NOVA IMS
Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Text Mining Techniques for Car Price Prediction

Ricardo Miguel Galvão Gonçalves

Project Work presented as partial requirement for obtaining the Master's degree in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

TEXT MINING TECHNIQUES FOR CAR PRICE PREDICTION

by

Ricardo Miguel Galvão Gonçalves

Project Work presented as partial requirement for obtaining the Master's degree in Advanced Analytics

Advisor / Co Advisor: Roberto André Pereira Henriques, Phd

September 2021

ABSTRACT

Modern data sources routinely contain information both in unstructured and structured forms, combining text with the usual numerical and categorical data. For instance, in websites dedicated for selling and buying cars the listings typically include a textual description of the car. Others also include a detailed list of numerical or categorical attributes, such as the total number of kilometers the car has, or it´s model.

In this work project we apply text mining techniques to create predictors for car price regression from unstructured data, the textual description in car listings. Two different types of predictors were studied, the tf-idf features obtained from the n-gram count matrix, or the singular vectors derived from the decomposition of the tf-idf matrix.

In this work we also examine the performance of reducing the vocabulary dimension by applying stemming, lemmatization or not applying either of those. We also compare the effects of creating the initial n-gram count matrix with only unigrams, unigrams and bigrams or only bigrams.

Our regression experiment shows that Support Vector Regression performs best at car price prediction using text data as predictors with $R^2 = 0.77$, MSE = 0.19 and MAE = 0.32. These results can be seen as respectable given the complex nature of the task.

KEYWORDS

Text Mining; Regression Analysis; Car Price Prediction.

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

BOW	Bag-of-words
MAE	Mean Absolute Error
MLP	Multi-layer Perceptron
MSE	Mean Squared Error
RFM	Recency, Frequency, Monetary Value
SVR	Support Vector Regression
TF-IDF	Term Frequency – Inverse Document Frequency
VSM	Vector Space Model

1. INTRODUCTION

As computer networks become the backbones of science and economy enormous quantities of machine-readable documents become available. There are estimates that 85% of business information lives in the form of text (Hotho, Nürnberger, and Paaß 2005), proving to be an invaluable data source.

However, texts written in natural language are an unstructured data source that is hard for machines to understand. They can be easily interpreted by a human, but it is still a complex task for computers to derive meaning from this data. Nevertheless, computers offer an important advantage over human capabilities: computing power, meaning they can find patterns, which are non-trivial recurrences, within data faster and more accurately than the human eye. Using automated text mining algorithms to discover knowledge from natural language texts, referred to as Natural Language Processing (Hotho et al. 2005), provides numerous challenges but also offers unique possibilities.

The era of Web 2.0 is witnessing the proliferation of online social media platforms and marketplaces, in which business models are developed by leveraging user generated content (Ngo-Ye and Sinha 2014). In these online marketplaces products are usually presented by images, attributes in a somewhat structured form and unstructured description texts.

Attributes of products are usually used to predict market prices. Knowing the exact market price is extremely useful when dealing with more expensive products that require a high investment, such as cars or houses. Online car markets provide a platform where sellers can present their cars by images, structured attributes like mileage and fuel type, and description texts.

Due to the individuality of cars, predicting one's price is a very difficult challenge. (Gegic et al. 2019) and (Pudaruth 2014) applied machine learning techniques to overcome this challenge, but in their studies only the structured attributes were used in the models. Because car prices often depend on car's individual conditions and configuration, sellers add user generated vehicle description texts to provide additional information about them. These texts are an important information source to give more price transparency and distinctiveness.

This work project will focus on the analysis of those user generated texts collected through web scraping of an online car market. Applying text mining techniques to help transform the unstructured data into variables capable of forecasting car prices and analyzing how effective those models are in doing so.

Two ways of creating predictors will be examined in this work, one is using the tf-idf scores obtained from the n-gram count matrix, the other is using the top singular vectors derived from the tf-idf score matrix. We will also compare the effects on regression performance of three types of pre-processing: stemming, lemmatization or not applying either of those. Finally, the length of the n-grams considered and it´s effects on performance will also be compared, between only using unigrams, unigrams and bigrams combined or only using bigrams.

The main goal of this work project is to understand to what extent can textual descriptions of cars accurately predict their price. Secondly, exploring and analysing the question of text pre-processing and feature extraction from text. Lastly, to explore the existing contributions related to the addressed topics, deepening our theoretical knowledge.

The first section of this project work addresses the past research done in the area of text regression. First exploring the efforts made in text pre-processing and feature representation and then looking at various types of text regression problems, as well as the different regression models used. Next, in the "Methodology" section, we present the methodology used in this work, describing the way the data was collected, processed, and analysed. Describing also the modelling process used to create the predictors and the regression models studied. Then, in the "Results and Discussion" section, we present the results of our regression models comparing their performances. Finally, in the last two sections we discuss our conclusions as well as the limitations and recommendations of this project work.

2. LITERATURE REVIEW

2.1. TEXT MINING

Text Mining or knowledge discovery from text (KDT) deals with the machine supported analysis of text (Hotho et al. 2005). The main goal of Text Mining is to disclose the concealed information by means of methods that can cope with the large number of words and structures in natural languages but also that allow to handle vagueness and uncertainty.

According to (Hotho et al. 2005) there are three main research areas related to Text Mining:

- Information Retrieval, which is the activity of finding information resources (usually documents) from a collection of unstructured data sets that satisfies the information need. It is mostly focused on facilitating information access rather than analyzing information (Allahyari et al. 2017).

- Natural Language processing, a sub-field of computer science, artificial intelligence and linguistics which aims at the understanding of natural language using computers.

- Information Extraction, the task of automatically extracting information or facts from unstructured or semi-structured documents. It usually serves as a starting point for other text mining algorithms (Allahyari et al. 2017).

With an ever-increasing share of human interaction, communication and culture being recorded as digital texts, the information encoded in text is a rich complement to the more structured kinds of data traditionally used in research. This suggests that text mining can be of high practical use and is able to fill the gap that conventional data mining has created in the discovery of knowledge in databases.

Conducting analysis on text can be done using a variety of approaches. This depends on the type of insight that is required. In their work, (Hotho et al. 2005) highlight 3 data mining methods that can be applied to text:

- Classification, which aims at assigning pre-defined classes to text documents. The most common classifiers used are Naïve Bayes, Nearest Neighbor, Decision Trees, Support Vector Machines and Kernel Methods.

- Clustering, which can be used to find groups of documents with similar content. Objects in the same cluster should be similar and dissimilar to documents of other clusters.

- Information extraction, which was previously explained. Can also be used for text summarization effects.

Adding to the methods described, in (Gentzkow, Kelly, and Taddy 2019) the authors provide and overview of methods for analyzing text and in all the cases they consider, the analysis consists of three steps:

1. Represent raw text *D* as a numerical array **C;**
2. Map **C** to predicted values $\widehat{\boldsymbol{V}}$ of unknown outcomes **V**; and
3. Use $\widehat{\boldsymbol{V}}$ in subsequent descriptive or causal analysis.

In this overview, some methods for predicting a numerical attribute $\boldsymbol{v_i}$ from counts $\boldsymbol{c_i}$, which is called text regression, are presented. This will be the focus of this work.

2.1.1. Text Preprocessing

Textual data is not structured as neatly as the more common types of data used in Machine Learning, making it a much more complex task to find useful information and patterns in the data in an automated way. The most important way it differs from other kinds of data is that text in inherently high dimensional (Gentzkow et al. 2019).

This means that the preprocessing step is crucial for the performance at the task at hand. Usually, the first step in the preprocessing task is the tokenization of the text. Tokenization consists in splitting a document into a stream of words by removing all punctuation marks and by replacing tabs and other non-text characters by single white spaces (Hotho et al. 2005). In (Kogan et al. 2009) lowercasing was also used, a common step in tokenization. (Foster, Liberman, and Stine 2013) also replaced rare words by an invariant “unknown” token.

In order to reduce the dimensionality of the documents in the corpus, the set of words/tokens describing those documents can be reduced by filtering and lemmatization or stemming methods.

Filtering is done on documents to remove some of the words. A common filtering is stop-words removal. Stop-words are words that frequently appear in the text without having much content information (Allahyari et al. 2017), like conjunctions or prepositions. Lemmatization methods try to map verb forms to the infinite tense and nouns to the singular form. However, in order to achieve this, the part of speech of every word in the text document has to be assigned (Hotho et al. 2005).

Stemming methods try to build the basic form of words, i.e. strip the plural ‘s’ from nouns, the ‘ing’ from verbs or other affixes. A stem is a natural group of words with equal meaning (Hotho et al. 2005).

Even tough the preprocessing stage may have noticeable influence on the success of the task at hand, the type of preprocessing done that produces the best results is problem dependent. For instance, a study by (Pak and Gunal 2017) on the impact of text representation and preprocessing on Turkish author identification with two different classification algorithms, Multinomial Naïve Bayes (MNB) and Sequential Minimal Optimization (SMO) , found that disabling stop-word removal and stemming achieved better performance than other combinations for MNB. For SMO enabling just stemming achieved the highest F-Score.

2.1.2. Feature Representation

In the literature, the most common way unstructured text is represented is by a vector space model. In VSM, the values of the elements are derived from event frequencies, such as the number of times a certain word appears in a particular document (Ngo-Ye and Sinha 2014).

The Bag-of-words (BOW) model is a specific type of vector space model. In BOW the order of the words is ignored, $\boldsymbol{c_i}$ is a vector whose length is equal to the number of words in the vocabulary and whose elements $\boldsymbol{c_{ij}}$ are the number of times word ***j*** appears in document ***i.*** This scheme can be extended to encode a limited amount of dependence by counting unique phrases rather than unique words. A phrase length of ***n*** is referred to as an *n-gram* (Gentzkow et al. 2019).

Another standard way of term weighting is the Term Frequency – Inverse Document Frequency (TF-IDF) score. This improves on normal Term Frequency weighting because it decreases the weight of terms occurring more frequently in the document collection, making sure the matching of documents is more affected by distinctive words which have relatively low frequencies in the collection (Allahyari et al. 2017).

The TF-IDF score is calculated as follows: $tf_{ij} \times idf_j$, where tf_{ij} is the term frequency for word j in document i and inverse document frequency (idf_j) is the log of one over the share of documents containing j : log(n/dj), where n is the total number of documents (Gentzkow et al. 2019).

Nassirtoussi et al. (2014), in their review of text mining for market prediction, concluded that there are, at least, 5 types of scores commonly used for representing features as numeric values: Information Gain, Chi-Square statistics, Document Frequency, Accuracy Balanced and TF-IDF.

Foster et al. (2013) proposed a new algorithm for converting text into numerical regressors. The method consists of 3 steps:

1. Convert the source text into lists of word types. A word type is a unique sequence of non-blank characters. Word types are not distinguished by meaning or use.

2. Compute matrices that (a) count the of times that word types appear within each document and (b) count the number of times that word types are found adjacent to each other.

3. Compute truncated singular value decompositions of the resulting matrices of counts. The leading singular vectors of these decompositions are the regressors.

Besides the normal feature representations (Kogan et al. 2009) also used a score called LOG1P**:** $\log\big(1 + freq(xj; d)\big)$, rather than normalizing word like TF, this score dampens them with a logarithm.

Liang et al. (2017) expand on the conventional methods of text feature extraction and outline some frequently used deep learning methods for text feature extraction. The first method described is the autoencoder, which is a feedforward network that can learn a compressed, distributed representation of data, usually with the goal of dimensionality reduction or manifold learning. An autoencoder usually contains one hidden layer between the input and the output layer. Also described is a stacked autoencoder, the deep counterpart of the autoencoder. It can be built simply by stacking layers. For every layer, its input is the learned representation of the former layer.

In their experiments it was compared the effect of three types of feature extraction methods – principal component analysis, a shallow sparse autoencoder and a deep sparse autoencoder – for pattern recognition. The proposed method of a deep sparse autoencoder enabled higher recognition accuracy, proving that the use of deep learning for text mining can make significant achievements.

2.2. REGRESSION

Predicting texts to a certain class can be highly useful and it can separate large collections of text into relevant groups. However, in some cases it can be more suitable to predict a specific value and not a global range. This type of prediction can be done using regression.

2.2.1. Linear Regression

The basic regression model is the linear regression. The aim of linear regression is to study the effect of one or more factors on a quantitative variable target. The theoretical model is: $y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_k x_{ki} + u_k$, where y_i is the target variable observed for individual *i*, β_i is the parameter associated with predictor variable *k*, x_{ki} is the predictor variable *k* for observed individual *i* and u_i is the error for individual *i* (Lehmann 2020). Usually the unknown parameters in linear regression are learned by minimizing the sum of squared errors (Nassirtoussi et al. 2014).

The most popular strategy for high-dimensional regression in contemporary statistics and machine learning is the estimation of penalized linear models, particularly with L1 penalization. For simple text regression tasks with input dimension on the same order as the sample size, penalized linear models typically perform close to the frontier in terms of out-of-sample prediction (Gentzkow et al. 2019). The most common cost functions are Lasso (L1 penalty), Ridge (L2 penalty) and Elastic net (mix of L1 and L2).

Lasso is a penalty regression with a quadratic loss function that introduces a penalty term associated with SSR (Guo et al. 2020). The cost function for Lasso regression can be written as: $\sum_{i=1}^{M}(y_i - \hat{y_i})^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j * x_{ij}\right)^2 + \lambda \sum_{j=0}^{p} |w_j|$. This type of regularization can lead to zero coefficients, i.e. some of the features are completely neglected for the evaluation of the output (Castelli et al. 2020)

Ridge regression is another penalty regression with a quadratic regularizer that inserts another penalty term into the original SSR term. The cost function is: $\sum_{i=1}^{M}(y_i - \hat{y_i})^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j * x_{ij}\right)^2 + \lambda \sum_{j=0}^{p} w_j^2$ and it minimizes the sum of squared residuals and the the λ* the slope^2 (Castelli et al. 2020). The parameter λ can range from 0 to positive infinity, increasing it will promote a smaller slope. Ridge regression can solve for parameters when there isn´t enough data samples.

Elastic net is a mixture of both L1 and L2 penalization where the penalty term, the one that is multiplied by the parameter λ, is given by: $\sum_{j=1}^{p}\left(\frac{1}{2}(1-\alpha)B_j^2 + \alpha|B_j|\right)$ where $\alpha \in (0,1)$ determines the trade-off between L1 and L2 regularization (Joshi et al. 2010).

2.2.2. Nonlinear Regression

However, linear models are limiting: text regression problems will often involve complex interactions between textual inputs, thus requiring a non-linear approach to properly capture such phenomena (Bitvai and Cohn 2015).

2.2.2.1. Regression Trees

Regression trees have become one of the most popular approaches for incorporating multi way predictor interactions into regression. A tree "grows" by sequentially sorting data observations into bins based on the value of the predictor variables. This partitions the data set into rectangular regions, and forms predictions as the average value of the outcome variable within each partition. This structure is an effective way to accommodate rich interactions and nonlinear dependencies (Gentzkow et al. 2019). Two extensions of the simple regression tree have been highly successful thanks to clever regularization approaches that minimize the need for tuning and avoid overfitting: Random Forests and Boosted Trees.

Random Forests, developed by Breiman, are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution as all trees in the forest (Breiman, 2001). The algorithm can be described as follows:

- From the training set create *t* bootstrap samples
- For the split of each node, in each of the base learners (Decision Trees), *x* attributes are randomly chosen out of *X* (with *x <= X*)
- Then an attribute out of *x* is chosen to split the node
- Each tree is grown to the largest extent possible
- Calculate the performance from out-of-bag observations

Boosted regression trees combine the strengths of regression trees and boosting, an adaptive method for improving model accuracy based on the idea that it is easier to find and average many rough rules of thumb, than to find a single, highly accurate prediction. These type of trees incorporate important advantages of tree-based methods, handling different types of predictor variables and accommodating missing data (Elith, Leathwick, and Hastie 2008).

2.2.2.2. Support Vector Regression

Although support vector machines are mainly used for classification tasks, they are also used for regression problems. The support vector regression model is trained by solving the following optimization problem (Kogan et al. 2009): $\min_{w \in R^d} \frac{1}{2}\left|\left|w\right|\right|^2 + \frac{C}{N}\sum_{i=1}^{N} \max(0, |v_i - f(d_i, w)| - \varepsilon)$, where *C* is a regularization constant, ε controls the training error. The training algorithm finds to weights *w* that define a parameterized function of the documents *d* and those weights to optimize the value of a continuous variable v.

2.2.2.3. Deep Learning

Various other machine learning techniques have been applied to text regression. The most common one in the deep learning area are neural networks, which typically allow the inputs to act on the response through one or more layers of interacting nonlinear basis functions. A main attraction of neural networks is their status as universal approximators, a theoretical result describing their ability to mimic general, smooth nonlinear associations (Gentzkow et al. 2019).

2.3. PREVIOUS WORK ON TEXT REGRESSION

The majority of work in the literature regarding text regression is related to stock market prediction using textual data from news websites or financial reports. (Nassirtoussi et al. 2014) provide an extensive overview of the literature related to this subject.

(Kogan et al. 2009) apply well known regression techniques to a large corpus of freely available financial reports, constructing regression models for volatility, which is the financial risk of investing in that company, for the period following a report. For those predictions, the models are trained using a support vector regression and the performance was reported using the mean squared error between the predicted and the true log-volatilities. In their results it´s reported that the models that used only text to predict volatility came very close to the historical baseline in some years. A text only model (LOG1P with bigrams) comes within 5% of the error of a strong baseline. A combined model, with text and historical data, improves substantially over the baseline in four out of six years (2003-2006) and this difference is robust to the representation used.

(Lerman et al. 2008) use computational linguistics to automatically predict the impact of news on public perception. This work uses the 2004 US Presidential election markets from Iowa Electronic Markets and the goal of the prediction system is to forecast the price prediction for the next day (up or down). The system operates in an iterative fashion, on each day the news for that day are used to construct a new instance. A logistic regression is trained on all previous days and the resulting classifier predicts the price movement of the new instance.

As far as feature representation they used BOW; news focus features, which represents differences between days of news coverage, the resulting value captures change is focus on day *t*, where a value greater than 0 means increased focus and a value less than 0 decreased focus; entity features, which are conjunctions of a certain word and the entity, defined a priori; and dependency features, which were extracted from dependency parses of the news articles, to capture dependency interactions.

In each market, the baseline news system makes a small profit, but the overall performance of the combined system is worse than the market history system alone, showing that the news baseline is ineffective. However, all news features improve over the market history system which means that the news information helps to explain market behavior.

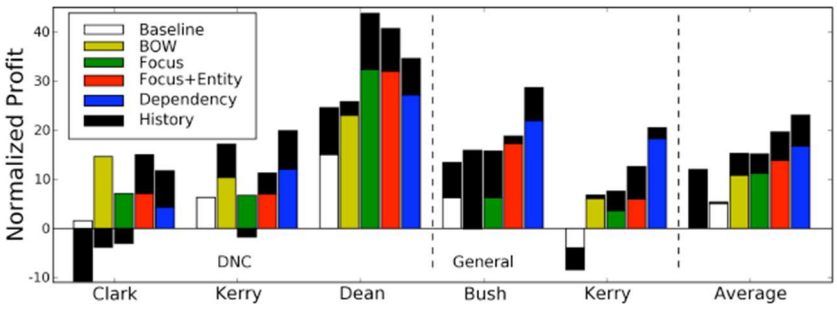

Figure 1- Normalized Profit for each candidate in different systems in (Lerman et al. 2008)

(Jin et al. 2013) present a system which mines news articles and makes forecasts about the movement of foreign exchange currency markets. The system uses a combination of language models, topic clustering and sentiment analysis to identify relevant news articles. These articles along with the historical stock index and currency exchange values are used in a linear regression model to make forecasts.

A language model was developed that classifies the incoming news articles as relevant or not relevant. Using the Latent Dirichlet Allocation model they classify these news articles into 30 topics and obtain each article's topic distribution. Then, the top topics are identified by manually aligning news articles with currency fluctuations, which are then identified as relevant topics. In order to classify incoming news articles, they estimate the topic distribution of each article and then decide whether its most prevalent topics fall into the set of relevant topics identified earlier.

According to the authors the system was able to forecast most of the studied appreciations and depreciations. For instance, the year 2012 saw the Brazilian Real's exchange rate significantly altered due to government interventions. On May 21st, the system predicted correctly that the BRL would continue depreciating as per trends from the previous weeks. However, on May 22nd, as the Brazilian government was intervening to reverse Real's fall, the system was able to correctly forecast the reversal of the currency movement.

Another topic of text regression approached in the literature is house price prediction. (Foster et al. 2013) convert text, obtained from house descriptions in real estate listings, and convert it into numerical regressors by exploiting methods from computational linguistics. The features are built using the algorithm previously explained and for modelling the response, which in this case is the log of the listed prices because there was some skewness in the property prices, they train a simple linear regression model.

A baseline model was provided for comparison by simply regressing *y* on the word counts in *W*, the matrix of counts, of the most common 2000 words. This model produces a fit of $\bar{R}^2$= 0,681. The next model uses regressors created from the SVD of the matrix *W*. A regression of log prices on the 100 leading singular vectors attains $\bar{R}^2$=0,49. Adding more singular vectors produces statistically significant, though diminishing improvements. With 500 singular vectors it produces $\bar{R}^2$=0,61.

Most of the main leading singular vectors are significant with an increasing proportion of insignificant variables as the position in the decomposition increases. According to the authors these singular values

are more consistently predictive with less noise in comparison to the significance for the coefficients of the word indicators. A third regression uses features derived from the SVD of the bigram matrix *B*, showing a slightly better performance than the previous.

This study is concluded by suggesting that the introduction of nonlinearity in these models could improve the results, also a brief analysis on the use of trigrams shows that they offer modest gains, albeit at a nontrivial increase in computation.

(Guo et al. 2020) study the performance of some machine learning algorithms associated with text mining from internet data in predicting house prices in China. To search for all possible representations of housing prices and consequently, obtain all possible keywords that are directly and indirectly related to housing prices through two channels: one is extracted from Baidu, a Chinese search engine, to display the preferences of ordinary Internet users – for example, buyers and sellers of real estate; while another is obtained by crawling the CNKI, the largest Chinese full-text database, covering academic journals in order to show the concept of housing prices from the viewpoint of scholars and experts. Through this text mining process, they obtained 29 keywords which were classified into 4 groups based on economic viewpoints: Macro-policies, local attributes, housing market characteristics and housing costs.

Economic aspects	**Keywords**
Macro policies (7)	Urbanization, rail transportation, real estate policy, pension fund, macro control, monetary policy, inflation
Local attributes (6)	Shanghai`s second-hand house, house web, house, house price, rental house and school district house
Housing market characteristics (9)	Second-hand house, second-hand web, housing price, housing frenzies, rising prices, price/income ratio, house, rent house web
Housing costs (7)	Housing fee, housing tax, mortgage calculator, mortgage interest, down payment, property tax, decoration

Table 1- List of keywords obtained in (Guo et al. 2020)

The machine learning models used were Generalized Linear Regression, Elastic Net Regression and Random Forests. The performance of the models are as follows:

Model / Index	Generalized Regression	Random Forest	Elastic Net
MSE	0.1021	0.0190	0.122
R^2	0.91	0.98	0.90

Table 2- Performance of the diferent models used in (Guo et al. 2020)

The authors conclude that this method, especially random forest not only detects turning points, but also offers prediction ability that clearly outperforms traditional regression analysis.

Another text regression task found in the literature is predicting movie revenues through critic reviews. (Joshi et al. 2010) used the text of film critics reviews from several sources to predict opening weekend revenue. Their data was gathered for movies released in 2005-2009. For those movies, metadata was obtained and a list of hyperlinks to movie reviews by crawling MetaCritic. The metadata retrieved includes the name of the movie, its production house, the set of genres it belongs to, the scriptwriter(s), the primary actors starring, running time and its MPAA rating. The reviews were scraped from the seven most frequent websites on MetaCritic.

Two response variables were considered, the total revenue on opening weekend and the per screen revenue. Both predictions were evaluated using mean absolute error and Pearson´s correlation between the actual and predicted value. A penalized linear regression, the elastic net model, was used to predict the response variables. As far as text features, the authors used n-grams of length 1,2 and 3, part-of-speech n-grams, obtained through the Stanford POS tagger and lastly dependency relations.

In these experiments three types of predictors were compared, predictors based on metadata, which was used as a baseline, predictors based on text and predictors that use both kinds of information. They reported that features from critic's reviews by themselves improve correlation on both response variables, however improvement in MAE is only observed for the per screen revenue prediction task. A combination of the meta and text features achieves the best performance in terms of MAE and Pearson's correlation. While the text only models have some high negative weight features, the combined models do not have any negatively weighted features and only very few metadata features, which leads to the conclusion that text features from pre-release reviews can substitute for and improve over a strong metadata-based opening weekend revenue prediction.

(Mishne and Glance 2006) studied whether applying sentiment analysis methods to weblog data results in better correlation than volume of discussion only. They analyzed the sentiment expressed in weblogs towards movies both before the movie's release and after, and test whether this sentiment correlates with the movie's box office information better than a simple count of the number of references in weblog does.

The authors show that, in the domain of movies, there is good correlation between references to movies in weblog posts – both before and after their release—and the movie's financial success. Furthermore, they demonstrate that shallow usage of sentiment analysis can improve this correlation. Specifically, the number of positive references correlates better than raw counts in the pre-release period. In of itself, the correlation between pre-release sentiment and sales is not high enough to suggest building a predictive model for sales based on sentiment alone. However, it is proposed that sentiment might be effectively used in predictive models for sales in conjunction with additional factors, such as movie genres and season.

Even tough the majority of text regression problems have been tackled using linear models or machine learning algorithms, (Bitvai and Cohn 2015) propose a nonlinear method based on a deep convolutional neural network to predict the future box-office takings of movies based on reviews by movie critics and attributes.

The model operates over unigrams, bigrams and trigrams. They use word embeddings to represent words in a low dimensional space, a convolutional network with max-pooling to represent documents in terms of n-grams, and several fully connected hidden layers to allow for learning of complex nonlinear interactions. Including nonlinearities is crucial for accurate modelling. A method for quantifying the effect of text n-grams on the prediction output is also shown. This allows for identification of the most important textual inputs and investigation of nonlinear interactions between these words and phrases in different data instances.

The network is trained with stochastic gradient descent and the Ada Delta update rule using random restarts. Stochastic gradient descent is noisier than batch training due to a local estimation of the gradient, but it can start converging much faster. Ada Delta keeps and exponentially decaying history of gradients and updates in order to adapt the learning rate for each parameter. Regularization and hyperparameter tuning were performed by early stopping on the development set. The outline of the convolutional network is shown:

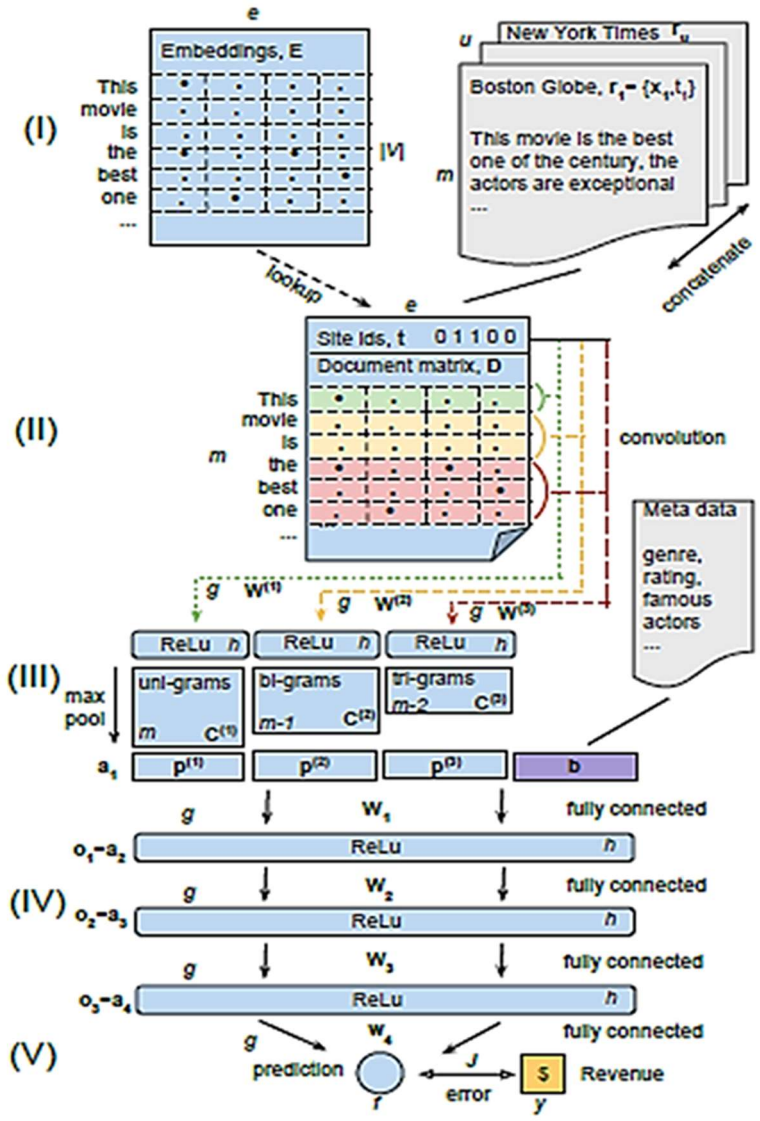

Figure 2- Outline of the CNN in (Bitvai and Cohn 2015)

The results show that the neural network performs very well, with around 40% improvement over the previous best (Joshi et al. 2010). Nonlinearities are clearly helpful as evidence by the ANN text model beating the BOW linear text model with a mean absolute test error of 6.0 vs 8.0.

(Ngo-Ye and Sinha 2014) develop and compare several text regression models for predicting the helpfulness of online reviews. The authors adopt a raw number of positive helpful votes of a review as the measure of helpfulness, the target variable. A new hybrid model was proposed, which incorporates

both the vector space model representation of review text and a reviewer's engagement profile. Three research question were addressed:

1. Examine if the VSM representation of review text improves the prediction of review helpfulness over a baseline model
2. Examine if the hybrid model is better than using VSM alone for the prediction of review helpfulness
3. Examine whether the hybrid model is better than using a reviewer's engagement profile alone

The conceptual models developed are presented in the following table:

Model type	Predictor variables
ZeroR model	NA
BOW Full model	All the words from review text collection
Dimension Reduced BOW model (BOW')	A subset of review words selected through applying CfsBF dimension reduction technique
RFM model	Reviewer's RFM dimensions (Recency, Frequency, and Monetary Value)
BOW' + RFM model	CfsBF selected review words + reviewer's RFM dimensions

Table 3- Conceptual models developed in (Ngo-Ye and Sinha 2014)

The ZeroR model always predicts the mean of the target variable, calculated based on prior observations, and it was used as a baseline.

The results are reported using CFS (correlation-based feature selection) for dimension reduction and support vector regression for predicting review helpfulness. The basic rationale for CFS is that the desired attribute subset includes attributes that are highly correlated with the target variable but, at the same time, have low correlations among the attributes themselves.

Based on the results, for all 32 scenarios for Yelp and Amazon – across all index weighting schemes and regression performance measures – BOW' + RFM models always perform better than BOW only models. The hybrid model is superior to the BOW only model in predicting review helpfulness. So, the implication for online platforms is that not only textual content is important in predicting review helpfulness but also that the reviewer's RFM dimensions matter.

In this literature review we first outline the 3 main research areas related with Text Mining. Then, we explored the problems of text pre-processing touching on the biggest problem associated with text, it´s inherently high dimensions. After that, some feature representation models were studied, with the most commonly used in the literature being the BOW model. Next, a review of the literature on text regression was made, briefly describing the mainly used regression models and then a review describing practical case studies was done. It was noted that the majority of the work in text regression was related to stock market prediction, (Nassirtoussi et al. 2014), (Kogan et al. 2009). Another thing to note in the literature is that when the authors used the option of a combined model with both features

derived from text and the usual numerical and categorical features it improved the results in some problems, such as in (Joshi et al. 2010) when trying to predict movie revenues. Also Kogan et al. (2009) show that a combined model, with text and historical data, improves substantially over the baseline. Ngo-Ye and Sinha (2014) also conclude that in predicting online reviews helpfulness a hybrid model considering both textual content and the reviewers RFM dimensions is superior to the BOW only model.

3. METHODOLOGY

This project was conducted following the flowchart depicted below:

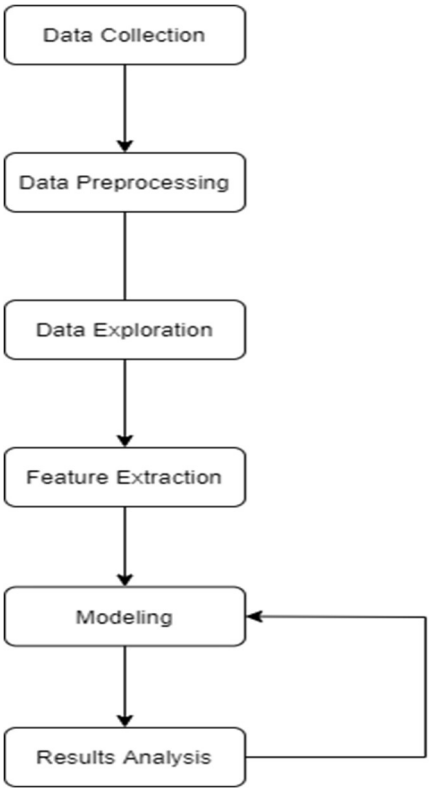

Figure 3 – Project Flowchart

The first step was to build the web scraper which would be used for collecting the necessary data for this project. The data was then pre-processed in order to make it viable for analysis. However, the data collection step and its pre-processing were intertwined given the fact that, to increase the used pre-processing tools vocabulary and increase their efficiency in normalizing the text, the more data collected the better the performance.

Next, a series of data exploration methods were used to extract information about the price value distribution, the number of words in the vocabulary and the ad length relation to price. Following that, a series of text regression models were developed, analyzing their performance results, and consequently improving them.

Each step will be further explained in the following chapters and subchapters.

3.1. DATA COLLECTION

A web scraper was written in *Python* with the help of the packages *BeautifulSoup* and *requests* (script in appendix 8.1) for the data collection process. The information was retrieved from the website*: www.olx.pt/carros-motos-e-barcos/carros/?search%5Bprivate_business%5D=private*. From every ad it would be collected the title, which usually contained the text that indicated the brand and the model of the car being sold, the price (which is the dependent variable studied in this work) and, after being redirected to that ad-specific page, the text description written by the seller and the model of the car was also retrieved from the specific location in the page, this was made to guarantee to always have the information of the car model because in some cases it wasn´t directly included in the title text. Some ads would redirect to the website: *www.standvirtual.com* but the information retrieved was identical since the structure of both websites was similar.

Every scrape collected between 970 and 975 ads, the process took an average time of one hour, given that to avoid overloading the website with requests and getting blocked, it had programmatically random sleep times (ScrapeHero 2020).

After every scrape, a *csv* file was created with 4 columns: title, model, desc and price. A total of 57 csv files were used for the final dataset.

3.2. DATA PREPARATION

The data preprocessing step is of extreme importance for this study, considering that the collected data is unstructured and generated by humans makes it prone to containing numerous errors and inconsistencies. Ensuring that the data used was as consistent as possible was the main goal of the preprocessing stage.

The initial dataframe, obtained after merging all csv files together in a single one using the language *Python* and the *pandas* library, contained 54573 rows of raw data. However, due to how long an ad stays posted on the website and because it can be reposted, it is normal that the web scraper would collect the same ad more than once. The first step in the preprocessing stage was to remove all duplicate ads. After that, an analysis on the data types of the dataframe showed that all columns, including "*price*", were of type *object*. So, transformations were made on the column to convert it to numeric type, and also filter out the observations that were labeled as "Troca" since what we are trying to predict is a numeric value. Subsequently, a brief exploration on the data found that some of the instances were about selling the car for its parts because it was no longer a working vehicle. These ads were predominantly at a cheaper price point than working cars, so a cut-off price of 1000€ was chosen to filter out these instances.

The next step consisted of merging all the text information contained in the 3 columns (*title, model* and *desc*) into one called *Text,* which will be used for price prediction. Before merging, the text in *title* and *model* was lowercased and striped of accents, given that its human generated it is best to normalize prior to merging so that information isn´t repeated due to human error on writing the words. The new column was created with the following rule: if *model* was already in *title* then *text* would be the combination of only the columns *title* and *desc;* and if *model* wasn´t in *title* then *text* would be the combination of all 3 columns.

The *text* column was then processed with the following steps:

- Lowercase all text;
- Remove accents;
- Remove the large white spaces at the beginning and end of the strings;
- Remove the special characters "\r" and "\n";
- Remove all punctuation signs;
- Remove white spaces with length of 2 or more in the middle of the strings and replacing them by a single white space;
- Add a white space to separate alphabetic characters and numeric characters, i.e.: tokens like "100km" get split into "100" and "km";
- Remove stop-words, common words that add no descriptive value. The stop word list used was the one provided by the *nltk.corpus* Portuguese stop words list;
- Replace and correct words with *word_replacer* dictionary;
- Correct words identified as wrong by the spell checker object with its suggested correction.

Considering that the only information used in this work for price prediction is the tokens in the *text* column, it is logical that observations with scarce information or, in other words, an insufficient number of words would not be helpful for the performance of the models. For this reason, another column was added to dataframe which contained the number of words in *text* corresponding to that instance. Every observation with a word count smaller than 15 was filtered out.

3.2.1. Word Correction and Vocabulary Building

The biggest problem in the preprocessing stage was the large number of badly written words causing the data to be very inconsistent. This may lead to having the same word, written in two different ways (one incorrectly and one correctly), having different impacts on the predictions.

The first step to address this problem was the use of the *SpellChecker* library, importing a spellchecker object with the predefined Portuguese vocabulary within that library. The default spell checker object uses a Levenshtein Distance algorithm to find permutations within an edit distance of 2 from the original word. It compares all permutations (insertions, deletions and substitutions) to known words in a word frequency list used by the spellchecker. The words that are found more often in the frequency list are more likely the correct results (Barrus 2018).

The Levenshtein distance is a measure of the similarity between two strings, the source string (*s*) and the target string (*t*). The distance is the number of deletions, insertions, or substitutions required to

transform *s* into *t.* The greater the Levenshtein distance, the more different the strings are. This metric is also sometimes referred as edit distance (Gilleland n.d.)

The algorithm is described as follows:

1.
    a. A) Set *n* to be the length of *s*
    b. Set *m* to be the length of *t*
    c. If *n*=0, return *m* and exit
    d. If *m*=0, return *n* and exit
    e. Construct a matrix containing 0..*m* and 0..*n* columns
2.
    a. Initialize the first row to 0..*n*
    b. Initialize the first column to 0..*m*
3. Examine each character of *s* (*i* from 1 to *n*)
4. Examine each character of *t* (*j* from 1 to *m*)
5.
    a. If s[*i*] equals t[j], the cost is 0
    b. If s[*i*] doesn't equal t[*j*], the cost is 1.
6. Set cell d[*i,j*] equal to the minimum of:
    a. The cell immediately above plus 1: d[*i-1,j*] + 1
    b. The cell immediately to the left plus 1: d[*I,j-1*] +1
    c. The cell diagonally above and to the left plus the cost: d[*i-1,j-1*] + cost
7. After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[*n,m*]

However, only using this spellchecker object was insufficient, and two problems were identified:

- The predefined vocabulary was quite small and because of that it would often identify a word as badly written when it was not the case. Given the context of this work, the vocabulary used is very specific and contains a high number of case-specific words, such as the car brand and model which sometimes would get identified as badly written words.

- The computing time for edit distances greater than one was infeasible. Consequently, the number of poorly written words identified and corrected in the process was insufficient.

The first solution was to create a word count dictionary, assuming that wrongly written words are less common than correctly written words. To that dictionary, words with a count greater or equal to 5 were added. The spellchecker object has an option to add a dictionary with word counts to its vocabulary, by doing this we are adding more context specific words to the vocabulary.

To further increase its vocabulary, words considered misspelled and with an incorrect proposed solution were also added to a list and then added to the spellchecker vocabulary.

The second problem was solved by creating a dictionary where the keys are the misspelled words, and the values are the corresponding correct way of spelling that specific word. This was done with the help of the *difflib* library which contains a function called *get_close_matches,* this function returns a

list of the best “good enough” matches according to a certain cutoff of the similarity score (Anon n.d.). The similarity score between two strings is calculated as follows: 2*M/T, where T is the total number of elements in both sequences, and M is the number of matches.

After a manual analysis on the list it produces, the incorrect words that differ from the correct spelling by 2 edits or more are added to that dictionary. Along with word correction, the dictionary was also used to standardize some words that could have multiple spellings to only one, i.e.: “quilometros”, ”kilometros”, ”kmts”,etc. were all changed to “km”. It would also separate words incorrectly joined together (whether through input error or the process of preprocessing).

After the preprocessing steps described previously, the initial dataframe containing 54573 rows was reduced to 18403 rows.

3.3. DATA EXPLORATION

In order to get a better perception and understanding about the data in this project, a brief exploratory analysis was made. A visual comprehension of the data helps build essential domain knowledge before the modelling application. It also helps in finding inconsistencies in the data and improving the processing task.

3.3.1. Description text data

For a first analysis, we checked the description text length distribution. In table 4 some statistics are presented related to the description text length (“count” column). As we can see, the word count distribution seems to be somewhat skewed to the right given that the median is smaller than the mean, also there is a big discrepancy between the third quartile value and the max value of 876.

TOTAL NUMBER OF ADS	18043
MEAN AD LENGTH	53.8148
STANDARD DEVIATION	47.3378
MINIMUM AD LENGTH	15
1ST QUARTILE	24
MEDIAN AD LENGTH	38
3RD QUARTILE	65
MAXIMUM AD LENGTH	876

Table 4 – Ad length statistics

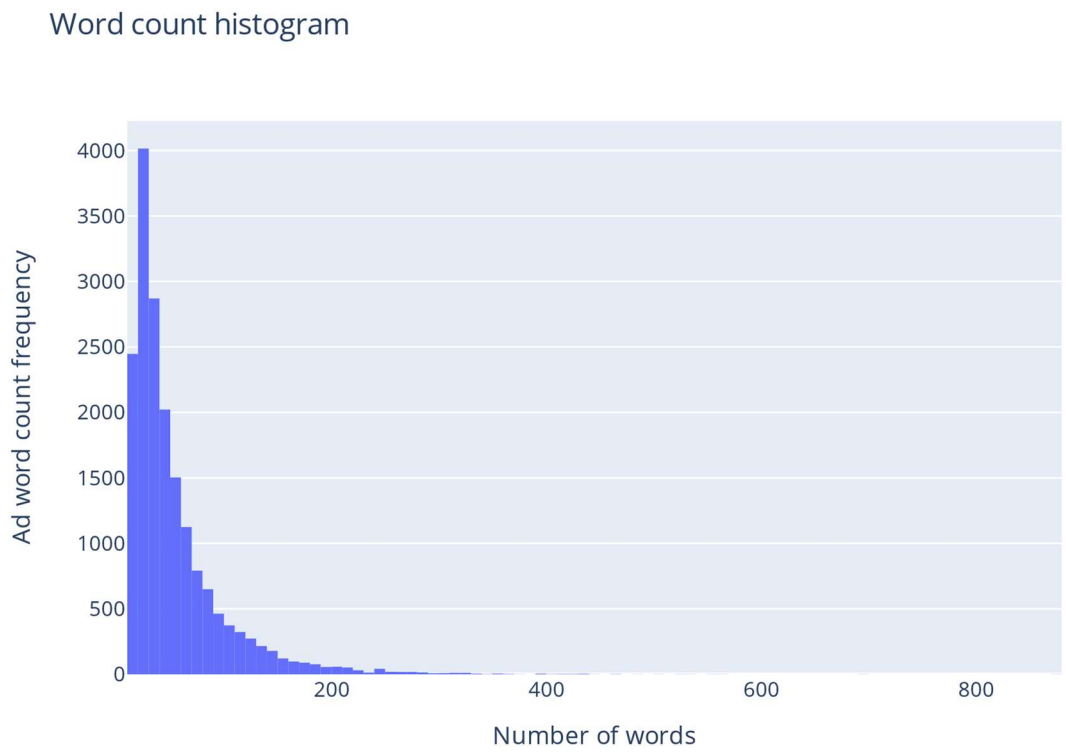

Figure 4- Ad word count histogram

Analyzing figure 4, an histogram whose bins correspond to intervals of length 10,first we can confirm that the ad length distribution is clearly skewed to the right. Additionally, we observe that the most common description text length is between 20 and 29 with a total of 4015 rows out of the total 18043 observations.

As far as vocabulary size, the current vocabulary after the previously described preprocessing steps contains a total of 18440 different words. This is a quite large vocabulary size, however there are a high number of extremely low frequency terms. For instance, there are a total of 6274 different words that only appear once in all the corpora, and a total of 8673 words that have a frequency of 2 or less. Filtering out these low frequency terms will reduce the size of the vocabulary to almost half and it can also help improve the performance of the task at hand. Here the decision was to filter out words that have a frequency of 1, with this the vocabulary contains a total of 12166 unique words.

Another way to reduce vocabulary size is the use of stemming and lemmatization. Applying stemming and removing words with frequency 1 reduces the vocabulary to 8476 unique words. The same procedure but with lemmatization reduces it to 10451 words.

Two wordclouds were also created to get a visual representation of the more frequent words in the vocabulary. The words with a higher frequency are the ones with a bigger size in the wordcloud. Figure 5 only considers unigrams opposed to figure 6 that also considers bigrams in its representation if they have a high enough frequency.

Figure 5 - Wordcloud with only unigrams

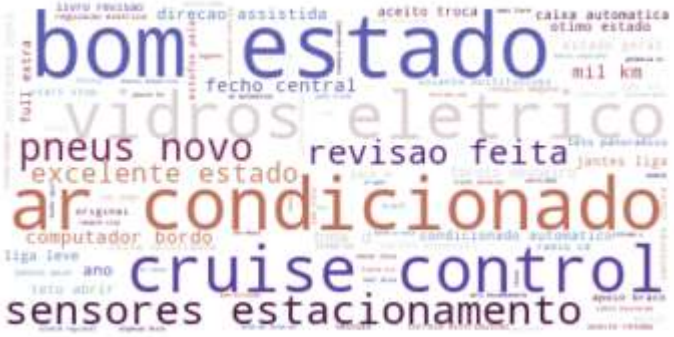

Figure 6 – Wordcloud with bigrams

As we can see, the most emphasized words in figure 5 are “km”, “estado”, “carro” and “novo”, followed by “revisao” and “eletrico”. When bigrams are considered in frequency analysis we see that some bigrams are very prominent in the vocabulary, given that bigrams like “bom estado”, “ar condicionado”, “cruise control” and “vidro eletrico” are represented in figure 6.

3.3.2. Target variable (price) distribution

The next analysis was related to the target variable distribution. In the following table we can see that the price data is very skewed to the right. 75% of the observations have a price of 13500€ or less, however, there is an extreme max value of 3333333€.

TOTAL NUMBER OF ADS	18043
MEAN PRICE	11047.9 €
STANDARD DEVIATION	31742.2 €
MINIMUM PRICE	1000 €
1ST QUARTILE	3500 €
MEDIAN PRICE	7499 €
3RD QUARTILE	13500 €
MAXIMUM PRICE	3333333 €

Table 5- Price Statistics

To get more insight on the distribution of the price variable a boxplot graph was also plotted with a log transformation applied to the target variable because a distribution that is symmetric or nearly so is often easier to handle and interpret than a skewed distribution. The natural log transformation can also improve the regression models prediction capabilities and will be later explored.

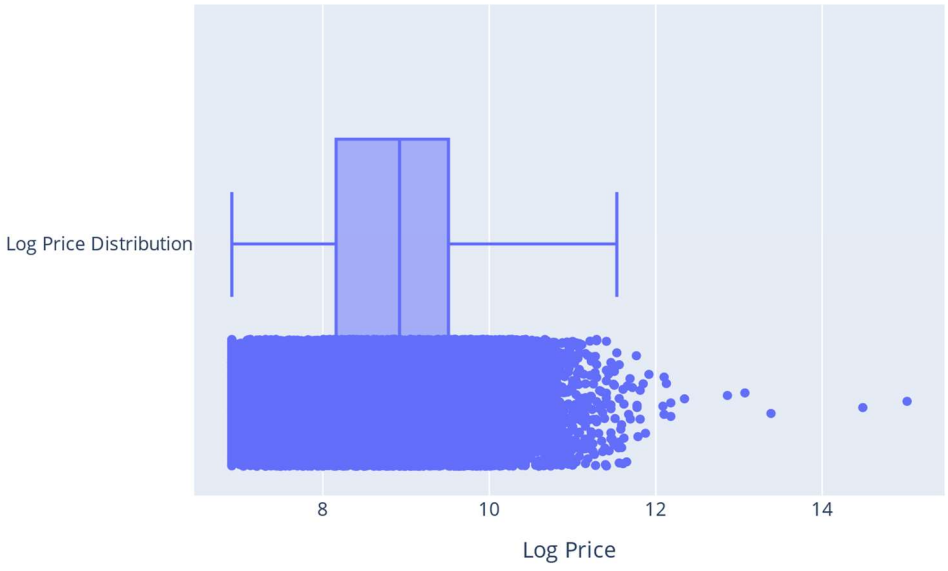

Figure 7- Log Price distribution

In figure 7 it is possible to observe that almost all cars are being sold under 100 000€, which is the upper fence of the boxplot (e^11.5). After that threshold, the data starts to become sparser, and those

values may be identified as outliers. To understand if there are any inconsistencies with those values, we looked at all the rows in the dataframe related to a price of 100 000€ or more.

After checking those observations some errors were identified which can be considered outliers and eliminated from the dataset, such as average cars being listed at incredibly high prices, probably due to human error.

7637	mitsubishi outlander phev	outlander	\n Quli...	3333333.0	mitsubishi outlander phev km 26000 km cilindra...
8111	renault megane dci confort	nan	\n Carr...	650050.0	renault megane dci confort carro imaculado ve ...
8682	fiat punto 2007 multijet 1.3	grande punto	\n Carr...	385000.0	fiat punto 2007 multijet 13 grande punto carro...

Figure 8 – Errors in price listings

In figure 8 we see an example of cars that are normally sold at a much lower price, being listed at a very high price. One of those ("Mitsubishi outlander phev") is even listed at the max price in the whole dataset. The logical conclusion is that these inconsistencies were due to human error in listing the price and the best procedure is to remove them from the dataset.

3.3.3. Price in relation to ad word count

The next analysis made was to understand if the number of words used in the description text, the column "text" in the dataframe, was correlated with the car price. In order words if the ad length was a good predictor of the car price, because it is normal to suppose that the higher priced cars would have a more detailed text description.

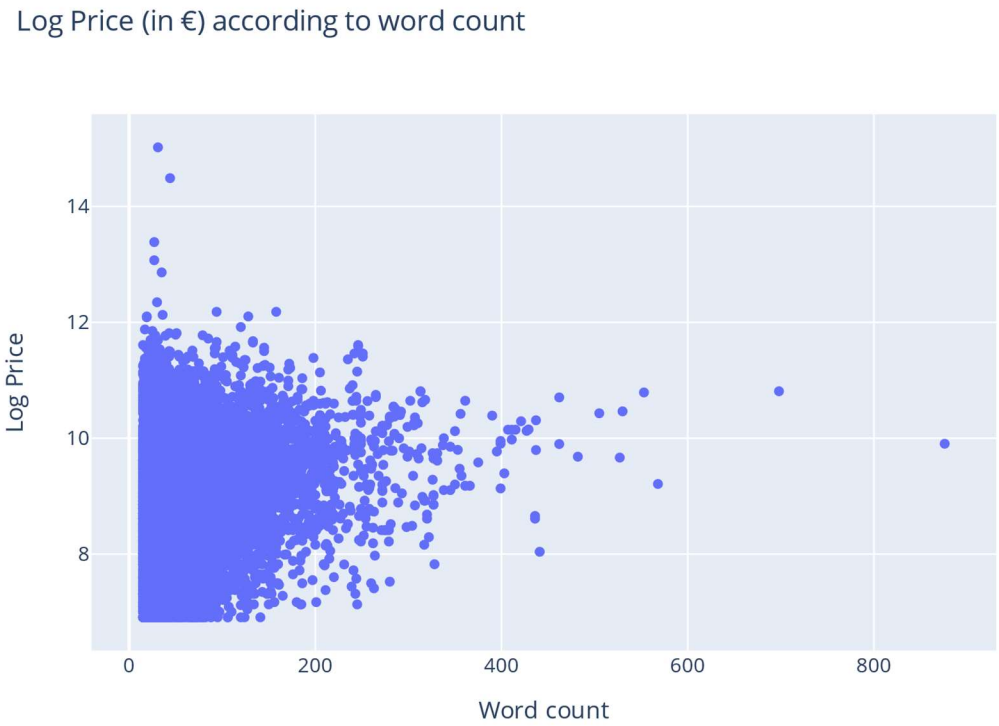

Figure 9- Log Price in relation to ad word count

Figure 9 shows the length of the description text plotted against the log price. As we can see, there seems to be no direct correlation between the number of words used and the price of the car. The scatter plot doesn´t show an upward trend along the x-axis (the number of words) which leads to the conclusion that the description length isn´t a good predictor of the car price.

3.3.4. Price in relation to number of badly written words

Given that the description texts were human created and so, prone to containing errors, another interesting analysis would be to check if the number of badly written words in the description text correlate with the car price. In this case, it would be a negative correlation, the higher the number of badly written words the lower the car price.

For this analysis, a new column was created, “wrong_word_count”, and the pre-processing steps that involved word correction were not applied to the initial dataframe (the use of the word_corrector dictionary and the word correction of the spell checker object), however all the other pre-processing steps were taken. A function was created that would use the spell checker object, with all the extra vocabulary added previously, and every time it would identify a word as being poorly written a +1 was added to a counter. The final value of the counter would the value of the “wrong_word_count” column in the corresponding row.

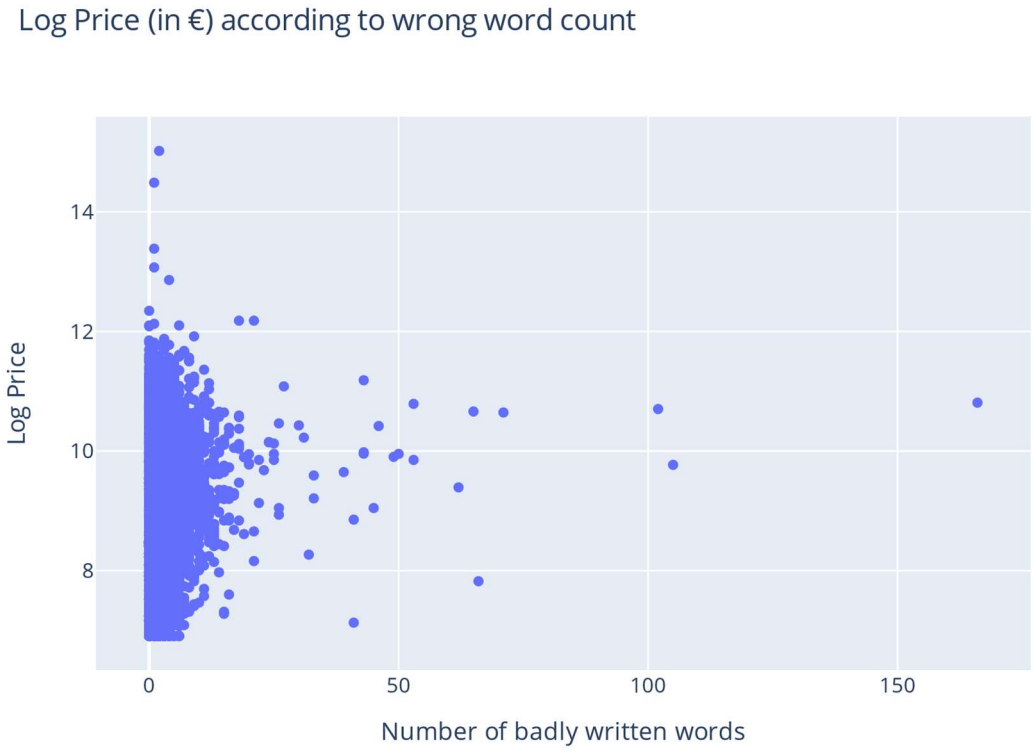

Figure 10- Log Price in relation to number of badly written words

Figure 10 shows no direct correlation between the number of badly written words and the log price, for instance considering an x value of 0 we observe car prices ranging from the minimum value (1000€) to e^12 (around 160 000€). After observing the figure above, we conclude that the number of badly written words is not a good predictor for car price.

3.4. Feature Extraction

Before applying regression models to predict car prices, the text used for the predictions needs to be represented in a way that allows those models to be applied. The chosen representation was the Bag-of-words model, previously explained. In this case, each document is the description text written by the seller and the vocabulary is the total number of unique words in all the description texts.

Using sklearn´s *CountVectorizer* a word count matrix is created, where each row corresponds to a certain ad and the columns are the words in the vocabulary. The elements in the matrix are the number of times a certain word appears in a certain document.

CountVectorizer allows this scheme to be extended to encode a limited amount of dependence with the option *ngram_range,* extracting information about the n-grams up until the maximum defined. For this work, unigrams and bigrams were considered.

Instead of the raw frequencies of occurrence of a token in a given document, one can also use the tf-idf term weighting scheme previously explained. This helps to scale down the impact of tokens that occur very frequently in the corpus and that are hence less informative than features that occur in a small fraction of the training corpus (Anon n.d.). For this project, both raw frequencies and tf-idf scores were used and their performances compared.

3.4.1. Dimensionality Reduction

The pre-processing steps previously described help to reduce the vocabulary dimension, however the word count matrix originated from *CountVectorizer* is still a very sparse matrix. Using all the features from the matrix for a value regression can negatively impact the model performance, seeing that a large part of the predictors have a value of 0 in almost all rows of the matrix.

To address this problem two types of dimensionality reduction were used:

- The first one was to only consider the top n most frequent features from the word count matrix;
- The second one was to apply a dimensionality reduction technique available in *sklearn* called *TruncatedSVD*.

The regular Singular Value Decomposition is a factorization of a real or complex matrix that generalizes the eigen decomposition of a square normal matrix to any $n \times p$ matrix. It is defined as follows: Let **X** be a $n \times p$ data matrix, the singular value decomposition of **X** is **X=UDV´**.

Where **U** is $n \times p$, **D** is $p \times p$ and **V** is $p \times p$ symmetric matrix. The columns of **V** give the eigenvectors of **X´X** matrix and the diagonal values of **D** matrix give the square root of the corresponding eigenvalues of the **X´X** matrix (Mendes 2017).

Truncated SVD is different from regular SVD in that it produces a factorization where the number of columns is equal to the specified truncation. Only the *t* columns vectors of **U** and *t* row vectors of **V´** corresponding to the *t* largest singular values are calculated.

3.5. MODELLING

Before running the regression models the data is split into training and testing sets with 80-20 ratio. In total we have 14429 instances for training and 3607 instances for testing our models' performances.

In this work different types of regression algorithms were tested: *Linear Regression* and it´s variants (*Lasso, Ridge and Elastic Net*), *Support Vector Regression (SVR), Nu Support Vector Regression (NuSVR), Linear Support Vector Regression (LinearSVR), Decision Tree Regressor, Gradient Boosting Regressor, Multi-Layer Perceptron Regressor (MLP Regressor).* All these estimators are available in the SciKit – Learn module (Pedregosa et al. 2011). The effects on the regression performance of applying stemming, lemmatization or none were also compared.

To summarize, the analysis is conducted according to the following steps:

1) Load the pre-processed dataset into a *Pandas* dataframe with two columns (the text column and the price column) and apply one of the following to the text:
    a. Stemming
    b. Lemmatization
    c. Basic (no stemming or lemmatization)
2) Split the dataframe into training set and testing set (80% training, 20% testing)
3) Create n-gram word count matrix considering:
    a. Only unigrams
    b. Unigrams and bigrams
    c. Only bigrams
4) Transform n-gram counts vectors into tf-idf scores
5) For dimensionality reduction apply one of the following:
    a. Keep a maximum of top n-attributes according to term frequency (n=1000,2000,3000)
    b. Compute truncated singular value decompositions of the resulting matrix, keeping the top k singular vectors, with k<n (k=100,200,300,400,500)
6) Fit the resulting training data into an estimator using the SciKit-Learn module. Fitting is done using each regression algorithm
7) Calculate performance indicators on the log price regression using the SciKit-Learn module.

3.5.1. Performance Indicators

To assess the quality of our regression models, we will use the coefficient of determination, denoted R^2, the Mean Squared Error (MSE) and the Mean Absolute Error (MAE) for evaluation. All metrics are calculated using the SciKit-Learn module.

R^2 is the ratio of the explained variation compared to the total variation, it is interpreted as the fraction of the sample variation in *y* (the dependent variable) that is explained by *x* (the independent variable(s)) (Damásio 2019). It is calculated as:

$$R^2 = 1 - \frac{SSR}{SST}, \qquad where\ SST = \sum_{i=1}^{n}(y_i - \bar{y})^2 \ \ and\ SSR = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Values of R^2 outside the range 0 to 1 can occur when the model fits the data worse than a horizontal hyperplane. This would occur when the wrong model was chosen.

The MSE is calculated as the mean of the squared differences between predicted and expected target values in a dataset:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where y_i is the i´th expected value in the dataset and $\hat{y}_i$ is the i´th predicted value. The difference between these two values is squared, which has the effect of removing the sign, resulting in a positive error value.

The squaring also has the effect of inflating or magnifying large errors. This has the effect of "punishing" models more for larger errors when it is used as a metric (Brownlee 2021).

The MAE score is calculated as the average of the absolute error values. It is computed as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

While MSE punishes larger errors more than smaller errors due to the square of the error value, the MAE does not give more or less weight to different types of errors and instead the scores increase linearly with the increases in error (Brownlee 2021).

4. RESULTS AND DISCUSSION

4.1. LINEAR REGRESSION RESULTS

For the default linear regression model on the word count matrix the best configuration was to consider only the top 2000 most frequent unigrams. Table 6 summarizes the results for the three pre-processing configurations analyzed. As we can observe from table 6, for 2000 tokens, lemmatization produces slightly better results.

	Basic	**Stemming**	**Lemmatization**
R2	0.58	0.58	0.59
MSE	0.35	0.35	0.34
MAE	0.46	0.46	0.45

Table 6 – Linear Regression results on the 2000 most frequent unigrams for basic, stemming and lemmatization configurations

In terms of the number of predictors used, considering 1000 most common words the results are the same for stemming, however for the other 2 options it produces worse results. For 3000 most frequent words the results were worse for all three pre-processing techniques. (See appendix 8.2.1)

In terms of the n-gram length used, there seems to be no improvement in results by considering both unigrams and bigrams instead of only unigrams (See appendix 8.2.1). Using only bigrams produces worse results.

The next linear regression model trained greatly improved on the baseline performance, as it is observed in figure 11. Instead of simply regressing the log car price on the 2000 most frequent words raw count, the matrix of counts is transformed to a normalized tf-idf representation. The configuration that achieved the best performance was: No stemming or lemmatization applied, unigrams only and considering the tf-idf features of the most frequent 2000 tokens.

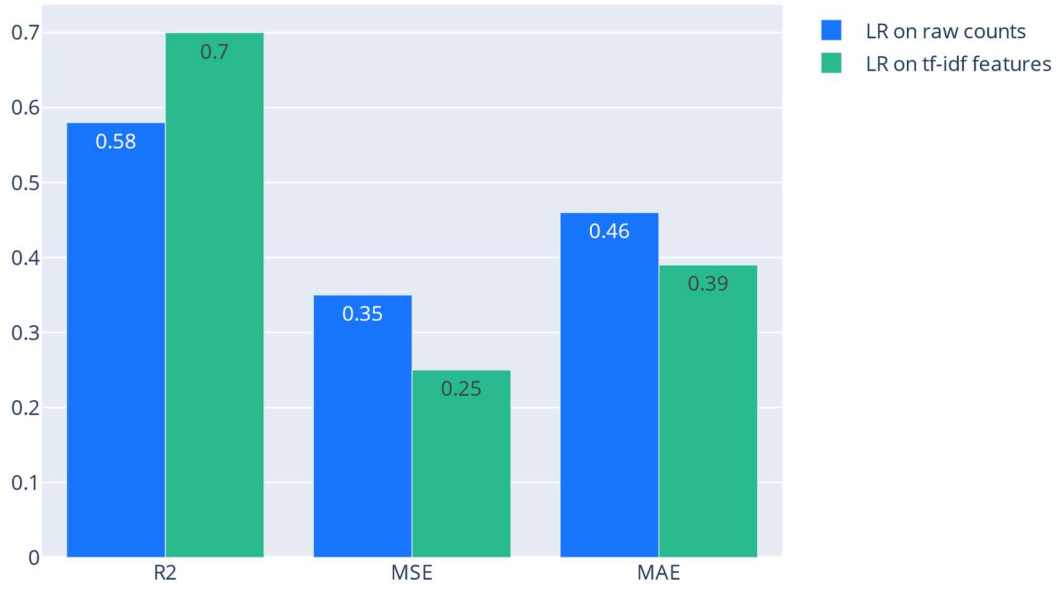

Figure 11- Performance comparison between Linear Regression on raw counts vs on tf-idf features. (No stemming or lemmatization used, unigrams only, 2000 most frequent words used as predictors)

The next model used regressors created from the SVD of the document/word count matrix. It was retained *k*=100,200,300,400,500 singular vectors of the word count matrix. The number of singular vectors retained from the word count matrix produced better results was 500. Table 7 summarizes the results for all three pre-processing techniques studied. As we can see applying stemming or lemmatization just slightly outperforms the basic preprocessing, however only in the R^2 metric.

Top 500 singular vectors	**Basic**	**Stemming**	**Lemmatization**
R2	0.57	0.58	0.58
MSE	0.36	0.36	0.36
MAE	0.47	0.47	0.47

Table 7- Linear Regression results using the top 500 singular vectors retained from the unigram count matrix

Retaining the top 500 singular vectors from the n-gram count matrix that contains both unigrams and bigrams instead of just unigrams hardly improves the results. There is an improvement of 0.01 in MAE when considering the basic or the stemming preprocessing, however for lemmatization there is a

decrease of 0.01 in R^2. If we consider only bigrams on the n-gram count matrix the results greatly decrease.

Similar to the previous results, instead of applying SVD to the word count matrix directly, retaining the top singular vectors from the tf-idf score matrix improves the results. The number of singular vectors analyzed was 100, 200, 300, 400 and 500. Again, keeping 500 singular vectors from the unigram and bigram count matrix seems to outperform the other configurations. As far as preprocessing configurations there seems to be no difference in the 3 different studied, as we can observe from table 8.

500 singular vectors	**Basic**	**Stemming**	**Lemmatization**
R2	0.69	0.69	0.69
MSE	0.26	0.26	0.26
MAE	0.39	0.39	0.39

Table 8 - Linear Regression results using the top 500 singular vectors retained from the tf-idf scores matrix of the unigram & bigram count matrix

As we can observe, for Linear Regression using the tf-idf scores from the top 2000 most frequent words, seems to outperform using the top 500 singular vectors from the tf-idf score matrix as regressors.

Another thing we can gather from these results is that when using tf-idf scores as regressors, considering both unigrams and bigrams for the tf-idf score matrix doesn´t improve the results. However, when using the singular vectors retained from the tf-idf score matrix as regressors, considering both unigrams and bigrams to create the matrix improves the results, albeit a very small improvement.

4.1.1. Linear Regression Variants Results

The different types of linear regression were also compared against the basic linear regression. These comparisons were made on the data with no stemming or lemmatization applied. The first comparison was made using as regressors the tf-idf scores of the 2000 most frequent unigrams. Ridge Regression slightly outperforms the basic Linear Regression model, on the other hand Lasso Regression and Elastic Net Regression perform much worse. Table 9 summarizes the results.

	Linear Regression	Ridge Regression	Lasso Regression	Elastic Net Regression
R2	0.70	0.72	0.00	0.00
MSE	0.25	0.24	0.84	0.84
MAE	0.39	0.37	0.75	0.75

Table 9 – Results for the different types of Linear Regression, using the tf-idf scores of the 2000 most frequent unigrams as predictors

The second comparison was made using the top 500 singular vectors from the tf-idf scores matrix obtained from the unigram count matrix. In this case Ridge Regression has the same results has the basic Linear Regression, the other two regression models show much worse performance. The results are shown in the following table:

500 singular vectors	**Linear Regression**	**Ridge Regression**	**Lasso Regression**	**Elastic Net Regression**
R2	0.68	0.68	0.00	0.00
MSE	0.27	0.27	0.84	0.84
MAE	0.40	0.40	0.75	0.75

Table 10 – Results for the different types of Linear Regression, using the top 500 singular vectors of the tf-idf score matrix as predictors

4.2. SUPPORT VECTOR REGRESSION RESULTS

For Support Vector three different classifiers were tested, as stated previously, *Support Vector Regression (SVR), Nu Support Vector Regression (NuSVR)* and *Linear Support Vector Regression (LinearSVR)*.

The same types of configurations tested for Linear Regression were used in this case. The three best configurations can be seen in Table 11. From our analysis we can see that NuSVR performs best among all classifiers. SVR achieves very similar results to NuSVR, usually only producing a worse result in MAE (See appendix 8.2.2). LSVR shows the worst performance for all metrics considered.

Pre-processing	Classifier	Variables	N-Gram length	R^2	MSE	MAE
Lemmatization/ Stemming	NuSVR	3000 tf-idf features	Unigrams only	0.77	0.19	0.32
Stemming	NuSVR	500 singular vectors obtained from tf-idf matrix	Unigrams only	0.77	0.20	0.32
Basic	NuSVR	500 singular vectors obtained from tf-idf matrix	Unigrams & Bigrams	0.77	0.20	0.32

Table 11 – Best Performing Support Vector Regression Configurations

As it is possible to observe from Table 11, both strategies for choosing the predictors are included in the top performing configurations. The results for R^2 and MAE are the same considering the top 500 singular vectors as predictors or keeping the top 3000 tf-idf features, the latter only slightly outperforming in terms of MSE.

4.2.1. Nu Value hyperparameter optimization

Given that NuSVR was the regressor that produced better results, an analysis on the model complexity measured against its performance on the results was made. The parameter *nu* controls the number of support vectors and it as value in the interval (0,1] (Anon n.d.), with the default value being 0.5. The higher the nu value the higher the number of support vectors.

Figures 12, 13 and 14 show the influence of increasing the model complexity on the measured performance metric, with the time the respective model takes to train being shown.

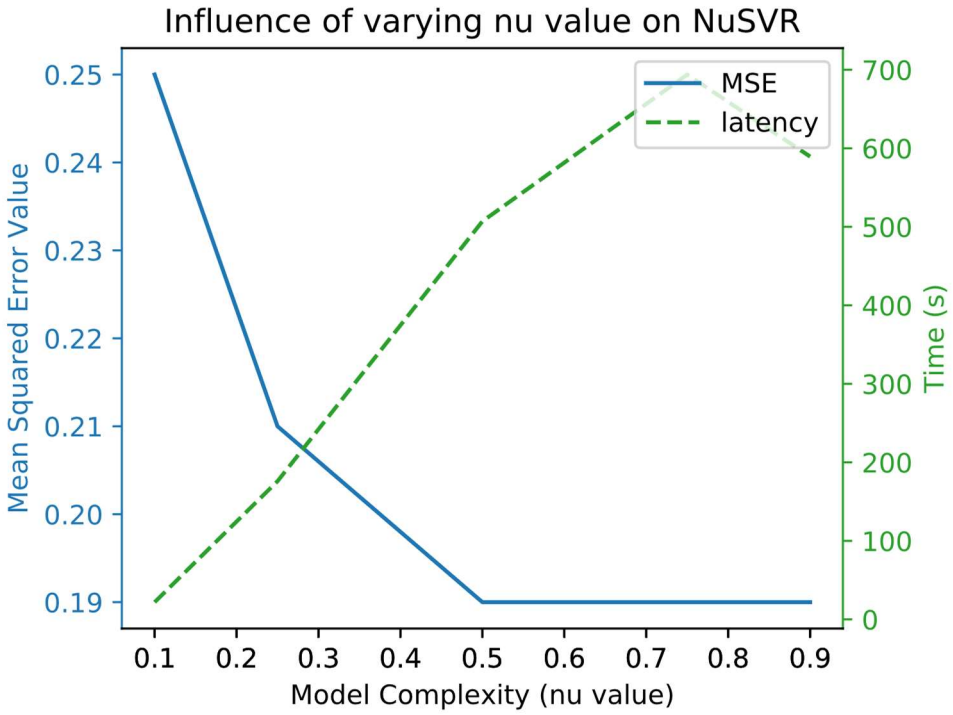

Figure 12 – Influence of varying nu value on NuSVr for MSE

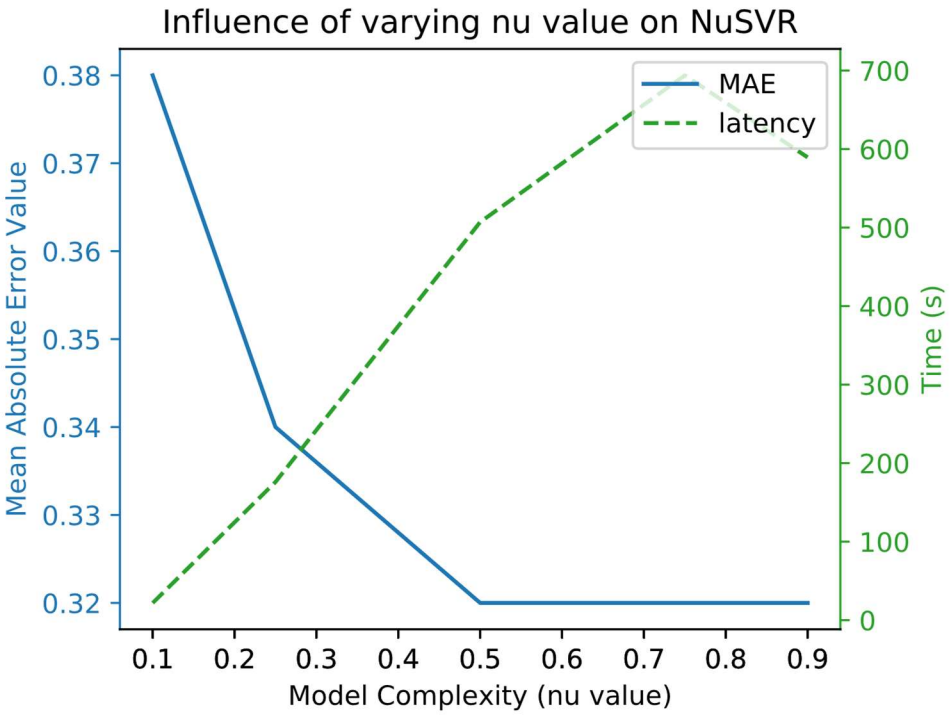

Figure 13 – Inlfuence of varying nu vale on NuSVR for MAE

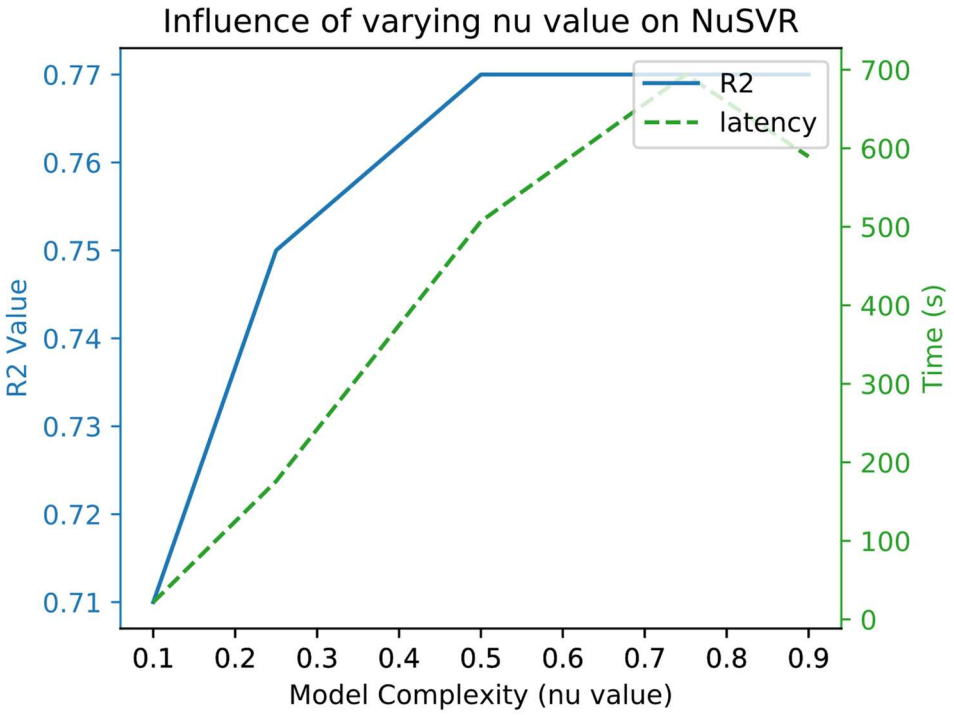

Figure 14 – Inlfuence of varying nu value on NuSVR for R^2

From the analysis of all three figures, we can observe that the better performing value of the nu parameter is 0.5, the default value. Figure 12 and 13 show that the error (MSE and MAE respectively) keeps decreasing until 0.5, after that it stabilizes and the increase in model complexity only increases the time the model takes to train. In figure 14 the conclusion is similar, the only difference is that since it is R^2 being measured, the line has a positive slope until nu=0.5.

4.3. Decision Tree Regressor and Gradient Boosting Regressor Results

The Decision Tree Regressor, available in Scikit-Learn, showed the worst performance for this regression task out of all the classifiers tested. The best configuration found for this model was to use 2000 tf-idf features derived from the unigram count matrix with no stemming or lemmatization applied. It produced an R^2 value of 0.28, a MSE value of 0.60 and a MAE value of 0.56. Since the performance of the model with the default parameters was so poor compared against the other models, hyperparameter optimization wasn´t explored.

Gradient Boosting Regressor provided better results than Decision Tree Regressor. The same types of configurations were analyzed and for Gradient Boosting Regressor the results are better using singular vectors as predictors. There are several combinations that produce the same results (See appendix 8.2.3), for instance using 100 singular vectors retained from the tf-idf matrix considering both unigrams and bigrams and with basic pre-processing originates the same results as using 500 singular vectors also retained from the tf-idf matrix originated from both unigrams and bigrams and applying stemming or lemmatization. However, with 100 singular vectors only the model had a smaller training time.

The best configuration that was chosen to use in further hyperparameter optimization was: 100 singular vectors derived from the tf-idf matrix considering both unigrams and bigrams and applying basic pre-processing. The respective metric values were 0.61 for R^2, 0.33 for MSE and 0.44 for MAE.

4.3.1. Gradient Boosting Regressor hyperparameter optimization

After choosing the configuration that performed best for Gradient Boosting Regressor, hyperparameter tuning was performed. The two parameters analyzed were the number of estimators used and the max depth of each individual regression tree estimator.

From figure 15 we can see that the performance for all three metrics keeps increasing the higher the number of estimators used, until 500 estimators, after 500 it doesn´t improve on all three metrics analyzed. With 500 estimators the R^2 value improves to 0.65, the MSE to 0.30 and the MAE to 0.41.

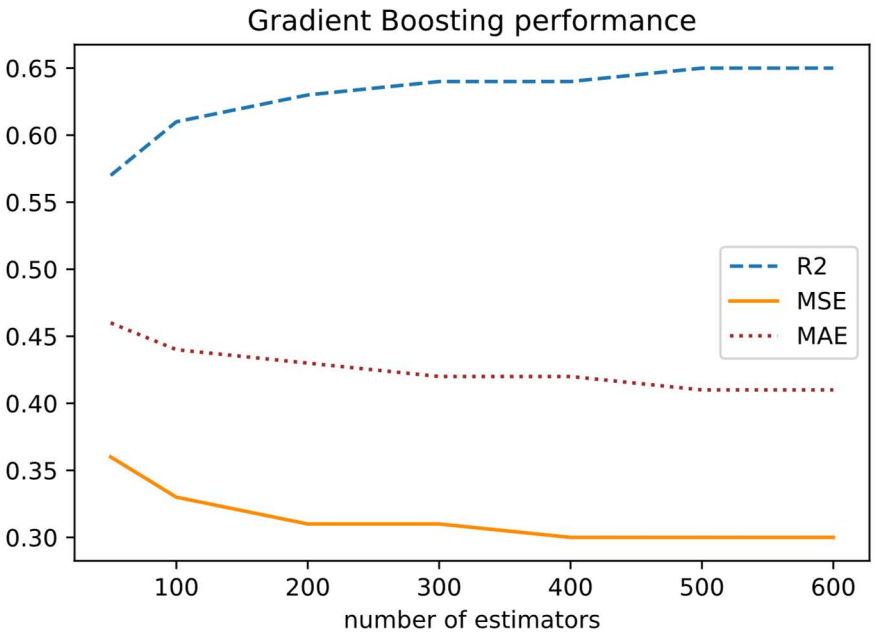

Figure 15 – Gradient Boosting Regressor performance when varying the number of estimators

The same type of analysis was made for the max depth value, but in this case the number of estimators was set to 500 instead of the default value of 100, seeing as it increases the model performance. Figure 16 shows that the performance of the estimator keeps increasing until a max depth value of 5 for both R^2 and MSE. For MAE, the value is lowest with a max depth value of 10, albeit very small decrease compared to max depth of 5 (just 0.01 of difference).

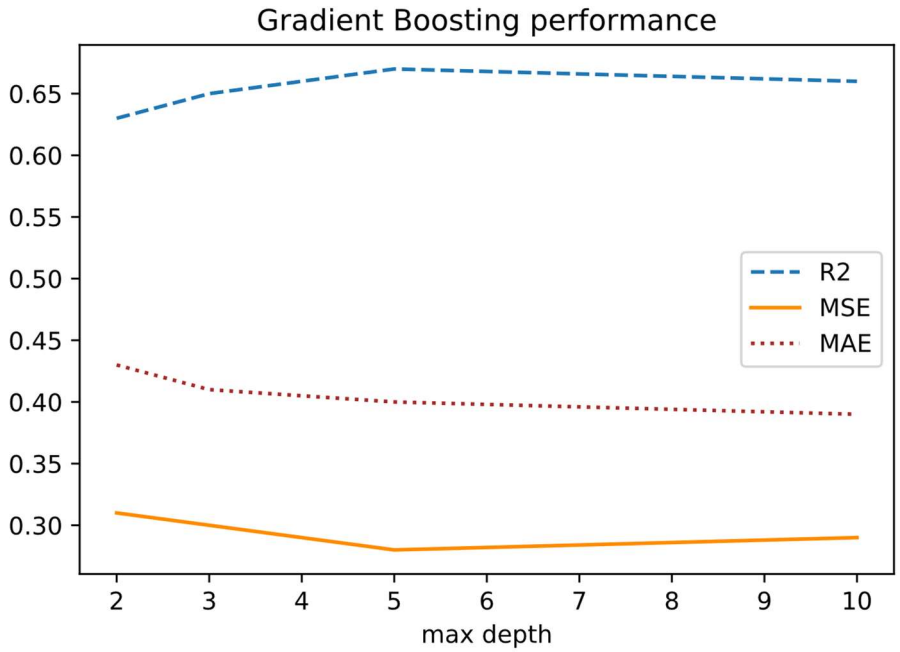

Figure 16 – Gradient Boosting Regressor performance when varying the max depth of the individual regression trees

Seeing as tough two of the three metrics analyzed in this work show better results with a max depth value of 5, this value was chosen as the one that optimized the performance of Gradient Boosting Regressor for this work. The final values for the performance metrics were 0.67 for R^2, 0.28 for MSE and 0.40 for MAE.

4.4. Multi-Layer Perceptron Results

The final regressor tested in this work was the Multi-Layer Perceptron Regressor. The same configurations were tested (See appendix 8.2.4). For this regressor using singular vectors provides better results than using the tf-idf features as regressors. The top two configurations in terms of performance on default hyperparameters can be seen in Table 12.

Pre-processing	Classifier	Variables	N-Gram length	R^2	MSE	MAE
Basic	MLPRegressor	100 singular vectors obtained from tf-idf matrix	Unigrams & Bigrams	0.67	0.28	0.40
Stemming	MLPRegressor	100 singular vectors obtained from tf-idf matrix	Unigrams & Bigrams	0.66	0.28	0.40

Table 12 – Best performing configurations for MLPRegressor

There is just a small difference in R^2 value between using the basic pre-processing and applying stemming, for the other two metrics there is no difference. In terms of the number of singular vectors used, we can observe that using the smaller number of 100 singular vectors provides the best results. (See appendix 8.2.4)

4.4.1. Multi-Layer Perceptron Regressor Hyperparameter optimization

After obtaining the configuration that achieves the better performance on this regressor with the default hyperparameters. In an attempt to improve the regressor performance, hyperparameter tuning was performed.

For this work the following Multi-Layer Perceptron parameters were analyzed:

- Hidden layer sizes
- Activation function
- Solver function
- Learning rate
- Learning rate initial value
- Alpha value
- Max iterations

The following tables show the results for each hyperparameter, and the different values tested. The parameters were tested in the order stated above, each time updating the hyperparameter configuration with the best value obtained.

Hidden layer sizes	**(10,)**	**(20,)**	**(40,)**	**(60,)**	**(100,)**	**(10,10)**	**(20,20)**	**(40,40)**	**(60,60)**	**(100,100)**
R2	0.63	0.62	0.65	0.66	**0.67**	0.65	0.66	0.66	0.66	0.64
MSE	0.31	0.32	0.29	0.29	**0.28**	0.29	0.29	0.29	0.28	0.31
MAE	0.42	0.42	0.41	0.40	**0.40**	0.41	0.40	0.41	0.40	0.42

Table 13 – Comparing various hidden layer sizes configurations

Activation function	**identity**	**logistic**	**tanh**	**relu**
R2	0.59	0.59	0.59	**0.67**
MSE	0.35	0.34	0.34	**0.28**
MAE	0.45	0.45	0.45	**0.40**

Table 14 – Comparing the different activation functions

Solver function	**lbfgs**	**sgd**	**adam**
R2	0.63	0.59	**0.67**
MSE	0.31	0.34	**0.28**
MAE	0.43	0.45	**0.40**

Table 15 – Comparing solver functions

Learning rate	**constant**	**invscaling**	**adaptive**
R2	0.67	0.67	0.67
MSE	0.28	0.28	0.28
MAE	0.40	0.40	0.40

Table 16 – Comparing Learning Rates

Learning rate init	**0.0001**	**0.001**	**0.005**	**0.01**	**0.05**	**0.1**
R2	0.62	**0.67**	0.64	0.63	0.64	0.64
MSE	0.32	**0.28**	0.30	0.31	0.30	0.30
MAE	0.43	**0.40**	0.42	0.42	0.42	0.42

Table 17 – Comparing different initial learning rate values

Alpha	**0.00001**	**0.0001**	**0.001**	**0.01**
R2	0.66	**0.67**	**0.67**	**0.67**
MSE	0.29	**0.28**	**0.28**	**0.28**
MAE	0.40	**0.40**	**0.40**	**0.40**

Table 18 – Comparing different alpha values

Max iterations	**100**	**200**	**300**	**400**	**500**	**1000**	**2000**
R2	0.65	**0.67**	0.66	0.66	0.66	0.66	0.66
MSE	0.29	**0.28**	0.29	0.29	0.29	0.29	0.29
MAE	0.41	**0.40**	0.40	0.41	0.41	0.41	0.41

Table 19 – Comparing the performance on different numbers of maximum iterations

From the results we can see that across all hyperparameters values tested, the default value is the better performing one. Learning rate and alpha also produce the same results with different values other than the default, however for all other hyperparameters that is not the case.

Since no further improvements in performance were made with hyperparameter tuning, given that the default values outperformed the values tested. The MLPRegressor model has a performance of 0.67 for R^2, 0.28 for MSE and 0.40 for MAE.

4.5. RESULTS SUMMARY

The following table provides a concise summary of all the methods applied and their respective best results in predicting the log car price.

Classifier	Pre-processing	Variables	N-Gram length	R^2	MSE	MAE
Linear Regression	Basic	Top 2000 tf-idf features	Unigrams only	0.70	0.25	0.39
Ridge Regression	Basic	Top 2000 tf-idf features	Unigrams only	0.72	0.24	0.37
Lasso Regression / Elastic Net Regression	Basic	Top 2000 tf-idf features	Unigrams only	0.00	0.84	0.75
NuSVR	Lemmatization/ Stemming	Top 3000 tf-idf features	Unigrams only	0.77	0.19	0.32
SVR	Lemmatization/ Stemming	Top 3000 tf-idf features	Unigrams only	0.77	0.20	0.33
LSVR	Basic	Top 3000 tf-idf features	Unigrams only	0.71	0.24	0.37
Decision Tree Regressor	Basic	Top 2000 tf-idf features	Unigrams only	0.60	0.28	0.56
Gradient Boosting Regressor	Basic	100 singular vectors derived from tf-idf matrix	Unigrams & Bigrams	0.67	0.28	0.40
MLP Regressor	Basic	100 singular vectors derived from tf-idf matrix	Unigrams & Bigrams	0.67	0.28	0.40

Table 20 – Applied methods and their respective best results

5. CONCLUSIONS

The research objective of this work project was to identify if the description text alone was a good enough predictor of car´s price. The experiment on creating regressors from unstructured text for this regression problem showed different results across the various regression models used.

Support Vector Regression outperformed all other models of regression tested in this work project. NuSVR was the better support vector regression model used, with it´s best configuration explaining 77% of the variation in the log price and also achieving good results in the MSE and MAE metrics. SVR also achieves very similar results to NuSVR, only producing slightly worse results in MAE.

The worst performing models were Lasso Regression and Elastic Net regression, both with an R^2 value of 0.00, a MSE value of 0.84 and a MAE value of 0.75. However, the other Linear Regression variant tested, Ridge Regression, showed better performance than the default Linear Regression model with an R^2 value of 0.72, a MSE of 0.24 and a MAE of 0.37, being the second-best performing model.

As far as pre-processing strategies applied in this work, results suggest that there wasn´t one that had an increased advantage in performance compared to the others, the difference usually being of just around 0.01/0.02 in either R^2, MSE or MAE. The performance varied according to the regressor used. For instance, for Linear Regression, Decision Tree Regressor, Gradient Boosting Regressor and Multi-Layer Perceptron Regressor the better configuration was found by not applying either stemming or lemmatization. However, for NuSVR the better configuration was found by using stemming/lemmatization.

The same conclusion can be derived for the n-gram length considered to create the initial n-gram count matrix, there was no option that consistently performed better for all types of regressors studied. For example, for Linear Regression when using the tf-idf scores as predictors considering both unigrams and bigrams doesn´t improve the model performance compared to only using unigrams. However, when using the singular vectors as predictors, the model performance is slightly increased when considering both unigrams and bigrams. Using only bigrams to create the initial n-gram count matrix proved to lead to worse results for all models analyzed.

In terms of predictors derived from the initial unstructured text, two types were analyzed: retaining the top n (n=1000, 2000, 3000) tf-idf features according to count frequency or using the top k (k=100, 200, 300, 400, 500) singular vectors derived from the tf-idf matrix. Our results suggest that, for this regression problem, the better performing option varies with the regression model being used and no option consistently outperforms the other. For instance, for Linear Regression and Support Vector Regression, which were the better performing models, using tf-idf features as predictors outperforms using singular vectors by a small margin. For Gradient Boosting Regressor and Multi-Layer Perceptron Regressor it was found that using singular vectors as predictors led to better performing models. Nevertheless, that difference in results was no greater than 0.05 in any of the three metrics analyzed, when compared to the best configuration that used the tf-idf features as predictors.

Applying the tf-idf vectorizer to transform the initial matrix of raw counts was the step that vastly improved the results, we can see from the baseline Linear Regression that used the raw word counts as predictors compared to the Linear Regression that retained the same number of tokens but with the tf-idf scores instead a great improvement. The metrics jumped from an R^2 of 0.58 to 0.70, a MSE

of 0.35 to 0.25 and a MAE of 0.46 to 0.39. This improvement was also verified when using Support Vector Regression and Multi-Layer Perceptron Regressor.

The analysis of the results found in the previous section, show that some regression models were more successful than others in managing to predict the log price a car being sold in an online car, using as predictors features created from unstructured text. These results demonstrate that text regression models for price prediction may be applicable as a complement to traditional methods of car price prediction through the use of the common attributes.

About the objectives initially proposed in this work, we believe they were achieved. At the end of this work project, we have a better understanding of the extent of the capabilities that textual descriptions have in accurately predicting car prices, with some regression models being more capable in using the features derived from those description texts to regress the price. Also, we deepened our knowledge on the question of text pre-processing and feature extraction from text, that lead to being able to create features capable of being used in our regression problem. Lastly, we increased our theoretical and practical knowledge on the addressed topics of Text Mining, text regression, the studied algorithms and the tools used.

6. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

Throughout the development of this work project, we have encountered some difficulties. One of those problems and perhaps the one that most influenced the progress was the treatment of the data, given that it all consisted of human inserted text it contained a very large amount of badly written words. This caused the data to be very inconsistent, having multiple variations of the same word, making the process of word correction quite time consuming.

Another problem related to word correction was the lack of tools for word correction that included in their vocabulary the context specific words, related to car descriptions, causing those tools to wrongly identify a large number of words as badly written.

Also, in the data collection process we encountered some deadlocks, given that the data was taken from ads published by the users of a website the amount of data collected was dependent on the number of new ads published by the users, which made the data collection process quite iterative and time consuming.

For future works in this area, it is recommended to create a pre-defined vocabulary with the context specific words to facilitate the data treatment process. Also, although the text was obtained from a Portuguese website, car description texts contain a great number of words that are anglicisms, such as “bluetooth” and “airbag”. With that being said, it is recommended that the word correction tool is able to recognize both languages, in this case Portuguese and English.

In terms of modelling and performance results, for future works instead of first finding the optimal configuration in terms of what type of features to use as predictors, with the default model hyperparameters, and only after proceeding to doing hyperparameter tuning. An option worth considering is to apply a Grid Search, available in Sci-Kit Learn, to find the optimal configuration with the corresponding hyperparameters that result in better performance. This could lead to an increase in the model’s performance, albeit with an increase in computation time.

7. BIBLIOGRAPHY

Allahyari, Mehdi, Seyedamin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys Kochut. 2017. “A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques.” *ArXiv:1707.02919 [Cs]*.

Anon. n.d. “1.4. Support Vector Machines — Scikit-Learn 0.24.2 Documentation.” Retrieved June 29, 2021a (https://scikit-learn.org/stable/modules/svm.html#svm-regression).

Anon. n.d. “Difflib — Helpers for Computing Deltas — Python 3.9.2 Documentation.” Retrieved April 2, 2021b (https://docs.python.org/3/library/difflib.html).

Anon. n.d. “Sklearn.Feature_extraction.Text.TfidfTransformer — Scikit-Learn 0.24.2 Documentation.” Retrieved April 28, 2021c (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html).

Barrus, Tyler. 2018. “Pyspellchecker — Pyspellchecker 0.6.1 Documentation.” Retrieved March 24, 2021 (https://pyspellchecker.readthedocs.io/en/latest/).

Bitvai, Zsolt, and Trevor Cohn. 2015. “Non-Linear Text Regression with a Deep Convolutional Neural Network.” Pp. 180–85 in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China: Association for Computational Linguistics.

Brownlee, Jason. 2021. “Regression Metrics for Machine Learning.” *Machine Learning Mastery*. Retrieved June 8, 2021 (https://machinelearningmastery.com/regression-metrics-for-machine-learning/).

Castelli, Mauro, Maria Dobreva, Roberto Henriques, and Leonardo Vanneschi. 2020. “Predicting Days on Market to Optimize Real Estate Sales Strategy.” *Complexity* 2020:1–22. doi: 10.1155/2020/4603190.

Damásio, Bruno. 2019. “Statistics for Data Science - The Linear Regression Model.” 120.

Elith, J., J. R. Leathwick, and T. Hastie. 2008. “A Working Guide to Boosted Regression Trees.” *Journal of Animal Ecology* 77(4):802–13. doi: 10.1111/j.1365-2656.2008.01390.x.

Foster, Dean P., Mark Liberman, and Robert A. Stine. 2013. “Featurizing Text: Converting Text into Predictors for Regression Analysis.” 37.

Gegic, Enis, Becir Isakovic, Dino Keco, Zerina Masetic, and Jasmin Kevric. n.d. “Car Price Prediction Using Machine Learning Techniques.” 8(1):6.

Gentzkow, Matthew, Bryan Kelly, and Matt Taddy. 2019. “Text as Data.” *Journal of Economic Literature* 57(3):535–74. doi: 10.1257/jel.20181020.

Gilleland, Michael. n.d. “Levenshtein Distance.” *Levenshtein Distance, in Three Flavors*. Retrieved August 30, 2021

(http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Fall2006/Assignments/editdistance/Levenshtein%20Distance.htm).

Guo, Jian-qiang, Shu-hen Chiang, Min Liu, Chi-Chun Yang, and Kai-yi Guo. 2020. "CAN MACHINE LEARNING ALGORITHMS ASSOCIATED WITH TEXT MINING FROM INTERNET DATA IMPROVE HOUSING PRICE PREDICTION PERFORMANCE?" *International Journal of Strategic Property Management* 24(5):300–312. doi: 10.3846/ijspm.2020.12742.

Hotho, Andreas, Andreas Nürnberger, and Gerhard Paaß. 2005. "A Brief Survey of Text Mining." 45.

Jin, Fang, Nathan Self, Parang Saraf, Patrick Butler, Wei Wang, and Naren Ramakrishnan. 2013. "Forex-Foreteller: Currency Trend Modeling Using News Articles." P. 1470 in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*. Chicago, Illinois, USA: ACM Press.

Joshi, Mahesh, Dipanjan Das, Kevin Gimpel, and Noah A. Smith. 2010. "Movie Reviews and Revenues: An Experiment in Text Regression." 4.

Kogan, Shimon, Dimitry Levin, Bryan R. Routledge, Jacob S. Sagi, and Noah A. Smith. 2009. "Predicting Risk from Financial Reports with Regression." P. 272 in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics on - NAACL '09*. Boulder, Colorado: Association for Computational Linguistics.

Lehmann, Carlotta. 2020. "Predictive Methods of Data Mining: Linear Regression."

Lerman, Kevin, Ari Gilder, Mark Dredze, and Fernando Pereira. 2008. "Reading the Markets: Forecasting Public Opinion of Political Candidates by News Analysis." Pp. 473–80 in *Proceedings of the 22nd International Conference on Computational Linguistics - COLING '08*. Vol. 1. Manchester, United Kingdom: Association for Computational Linguistics.

Liang, Hong, Xiao Sun, Yunlei Sun, and Yuan Gao. 2017. "Text Feature Extraction Based on Deep Learning: A Review." *EURASIP Journal on Wireless Communications and Networking* 2017(1):211. doi: 10.1186/s13638-017-0993-1.

Mendes, Jorge M. 2017. "Principal Components Analysis." 336.

Mishne, Gilad, and Natalie Glance. 2006. "Predicting Movie Sales from Blogger Sentiment." 4.

Nassirtoussi, Arman, Saeed Aghabozorgi, Teh Ying Wah, and David Chek Ling Ngo. 2014. "Text Mining for Market Prediction: A Systematic Review." *Expert Systems with Applications* 41(16):7653–70. doi: 10.1016/j.eswa.2014.06.009.

Ngo-Ye, Thomas L., and Atish P. Sinha. 2014. "The Influence of Reviewer Engagement Characteristics on Online Review Helpfulness: A Text Regression Model." *Decision Support Systems* 61:47–58. doi: 10.1016/j.dss.2014.01.011.

Pak, Muhammet Yasin, and Serkan Gunal. 2017. "THE IMPACT OF TEXT REPRESENTATION AND PREPROCESSING ON AUTHOR IDENTIFICATION." *ANADOLU UNIVERSITY JOURNAL OF SCIENCE AND TECHNOLOGY A - Applied Sciences and Engineering* 18(1):218–218. doi: 10.18038/aubtda.270276.

Pedregosa, Fabian, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, and David Cournapeau. 2011. “Scikit-Learn: Machine Learning in Python.” *MACHINE LEARNING IN PYTHON* 6.

Pudaruth, Sameerchand. n.d. “Predicting the Price of Used Cars Using Machine Learning Techniques.” 13.

ScrapeHero. 2020. “How to Scrape Websites without Getting Blocked.” Retrieved (https://www.scrapehero.com/how-to-prevent-getting-blacklisted-while-scraping/).

8. APPENDIX

8.1. DATA COLLECTION SCRIPT

```
import requests
from bs4 import BeautifulSoup
import re
import pandas as pd
from tqdm import tqdm_notebook as tqdm
import time
import random

dic = {'title':[], 'model':[], 'desc':[],'price' : []}
#setting my user agent
headers = {
      'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.198 Safari/537.36'

       }


URL = 'https://www.olx.pt/carros-motos-e-barcos/carros/?search%5Bprivate_business%5D=private&page=0'

for j in tqdm(range(1,26)):
   URL = URL[:-1] + str(j) # changing the last character of the URL so that it changes pages
   print("scraping page: " + str(j))

   sleeptime = random.randint(2,20)
  time.sleep(sleeptime)

   page = requests.get(URL,headers = headers)

   soup = BeautifulSoup(page.content, 'html.parser')
   data = soup.find('table', class_='fixed offers breakword redesigned')
   ads = data.find_all('tr', class_='wrap')

   for ad in ads:
      title_elem = ad.find('td',class_ = 'title-cell')
      title = title_elem.div.h3.strong.text
      link = title_elem.find('a')['href']

      price_elem = ad.find('td', class_ = 'wwnormal tright td-price')
      price = price_elem.div.p.strong.text
```

```
sleeptime = random.choice([3.4,4.5,2.1,6.7,2.9,3,1.4,9.8,2.3])
time.sleep(sleeptime)

ad_specific_page = requests.get(link)
 ad_specific_soup = BeautifulSoup(ad_specific_page.text,'html.parser')

#resetting the model value so it doesn´t overlap
model = "N/A"

if 'https://www.olx.pt' in link:
   full_desc_soup = ad_specific_soup.find('div', class_ = 'clr descriptioncontent marginbott20')

   if full_desc_soup is None:
      continue

   options = full_desc_soup.find_all('a')

   for option in options:
      if option is None:
         continue

      if option.span.text == 'Modelo':
         model = option.strong.text
         break # already found the model name, no need to keep looping

   text_desc = full_desc_soup.find('div', class_ = 'clr lheight20 large')
   desc = text_desc.text

else: # if its redirected to standvirtual website
   full_desc_soup = ad_specific_soup.find('div', class_='offer-content__row om-offer-main')

   if full_desc_soup is None:
      continue

   options = full_desc_soup.select('.offer-params__item')

   for option in options:
      if option is None:
         continue

      if option.span.text == 'Modelo':
         model = option.div.a.text
         break

   text_desc = full_desc_soup.find('div', class_ ='offer-description__description' )
```

```
            desc = text_desc.text


        dic['title'].append(title)
        dic['model'].append(model)
        dic['desc'].append(desc)
        dic['price'].append(price)


test_df = pd.DataFrame.from_dict(dic)
# exporting the data to a csv file

test_df.to_csv(r'C:\Users\ricar\Mestrado\2_Ano\olx_scrape\Dataframes\scrapeXX.csv', index=False)
```

8.2. RESULTS TABLES

8.2.1. Linear Regression

1000 top frequent tokens	**Basic**	**Stemming**	**Lemmatization**
R2	0.57 (0.56)	0.58 (0.56)	0.58 (0.56)
MSE	0.36 (0.37)	0.35 (0.37)	0.35 (0.37)
MAE	0.46 (0.47)	0.46 (0.47)	0.46 (0.48)

Table 21 – Linear Regression results for 1000 most frequent unigrams (unigrams & bigrams)

3000 top frequent tokens	**Basic**	**Stemming**	**Lemmatization**
R2	0.54 (0.57)	0.55 (0.56)	0.56 (0.56)
MSE	0.39 (0.36)	0.38 (0.37)	0.37 (0.37)
MAE	0.47 (0.46)	0.47 (0.47)	0.47 (0.47)

Table 22 – Linear Regression results for the 3000 most frequent unigrams (unigrams & bigrams)

1000 tf-idf tokens	**Basic**	**Stemming**	**Lemmatization**
R2	0.67 (0.66)	0.68 (0.67)	0.68 (0.66)
MSE	0.28 (0.29)	0.26 (0.28)	0.27 (0.29)
MAE	0.40 (0.41)	0.40 (0.41)	0.40 (0.41)

Table 23 - Linear Regression results for the top 1000 tf-idf scores considering only unigrams (unigrams & bigrams)

2000 tokens	Basic	Stemming	Lemmatization
R2	0.70 (0.69)	0.69 (0.69)	0.70 (0.69)
MSE	0.25 (0.26)	0.26 (0.26)	0.26 (0.26)
MAE	0.39 (0.39)	0.39 (0.39)	0.39 (0.39)

Table 24 - Linear Regression results for the top 2000 tf-idf scores considering only unigrams (unigrams & bigrams)

3000 tokens	Basic	Stemming	Lemmatization
R2	0.68 (0.69)	0.69 (0.68)	0.69 (0.67)
MSE	0.26 (0.26)	0.26 (0.27)	0.26 (0.27)
MAE	0.39 (0.39)	0.39 (0.40)	0.39 (0.40)

Table 25 - Linear Regression results for the top 3000 tf-idf scores considering only unigrams (unigrams & bigrams)

Top 100 singular vectors	Basic	Stemming	Lemmatization
R2	0.49 (0.48)	0.48 (0.48)	0.48 (0.47)
MSE	0.43 (0.44)	0.43 (0.44)	0.44 (0.44)
MAE	0.52 (0.52)	0.52 (0.52)	0.52 (0.52)

Table 26 – Linear Regression results for the top 100 singular vectors retained from the word count matrix using only unigrams (unigrams & bigrams)

Top 200 singular vectors	Basic	Stemming	Lemmatization
R2	0.53 (0.53)	0.53 (0.53)	0.52 (0.52)
MSE	0.40 (0.40)	0.40 (0.39)	0.41 (0.40)
MAE	0.49 (0.49)	0.49 (0.49)	0.50 (0.49)

Table 27 - Linear Regression results for the top 200 singular vectors retained from the word count matrix using only unigrams (unigrams & bigrams)

Top 300 singular vectors	**Basic**	**Stemming**	**Lemmatization**
R2	0.55 (0.54)	0.54 (0.55)	0.54 (0.54)
MSE	0.38 (0.38)	0.39 (0.38)	0.38 (0.38)
MAE	0.48 (0.48)	0.49 (0.48)	0.48 (0.48)

Table 28 - Linear Regression results for the top 300 singular vectors retained from the word count matrix using only unigrams (unigrams & bigrams)

Top 400 singular vectors	**Basic**	**Stemming**	**Lemmatization**
R2	0.56 (0.56)	0.56 (0.56)	0.56 (0.56)
MSE	0.37 (0.37)	0.37 (0.37)	0.37 (0.37)
MAE	0.47 (0.47)	0.47 (0.47)	0.47 (0.47)

Table 29 - Linear Regression results for the top 400 singular vectors retained from the word count matrix using only unigrams (unigrams & bigrams)

Top 500 singular vectors	**Basic**	**Stemming**	**Lemmatization**
R2	0.57 (0.57)	0.58 (0.58)	0.58 (0.57)
MSE	0.36 (0.36)	0.36 (0.36)	0.36 (0.36)
MAE	0.47 (0.46)	0.47 (0.46)	0.47 (0.47)

Table 30 - Linear Regression results for the top 500 singular vectors retained from the word count matrix using only unigrams (unigrams & bigrams)

500 singular vectors	**Basic**	**Stemming**	**Lemmatization**
R2	0.68 (0.69)	0.68 (0.69)	0.68 (0.69)
MSE	0.27 (0.26)	0.27 (0.26)	0.27 (0.26)
MAE	0.40 (0.39)	0.40 (0.39)	0.40 (0.39)

Table 31 – Linear Regression results for the top 500 singular vectors retained from the tf-idf score matrix using only unigrams (unigrams & bigrams)

	Linear Regression	Ridge Regression	Lasso Regression	Elastic Net Regression
R2	0.70 (0.69)	0.72 (0.71)	0.00 (0.00)	0.00 (0.00)
MSE	0.25 (0.26)	0.24 (0.24)	0.84 (0.84)	0.84 (0.84)
MAE	0.39 (0.39)	0.37 (0.38)	0.75 (0.75)	0.75 (0.75)

Table 32 – Comparing the diferente types of Linear Regression on the top 2000 tf-idf scores considering only unigrams (unigrams & bigrams) all with basic pre-processing

500 singular vectors	**Linear Regression**	**Ridge Regression**	**Lasso Regression**	**Elastic Net Regression**
R2	0.68 (0.69)	0.68 (0.69)	0.00 (0.00)	0.00 (0.00)
MSE	0.27 (0.26)	0.27 (0.26)	0.84 (0.84)	0.84 (0.84)
MAE	0.40 (0.39)	0.40 (0.39)	0.75 (0.75)	0.75 (0.75)

Table 33 – Comparing the diferente types of Linear Regression on the top 500 singular vectors retained from the tf-idf score matrix considering only unigrams (unigrams & bigrams) all with basic pre-processing

8.2.2. Support Vector Regression Results

Basic Pre-Processing:

2000 features	**LSvr**	**SVR**	**NuSvr**
R2	0.70 (70)	0.76 (0.75)	0.76 (0.75)
MSE	0.25 (0.26)	0.20 (0.21)	0.20 (0.21)
MAE	0.38 (0.38)	0.33 (0.34)	0.33 (0.33)

Table 34 - Comparing different types of support vector regression using the top 2000 tf-idf features, unigrams only (unigrams & bigrams)

3000 features	LSvr	SVR	NuSvr
R2	0.71 (0.70)	0.76 (0.75)	0.77 (0.76)
MSE	0.24 (0.25)	0.20 (0.21)	0.20 (0.20)
MAE	0.37 (0.38)	0.33 (0.34)	0.32 (0.33)

Table 35 - Comparing different types of support vector regression using the top 3000 tf-idf features, unigrams only (unigrams & bigrams)

Stemming Pre-Processing:

2000 features	SVR	NuSvr
R2	0.76 (0.75)	0.76 (0.76)
MSE	0.20 (0.21)	0.20 (0.21)
MAE	0.33 (0.34)	0.32 (0.33)

Table 36 - Comparing different types of support vector regression using the top 2000 tf-idf features, unigrams only (unigrams & bigrams)

3000 features	SVR	NuSvr
R2	0.77 (0.75)	0.77 (0.76)
MSE	0.20 (0.21)	0.19 (0.20)
MAE	0.33 (0.33)	0.32 (0.33)

Table 37 - Comparing different types of support vector regression using the top 3000 tf-idf features, unigrams only (unigrams & bigrams)

Lemmatization Pre-Processing:

2000 features	**SVR**	**NuSvr**
R2	0.76 (0.75)	0.76 (0.75)
MSE	0.20 (0.21)	0.20 (0.21)
MAE	0.33 (0.34)	0.32 (0.33)

Table 38 - Comparing different types of support vector regression using the top 2000 tf-idf features, unigrams only (unigrams & bigrams)

3000 features	**SVR**	**NuSvr**
R2	0.77 (0.75)	0.77 (0.76)
MSE	0.20 (0.21)	0.19 (0.21)
MAE	0.33 (0.34)	0.32 (0.33)

Table 39 - Comparing different types of support vector regression using the top 3000 tf-idf features, unigrams only (unigrams & bigrams)

Using Singular Vectors and basic pre-processing:

100 singular vectors	**LSvr**	**SVR**	**NuSvr**
R2	0.58	0.70 (0.70)	0.70 (0.70)
MSE	0.35	0.26 (0.25)	0.26 (0.25)
MAE	0.46	0.38 (0.37)	0.38 (0.37)

Table 40 – Comparing different types of support vector regression using the top 100 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams)

200 singular vectors	**LSvr**	**SVR**	**NuSvr**
R2	0.61	0.73 (0.73)	0.73 (0.73)
MSE	0.32	0.21 (0.25)	0.23 (0.23)
MAE	0.44	0.35 (0.35)	0.35 (0.38)

Table 41 - Comparing different types of support vector regression using the top 200 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams)

300 singular vectors	**LSvr**	**SVR**	**NuSvr**
R2	0.65	0.75 (0.75)	0.75 (0.75)
MSE	0.29	0.21 (0.21)	0.21 (0.21)
MAE	0.41	0.34 (0.33)	0.33 (0.33)

Table 42 - Comparing different types of support vector regression using the top 300 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams)

400 singular vectors	**LSvr**	**SVR**	**NuSvr**
R2	0.66	0.76 (0.76)	0.76 (0.76)
MSE	0.28	0.20 (0.20)	0.20 (0.20)
MAE	0.41	0.33 (0.33)	0.33 (0.33)

Table 43 - Comparing different types of support vector regression using the top 400 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams)

500 singular vectors	**LSvr**	**SVR**	**NuSvr**
R2	0.67	0.76 (0.76)	0.76 (0.77)
MSE	0.28	0.20 (0.20)	0.20 (0.20)
MAE	0.40	0.33 (0.33)	0.33 (0.32)

Table 44 - Comparing different types of support vector regression using the top 500 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams)

Using Singular Vectors and stemming pre-processing:

100 singular vectors	**SVR**	**NuSvr**
R2	0.69 (0.70)	0.69 (0.70)
MSE	0.26 (0.26)	0.26 (0.26)
MAE	0.38 (0.37)	0.38 (0.37)

Table 45 - Comparing different types of support vector regression using the top 100 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams), stemming pre-processing

200 singular vectors	**SVR**	**NuSvr**
R2	0.73 (0.74)	0.73 (0.74)
MSE	0.22 (0.22)	0.22 (0.22)
MAE	0.35 (0.34)	0.35 (0.34)

Table 46 - Comparing different types of support vector regression using the top 200 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams), stemming pre-processing

300 singular vectors	**SVR**	**NuSvr**
R2	0.75 (0.75)	0.75 (0.75)
MSE	0.21 (0.21)	0.21 (0.21)
MAE	0.34 (0.33)	0.33 (0.33)

Table 47 - Comparing different types of support vector regression using the top 300 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams), stemming pre-processing

400 singular vectors	**SVR**	**NuSvr**
R2	0.76 (0.76)	0.76 (0.76)
MSE	0.20 (0.20)	0.20 (0.20)
MAE	0.33 (0.33)	0.33 (0.33)

Table 48 - Comparing different types of support vector regression using the top 400 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams), stemming pre-processing

500 singular vectors	**SVR**	**NuSvr**
R2	0.76 (0.76)	0.77 (0.77)
MSE	0.20 (0.20)	0.20 (0.20)
MAE	0.33 (0.33)	0.32 (0.32)

Table 49 - Comparing different types of support vector regression using the top 500 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams), stemming pre-processing

Using Singular Vectors and lemmatization pre-processing:

100 singular vectors	**SVR**	**NuSvr**
R2	0.69 (0.70)	0.69 (0.70)
MSE	0.26 (0.26)	0.26 (0.26)
MAE	0.38 (0.37)	0.38 (0.37)

Table 50 - Comparing different types of support vector regression using the top 100 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams), lemmatization pre-processing

200 singular vectors	**SVR**	**NuSvr**
R2	0.73 (0.73)	0.73 (0.73)
MSE	0.23 (0.22)	0.23 (0.23)
MAE	0.35 (0.35)	0.35 (0.35)

Table 51 - Comparing different types of support vector regression using the top 200 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams), lemmatization pre-processing

300 singular vectors	**SVR**	**NuSvr**
R2	0.75 (0.75)	0.75 (0.75)
MSE	0.21 (0.21)	0.21 (0.21)
MAE	0.34 (0.33)	0.34 (0.33)

Table 52 - Comparing different types of support vector regression using the top 300 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams), lemmatization pre-processing

400 singular vectors	SVR	NuSvr
R2	0.76 (0.76)	0.76 (0.76)
MSE	0.20 (0.20)	0.20 (0.20)
MAE	0.33 (0.33)	0.33 (0.33)

Table 53 - Comparing different types of support vector regression using the top 400 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams), lemmatization pre-processing

500 singular vectors	SVR	NuSvr
R2	0.76 (0.76)	0.76 (0.76)
MSE	0.20 (0.20)	0.20 (0.20)
MAE	0.33 (0.33)	0.32 (0.32)

Table 54 - Comparing different types of support vector regression using the top 500 singular vectors retained from the tf-idf score matrix, unigrams only (unigrams & bigrams), lemmatization pre-processing

8.2.3. Decision Tree Regressor and Gradient Boosting Regressor results

1000 features	Basic	Stemming	Lemmatization
R2	0.21 (0.25)	0.23 (0.19)	0.18 (0.18)
MSE	0.66 (0.63)	0.65 (0.68)	0.69 (0.69)
MAE	0.59 (0.58)	0.58 (0.59)	0.60 (0.60)

Table 55 – Decision Tree Regressor results using the top 1000 tf-idf features, considering only unigrams (unigrams & bigrams)

2000 features	Basic	Stemming	Lemmatization
R2	0.28 (0.28)	0.22 (0.24)	0.19 (0.23)
MSE	0.60 (0.60)	0.65 (0.64)	0.68 (0.65)
MAE	0.56 (0.56)	0.58 (0.58)	0.60 (0.58)

Table 56 - Decision Tree Regressor results using the top 2000 tf-idf features, considering only unigrams (unigrams & bigrams)

3000 features	Basic	Stemming	Lemmatization
R2	0.25 (0.26)	0.24 (0.20)	0.20 (0.28)
MSE	0.63 (0.62)	0.64 (0.67)	0.67 (0.61)
MAE	0.57 (0.57)	0.58 (0.59)	0.59 (0.56)

Table 57 - Decision Tree Regressor results using the top 2000 tf-idf features, considering only unigrams (unigrams & bigrams)

Number of singular vectors	100	200	300	400	500
R2	0.20	0.16	0.16	0.16	0.17
MSE	0.67	0.70	0.71	0.71	0.70
MAE	0.60	0.61	0.62	0.62	0.61

Table 58 – Comparing the performance of Decision Tree Regressor on the number of singular vectors, using basic pre-processing and only unigrams

Gradient Boosting Regressor

1000 features	Basic	Stemming	Lemmatization
R2	0.54 (0.54)	0.55 (0.55)	0.55 (0.54)
MSE	0.38 (0.38)	0.38 (0.38)	0.38 (0.39)
MAE	0.48 (0.48)	0.48 (0.48)	0.48 (0.48)

Table 59 – Comparing the performance of Gradient Boosting Regressor results using the top 1000 tf-idf features, considering only unigrams (unigrams & bigrams)

2000 features	Normal	Stemming	Lemmatization
R2	0.55 (0.55)	0.55 (0.56)	0.54 (0.55)
MSE	0.38 (0.38)	0.38 (0.37)	0.38 (0.38)
MAE	0.48 (0.48)	0.48 (0.47)	0.48 (0.48)

Table 60 - Comparing the performance of Gradient Boosting Regressor results using the top 2000 tf-idf features, considering only unigrams (unigrams & bigrams)

3000 features	Normal	Stemming	Lemmatization
R2	0.55 (0.55)	0.55 (0.56)	0.55 (0.55)
MSE	0.38 (0.38)	0.37 (0.37)	0.38 (0.38)
MAE	0.48 (0.48)	0.47 (0.47)	0.48 (0.48)

Table 61 - Comparing the performance of Gradient Boosting Regressor results using the top 3000 tf-idf features, considering only unigrams (unigrams & bigrams)

100 singular vectors	Normal	Stemming	Lemmatization
R2	0.59 (0.61)	0.58 (0.60)	0.58 (0.60)
MSE	0.35 (0.33)	0.35 (0.34)	0.35 (0.34)
MAE	0.46 (0.44)	0.46 (0.45)	0.46 (0.45)

Table 62 – Gradient Boosting Regressor results using the top 100 singular vectors retained from the tf-idf matrix, considering only unigrams (unigrams & bigrams)

200 singular vectors	Normal	Stemming	Lemmatization
R2	0.59 (0.61)	0.59 (0.61)	0.59 (0.61)
MSE	0.34 (0.33)	0.34 (0.33)	0.35 (0.33)
MAE	0.45 (0.44)	0.46 (0.44)	0.45 (0.44)

Table 63 - Gradient Boosting Regressor results using the top 200 singular vectors retained from the tf-idf matrix, considering only unigrams (unigrams & bigrams)

300 singular vectors	Normal	Stemming	Lemmatization
R2	0.60 (0.61)	0.59 (0.60)	0.59 (0.61)
MSE	0.34 (0.33)	0.34 (0.33)	0.35 (0.33)
MAE	0.45 (0.44)	0.46 (0.44)	0.45 (0.44)

Table 64 - Gradient Boosting Regressor results using the top 300 singular vectors retained from the tf-idf matrix, considering only unigrams (unigrams & bigrams)

400 singular vectors	Normal	Stemming	Lemmatization
R2	0.59 (0.61)	0.59 (0.61)	0.59 (0.61)
MSE	0.34 (0.33)	0.34 (0.33)	0.35 (0.33)
MAE	0.45 (0.44)	0.45 (0.44)	0.46 (0.44)

Table 65 - Gradient Boosting Regressor results using the top 400 singular vectors retained from the tf-idf matrix, considering only unigrams (unigrams & bigrams)

500 singular vectors	Normal	Stemming	Lemmatization
R2	0.59 (0.61)	0.59 (0.61)	0.58 (0.61)
MSE	0.34 (0.33)	0.34 (0.33)	0.35 (0.33)
MAE	0.45 (0.44)	0.46 (0.44)	0.46 (0.44)

Table 66 - Gradient Boosting Regressor results using the top 500 singular vectors retained from the tf-idf matrix, considering only unigrams (unigrams & bigrams)

8.2.4. Multi-Layer Perceptron results

1000 features	Normal	Stemming	Lemmatization
R2	0.58 (0.54)	0.59 (0.56)	0.58 (0.56)
MSE	0.35 (0.39)	0.34 (0.37)	0.35 (0.37)
MAE	0.44 (0.46)	0.43 (0.45)	0.44 (0.45)

Table 67 – MLP Regressor results using the tf-idf scores of the 1000 most frequent unigrams (unigrams & bigrams)

2000 features	Normal	Stemming	Lemmatization
R2	0.62 (0.62)	0.62 (0.59)	0.63 (0.58)
MSE	0.32 (0.32)	0.32 (0.34)	0.31 (0.35)
MAE	0.42 (0.43)	0.42 (0.44)	0.41 (0.44)

Table 68 - MLP Regressor results using the tf-idf scores of the 2000 most frequent unigrams (unigrams & bigrams)

3000 features	**Normal**	**Stemming**	**Lemmatization**
R2	0.63 (0.61)	0.64 (0.62)	0.63 (0.56)
MSE	0.31 (0.33)	0.30 (0.32)	0.31 (0.37)
MAE	0.42 (0.43)	0.41 (0.42)	0.41 (0.45)

Table 69 - MLP Regressor results using the tf-idf scores of the 3000 most frequent unigrams (unigrams & bigrams)

100 singular vectors	**Normal**	**Stemming**	**Lemmatization**
R2	0.65 (0.67)	0.64 (0.66)	0.64 (0.66)
MSE	0.30 (0.28)	0.30 (0.28)	0.30 (0.29)
MAE	0.42 (0.40)	0.42 (0.40)	0.42 (0.40)

Table 70 – MLP Regressor results using the top 100 singular vectors retained from the tf-idf matrix considering only unigrams (unigrams & bigrams)

200 singular vectors	**Normal**	**Stemming**	**Lemmatization**
R2	0.63 (0.65)	0.62 (0.65)	0.63 (0.65)
MSE	0.31 (0.29)	0.32 (0.30)	0.31 (0.30)
MAE	0.42 (0.41)	0.43 (0.41)	0.42 (0.41)

Table 71 - MLP Regressor results using the top 200 singular vectors retained from the tf-idf matrix considering only unigrams (unigrams & bigrams)

300 singular vectors	**Normal**	**Stemming**	**Lemmatization**
R2	0.56 (0.64)	0.57 (0.64)	0.53 (0.63)
MSE	0.37 (0.30)	0.36 (0.30)	0.39 (0.31)
MAE	0.46 (0.42)	0.46 (0.42)	0.48 (0.43)

Table 72 - MLP Regressor results using the top 300 singular vectors retained from the tf-idf matrix considering only unigrams (unigrams & bigrams)

400 singular vectors	**Normal**	**Stemming**	**Lemmatization**
R2	0.52 (0.61)	0.52 (0.62)	0.52 (0.60)
MSE	0.40 (0.33)	0.40 (0.32)	0.40 (0.34)
MAE	0.48 (0.44)	0.48 (0.43)	0.48 (0.44)

Table 73 - MLP Regressor results using the top 400 singular vectors retained from the tf-idf matrix considering only unigrams (unigrams & bigrams)

500 singular vectors	**Normal**	**Stemming**	**Lemmatization**
R2	0.52 (0.61)	0.53 (0.60)	0.48 (0.60)
MSE	0.40 (0.33)	0.40 (0.34)	0.43 (0.34)
MAE	0.48 (0.44)	0.48 (0.44)	0.51 (0.44)

Table 74 - - MLP Regressor results using the top 400 singular vectors retained from the tf-idf matrix considering only unigrams (unigrams & bigrams)