()

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Fig. 6 Transformed remaining features

Heading towards end of cleaning process, a visualization approach has to be taken in order to detect repeated values.

ds\_bst.**hist**(*bins* = 30, *figsize* = (20, 20), *color* = 'r')

Gráfico

Descripción generada automáticamenteGráfico

Descripción generada automáticamente

Fig. 7 Histograms

This extra analysis was helpful, because it ultimately helped in the exclusion of another two useless features.

ds\_bst.**drop**(['aircraft\_type', 'number\_of\_people\_injured'], *inplace* = True, *axis* = 1)

With that last step, the process of dataset cleaning has concluded. Additional analysis and clean dataset file generation can be found inside the jupyter notebook file.

# Model implementation introduction

Considering the previous steps, the selected dataset has been cleaned through multiple programming and analysis techniques in order for it to be ready for machine learning algorithms, this with the purpose of training an effective and efficient set of models, from which one will be determined to be the best amongst them, at least for this problem.

As it has been used in past steps, Jupyter [13] will act as the container/holder for the computational operations results and outcomes from the algorithms.

# Model selection and motivation(s)

For the analysis and comparison between results, and for class material comprehension purposes, the following Machine Learning Models will be implemented:

1. *Linear Regression (Normal)*

As it is the most common type of technique and usually one of the first concepts used to teach about ML, this widely used model will function as the main comparison and example for further upgrading in next model implementations.

Although the concepts involved in LR are fairly basic, these tools are still very useful and serve as a comparison entry point.

1. *Neural Network*

Another broadly known technique when discussing about Machine/Deep Learning. This model has gained plenty of attention over recent years, as it’s being used among a great range of modern-day problems, such as facial recognition, stock market predictions, signature verification, etc. [14] Thus, making it a great opportunity for a demonstration of this model for yet another contemporary problem.

1. Decision Tree

Lastly, this model will be visited as an alternative to classic statistic methods, as DT’s support nonlinear data and makes for a great visual resource that involves several categories/features found in the dataset of analysis. This highly customizable model allows for fine-grain knobbing/adjusting for better result outcome and can be easily compared against other models.

# Implementation

Full model implementation can be found inside the jupyter notebook created specifically for this part of the project, giving an extensive explanation of its steps and result retrieving.

This file its embedded in the next figure (only accessible by Word; request original file at is727272@iteso.mx).

First, and as per usual, libraries have to be loaded.

*# Libraries*

*import* math

*import* numpy *as* np

*import* pandas *as* pd

*import* seaborn *as* sns

*from* sklearn *import* tree

*from* sklearn *import* metrics

*import* matplotlib.pyplot *as* plt

*from* sklearn.metrics *import* accuracy\_score

*from* sklearn.metrics *import* confusion\_matrix

*from* sklearn.tree *import* DecisionTreeClassifier

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.metrics *import* classification\_report, confusion\_matrix

Then the cleaned dataset itself.

*# Read dataset*

ds\_bst = pd.**read\_csv**('bird\_strikes\_clean.csv')

Pantalla de computadora

Descripción generada automáticamente con confianza baja

Fig. 7 Trimmed dataset features

For ease of manipulation, feature swapping it’s made by the following lines.

ds\_bst['effect\_indicated\_damage'], ds\_bst['is\_aircraft\_large'] = ds\_bst['is\_aircraft\_large'], ds\_bst['effect\_indicated\_damage']

ds\_bst.**rename**({'effect\_indicated\_damage': 'is\_aircraft\_large', 'is\_aircraft\_large': 'effect\_indicated\_damage'}, *axis*=1, *inplace*=True)

## Linear Regression

**Steps:**

1. Convert dataset to numpy array

2. Add the columns of number 1

3. Split the dataset into Training and Testing sets

4. Using the xTrain and yTrain (Training dataset) and Linear Regression function from sklearn library, obtain the model (W's). Then make predictions using the Testing dataset, and obtain the R2 score for predictions.

5. Using Ridge function from sklearn library, obtain the model (W's) and then make predictions using the Testing dataset, and obtain the R2 score for predictions.

6. Increment alpha value in logarithmic form: 10, 100, 1000, 10000, 100000, 1e6, 1e7, then graph ridge score behaviour for each alpha value

*# Variable definition*

ds\_bst\_np\_lr = np.**array**(ds\_bst)

x = ds\_bst\_np\_lr[:, :-1]

y = ds\_bst\_np\_lr[:, -1]

y = y.reshape(-1, 1)

*# Add the columns of 1's*

def **addones**(*X*):

    X1 = np.**array**(*X*)

    m, n = np.**shape**(X1)

    ones = np.**ones**((m, 1))

    X1 = np.**concatenate**((ones, X1), *axis* = 1)

*return* X1

x = **addones**(x)

*# Split the dataset into Training and Testing sets, test size of 33%, and random\_state= 1*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, *test\_size* = 0.33, *random\_state* = 1)

**print**('Shape of Training data: ', np.**shape**(X\_train), np.**shape**(y\_train))

**print**('Shape of Testing data: ', np.**shape**(X\_test), np.**shape**(y\_test))



Fig 8. Training and testing sub-datasets shapes

*# Using the xTrain and yTrain (Training dataset) and Linear Regression function from sklearn library, obtain the model (W's).*

*# Then make predictions using the Testing dataset, and obtain the  𝑅2  score of your predictions.*

*from* sklearn.linear\_model *import* LinearRegression

*# Fit the data to training dataset*

reg = LinearRegression().fit(X\_train, y\_train)

*# Obtain and print the score*

cost = reg.score(X\_test, y\_test)

**print**(f'Error (R2): {cost}')

*# Obtain and print the W's coefficients*

w = reg.coef\_

**print**(f'W: {w}')

*# Obtain and print the intercept*

intercept = reg.intercept\_

**print**(f'W0: {intercept}')

*# Add the intercept value to the W's array and print W*

w[0][0] = intercept

**print**(w)

Texto

Descripción generada automáticamente

Fig. 9 Cost and weights

def **r2**(*Y*, *Yt*):

    error = *Y* - *Yt*

    variance = (*Y* - np.**average**(*Y*)) \*\* 2

    cost = 1 - (np.**sum**(error \*\* 2)) / np.**sum**(variance)

*return* cost

*# Predictions for Testing dataset*

yt = np.**dot**(w, X\_test.T).T

**print**(np.**shape**(yt))

*# Obtain and print the R2 score*

cost = **r2**(y\_test, yt)

**print**(cost)

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Fig. 10 R2 Output

*# Linear regression con regularizacion "Ridge"*

*# Using Ridge function from sklearn library, obtain the model (W's) and then make predictions using the Testing dataset*

*# and obtain the  𝑅2  score of your predictions.*

*from* sklearn.linear\_model *import* Ridge

*# Define the clf method using alpha = 10*

clf = Ridge(*alpha* = 10.0)

*# Fit to the training dataset*

ridge = clf.fit(X\_train, y\_train)

*# Obtain and print the score*

Score2 = ridge.score(X\_test, y\_test)

**print**(f'R2: {Score2}')

*# Obtain and print the W's coefficients*

w2 = ridge.coef\_

**print**(w2)

*# Obtain and print the intercept*

intercept2 = ridge.intercept\_

**print**(intercept2)

*# Add the intercept value to the W's array and print W*

w2[0][0] = intercept2

**print**(w2)

Texto

Descripción generada automáticamente

Fig. 11 Costs and weights for ridge implementation

*# Increment alpha value in logarithmic form: 10, 100, 1000, 10000, 100000, 1e6, 1e7*

*# then graph ridge score behaviour for each alpha value*

alphas = [10, 100, 1000, 10000, 100000, 1e6, 1e7, 1e8]

J = []

*for* a *in* alphas:

*# Define the clf method using distinct alphas*

    clf = Ridge(*alpha* = a)

*# Fit to the training dataset*

    ridge = clf.fit(X\_train, y\_train)

*# Obtain and print the score*

    Score = ridge.score(X\_test, y\_test)

*# Obtain and print the W's coefficients*

    w = ridge.coef\_

*# Obtain and print the intercept*

    intercepto = ridge.intercept\_

*# Add the intercept value to the W's array and print W*

    w[0][0] = intercepto

*# Predictions for Testing dataset for Ridge algorithm*

    yt = np.**dot**(w, X\_test.T).T

*# Obtain and print the R2 score for Ridge Algorithm*

    cost = **r2**(y\_test, yt)

    J.**append**(cost)

plt.**plot**(alphas, J, 'b')

The generated graph will be displayed in results description section.

## Neural Network

**Steps:**

1. Data loading

2. Plot the data

3. W function initialization, Sigmoid, Cost and Forward

4. Prediction, Accuracy and Decission Boundary

5. Model definition

6. Results visualization

*# Plot the training dataset*

f, ax = plt.**subplots**()

ax.**plot**(X\_train, y\_train)

plt.**xlabel**('X\_test')

plt.**ylabel**('y\_test')

plt.**show**()

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

Fig. 12 Dataset plotting

*# Initialize W's and b's*

def **init\_w**(*m*, *nh*, *ny*):

    np.random.**seed**(2)

*# w´s willbe created randomly*

*# b's will be zeros*

    W1 = np.random.**randn**(*nh*, *m*) \* 0.01

    b1 = np.**zeros**((1,*nh*))

    W2 = np.random.**randn**(*nh*, *nh*) \* 0.01

    b2 = np.**zeros**((1,*nh*))

    W3 = np.random.**randn**(*ny*, *nh*) \* 0.01

    b3= np.**zeros**((*ny*,1))

    W = {"W1": W1, "b1": b1, "W2": W2, "b2": b2, "W3": W3, "b3": b3}

*return* W

*# Testing the function*

m = x.shape[1] *# features on x*

nh = 2 *# hidden neurons*

ny = 1 *# outputs units*

W = **init\_w**(m, nh, ny)

**print**(W['W1'].shape, 'W1:\n', W['W1'])

**print**(W['b1'].shape, 'b1:\n', W['b1'])

**print**(W['W2'].shape, 'W2:\n', W['W2'])

**print**(W['b2'].shape, 'b2:\n', W['b2'])

**print**(W['W3'].shape, 'W3:\n', W['W3'])

**print**(W['b3'].shape, 'b3:\n', W['b3'])

Texto

Descripción generada automáticamente

Fig. 12 Measures generated for the network

*# Sigmoid function*

def **sigmoid**(*z*):

    g = 1/(1+ np.exp(-*z*))

*return* g

*# Forward propagation to calculate ouput probabilites*

def **forward**(*x*, *W*):

    W1 = *W*['W1']

    b1 = *W*['b1']

    W2 = *W*['W2']

    b2 = *W*['b2']

    W3 = *W*['W3']

    b3 = *W*['b3']

    a1 = *x*

    Z2 = np.**dot**(a1, W1.T) + b1

    a2 = **sigmoid**(Z2)

    Z3 = np.**dot**(a2, W2.T) + b2

    a3 = **sigmoid**(Z3)

    Z4 = np.**dot**(a3, W3.T) + b3

    a4 = **sigmoid**(Z4)

    Z = {'Z2': Z2, 'a2': a2, 'Z3': Z3, 'a3': a3, 'Z4': Z4, 'a4': a4}

*return* a4, Z

*# Cost function*

def **cost**(*a*, *y*):

    J = 1/2 \* np.**sum**((*a* - *y*)\*\*2)

*#J = np.sum((a - y)\*\*2)*

*return* J

*# Derivative of sigmoid function*

def **d\_sigmoid**(*z*):

    ds = **sigmoid**(*z*) \* (1 - **sigmoid**(*z*))

*return* ds

*# Backpropagation algorithm*

def **backp**(*W*, *Z*, *X*, *y*):

    m = *X*.shape[1]

    W1 = *W*['W1']

    W2 = *W*['W2']

    W3 = *W*['W3']

    a2 = *Z*['a2']

    a3 = *Z*['a3']

    a4 = *Z*['a4']

    Z2 = *Z*['Z2']

    Z3 = *Z*['Z3']

    Z4 = *Z*['Z4']

    d4 = a4 - *y*

    d3 = np.**dot**(d4, W3) \* **d\_sigmoid**(Z3)

    d2 = np.**dot**(d3, W2) \* **d\_sigmoid**(Z2)

    dW1 = (1/m) \* np.**dot**(d2.T, *X*)

    dW2 = (1/m) \* np.**dot**(d3.T, a2)

    dW3 = (1/m) \* np.**dot**(d4.T, a3)

    db1 = (1/m) \* np.**sum**(d2, *axis* = 0)

    db2 = (1/m) \* np.**sum**(d3, *axis* = 0)

    db3 = (1/m) \* np.**sum**(d4)

    grad = {'dW1': dW1, 'dW2': dW2, 'dW3': dW3, 'db1': db1, 'db2': db2, 'db3': db3}

*return* grad

*# Implement and execute the NN model*

def **bird\_strikes\_model**(*x*, *y*, *nh*, *alpha* = 0.001, *epochs* = 10000):

    np.random.**seed**(2)

    m = *x*.shape[1]

    ny = 1

    W = **init\_w**(m, *nh*, ny)

    a4, z = **forward**(*x*, W)

**print**('Initial cost:', **cost**(a4, *y*))

    J = []

*for* i *in* range(*epochs*):

        a4, Z = **forward**(*x*, W)

        J.**append**(**cost**(a4, *y*))

        grad = **backp**(W, Z, *x*, *y*)

        W['W1'] = W['W1'] - *alpha* \* grad['dW1']

        W['W2'] = W['W2'] - *alpha* \* grad['dW2']

        W['W3'] = W['W3'] - *alpha* \* grad['dW3']

        W['b1'] = W['b1'] - *alpha* \* grad['db1']

        W['b2'] = W['b2'] - *alpha* \* grad['db2']

        W['b3'] = W['b3'] - *alpha* \* grad['db3']

**print**('Final cost:', J[*epochs*-1])

*return* W, J

W, J = **bird\_strikes\_model**(X\_train, y\_train, nh, *alpha*= 0.0001, *epochs*=1000)

**print**('W1 =', W['W1'])

**print**("b1 = ", W['b1'])

**print**("W2 = ", W['W2'])

**print**("b2 = ", W['b2'])

**print**("W3 = ", W['W3'])

**print**("b3 = ", W['b3'])

plt.**plot**(J)

plt.**title**('Cost over epochs')

plt.**xlabel**('epochs')

plt.**ylabel**('cost');

Texto

Descripción generada automáticamente

Fig. 13 Costs and measures for model implementation

The generated graph will be displayed in results description section.

*# Implement prediction, accuracy, and decision boundary functions*

def **predict**(*x*, *W*):

    a4, Z = **forward**(*x*, *W*)

    y\_hat = list(map(lambda *x*: 1 *if* *x* > 0.5 *else* 0, a4))

    y\_hat = np.**array**(y\_hat)

    y\_hat = y\_hat.reshape(-1, 1)

*return* y\_hat

def **accuracy**(*y\_hat*, *y*):

    m = **len**(*y*)

    tptn = (*y* == *y\_hat*).sum()

    acc = tptn / m

*return* acc

def **decision\_boundary**(*x*, *y*, *w*, *ax*):

    x\_min, x\_max = *x*[:, 0].min() - 0.5, *x*[:, 0].max() + 0.5

    y\_min, y\_max = *x*[:, 1].min() - 0.5, *x*[:, 1].max() + 0.5

    h = 0.01

    xx, yy = np.**meshgrid**(np.**arange**(x\_min, x\_max, h), np.**arange**(y\_min, y\_max, h))

    Z1 = **predict**(np.c\_[xx.ravel(), yy.ravel()], *w*)

    Z1 = Z1.reshape(xx.shape)

*ax*.contourf(xx, yy, Z1, *cmap* =plt.cm.tab20c)

*ax*.scatter(*x*[:, 0], *x*[:, 1], *c* = *y*.squeeze(), *cmap*=plt.cm.tab20c)

*# Prediction for Training dataset*

y\_hat = **predict**(X\_train, W)

acc = **accuracy**(y\_hat, y\_train)

**print**(f'2 Neurons, accuracy = {str(acc)}')



Fig. 14 Model accuracy (Best outcome)

*# Training with more neurons*

hidden = [3, 4, 5, 6]

*for* h *in* hidden:

    W, J = **bird\_strikes\_model**(X\_train, y\_train, h, *alpha*= 0.0001, *epochs*=1000)

    y\_hat = **predict**(X\_train, W)

    acc = **accuracy**(y\_hat, y\_train)

**print**(f'{h} Neurons, accuracy = {acc}')

Texto

Descripción generada automáticamente

Fig. 15 Neuron addition costs and accuracy (training)

*# Testing with same amount of neurons as training*

y\_hat = **predict**(X\_test, W)

acc = **accuracy**(y\_hat, y\_test)

**print**(f'2 Neurons, accuracy = {acc}')

hidden = [3,4,5,6]

*for* h *in* hidden:

    W, J = **bird\_strikes\_model**(X\_test, y\_test, h, *alpha*= 0.0001, *epochs*=1000)

    y\_hat = **predict**(X\_test, W)

    acc = **accuracy**(y\_hat, y\_test)

**print**(f'{h} Neurons, accuracy = {acc}')

Texto

Descripción generada automáticamente

Fig. 16 Neuron addition costs and accuracy (testing)

## Decision tree

**Steps**:

1. Data loading

2. Data analysis

3. Training and test separation

4. Gini and Entropy definition

5. Predictions

6. Tree plotting

7. Confusion matrix (both models)

8. Comparisons

X = ds\_bst.values[:, :-1]

**print**(X)

Y = ds\_bst.values[:, -1]

np.**unique**(Y, *return\_counts* = True)

Calendario

Descripción generada automáticamente

Fig. 17 Dataset uniqueness verification

*# Gini model*

clf\_gini = DecisionTreeClassifier(*criterion* = 'gini', *random\_state* = 100, *max\_depth* = 3, *min\_samples\_leaf* = 5)

clf\_gini = clf\_gini.fit(X\_train, y\_train)

*# Entropy model*

clf\_entropy = DecisionTreeClassifier(*criterion* = 'entropy', *random\_state* = 100, *max\_depth* = 3, *min\_samples\_leaf* = 5)

clf\_entropy = clf\_entropy.fit(X\_train, y\_train)

*# Predictions*

y\_pred\_gini = clf\_gini.predict(X\_test)

y\_pred\_entropy = clf\_entropy.predict(X\_test)

**print**(classification\_report(y\_test, y\_pred\_gini), '\n')

**print**(classification\_report(y\_test, y\_pred\_entropy))

Imagen de la pantalla de un celular de un mensaje en letras negras

Descripción generada automáticamente con confianza baja

Fig. 18 DT Predictions

*# Tree ploting*

plt.**figure**(*figsize* = (25, 10))

a = tree.plot\_tree(clf\_gini, *filled* = True, *rounded* = True, *fontsize* = 14)

plt.**figure**(*figsize* = (25, 10))

a = tree.plot\_tree(clf\_entropy, *filled* = True, *rounded* = True, *fontsize* = 14)

**print**('Train matrices')

cfm\_train\_gini = confusion\_matrix(y\_test, y\_pred\_gini)

**print**(cfm\_train\_gini, '\n')

cfm\_train\_entropy = confusion\_matrix(y\_test, y\_pred\_entropy)

**print**(cfm\_train\_entropy)

**print**('Test matrices')

*# Gini model*

clf\_gini = DecisionTreeClassifier(*criterion* = 'gini', *random\_state* = 100, *max\_depth* = 3, *min\_samples\_leaf* = 5)

clf\_gini = clf\_gini.fit(X\_test, y\_test)

*# Entropy model*

clf\_entropy = DecisionTreeClassifier(*criterion* = 'entropy', *random\_state* = 100, *max\_depth* = 3, *min\_samples\_leaf* = 5)

clf\_entropy = clf\_entropy.fit(X\_test, y\_test)

*# Predictions*

y\_pred\_gini = clf\_gini.predict(X\_test)

y\_pred\_entropy = clf\_entropy.predict(X\_test)

cfm\_train\_gini = confusion\_matrix(y\_test, y\_pred\_gini)

**print**(cfm\_train\_gini, '\n')

cfm\_train\_entropy = confusion\_matrix(y\_test, y\_pred\_entropy)

**print**(cfm\_train\_entropy)

The generated graphs and matrix will be displayed in results description section.

# Results description

*Graph generated for Linear Regression*

Gráfico

Descripción generada automáticamente

Fig. 19 Linear regression graph and error

For this model, great results have been achieved, as error is fairly low, and the graph demonstrates how cost descends elegantly, approaching zero.

*Graph generated for Neural Network*

Imagen de la pantalla de un celular con texto

Descripción generada automáticamente con confianza mediaGráfico

Descripción generada automáticamente

Fig. 20 Epoch graph and costs of neural network neuron addition

Yet again, costs descend as epochs augment, but this is usual behavior for NN. When analyzing cost, a fluctuation can be spotted, with similar costs repeating themselves, but with zero to no accuracy upgrades. This doesn’t mean something has gone wrong or similar, but perhaps this could indicate that this model may not be adequate for this particular problem.

*Tree graph and related data*

Texto

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

Fig. 21 Gini model prediction tree and matrices

Lastly, another common behavior can be spotted with this model execution when matrix analysis is made. Little under half the data it’s being correctly categorized, with about 60% of accuracy. It cannot be said that this model it’s bad or wrong, again, it’s just not the perfect fit for what it tried to address.

# Performance comparative

Talking about time, the classifications remains as it follows (in ascending time order):

1. Linear Regression
2. Decision Tree
3. Neural Network

It has to be kept in mind that NN works closely with epoch concept and implementation, giving it the unfortunate last place, at least for performance.

Now, comparing results, it’s easy to place LR as the best model for the current problem, as it showed tiny error, meaning virtually no cost whatsoever, but something else can be said that would be more appropriate: further analysis has to be made. Whether is dataset refining, or data manipulation, or model selection, the results gathered in this part of the project can’t be totally seen as conclusive. But maybe that is what all of this it’s all about, about searching and building better and better models. The results obtained are not wrong, but maybe they would be serving a greater purpose as a simple entry point.

As for now, LR it’s the undeniable outstanding model. This’ll be discussed in the final document.

# Conclusions and pending work

It can be considered now that the project has concluded, as the models for predictions has been implemented and its development has been demonstrated. Taking into account the level of detail required, the model implemented perhaps it’s not the best suited for a new state of the art fatality outcome alert system, but, with some grain level refining, this can be achieved easily; in the end, the goal of the project was always aiming to improve modern aviation systems for future possible disasters involving mid-air strikes and/or collisions.

As an ambitious future goal, further data collection and manipulation could be done in order for a better model to be made, a model that could potentially save thousands of lives, endangered by air treacherous obstacles, such as birds and incompetence.

Talking about difficulties faced in project development, the utilized dataset had to be though in an abstract manner, thus being more than a collection of data, but rather a compendium of crammed information. Although it wasn’t that much of a big deal, but the cleaning process was a cornerstone for the vast majority of model implementation, this being because the ultimate features selected to appear as critical and exclusively numerical data made possible the great results obtained, whereas leaving all information could harm the model(s) execution.

With this said, it can be asserted that probably it is not the model implementation what’ll determine a successful outcome, but the cleaning process and data structure itself. Even the best of the models will execute poorly if random and uncorrelated values are used as input.

As for now, this project has concluded, but not before saying how much impact the knowledge gathered along its realization and the course that made it possible could potentially have in future. Today, the entry point for a collision fatality prediction system is set, tomorrow, never knows…

# References

|  |  |
| --- | --- |
| [1] | National Safety Council, «Odds of Dying,» NSC Injury Facts, 2020. [Online]. Available: https://injuryfacts.nsc.org/all-injuries/preventable-death-overview/odds-of-dying/. [Last access: 14 September 2022]. |
| [2] | K. Hoke, «AeroSavvy,» AeroSavvy , 25 April 2018. [Online]. Available: https://aerosavvy.com/recurrent-training/. [Last access: 14 September 2022]. |
| [3] | Clifford Law Offices PC, «The National Law Review,» The National Law Review, 8 December 2020. [Online]. Available: https://www.natlawreview.com/article/most-common-causes-aviation-accidents. [Last access: 2022 September 14]. |
| [4] | S. Brown, «MIT Sloan School of Management,» MIT, 21 April 2021. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained. [Last access: 14 September 2022]. |
| [5] | S. Gupta, «Springboard,» Springboard, 25 September 2020. [Online]. Available: https://www.springboard.com/blog/data-science/when-not-to-use-ml/. [Last access: 14 September 2022]. |
| [6] | Uniqtech, «Medium,» Data Science Bootcamp, 22 December 2018. [Online]. Available: https://medium.com/data-science-bootcamp/understand-dot-products-matrix-multiplications-usage-in-deep-learning-in-minutes-beginner-95edf2e66155. [Last access: 2022 September 2022]. |
| [7] | J. A. Richards y X. Jia, «Supervised Classification Techniques,» from *Remote Sensing Digital Image Analysis*, Berlin, Springe, 1999, pp. 181-222. |
| [8] | C. Eastwood, Dirección, *Sully.* [Movie]. United States: Flashlight Films, 2016. |
| [9] | S. Lanfermeijer, «Tailstrike,» Tailstrike Consultancy, [Online]. Available: 2022. [Last access: 15 September 2022]. |
| [10] | D. Null, «The Guardian,» 2011. [Online]. Available: https://www.theguardian.com/notesandqueries/query/0,5753,-10081,00.html. [Last access: 2015 September 2022]. |
| [11] | J. Shih, «data.world,» 2016. [Online]. Available: https://data.world/shihzy/2000-2011-birds-strikes-planes. [Last access: 15 September 2022]. |
| [12] | pandas, «pandas documentation,» 12 September 2022. [Online]. Available: https://pandas.pydata.org/docs/. [Last access: 15 September 2022]. |
| [13] | F. Pérez & B. Granger, «Jupyter project,» Jupyter project, 2022. [Online]. Available: https://jupyter.org/. [Last access: 14 November 2022]. |
| [14] | V. Kaushik, «Analytics Steps,» Analytics Steps Infomedia LLP, 26 August 2021. [Online]. Available: https://www.analyticssteps.com/blogs/8-applications-neural-networks. [Last acess: 15 November 2022]. |