Neural Networks applied to business service improvement

A Degree Thesis
Submitted to the Faculty of the
Escola d'Enginyeria de Barcelona Est

Universitat Politècnica de Catalunya

by

Joao Marcos Pifer da Silva

In partial fulfilment of the requirements for the degree in
INDUSTRIAL ELECTRONICS AND AUTOMATICS ENGINEERING

Director: Samir Kaanan Izquierdo
Co Director: Miguel Montano Ibañez

Barcelona, January 2021

Abstract

Artificial Neural Networks are nothing but an algorithm which endeavors to emulate biological neural networks with the purpose of finding a relationship between an input and an output the network is being trained for. With the aim of applying the Natural Language Processing state of the art algorithms such as Transformers, is where the subsequent project comes into play.

In this project two artificial intelligence models will be implemented in order to improve business service and user experience. The first model consists in predicting the time in which a problem will be solved using BERT model and also as a way to introduce deep learning techniques to the business department. The second model consists in predicting if a problem will affect a larger number of users than indicated before or during the event, also introducing unsupervised techniques.

Resum

Xarxes Neuronals Artificials no són res més que un algoritme que intenta simular xarxes neurals biològiques amb la intenció de trobar una relació entre la informació d'entrada i una sortida per les quals la xarxa està sent entrenada. Amb l'objectiu d'aplicar algoritmes de l'estat de l'art de Processament del Llenguatge Natural, és on el següent projecte entra en joc.

En aquest projecte seran implementades dues solucions d'intel·ligència artificial amb l'objectiu de millorar el servei d'un negoci i l'experiència d'usuari. Per una banda, el primer model consisteix a predir el temps en el qual un problema serà resolt utilitzant el model BERT i també com a mètode d'introducció del deep learning al departament. Per l'altre, el segon model consisteix a predir si un problema afectarà un nombre major d'usuaris que l'indicat abans o durant l'esdeveniment té lloc, també com a mètode d'introducció de tècniques d'aprenentatge no supervisat.

Resumen

Redes Neuronales Artificiales no son más que un algoritmo que intenta simular redes neuronales biológicas con la intención de encontrar una relación entre la entrada y la salida para las cuales la red está siendo entrenada. Con el objetivo de aplicar algoritmos del estado del arte del Procesamiento del Lenguaje Natural, es donde el siguiente proyecto entra en juego.

En este proyecto dos soluciones de inteligencia artificial serán implementadas con el objetivo de mejorar el servicio de un negocio y la experiencia de usuario. Por un lado, el primer modelo consiste en la predicción del tiempo de resolución de un problema utilizando el modelo BERT y también como método de introducción del deep learning al departamento. Por otro lado, el segundo modelo consiste en predecir si un problema afectará a un número mayor de usuarios que el indicado antes o durante el evento tiene lugar, además como método de introducción de técnicas que aprendizaje no supervisado.

Acknowledgements

I would first like to thank my thesis advisor Samir Kanaan of the EEBE at Polytechnic University of Catalonia, due to guiding me with artificial intelligence when I did not know how to proceed.

I would also like to thank my supervisor Miguel Montano from SEAT S.A., the company I was when this project was carried out, due to give me such an opportunity to develop what I like the most and put a lot of interest in investigating and introducing new techniques.

Finally, I would like to thank my family who have put a lot of effort in order to be at this point in my life. Thanks to each friend, professor and anyone who has motivated me to be passionate through this project development.

Contents

List of Figures

List of Tables

Acronyms

NNs: Neural Networks
NLP: Natural Language Processing
AI: Artificial Intelligence
ML: Machine Learning
AI: Deep Learning
MLP: Multi-Layer Perceptron
MSE: Mean Squared Error
MAE: Mean Absolute Error
MSLE: Mean Squared Logarithmic Error
BERT: Bidirectional Encoders Representations from Transformers
RTP: Resolution Time Prediction
WIA: Wrong Impact Assigment
AWS: Amazon Web Services

1 Introduction and Project Planning

1.1 Context

The project is carried out at SEAT, an automotive company part of Volkswagen Group, which apart from manufacturing personal vehicles, designs and develops mobility products and services at SEAT:CODE and is currently introducing e-mobility solutions to Barcelona at SEAT:MO. More precisely, this Artificial Intelligence solution takes place during an internship as a Data Scientist at SEAT's IT Department.

1.2 Motivation and purpose of the project

The main motivation of this project is the need of an engineer to apply the new upcoming technologies to the exponential growing industry apart from serving as a first contact to AI. As a purpose, the aim is to improve the service quality and introduce for the first time deep learning techniques to the department.

The project main goals are:

- Get a deep knowledge on NLP State of the Art techniques.
- Introduce the acquired knowledge to the department.
- Develop algorithms to improve the business service quality.

1.3 Requirements and Specifications

The requirements are based mainly in two ideas that wanted to be included to the workflow previously:

- Predict the time in which a problem will be solved.
- Predict if a problem will affect a larger group of users before or during it occurs.

Referring to the first requirement specifications:

- The model must give the predicted time in minutes.
- The model must predict the time of Priority 1,2 and 3 only.
- Resolution times over 120 minutes must be excluded from the supervised dataset given the fact that they are considered outliers.
- The model must work completely offline.
- The model must use deep learning.
- Any sensitive information must be shown in the project documentation.

Referring to the second requirement specifications:

- The application must give an easily readable Excel file with most relevant tickets suspicious of having a higher impact in the future.
- The supervised dataset must only include Ticket Type 'Incident'.

- The supervised dataset must only include Master/Child field with content different from 'Child'.
- The application must be implemented in a jupyter notebook.
- It is considered that a problem is affecting a group of users when there are 5 or more tickets with the same or similar content.
- Any sensitive information must be shown in the project documentation.
- Any code developed in this section must be shown in the project documentation.

1.4 Project Background

The previous project background only consisted in a approach to the time prediction which did not succeed. This first approach consisted in a code based in time series without taking account of text, the following image reflexes this work:

Figure 1: Resolution Time Prediction project background. Source: SEAT S.A.

In the above image, the Y axes represents the predicted time in logarithmic scale, moreover it is indicated the maximum time depending on the Priority value. In the X axis it is represented the moment in which the problem was created.

The idea of this prediction based in time series, is to adjust the curve depending on the previous resolution times and adjusting to an specific case for each application. The gray dots corresponds to all priority 6 tickets, the orange dots to the a more specific type of tickets and the green dots, which the prediction (green line) was fitted to, corresponds with the application subfield.

Finally, this model could end up having a good approximation but nevertheless it has two main disadvantages. On the one hand, to have a good approximation it is necessary to do an exhaustive work adjusting parameters and creating specific models for an infinity of cases. On the other hand, this type of model is limited to applications with at least a minimum number of tickets per case.

1.5 Work Plan

As it was mentioned previously, one of the main goals is to introduce deep learning techniques to SEAT IT Department. In addition, the previous deep learning background knowledge was generally theoretical, thus Kanban could be the appropriate working methodology for this project because of its adaptability to the project needs and requirements/specifications changes.

Hence, the following tables sums up the main tasks through the project development, where:

- **P**: corresponds project management related tasks.
- **I**: corresponds to investigation related tasks.
- **D**: corresponds to project content development.

Ref	Title	Short Description
P01	Scope of the project	Define the main content of the project, may vary through the project development.
P02	Project requirements and specifications	Define project requirements and specifications based on needs, technologies, scope...
I01	First NN Regression Model	First approach to deeplearning by implementing a basic MLP regression model with Word2Vec embeddings.
P03	Project Architecture	Define the project architecture in order to assure scalability.

Table 1: Work Plan for Project Week 01

Ref	Title	Short Description
I02	Multi-Input investigation	Investigation about mixing text and numerical information.
I03	First NN Classification Model	First approach to classification by implementing a basic MLP regression model with Word2Vec embeddings.
D01	Preprocessor architecture	Define the architecture of Preprocessor class which might be used for both regression and classification models.

Table 2: Work Plan for Project Week 02

Ref	Title	Short Description
I04	BERT Model investigation	Investigation about State of the Art NLP classification model.
D02	RTP model approach	Resolution Times Prediction model approach, using BERT for the first time.

Table 3: Work Plan for Project Week 03

Ref	Title	Short Description
I05	RTP high loss	Investigate about data scaling to avoid super high loss.
D03	Cannot Save Model bug	Model could not be saved in a local file.

Table 4: Work Plan for Project Week 06

Ref	Title	Short Description
D04	Run Model locally	Eliminate completely any internet access the model needs.
D05	Run Model on GPU	Run model on local GPU to improve training time performance.
D06	Run Model in external machine	Run model in external machine which has a better GPU.

Table 5: Work Plan for Project Week 07

Ref	Title	Short Description
D07	WIA model approach	Wrong Impact Assigment model first approach
D08	WIA ML Classification Model	Create an easy classification model based on classical ML techniques
D09	Operational Trainer script	Create an operation trainer script in order to facilitate changes through training.

Table 6: Work Plan for Project Week 10

Ref	Title	Short Description
D10	RTP in production	Put RTP model in production

Table 7: Work Plan for Project Week 11

Ref	Title	Short Description
I05	BERT Layers	Deep investigation about BERT architecture and ways to fine-tune.

Table 8: Work Plan for Project Week 12

Ref	Title	Short Description
I06	Custom Loss function	Create a custom loss function for RTP model in order to penalize higher errors

Table 9: Work Plan for Project Week 14

Ref	Title	Short Description
D11	WIA classification model	Compare how a classical ML model performs over the BERT model.
D12	WIA mix clustering and classification	Mix machine learning classification model with clustering in order to give a better approximation

Table 10: Work Plan for Project Week 15

Ref	Title	Short Description
D13	WIA In production	Put WIA model in production.

Table 11: Work Plan for Project Week 16

Ref	Title	Short Description
D14	WIA corrections	Sort by most relevant cases.

Table 12: Work Plan for Project Week 17

1.6 Work Plan Deviations

For this project, a weekly meeting was established to follow the project progress and set next week tasks. Despite of that, as every software project there are some deviations and delays through investigation and development phases. The following table sums up the project main deviations for each Project Week (PW).

Ref	Title	Reason	Planned	Done
D02	RTP model approach	Vague knowledge about BERT architecture and meaning of its inputs.	PW03	PW5
D05	Run Model on GPU	Setting up tensorflow-gpu for the first time and finally realizing it only worked with NVIDIA GPUs.	PW07	PW8
D06	Run Model in external machine	Eliminating code,internet dependences,setting up tensorflow-gpu and installing NVIDIA GPU framework in a machine with no internet access and no IDE caused this delay	PW07	PW9
I05	BERT Layers	Vague knowledge about how to access, modify and combine BERT intern layers outputs with tensorflow backend functions.	PW12	PW13

Table 13: Differences with initial planning.

2 State of the art of the NLP

Natural Language Processing, or NLP for short, is broadly defined as the automatic manipulation of natural language, for instance speech or text, by software. As most of the content on the internet is basically text, it is not conspicuous to find a technologie for that purpose.
In this section the classical machine learning techniques will be overviewed in order to be more focused on the upcoming technologies such as deep learning.

First of all, the misconception caused by the media to use *Artificial Intelligence* (AI), *Machine Learning* (ML) and *Deep Learning*(DL) as synonyms is something to be clarified.AI is nothing but the purpose of application of ML or DL algorithms, this purpose is commonly to automate tasks that could be done by the human being. In the NLP field, these tasks could be *Automatic Text Summary*, *Text Classification*, *Sentimental Analysis*,etc. In respect to ML, being a field of AI, compounds a bunch of algorithms and techniques that has been around for more than 50 years. Moreover, this is the current technique used in the department to solve NLP problems that will be explained more in detail in a further section. Finally, DL is considered a specific field of ML where it contributes with its own neural networks based algorithms.

2.1 Machine Learning

The classical way to create a NLP ML model could be divided in two phases, preprocessing and modelization.The preprocessing phase consists mainly in the noise extraction to improve the performance and to facilitate the job for the model in order to get a better accuracy. Furthermore, this phase is also important to DL as the unique difference between ML and DL is the modelization algorithms and techniques.

Thus, the following techniques are the commonly used in the preprocessing phase:

- **Stopwords Extraction**: This implementation basically extract a list of words from the main text which do not add much meaning to the sentence. Those words are generally articles, prepositions, adverbs,etc.
- **Word File Frequency Extraction**: This implementation means to extract words depending on its file frequency, mainly to reduce noise. It is understood as file frequency the appearance of a certain word in different samples. Extracting words with low file frequency will help the model to perform faster, otherwise extracting high frequency words will help the model to decide between two or more categories better if those common words were not extracted.
- **Stemming**: Stemming is kind of normalization for words. The words which have the same meaning but have some variation according to the context, for instance plurals or genre, are normalized extracting the prefixes and sufixes.
- **Lemmatization**: Lemmatization is the process of finding the lemma of a word depending on their meaning. It helps in returning the base or dictionary form of a word, for instance in Lemmatization the word 'was' would be repliced by 'be' while in Stemming it would not suffer any change.
- **Tokenization**: As the model to minimize the error uses numbers, in some way it is necessary to transform the words into this format. There are many different ways to do this, the simplest one is tokenization in which every word is converted into a number.

Referring to the modelization phase, as the goal for this project is to introduce deep learning techniques to the department, the machine learning techniques will be only overviewed. Currently, most of the NLP projects developed are based in supervised learning, then the common algorithms used for this are the following:

- Random Forest Classifier
- Linear SVC
- Multinomial Naive Byas
- Logistic Regression

All the techniques and algorithms mentioned can be easily found in scikit-learn [8] and nltk [2] libraries.

2.2 Deep Learning

Even though the scope of this project is to apply neural networks, it is not superfluous to make an overview on how a neural network works. As it was mentioned before, deep learning is a subsection of machine learning which contributes with its neural network based algorithms. A neural network is a chain of simple neurons that aims to recognize the relationships in a set of known data provided through a process that mimics the way human brain operates and analyse. Neural networks can adapt itself transforming the input through the net in order to get the best result as similar as possible to the supervised output.

The first image that comes to mind of a NN is the represented in the Figure 2, also known as Multilayer Perceptron. Every NN has Input, Hidden and Output layers where Input and Output layers are the interface with the user and the Hidden layers is where the magic takes place. In this net every neuron is known as a perceptron, even though there are many distributions of networks their behavior are basically the same.

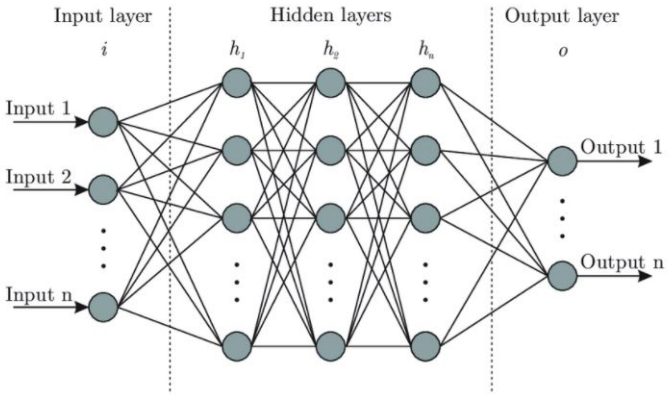


Figure 2: Multilayer Perceptron representation. Source: Designing your neural networks by Lavanya Shukla [10]

2.2.1 Activation function

In every perceptron the inputs are combined using the net input function and transformed by an activation function to approximate an output with the minimum error possible, this error is

calculated through a loss function and finally an optimizer is used to find the error global minimum and update the weights that transform each input through the gradient descent algorithm.

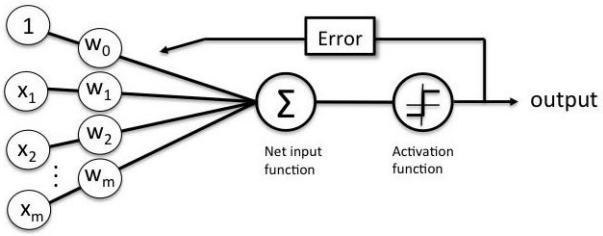


Figure 3: Perceptron behavior representation. Source: What is Perceptron: A Beginners Tutorial for Perceptron by Simplilearn [11]

A perceptron is a function that maps its input x, which is multiplied with the learnt weight coefficient; an output value f(x) is generated as following:

$$f(x) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} + b > 0 \\ 0 & otherwise \end{cases} \tag{1}$$

In the above equation:

- w: vector of weights.
- b: bias, an element that adjusts the boundary away from origin without any dependence on the input value.
- x: vector of input values.

Then the net input function follows the next formula:

$$F(x_0, x_1, ..., x_n) = \sum_{i=0}^{N} w_i \cdot x_i + b \tag{2}$$

The output of each perceptron can be represented as 1 or 0, but depending on the task here is where the activation functions takes a really important role.With an eye on them, depending on the task it is convenient to select one or another.

Binary classification	**Multi-class classification**	**Regression**
Sigmoid	Softmax	Relu
Tanh		Linear

Table 14: Common used activation functions depending on task.

2.2.2 Loss function

More in detail, in respect to the loss functions, the errors for each perceptron are calculated through them. A certain loss function is a way to evaluate how good is the model performing.

Classification	Regression
Hinge Loss	Mean Squared Error
Cross Entropy Loss	Mean Absolute Error
	Mean Squared Logaritmic

Table 15: Common used loss functions depending on task.

For the regression models, for instance, each loss functions evaluates differently the differences between predicted and supervised outputs. MAE penalizes huge and small errors with the same exponential rate causing huge errors to not being enough penalized respect a considerable amount of small errors . However, MSE makes huge errors become even higher compensating the high mount of small errors problem. Finally, MSLE does not differentiate between huge and small errors, meaning that the logarithmic error between 20 and 10 is almost the same than 20000 and 10000.

2.2.3 Optimizer

The optimizer uses the gradient descent algorithm to update the weights in order to minimize the error. Gradient indicates the direction of increase. As the minimum point in the valley is the target, it is necessary to go in the opposite direction of the gradient and then updating the parameters in the negative direction to minimize the loss. There are many different algorithms that performs better depending on the application, but Adam optimizer is the standard one that works independently on the task with an acceptable result and is the one used in this project.

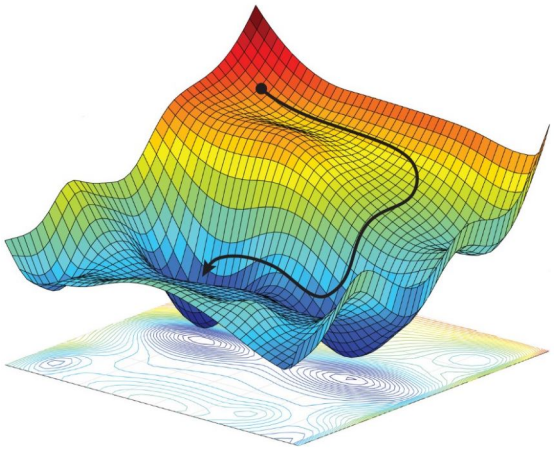


Figure 4: Gradient descent visualization . Source: A. Amini et al. "Spatial Uncertainty Sampling for End-to-End Control". NeurIPS Bayesian Deep Learning 2018

Normally, a random point is set to the model to calculate the mean error and once this is done, the optimizer operates between each epoch calculating the gradient descent direction to step forward and the recalculate the mean error and so on.
An important concept to comment is that the speed the optimizer when moving through the error field is dependent to the batch size. Batch size could be understood as the quantity of samples put at once in the net, reducing the training time drastically as bigger this parameter gets.

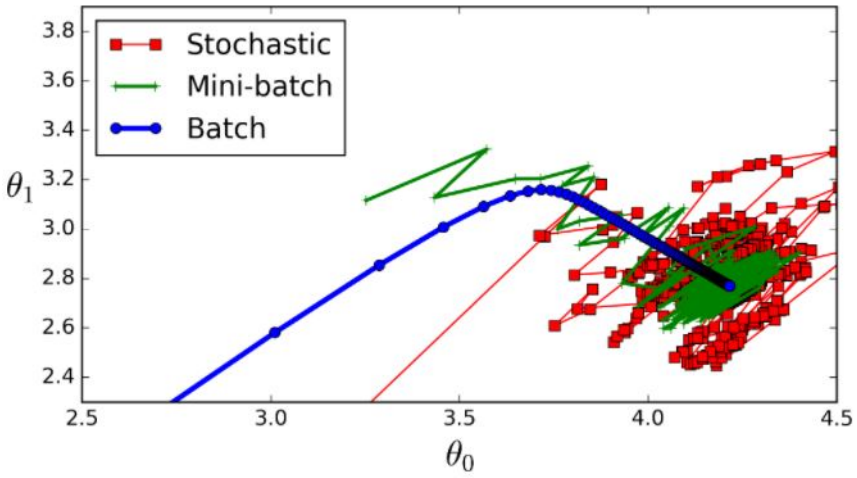


Figure 5: Effect of batch size in gradient descent . Source: What is batch size in neural network? Answer by itdxer [5]

The smaller the batch the less accurate the estimate of the gradient will be. On the one hand, in the above figure, the direction of the mini-batch gradient (green color) fluctuates much more in comparison to the direction of the full batch gradient (blue color).On the other hand, stochastic is just a mini-batch with batch size equal to 1. In that case, the gradient changes its direction even more often than a mini-batch gradient.

2.2.4 Underfitting and Overfitting

Underfitting and overfitting are the typical issues a deep learning scientist must face every time a model is being created. On the one hand, underfitting basically corresponds to a vague model that can not approximate properly to real values. On the other hand, overfitting corresponds to a model that approximates too much to the cases studied that performs even much worse with data ever seen.

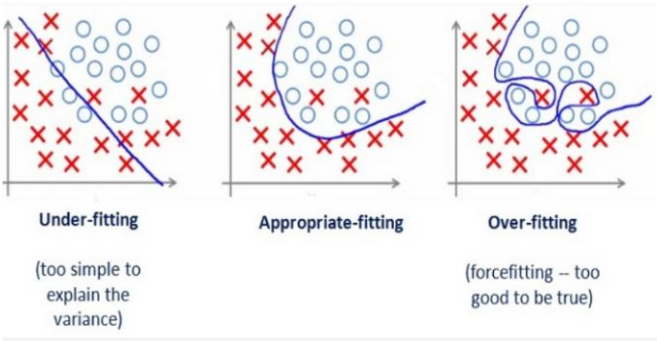


Figure 6: Overfitting and Underfitting representation. Source: What is underfitting and overfitting in machine learning and how to deal with it by Anup Bhande [1].

The way to differentiate underfitting from overfitting is using the train/validation loss graph through epochs. While training set is unable to understand the dependencies in underfitting, in overfitting it does but validation set does not.

Fortunately, there are techniques in DL in order to fix these problems.

Techniques to prevent Underfitting:

- Creating a more complex neural network. The model is too simple to understand the complex dependency.
- Training Longer. The model has not found the appropriate weights to fit the values.

Techniques to prevent Overfitting:

- Using Dropout layers. Basically, the operation mode of this layers are deactivating random neurons of the previous layer in order to increase the model robustness.
- L2/L1 Regularization. DL models tend to work unpredictably when the weights get high values, but this regularization methods joins a weight dependent addend to the loss function in order to prevent this to happen.
- Data augmentation, the model does not understand the relation between the inputs and outputs and more data is needed.
- EarlyStopping, DL model usually tends to work better in initial epochs and stopping it before training and validation loss diverges is way to stop the model from worsen.

Finally, to evaluate the model performance it is common to separate the dataset in three sections: training set, validation set and testing set. Training set corresponds to the major amount all three and is used to train the model, validation set corresponds to the minor amount and it is used to evaluate the model performance while training without affecting the weights and having a approach on how the model will perform with the test set which is finally used to see how the model will work in 'real life'.

2.3 BERT Model

Google's BERT model[4] has marked a turning point for the field of NLP due to its superior performance over a wide variety of tasks. BERT builds on two key ideas that have been responsible for many of the recent advances in NLP: (1) the transformer architecture and (2) unsupervised pre-training[13]. Moreover, this model was open-sourced with two versions of pre-trained model $BERT_{BASE}$ and $BERT_{LARGE}$ where both of them were pre-trained in English.
The language the model was trained is as important as the model itself because the way it achieves state of the art results is due to its pre-training with a very large amount of data in that specific language which reduces drastically the training time when fine-tuning BERT model for a specific task. As a consequence, the model used in this project is a Spanish version of $BERT_{BASE}$ called BETO [3] because of the dataset nature.

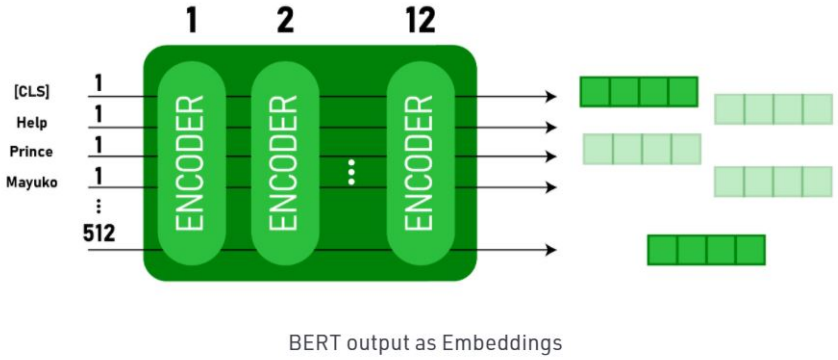


Figure 7: $BERT_{BASE}$ Layers representation. Source: Explanation of BERT Model – NLP by pawangfg [7]

Going further on $BERT_{BASE}$ architecture, it is consisted of a 12-layer model (while $BERT_{LARGE}$ is 24-layer long) formed by Encoders. An Encoder is the neural network architecture that has let this model obtain state of the art results and it is formed by 6 identical layers where each layer has two sublayers. The first layer is a multi-head self-attention mechanism and the second is a position wise fully connected feed-forward network. There is a residual connection around each of the two sub layers followed by layer normalization [9], see Figure 8:

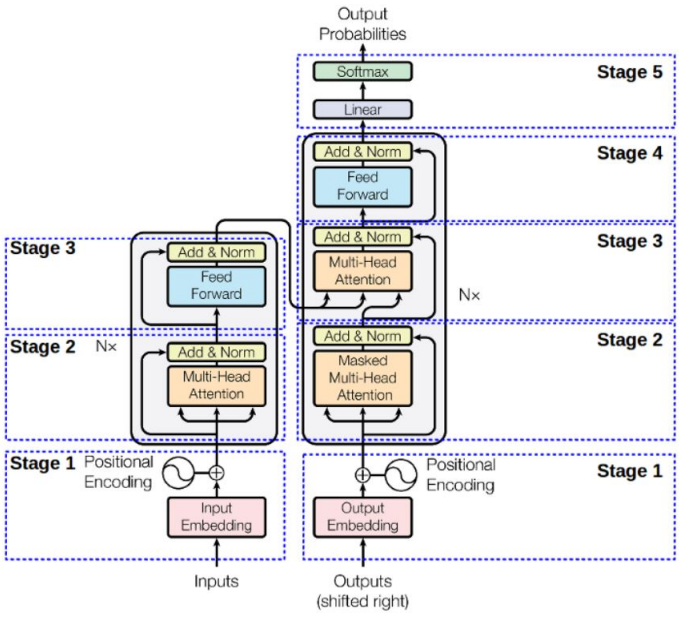


Figure 8: BERT: Bidirectional Encoder Representations from Transformers by Akshay Prakash [9]

The purpose of this architecture is doing what BERT name exactly means, a Bidirectional Encoders Representations from Transformers. This means that this architecture endeavors to look for a representation of each word going on both directions in a sentence to get a better representation. Despite of the architecture complexity, for this project an Encoder will be con-

sidered as black box and the field of operation will only reach level of Figure 7.

As it was mentioned before, the second key of this model is the unsupervised learning. The pre-trained weights are achieved through two unsupervised tasks:

- **Masked Language Modeling**: Consists of predicting a missing word given the left and right context.
- **Next Sequence Prediction**: Predicting whether one sentence follows another.

However, using the pre-trained weights that facilitates the training, it is possible to fine-tune BERT to perform many common tasks such as Binary/Multi-class Classification and Question Answering but also others not very used to be seen as Regression, which is the goal of this project.

Therefore, $BERT_{BASE}$ gives a 768 output vector per each word analysed ,where it is included a sentence representation as the [CLS] token vector, per each Encoder layer. An important fact about $BERT_{BASE}$ is that the input sentence is limited to 512 words, having then an output layer of maximum dimensions 12 x 512 x 768 where first dimension points to each layer, the second one to the sentence and the last one to each word or token.
Consequently, this produces the model to have approximately 110 Million parameters requiring a minimum hardware characteristics to be feasible.

Finally, in respect to the model inputs, even though it is possible to use any word embedding such as a simple Tokenizer it is recommended to use the word embedding the model was trained with, the BerTokenizer. BerTokenizer uses word piece embedding algorithm to separate each word from its prefix/suffix differentiating, for instance, between plural and singular words because of its special token specified in the vocabulary file.
This specific tokenizer produces the following inputs for BERT model:

- **Token embeddings**: A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
- **Segment embeddings**: A marker indicating Sentence A or Sentence B is added to each token. This allows the encoder to distinguish between sentences.
- **Positional embeddings**: A positional embedding is added to each token to indicate its position in the sentence.

3 Project Development

As a first version, the idea is to get the better result as possible by just using the text as input information,and finally including extra information to the net.
One the one hand, to predict the time in which a problem/ticket will be solved a regression model based in NNs will be considered. On the other hand, to predict the range of affectation a problem could have, a non-supervised model mixed with a supervised model will be used. In order to simplify the nomenclature, the regression model will be called Resolution Time Prediction (RTP) and the affectation model will be called Wrong Impact Assignment (WIA).

3.1 Environment Setup

The environment the model was run took a really important role when referring to time performance. In the past years, GPUs which initially were build for supporting games graphics, ended up being a really good tool for deep learning because of its treatment with tensors. The machine located in the department contains a fully operating Quadro P5000 GPU which helped reducing the time performance drastically despite of its lack of internet access. With the simplest possible BERT architecture, here is the time comparison while running the model on a 16GB GPU and a 16GB CPU:

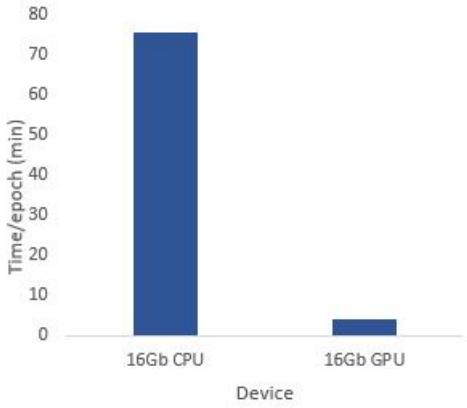


Figure 9: CPU vs GPU time performance per epoch in minutes.

It is important to mention that the Figure 9 is to only represent the improvement scale given that training time per epoch depends also on the dataset size, batch size and many different hyper parameters. Then, to achieve this time improvement it is necessary to install a specific library that works with GPU, tensorflow-gpu [12].

3.2 Dataset structure

Despite the fact that the same dataset will be used for both solutions, each subset might or might not contain the same tickets because each output is supervised differently.

In respect to the dataset content, from the typical problem resolution management fields the following will be used:

- **Title**: corresponds to a short description of the problem.
- **Description**: corresponds to a more detailed description of the problem.
- **Solution**: if the problem is in Status Closed/Resolved, this field corresponds to the problem solution.
- **Impact**: this field corresponds to the problem field of affectation. In a scale 1 to 4, where one 1 means that the problem is affecting a large number of users in a certain application and 4 to a single user or a negligible number of users compared to all users.
- **Priority/Prio**: Depending on the application field of operation, this number in a scale 1 to 8, where 1 is the more critical and 8 is the least, means how critical is that problem.
- **Time Class**: corresponds to the application technical support operating hours. It corresponds to a value which goes from 1 to 4 indicating the range of hours a problem could be solved by the application support team.
- **Open Time**: corresponds to the date when the problem was created.
- **Resolution count**: corresponds to the time in which the problem was resolved.
- **Status**: corresponds to the current status of the problem, this status could be Open, Resolved, Closed, Wait on User...
- **Ticket Type**: corresponds to Incident, Service Request or Complaint. Incident corresponds to a problem with an application, Service Request is when an user ask for access or installation and Complaint when a user has any complaint about the service given.
- **Master/Child**: Master if it was found a ticket that once solved all the referenced child tickets will solve automatically, Child if the same ticket or its superior is already being treated and finally this field is let empty if there is no trace about dependency between tickets.

3.3 Resolution Time Prediction

The aim of this model is to improve user experience by informing the time in which his problem will be solved. In addition, this section will serve as a deep learning applied introduction to the department.

3.3.1 Dataset distribution

For this model, from the entire dataset of tickets only the ones in status Closed and Title/Description fields will be used to get the required supervised condition. However, as specification for this tool it has been established that the time prediction should only consider tickets with priority equals or fewer than 3 consequently having 13389 samples. Then the dataset distribution given these conditions could be seen in the Figure 10.

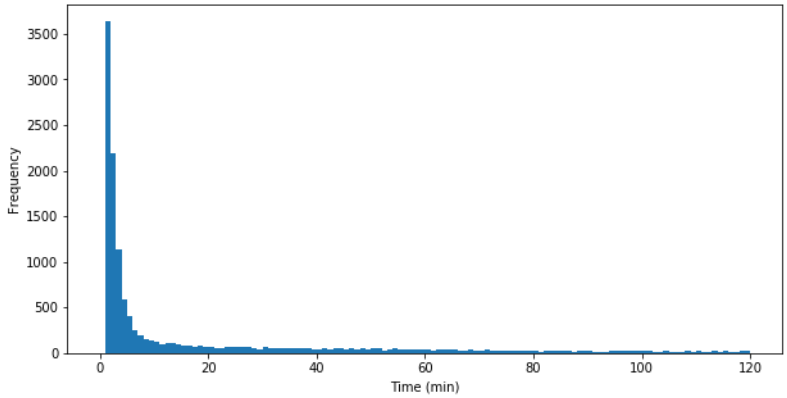


Figure 10: Resolution Times dataset distribution

In respect to the dataset, the tickets with resolution time higher than 120 minutes will considered as outliers because of specifications. The hyperbolic form of the distribution is mainly caused because there are a comparable quantity of problems that are solved while a phone call is taking place or even after when the operator must document the problem creating and closing it an instant later.

3.3.2 Solution Implementation

The process followed to implement the RTP model is consisted of improving preprocessing and modelization phases. The first step to take is to establish a first model architecture in order to compare whether the model improves or not.

First approach implementation

This model will basically follow the same architecture as the model proposed by Emil Lykke Jensen[6] adapted to a regression model:

```
1
2 # Load model config
3 model_config = BertConfig.from_json_file(MODEL_CONFIG_PATH)
4 model_config.output_hidden_states = True # Enable all BERT Layers access.
5
6 # Load the Transformers BERT model
```

```
transformer_model = TFBertModel.from_pretrained(MODEL_PATH,config = model_config, from_pt=True)
transformer_model.trainable = True # False fixes the weights.
bert = transformer_model.layers[0]

# Building model Inputs
input_ids = Input(shape=(MAX_LEN), name='input_ids', dtype='int32')
token_type_ids = Input(shape=(MAX_LEN), name='token_type_ids', dtype='int32')
attention_mask = Input(shape=(MAX_LEN), name='attention_mask', dtype='int32')
bert_inputs = {'input_ids': input_ids,
        'token_type_ids':token_type_ids,
        'attention_mask': attention_mask}

inputs = bert_inputs
x = bert(bert_inputs)[1] # BERT 12th layer [CLS] token


outputs = Dense(units=1,
                activation='relu',
                name='output_layer',
                kernel_initializer=TruncatedNormal(stddev=model_config.initializer_range))(x)

model = Model(inputs=inputs, outputs=outputs, name='BERT_regression')
```

This first model gives the following result:

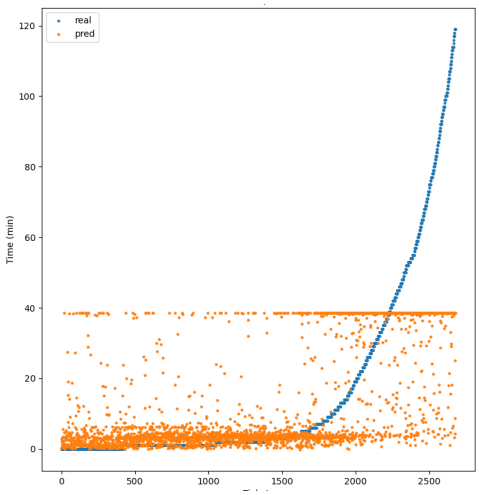


Figure 11: RTP model first approach. Source: self-elaboration.

This figure contains lots of information. First of all, the Y axis represents the time while X axis represent the ticket/problem index a way to represent the sentence contained in each ticket. The dotted blue line represents the supervised real time the ticket took to be solved and the orange dots represent the predicted time using BERT, then the closer the orange dots are to the blue line the more accurate the model is.

In respect to the graphic form itself, it is possible to see that BERT model really differs from fewer resolution times from higher ones but it seems to be saturating near 40 minutes. This is often caused because of the scale of the data given that the neural network saturates the higher prediction times in order to predict better the fewer, which are majority.

Output Scaling implementation

Scaling the data from 0 to 12, to maintain direct proportionality, seems that the model left the saturating condition. In order to scale the outputs, the following code is given as example:

```
scaler = MinMaxScaler(feature_range=(0,12)) # Selected Scaler from sklearn
Y_scaled = scaler.fit_transform(Y)  # Scaling the outputs
```

It is important to mention that as low as the scaling range gets the more probable is the model to get in an underfitting condition.

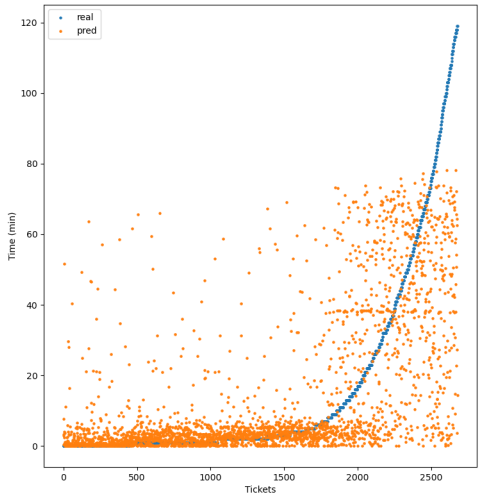


Figure 12: RTP model first approach scaled. Source: self-elaboration.

Moreover, in order to compare the model improvement, it is necessary to have the data in the same scale if not the loss would have very different values between further implementations. At this point, the model loss is 4.99 but as it is always recommended to use more than one indicator, then the r^2 correlation at this point is 0.355. Therefore, the goal is to reduce as much as possible the MSE and increase the r^2.

In respect to the model loss, at this point it completely presents an overfitting behavior due to validation and train loss divergence as the following figure shows:

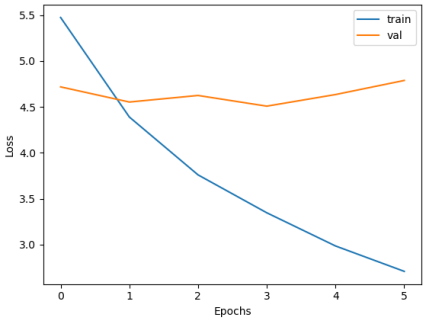


Figure 13: Output scaling implementation, model loss. Source: self-elaboration.

As it was mentioned in previous sections, there are different techniques to prevent the model to overfit, then the first implementation that could be done is including a callback function in order to evaluate the validation loss during the training process.

```
callback = EarlyStopping(monitor=monitor,
                                 min_delta=min_delta,
                                 patience=patience,
                                 restore_best_weights=True)

history = model.fit(x=X_train,
                y=y_train,
                validation_split=val_size,
                batch_size=batch_size,
                epochs=epochs,
                callbacks=[callback])
```

BERT Hidden Layers implementation

As it is proposed by the github user Han Xiao[14] it is recommended to go a level deeper in BERT architecture and see how the model performs per each layer. As it was mentioned in a previous section, each encoder layer has an output of size 512x768 where 512 is the maximum sentence length and 768 is the word embedding. Given the fact that the previous model uses the last [CLS] token vector from layer 12 as a representation to a sentence, in previous layers it is recommended to create an own sentence embedding (also commented in Han Xiao's post). In the following figure Han Xiao shows how two different sentence embedding performs per each layer:

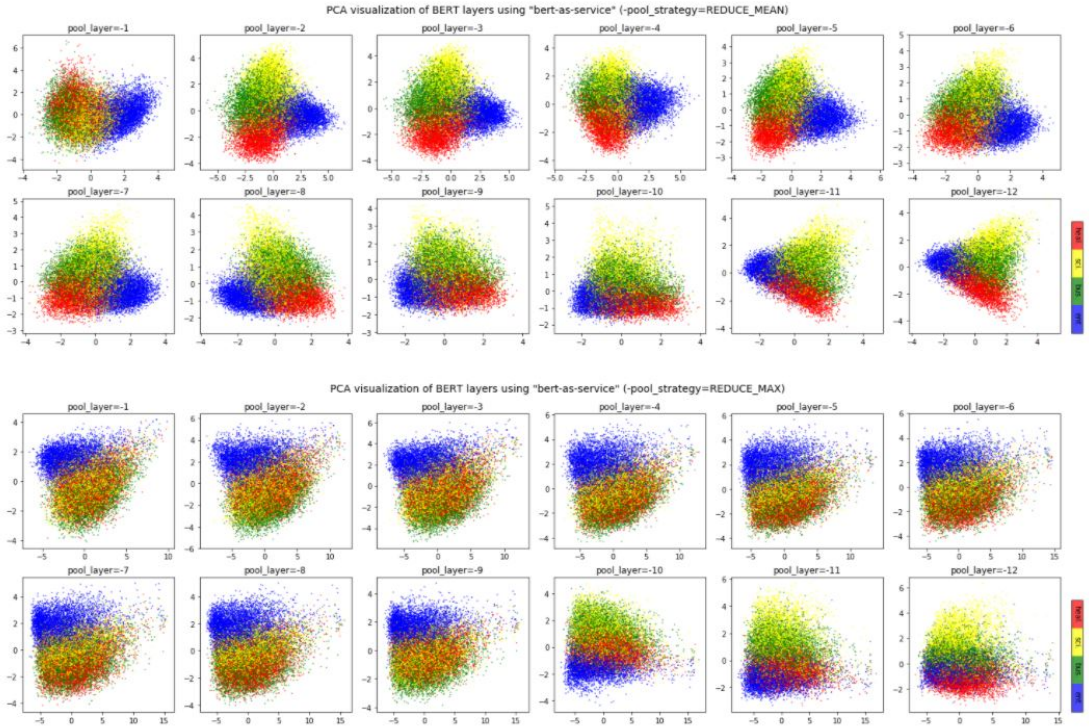


Figure 14: Sentence embedding per each $BERT_{BASE}$ layer. Source: Q: So which layer and which pooling strategy is the best? answer by Han Xiao[14]

It seems that meaning the array along word dimension as a way to get a sentence representation gives a better result. To perform this implementation is necessary to access all BERT Layers modifying the previous BERT architecture. The following code is given as example:

```
hidden_state = 2
'''
hidden_state = 0 -> corresponds to BERT last layer, shape = (None,512,768)
hidden_state = 1 -> corresponds to BERT last [CLS] layer, shape = (None,768)
hidden_state = 2 -> corresponds to all BERT Layers,shape = (12,None,512,768)
'''

hidden_layer = 1 # values corresponded to BERT Hidden Layer, from 1 to 12.

x = bert(bert_inputs)[hidden_layer]

# The AveragePooling layer is used to do the Mean of all words in a sentence,
    having a sentence unique representation
x = AveragePooling1D(pool_size=(MAX_LEN),
                     data_format='channels_last',
                     name='sentence_embedding')(x)

# The squeeze layer is to reduce the resulting non significant dimension.
x = tensorflow.squeeze(x,axis=1,name='squeezer')
```

Then applying this concept to each layer these are MSE and r^2 given for the test set:

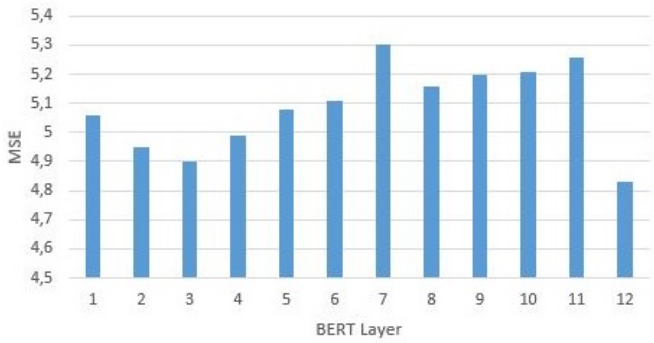


Figure 15: Model MSE per each $BERT_{BASE}$ layer. Source: self-elaboration.

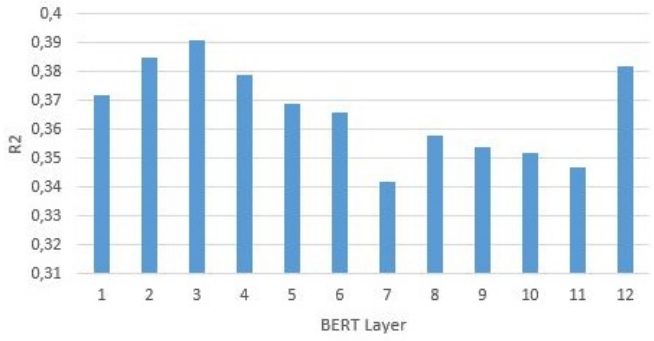


Figure 16: Model r^2 per each $BERT_{BASE}$ layer. Source: self-elaboration.

For this specific task, at first sight the first layers give a general approach of the problem

but the last layer also seems to give a good result. A good practice while working with BERT intermediate layers is to concatenate them, then the decision taken was to concatenate layers 2,3,4 and 12 to see if the model performance improves, having a 768x4=3072 output. There are many combinations such as summing or meaning but it is common to concatenate 4 layers and it is always recommended to try which combination is better for an specific task. The following table is given for a better understanding on what is happening to each dimension and shows the required dimension transformations in order to concatenate the layers:

Layer	Type	Shape	Shape concept
attention_mask	Input	(None,256)	(batch,max_len)
input_ids	Input	(None,256)	(batch,max_len)
token_type_ids	Input	(None,256)	(batch,max_len)
hidden_layers	Tensor	(4,None,512,768)	(layer,batch,max_len,word_embedding)
permute_dimensions	Transpose	(None,4,512,768)	(batch,layer,max_len,word_embedding)
sentence_embbeding	AveragePooling2d	(None,4,1,768)	(batch,layer,max_len,sentece_embedding)
squeezer	Squeeze	(None,4,768)	(batch,layer,sentece_embedding)
concatenate	Concatenate	(None,3072)	(batch,layers_sentece_embedding)
output	Dense	(None,3072)	(batch,layers_sentece_embedding)

Table 16: BERT model hidden layers concatenation, shape transformation.

In Figure 16 batch represents the selected batch size for the model, max_len to the maximum sentence length, word_embedding is the fixed 768 word embedding fixed by $BERT_{BASE}$, sentence_embedding represents the sentence representation after averaging the words embeddings and finally layers_sentence_embedding corresponds to the final four layer representation of each sentence. In addition, to perform this implementation the following code is given as example:

```
1
2  hidden_layers = bert(bert_inputs)[2]
3  hidden_layers = (hidden_layers[2],
4                   hidden_layers[3],
5                   hidden_layers[4],
6                   hidden_layers[-1])
7  hidden_layers = tensorflow.convert_to_tensor(list(hidden_layers))
8
9
10 # hidden layers shape is (layer,batch,max_len,word_embedding)
11 # The permutation is necessary to have batch_dimension in the first position in
       order to perform the AveragePooling.
12 x = tensorflow.keras.backend.permute_dimensions(hidden_layers,(1,0,2,3))
13
14 # Average through the sentence_dimension
15 x = AveragePooling2D(pool_size=(max_len,1),
16                      data_format='channels_first',
17                      name='sentence_smbedding')(x)
18
19 # Eliminates the non significant dimension
20 x = tensorflow.squeeze(x,axis=2,name='squeezer')
21
22 to_concat = []
23 for i in range(x.shape[1]):
24   to_concat.append(x[:,i,:])
25
26 # layers concatenation
27 x = concatenate(to_concat,axis=-1)
```

The MSE and r^2 concatenating this layers are 4.74 and 0.381 respectively, improving the performance respect previous implementations.

Stopwords implementation

The next step taken was the attempt to improve the preprocessing phase extracting stopwords. As it was expected, MSE increased to 4.83 and r^2 dropped to 0.366. It was supposed that this behavior is mainly because BERT and BETO were trained with fully constructed sentences, and extracting some words would make the model to develop worse so this implementation is not included in next steps.

SEAT vocab implementation

At this point, it was noticed that it was a hard task to improve the model drastically but something came to mind. What happen to words that are not within BETO vocabulary? Looking inside BETO vocabulary file, there were some tokens called [unused] which goes from [unused0] to [unused970]. So the next step taken was to included the 971 most repeated words not included in the original vocabulary file giving a result of MSE equals to 4.81 and r^2 equals to 0.394. Even though the MSE increased this implementation was included because r^2 increased from 0.381 to 0.394 which could be considered an improvement.

Custom Loss Function implementation

An important concept to take care of in Figure 11 is that the estimated resolution times under the exponential curve are the most critical ones given the fact that the model is predicting a very low time while it should have been much higher. In order to penalize more the predicted under the curve a custom loss function will be used instead of the standard Mean Square Error function. The main problem about the loss function is that it must be derivable because of the gradient descent.

The idea is to penalize all negative errors using the most similar function to a step function known, the sigmoid function. To achieve this special penalization the following loss function is proposed, where y belongs to the real value and y' to the predicted value.:

$$cMSE = \frac{1}{N}\sum_{k=0}^{N}(y_k - y'_k)^2 + \frac{(|y_k - y'_k|)^n}{1 + e^{y_k - y'_k}} \tag{3}$$

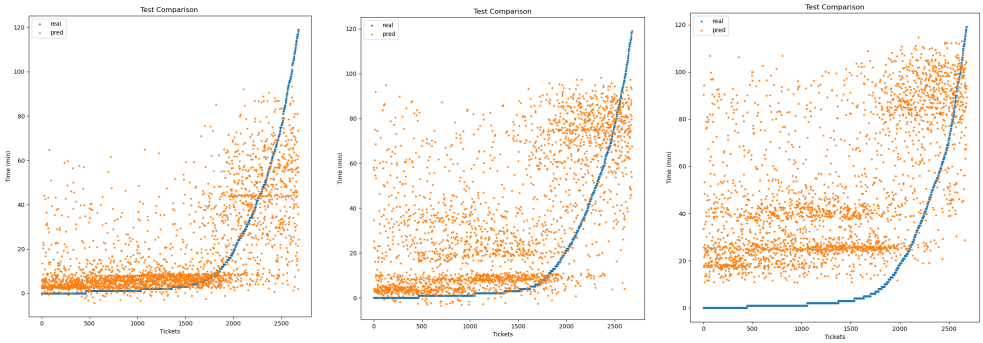
Analyzing the above formula, if $y - y$ results in a negative value the second addend tends to be $(y_k - y'_k)^n$ provoking the negative error formula to be $(y_k - y'_k)^2 + (y_k - y'_k)^n$ a way bigger than the positive errors where the second addend tends to be 0 and its error formula to be $(y_k - y'_k)^2$. The second addend uses the sigmoid function which does not change abruptly from 1 to 0 between negative and positive values giving a percentage of the second addend. Approximately, sigmoid function gives the following relation:

$$sigmoid(x) = \begin{cases} 1 & \text{if } x < -5 \\ 0 & \text{if } x > 5 \end{cases} \tag{4}$$

In consequence, the escalation also produces an extra penalization of errors in the range -50 minutes $<$ error $<$ +50 minutes.

In addition, the exponent n is the value to be set in order to penalize the negative error over the rest of errors. Eventually this will produce the model to create an offset to the prediction.

The problem found about this implementation is that the loss is no more comparable with the previous ones and r^2 gets negative values which could be interpreted as 0, hence the model performs worse, or not?



(a) n=2 (b) n=3 (c) n=4

Figure 17: n effect on the model prediction

As seen in the above figure, by increasing the value of n the model adapts its prediction to perform better on higher values sacrificing the accuracy in fewer values.

In conclusion, this implementation does not give any numerical improvement basing the criteria on the MSE and r^2 indicators but gives a conceptual improvement when referring to the errors under the curve mentioned previously. For this reason, the resource of creating a custom loss function for an specific task could set an important milestone in the model development, but for now this implementation wont be included.

Extra Features implementation

The main advantage of deep learning respect classical machine learning techniques, is the possibility of mixing different types of data giving different priorities. In this implementation the goal is to mix NLP with extra supervised features. The features selected were basically the ones which were considered more relevant to this problem such as Impact, Priority, Time Class, problem opened hour and problem opened month. To implement the extra features it is also necessary to scale the data in order to not giving priority and let this job be done by the network.

This implementation could be easily done by adding an extra input with the extra features array. In addition, in contemplation of mixing the processed text and these numerical data, a simple Dense layer is included. The following code sums up this implementation:

```
1
2  # Load model config
3  model_config = BertConfig.from_json_file(MODEL_CONFIG_PATH)
4  model_config.output_hidden_states = True # Enable all BERT Layers access.
5
6  # Load the Transformers BERT model
7  transformer_model = TFBertModel.from_pretrained(MODEL_PATH,config = model_config,
       from_pt=True)
8  transformer_model.trainable = True # False fixes the weights.
9  bert = transformer_model.layers[0]
10
11 # Building model Inputs
12 input_ids = Input(shape=(MAX_LEN), name='input_ids', dtype='int32')
13 token_type_ids = Input(shape=(MAX_LEN), name='token_type_ids', dtype='int32')
14 attention_mask = Input(shape=(MAX_LEN), name='attention_mask', dtype='int32')
```

```
extra_features = Input(shape=extra_features_shape,name='extra_features',dtype='
    float32')

# Current model inputs
inputs= {'input_ids': input_ids,'token_type_ids':token_type_ids, 'attention_mask'
    : attention_mask,'extra_features': extra_features}

# BERT model inputs, used as layer
bert_inputs = {'input_ids': input_ids,
        'token_type_ids':token_type_ids,
        'attention_mask': attention_mask}


hidden_layers = bert(bert_inputs)[2]
hidden_layers = (hidden_layers[2],
                 hidden_layers[3],
                 hidden_layers[4],
                 hidden_layers[-1])
hidden_layers = tensorflow.convert_to_tensor(list(hidden_layers))


# hidden layers shape is (layer_dimension, batch_dimension, sentence_dimension,
    word_dimesion)
# The permutation is necessary to have batch_dimension in the first position in
    order to perform the AveragePooling.
x = tensorflow.keras.backend.permute_dimensions(hidden_layers,(1,0,2,3))

# Average through the sentence_dimension
x = AveragePooling2D(pool_size=(max_len,1),
                     data_format='channels_first',
                     name='sentence_smbedding')(x)

# Eliminates the non significant dimension
x = tensorflow.squeeze(x,axis=2,name='squeezer')

to_concat = []
for i in range(x.shape[1]):
  to_concat.append(x[:,i,:])

# Layers concatenation
layers_sentence_embedding = concatenate(to_concat,axis=-1)

# Concatenates text processed and numerical features
x = concatenate([layers_sentence_embedding,extra_features],axis=-1)

# Mix text processed and numerical features
x = Dense(3072+extra_features_shape,activation='relu')(x)

# Output layer
outputs = Dense(units=1,activation='relu',
                name='output_layer',
                kernel_initializer=TruncatedNormal(stddev=model_config.
    initializer_range))(x)

model = Model(inputs=inputs, outputs=outputs, name='BERT_regression')
```

The prediction loss with this implementation dropped to 4.14 and r^2 increased to 0.403. With this significant improvement it is important to highlight implementation of extra features to NLP models.

Dropout Layers implementation

In order to reduce the overfitting, a common implementation is including Dropout Layers after dense layers. Usually by adding this layer the model robustness improves and it is able to generalize the input given while training by omitting some perceptrons. The following code gives an example on how to implement a Dropout Layer.

```
x = bert(bert_inputs)[1] # shape = (None,768), BERT 12th layer [CLS] token
x = Dropout(0.2)(x) # Deactivates 20% of the previous layer neurons randomly
```

However, for this specific task by including Dropout layers the MSE increased drastically to 5.09 and r^2 decreased to 0.36. For this specific task the model worsen when including Dropout layers and any conclusion has been reached while looking for the root cause.

Final results

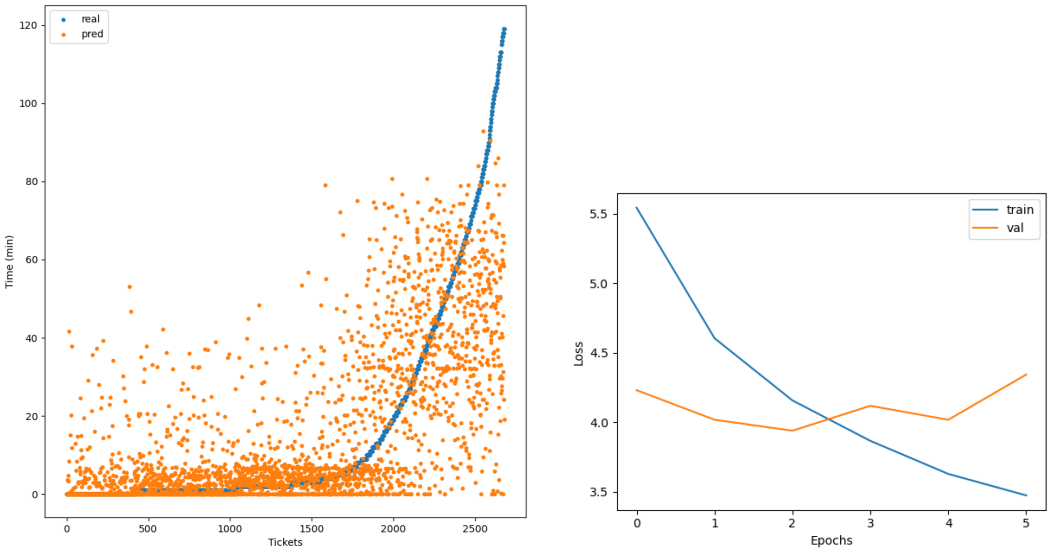
The final result is consisted in implementing all the steps that in a way or another improved the indicators established in the beginning of this section. In the following table it is possible to see a summary with the results of each implementation:

Implementation	MSE	r^2	Implemented
Output scaling	4.99	0.355	Yes
BERT Hidden Layers	4.741	0.381	Yes
Stopwords	4.81	0.366	No
SEAT vocab	4.83	0.394	Yes
Custom Loss Function	NA	NA	No
Extra Features	4.14	0.403	Yes
Dropout Layers	5.09	0.368	No

Table 17: RTP Model implementations results summary.

In respect to the model loss represented in Figure 18b, overfitting is still present a this point with a well drawn divergence between training set loss and validation set loss. The importance of implementing an EarlyStopping function and restoring the weights which gave the better result for validation, is a turning point to prevent the model to work worse in real cases.

In respect to the model prediction represented in Figure 18a, it is possible to see that the model can differ from low and higher times. However, it is necessary to study how to improve the model in order to reduce the number of points in the graph low-right zone.



(a) RTP Model prediction

(b) RTP Model loss.

Figure 18: RTP Model, final implementation results. Source: self-elaboration.

The following figure shows the final model architecture which not only have common used Tensorflow layers such as Dense and AveraPooling Layers, if not it also includes tensor operators from Tensorflow backend, for instance concatenations and squeezing operations. It also reflexes the two branches corresponded to text and numerical processing showing this specific deep learning resource of mixing very different types of data.

The architecture also exhibits the possibility of using large pretrained models basically as simple layers and creating custom models to perform an specific task.

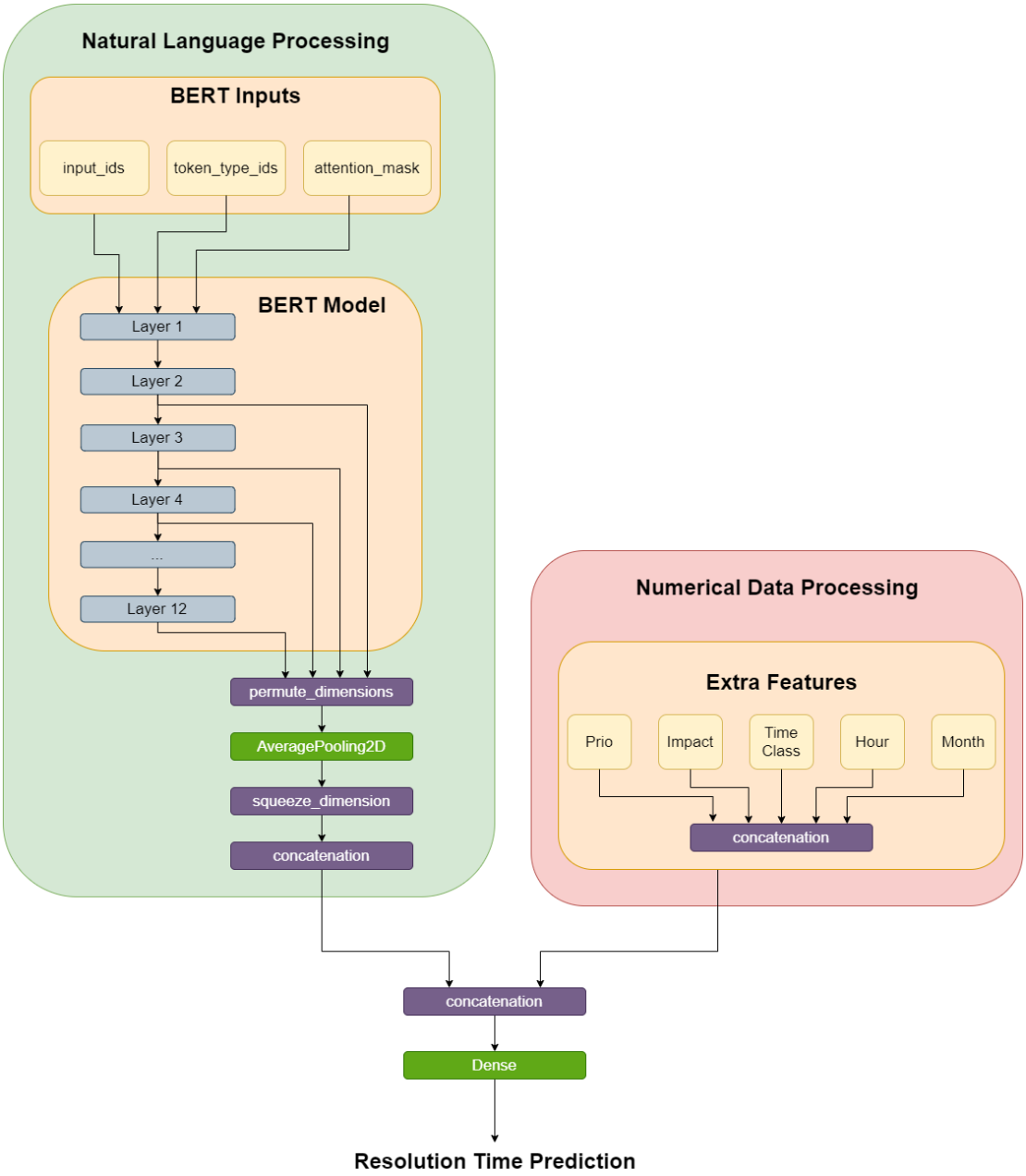


Figure 19: RTP Model architecture. Source: self-elaboration.

Finally, dealing with overfitting have been the main impediment to succeed in this prediction model. Despite this, the model can actually differentiate intermediate and low resolution times but higher ones needs extra data or a custom loss functions as proposed to perform better. Consequently, further investigation is needed.

3.4 Wrong Impact Assignment

The aim of this model is predict if a problem will affect a larger group of users than indicated before or during the event.

3.4.1 Dataset distribution

For this task, only fields Title and Description will be used to create this NLP model to predict the Impact. Furthermore, given the fact that the Impact is sensitive information the histogram is given without specifying the quantity of each class.

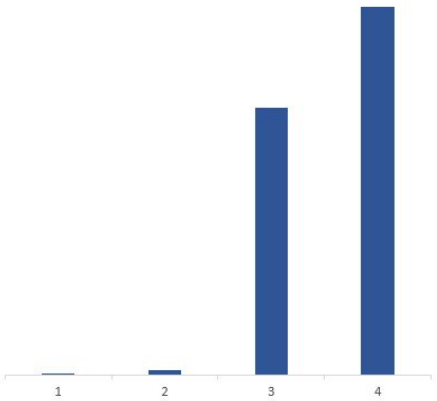


Figure 20: Wrong Impact Assignment dataset distribution

In respect to the dataset, in order to have an equally distributed number of samples per class, random samples will be selected from the higher classes to have the same size as Impact 1 class.

3.4.2 Solution Implementation

The scope of this model is to detect a problem or group of problems that will have an impact actually higher than the indicated when it was created. In order to achieve this, the idea is to mix both supervised and unsupervised learning as mentioned previously. Impact is the parameter that indicates in a range 1 to 4 how many users are affected by the same problem, then it makes sense to create a simple multi-class classification model to predict in which category a problem would fall but also makes sense to use clustering as a unsupervised technique given that fact that the closer the points are the similar the problem different users are having.

Despite in this project the goal is to introduce BERT model, deep learning is far from being the first, easiest and better solution. Deep learning requires a deep investigation background to start a predicting model using this branch of machine learning. Once the BERT model was explained and applied in previous section, in this section will only be mentioned that using BERT the result for this classification task was 62.80% while using classical machine learning techniques such the ones explained in section 2.1 the accuracy is around 65.23%. Consequently, for this application the ML solution will be used while the DL technique does not overtakes ML.

An important assumption taken for this application is that in average all suppliers indicate correctly if the problem is affecting the correct amount of users.

First of all, comparing the indicated Impact with the predicted Impact:

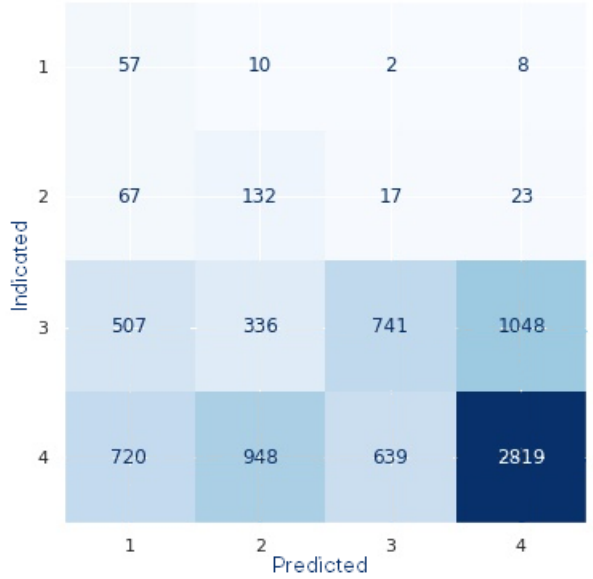


Figure 21: Comparison between indicated and predicted Impact. Source: self-elaboration

This matrix reflexes the indicated and predicted impact in a certain period of time. To understand what this matrix means, it is as easy as looking at a sigle position, for instance at row 4 and column 1, the 720 value means that suppliers are indicating that there are 720 classified as Impact 4 but the classification model predicts that it should have been Impact 1.

An important concept to mention about this subset is that tickets in field **Ticket Type** with value different from 'Incident' and in field **Master/Child** with value 'Child' were excluded. On the one hand, values different from Incident were excluded because services request are very common in an application Life-cycle and were not necessary for this application. On the other hand, Child were excluded because they are very probable to be wrong assigned given the fact that only the Master ticket fields are usually updated in order to have a traceback about the incident.

All the tickets shown in the Figure 21 actually corresponds to many different applications. In order to show the most relevant applications, it is selected the n group of applications with the most number of errors comparing the indicated and predicted impact getting a list similar to the following one:

Application	nº of errors
Application 1	367
Application 2	352
Application 3	283
Application 4	277
Application 5	260
Application 6	236
Application 7	228
Application 8	199
Application 9	196
Application 10	193
...	...

Table 18: Number of errors per application.

The next step needed in order to perform the clustering is the vectorization of the text. The algorithm commonly used to do this task is TfidfVectorizer from scikit-learn[8] library, giving an output of shape (7640,9299) where the first dimension corresponds to each ticket and the second dimension to the vector representation of each one with a size of 9299 axis. Thus, to plot each ticket it is necessary to reduce the dimensionality to 2 axis, there are many algorithms to perform this but in this application t-SNE is used, also from scikit-learn[8].

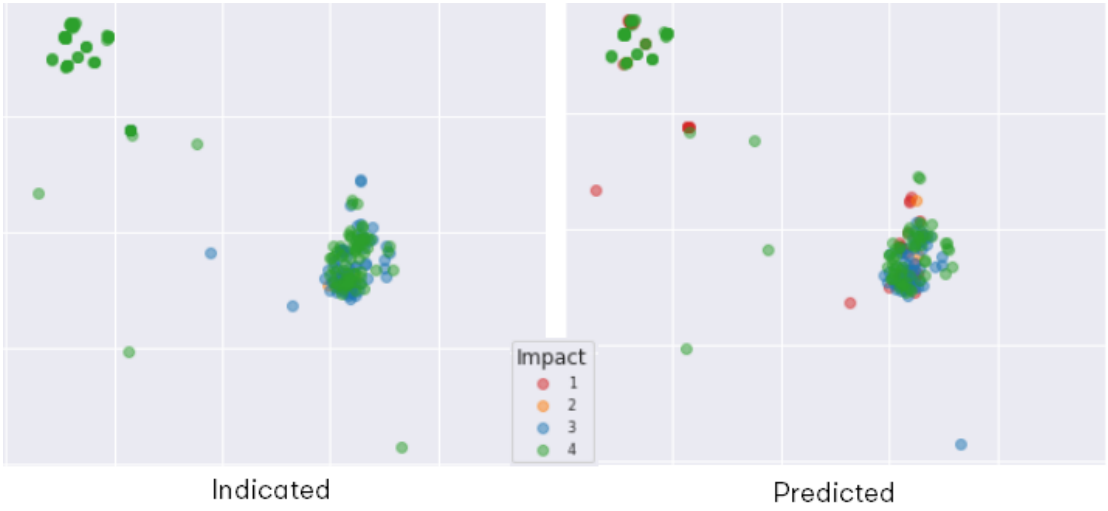


Figure 22: Graphical comparison between indicated and predicted Impact, good practice. Source: self-elaboration.

In the above figure, each point represents a ticket created by a different user of a single application, this implicates that the closer two points are the similar they are as mentioned before. What is more, a common case when documenting the problems the users are having in a phone call, the operator usually copy/paste the same description in order to save time and eventually this produces tickets to be exactly the same producing darker points in this figure.

Analysing the distribution, note that clusters are basically the same color except from lonely points and some possible errors caused by the model accuracy. Thus, the supplier seems to react

fast and document the Impact properly. The cases which wanted to be detected are the clusters where there are many different wrong assigned impact that could be seen even at first glance.

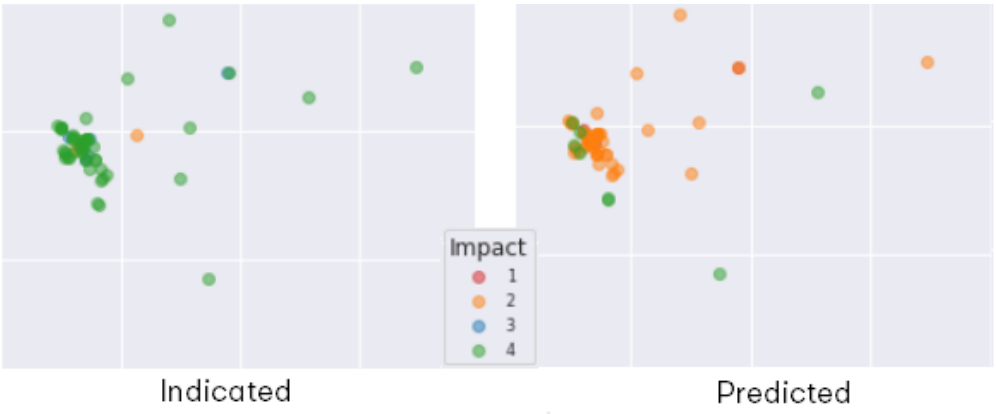


Figure 23: Graphical comparison between indicated and predicted Impact, bad practice. Source: self-elaboration.

The above figure shows the exactly behavior that wanted to be detected, which corresponds to tickets 2d representation of a single application in a period of time. Till this moment only supervised learning has been used, but in order to mix with unsupervised learning the algorithm DBSCAN will be used. Applying this algorithm to the 2d representation of tickets without taking in account the impact the following distribution is given:

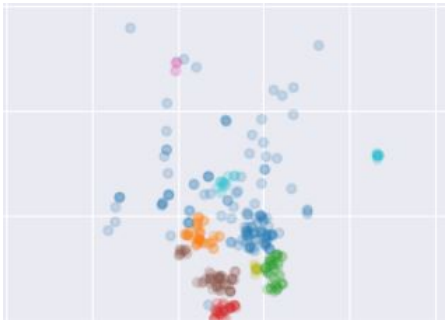


Figure 24: DBSCAN clustering classification. Source: self-elaboration.

The above figure shows the DBSCAN classification, this algorithm receives a parameter called epsilon which could be adjusted to what is considered a 'similar problem' by its Cartesian distance. Hence, the goal of this is to find similar tickets.

The next step is to clear lonely tickets or very dispersed clusters such as the blue cluster in Figure 24. This cleaning will be done basically setting a standard deviation filter per cluster and in order to avoid clusters with only one or few tickets, it will only be considered clusters with at least 5 components as set in requirement specifications. The following figure shows the result once this filter is applied:

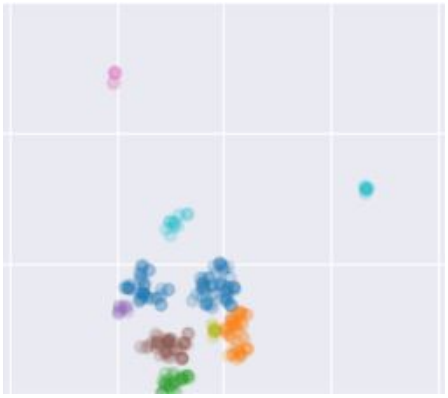


Figure 25: DBSCAN clustering classification with standard deviation filter. Source: self-elaboration.

The step is mixing clustering and classical classification algorithms. The idea is to analyse for each application every single cluster and compare this cluster's confusion matrix with the prediction model confusion matrix. Then to decide if this cluster is suspicious of probably having a higher impact in the future, the error percentage of each class will be compared. To understand this concept, the following example is given:

(a) Indicated (b) Predicted (c) Clustered

Figure 26: Comparison between indicated, predicted and clustered impact. Source: self-elaboration.

In the above figure, after the filter phase, it was detected two clusters indicated in Figure 26c. In the example, it will be focusing in the blue cluster. Then this cluster presents 22 tickets that have been considered similar, by DBSCAN, which has the following confusion matrix where row means indicated Impact and columns the predicted one:

1	0	0	0	0
2	0	3	0	0
3	0	1	0	0
4	3	15	0	0
	1	**2**	**3**	**4**

Table 19: Blue cluster confusion matrix from Figure 26c

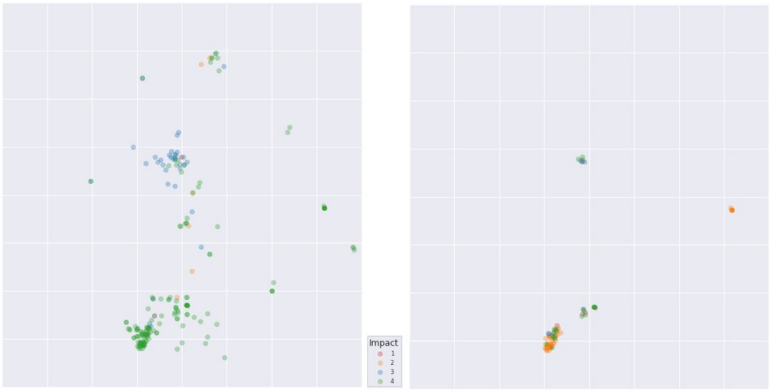
The next step is to normalize this confusion matrix in order to compare with the model confusion matrix. Here M corresponds to the model confusion matrix and C to the individual cluster confusion matrix.

$$M = \begin{bmatrix} 0.763 & 0.169 & 0.062 & 0.078 \\ 0.163 & 0.657 & 0.050 & 0.128 \\ 0.024 & 0.053 & 0.643 & 0.241 \\ 0.049 & 0.119 & 0.243 & 0.551 \end{bmatrix}$$

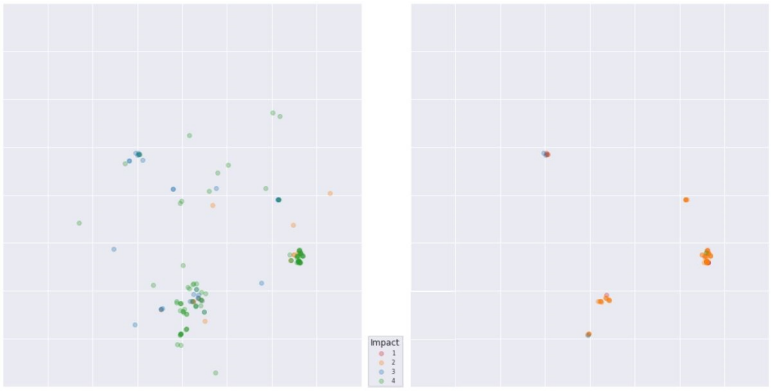
$$C = \begin{bmatrix} 0. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0.167 & 0.833 & 0. & 0. \end{bmatrix}$$

Once having the C confusion matrix normalized, the next step is to compare M_{ij} and C_{ij} where $i < j$. This under-diagonal condition is due to the fact that predicting a higher impact than the indicated does not suppose any dilemma for this application.

Thus, as $C_{42} >> M_{42}$ this cluster should not be discarded because the model probability to be wrong is 11.9% when predicting Impact 2 while it should have been actually Impact 4 and 83.3% where predicted with Impact 2. Despite this, a second indication is that 16.7% of the total where predicted as Impact 1 while the probability of error is 4.9% for this Impact combination. This procedure is automatically done for each application cluster.



(a) Application 1



(b) Application 2

Figure 27: Suspicious tickets of becoming higher in Impact. Source: self-elaboration.

The above figure shows the final result of suspicious tickets of becoming higher in Impact. However, because of requirements it is needed to create an Excel sheet where it shows the most relevant ones. In order to do this, a system of punctuation was created which follows the next formula:

$$points = \frac{1}{N} \sum_{k=0}^{N} I_k - I'_k \tag{5}$$

In Equation 5, I refers to the indicated Impact and I' to the predicted Impact. The subtraction makes higher differences being more relevant and also taking into account if there are a significant number of tickets with a higher prediction than indication. In addition, dividing by the number of components the cluster is formed, avoids highlighting a cluster with 30 errors in a group of 100 over a cluster with 15 errors in a group of 20.

Application	Cluster	Points	Size
Application 1	0	3	6
Application 2	1	3	6
Application 3	0	3	6
Application 4	0	2.93	31
Application 5	1	2.88	9
Application 6	0	2.57	7
Application 7	1	2.33	9
Application 8	1	2.16	6
Application 9	0	2.14	7
Application 10	0	2.13	15

Table 20: Top 10 groups of tickets suspicious of becoming higher in Impact.

Finally, commenting two different cases given in the output file, the first one shows a similar problem different users are having with Application 2, occurring in a short period of time. Because of confidentiality terms, more information about tickets can not be shown.

Application	Impact	Impact Pred	Title	Open Time
Application 2	4	1	File recovery	13:59:54
Application 2	4	1	File recovery	13:54:57
Application 2	4	1	File recovery	13:50:04
Application 2	4	1	File recovery	13:45:40
Application 2	4	1	File recovery	13:41:14
Application 2	4	1	File recovery	13:36:56

Table 21: Application 2 example of tickets suspicious of having a higher impact in the future.

Another case could be the shown in the following table where the same problem is happening but in two consecutive days. This case could reflect problems that even though they are locally solved, they could occur periodically every day and should be considered to study the root cause of this problem.

Application	Impact	Impact Pred	Title	Open Time
Application 7	3	1	Unable to open Application X	15:21:00
Application 7	4	1	Unable to open Application X	10:55:46
Application 7	4	1	Unable to open Application X	10:01:12
Application 7	4	1	Unable to open Application X	16:52:58
Application 7	3	1	Unable to open Application X	8:20:00
Application 7	4	1	Unable to open Application X	15:25:37
Application 7	4	1	Unable to open Application X	11:53:53
Application 7	3	1	Unable to open Application X	8:33:41

Table 22: Application 7 example of tickets suspicious of having a higher impact in the future.

4 Budget

This project consisted mainly in research and development, hence the budget is adjusted to these two tasks, regarding licenses and tools used for the development, most of resources were Open Source and free community licenses.

The resulting costs based on the above explanations are generally the minimum price per hour for an internship student set by the EEBE, as 8€/h for the 2020/2021 university season. Thus, the budget of this project is:

Hours (h)	**Price per hours (€/h)**	**Total (€)**
540	8	4320

Table 23: Project budget

However, it is important to highlight that the task related to D06 in Table 5 has caused an important reduction in cost/time due to the fact that another solution would have been implementing the model in AWS instead of locally. In order to have the same conditions of development as locally, training model without any restriction and during 6 months, the following table reflexes the cost difference:

	Cost €
Local P5000 16Gb GPU	1727.27
AWS 16Gb GPU	4711.84
Difference	+2984.57

Table 24: Cost difference between implementing models locally and in AWS.

5 Future development

In respect to the RTP model, there are different implementations or improvements that could be done in the future. Firstly, studying deeply the effect of scaling the output data and which range to select could facilitate relationship understanding to the model and having a great impact to the results. Secondly, the implementation of a custom loss function could help the model to predict higher resolution times despite of sacrificing precision as mentioned previously. Besides, the importance of including extra features can improve the model drastically as seen, for instance by including extra features as text length or the technical support team backlog of tickets . Finally, there are overfitting prevention techniques not applied such as L2 and L1 regularization which could also be considered.

In respect to the WIA model, by improving the classification accuracy using classical machine learning techniques or deep learning could give a better approach and certainty to the prediction.

To close this section, it is necessary to mention that there are other models architectures based in transformers that must be considered in future implementations such as GPT-3 model which at the moment this project was developed it was not open source but it seems to have promising results.

6 Conclusion

In conclusion, the goal of getting a deep knowledge in NLP State of the Art techniques and introducing the acquired knowledge to the department have been accomplished successfully. However, due to this project range the degree of affectation wont be seen still the application is completely integrated. In spite of this, a regression model based in NLP is not a problem usually studied so the model developed in this project could be considered unique in its category setting a turning point to AI applications based in this concept.

Furthermore, as it could be seen through this project development, deep learning disposes of innumerable resources to improve accuracy/precision in respect to machine learning, in spite of requiring a much higher number of samples compared to the second option. Hence, having a much better DL model than a ML model also requires much more effort.

Once the above concepts are said, in almost every NLP application it is always recommended to have a Minimum Viable Product (MVP) using machine learning but finally improve the prediction model using advanced techniques as proposed in the Impact prediction. Thus, deep learning is far from being the first, easier, and best solution in a first approach to a problem. It is very important to mention that perfectionism plays a very important role while training deep learning models and it has to be taken into account and preventing this to become the antagonist.

Finally, the journey through Industrial Electronics and Automatics Engineering has shown the hardware part of the current industrial revolution where artificial intelligence role becomes bigger in a very short period of time. Hence, with the acquired knowledge in this project the capacity of merging both worlds has become an important turning point in an engineer career.

References

[1] Anup Bhande. *What is underfitting and overfitting in machine learning and how to deal with it.* URL: https://medium.com. (accessed: October 2020).

[2] Edward Loper Bird Steven and Ewan Klein. "Nltk: The natural language toolkit". In: (2009).

[3] José Cañete et al. "Spanish Pre-Trained BERT Model and Evaluation Data". In: *PML4DC at ICLR 2020*. (accessed: January 2021).

[4] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[5] itdxer. *What is batch size in neural network?* URL: https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network. (accessed: January 2021).

[6] Emil Likke Jensen. *Multi-Label, Multi-Class Text Classification with BERT, Transformers and Keras*. URL: https://towardsdatascience.com. (accessed: November 2020).

[7] pawangfg. *Explanation of BERT Model – NLP*. URL: https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/. (accessed: January 2021).

[8] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[9] Akshay Prakash. *BERT: Bidirectional Encoder Representations from Transformers*. URL: https://medium.com. (accessed: January 2021).

[10] Lavanya Shukla. *Designing your neural networks*. URL: https://www.kdnuggets.com/2019/11/designing-neural-networks.html. (accessed: January 2021).

[11] Simplilearn. *Knuth: Computers and Typesetting*. URL: https://www.simplilearn.com/what-is-perceptron-tutorial. (accessed: January 2021).

[12] TensorFlow. *GPU support*. URL: https://www.tensorflow.org/install/gpu. (accessed: October 2020).

[13] Jesse Vig. *Deconstructing BERT: Distilling 6 Patterns from 100 Million Parameters*. URL: https://towardsdatascience.com. (accessed: October 2020).

[14] Han Xiao. *bert-as-service*. URL: https://github.com/hanxiao/bert-as-service. (accessed: January 2021).