University of Padova

Department of Mathematics

Master Thesis in Data Science

Optimal initial bid and new keywords suggestion in Sponsored Search Auctions

Supervisor

Prof. Michele Rossi

University of Padova

Co-supervisor

Dr. Marco Boresta

ACT Operations Research

Master Candidate

Giuliano Squarcina

Academic Year

2020-2021

Abstract

The suggestion of profitable new keywords and the estimation of the optimal initial bid for each of them, are two crucial problems that every *bidding strategy* has to address in order to maxime profits in *Sponsored Search Auctions* (SSA).

In this thesis, both problems are solved from the perspective of an economic agent not considered in the research developed so far: the *broker*. It is an intermediary between search engine and advertisers, purchasing *ad slots* from the former and selling them to the latters.

The optimal initial bid is cast as a regression problem, where regressors are the keyword itself (encoded in a numerical vector) and a set of features known a-priori, provided by the search engine. The dependent variable is the actual bid observed. The complex highly non-linear relationship is learnt by a Fully Connected Neural Network. The observations used for training and testing are those belonging to keywords with a positive average profit, in order to ensure that the relation learnt leads to positive profitability.

The new keywords suggestion problem is solved defining a general web scraping method, designed to work on almost every webpage returned from a query, followed by keywords extractors algorithms to retrieve relevant terms. A filter is created to clean the extracted keywords, improving the quality of suggestions.

The proposed neural network reaches a prediction error in the test set of just 0.16\$, while performance of the new keywords suggestion tool is assessed by human evaluators, using broadly adopted metrics in the field (*Relatedness*, *Non-obviosness* and *F-score*). The best keywords extractor model reaches an F-score higher than 0.6 for every scrap level and number of suggestions requiered.

The synergy between the new keywords suggestion tool, able to provide related but non-obvious (and so cheap) keywords, and the optimal initial bid tool, able to estimate a reliable starting bid, should increase the overall broker's profit.

Contents

Listing of figures

Listing of tables

Listing of acronyms

ad . . . . . advertisement

CP . . . . . Change Point

CPC . . . . . Cost Per Click

IDCG . . . . . Ideal Discounted Cumulative Gain

MAP . . . . . Mean Average Precision

MRR . . . . . Mean Reciprocal Rank

nDCG . . . . . n-Discounted Cumulative Gain

POS . . . . . Part Of Speech

PPC . . . . . Pay Per Click

QS . . . . . Quality Score

RAKE . . . . . Rapid Automatic Keyword Extraction

RPC . . . . . Revenue Per Click

SERP . . . . . Search Engine Results Page

SSA . . . . . Sponsored Search Auction

VCG . . . . . Vickrey Clarke Groves

YAKE! . . . . . Yet Another Keyword Extractor!

1 Introduction

1.1 Presentation Of The Problem

The problem presented hereafter arises in a real world business project, developed by ACTOR, on the behalf of a Media Technology and Digital Advertising company, which acts as a broker in *Sponsored Search Auction* (SSA). In detail, the problem consists of two business related sub-problems:

1. the recommendation of profitable new keywords,
2. the estimation of the optimal initial bid for each of them.

In order to better understand the problem, it is useful to introduce the framework of SSA and the concept of *bidding strategies*.

The *bidding strategy* (BS) arises in the context of SSA, which is a virtual auction where advertisers try to get a position on the *search engine results page* (SERP) presenting a bid for each relevant keyword. Every time a query is submitted, an instantaneous auction is done among all bids associated with keywords matching the query. In this way, a position to each *advertisement* (ad) is assigned. The ranking of an ad is determined by a combination of its bid and quality score, which is meant to capture the ad's relevance to the keyword. The value of the bid is crucial: if the bid is too low, it appears in a low rank position with a low probability to be clicked; if the bid is too high, the profit is reduced and could even turn into a loss. The *quality*

score (QS) is also important: given the same bid, the ad with higher quality wins. The auction mechanism is *Generalize Second Price* (GSP): the advertiser j, who gets position i, pays the bid presented by advertiser who ranked one position below ($i+1$) divided by QS_j. The payment is *Pay Per Click* (PPC): only if the sponsored link is clicked, the advertiser pays the search engine for sending the user to its webpage. The BS is said to be optimal for a reference economic agent (e.g. advertiser) if it maximizes his cumulated future profits.

The *broker bidding strategy* (BBS) is the extension of the above framework inserting an intermediate economic agent (broker) between the search engine and the advertisers. The broker bids on various keywords to get its sponsored links placed in a good position in SERP. When a customer clicks on the broker's link (*click-in*), he lands on a webpage where several outgoing links belonging to different advertisers are listed. The outgoing links' positions are determined through an auction, where the seller is the broker. He pays the *Cost per Click* (CPC), to the search engine, for each click-in and gets a *Revenue per Click* (RPC), from the advertisers, for each *click-out*. The profitability is not simply determined by setting RPC $>$ CPC, but depends on the ratio click-out/click-in, that is, the number of click-out for one click-in. Note that the CPC can only be estimated by excess (it is upper-bounded by the broker's bid) while the RPC cannot be estimated at all (it depends on the advertisers' bids which are unknown before the eventual click-out). The BBS is said to be optimal for the broker if it maximizes his cumulated future profits. In this work, the reference economic agent is the broker.

It is important to notice that the problem presented at the beginning of the current section is part of the BBS problem. In fact, to maximize his profits, the broker has to find first of all the most profitable keywords. Second, he has to determine the optimal bidding for each of them for every time period in the forecast horizon. Regarding this last point, the goal of this work is limited to estimating only the optimal *initial* bid. The reason is that exists an optimizing tool, the *CPC-optimizer*, developed by ACTOR in the business project perimeter, able to dynamically adjust an existing bid, but not able to propose an optimal initial bid. Hence, it is necessary to create a tool complementary to the *CPC-optimizer*.

1.2 Literature Review

An in-depth review about SSA, their mechanisms, role in business, evolution and technology which supports them, can be found in [1].

A broad scientific literature exists, which tries to find an optimal BS tackling the problem in different research fields: optimization, game theory, machine learning and a combination of

them.

In a paper of 2004 [2], Kitts and Leblanc present a trading agent for PPC auctions who uses an explicit profit objective function. The agent employs reinforcement learning to create a look-ahead plan of its desired bids, allocating resources among auctions and through periods subject to budgets and positions constraints in order to maximize the overall profits.

In [3], Edelman et al. investigate the GSP and compare it with *Vickrey-Clarke-Groves* (VCG) auction mechanism. If a search engine offered only one advertisement per result page, GSP and VCG would be equivalent. With multiple positions available, GSP auction is no longer equivalent to the VCG one and does not have *truth-telling behavior* as an equilibrium in dominant strategies. In other words, it is not in the best interest of the participating advertisers to bid their true valuation of a click and so are forced to elaborate a bidding strategy.

Cary et al. [4] show a greedy bidding strategy that a software might use in a repeated single keyword auction in order to maximize its utility, assuming to know the other competitors' bids in the next round. Given a target advertising slot s at price p_s, and the one position above slot $s-1$ with price p_{s-1}, the GSP mechanism allows a range of bid values that result in the same outcome (any bid between p_s and p_{s-1}). If all the advertisers choose their next bid b to be indifferent between successfully winning the targeted slot s at price p_s, or winning a slightly more desirable slot at price b, there exists an equilibrium where the payments to the search engine are identical to those of the VCG mechanism.

In [5], a single keyword bidding strategy is optimized over multiple periods with a fresh budget B allocated at the beginning of each period to reflect the manner in which advertisers actually specify their budgets in real SSA. Specifically, the problem is formulated as a Markov Decision Process (MDP) defined as $M = (S, [A_s]_{s \in S}, \mu, r)$ where S is a set of states and A_s are the set of actions available to the agent in each state s. For $a \in A_s$, $\mu(a, s, s')$ is the probability of transitioning from state s to state s' when taking action a in state s. $r(a, s, s')$ is the expected reward received after taking action a in state s and transitioning to state s'.

Powell et al. in [6] formulate the learning of the optimal bidding policy for SSA as a stochastic optimization problem in a single ad campaign. The agent tries to learn the *bidding policy* which maximizes the cumulative profits under different *learning policies* from a dataset of hourly frequency data.

Zhang et al. [7] set a more realistic scenario where the advertiser can create a number of campaigns and set a budget for each of them. In a campaign, he can further create several ad-groups with bid keywords and bid prices. The problem is formulated as a constrained optimization, which takes the campaign budgets and the keyword bid prices as variables and finds their opti-

mal values by maximizing the advertiser revenue, with the constraint of the account-level budget.

Some works tackle the BS from the search engine's perspective. In [8], a neural network is used to predict bidders' behavior using a set of features that are observable by the bidder. Based on the bidder behavior model, a Markov Decision Process (MDP) is defined and solved with reinforcement learning techniques.

A different line of research focuses on the choice of the set of keywords. Some techniques broadly adopted are presented in the next paragraphs.

Many high ranked websites include relevant keywords in their meta-tags. A meta-tag spider, like *Wordtracker* [9], queries search engine for seed keywords and extracts meta-tag words from these highly ranked webpages.

Another approach to extract related words is to use the Metacrawler Search Network's related keyword lists. Search engines maintain a list of few related keywords used for query expansion. To gather more words, current tools re-spider the first list of resulting keywords. This gives popular keywords closely related to the base keyword, but the number of relevant keywords generated is still low.

Proximity-based tools issue queries to a search engine to get highly ranked webpages for the seed keyword and expand the seed with words found in its proximity. For example for the seed keyword "hawaii vacations", this tool will find keywords like: "hawaii family vacations", "discount hawaii vacations", etc. Though this tool finds a large number of keywords, it cannot find relevant keywords not containing the exact seed query words.

Google Ads [10] relies on query log mining for keyword generation. It presents frequent queries that contain the entire search term. Similarly, *Overture*'s Keyword Selection Tool lists frequent queries of recent past containing the seed terms. Both these techniques suffer from two main drawbacks. The first one is proximity-based searches, i.e. failure to generate relevant keywords not containing search terms. To generate additional keywords, Google Ads mines advertiser logs. The second one is that suggestions are limited to those words that occur frequently in advertiser search logs. Such frequent words have a good chance of being among expensive keywords, as they are already popular in the advertising community.

Moreover, all above techniques fail to take semantic relationships into account. Uncommon relevant terms, not containing the input query term, are often ignored. In [11], Joshi and Motwani develop a new technique called *TermsNet*, which represents each term and its context with a document containing text-snippets from top 50 search-hits for that term. Then, it determines relevance between terms and captures their semantic relationships as a directed

graph. Neighbors of a term in such a graph are used to generate the common as well as the non-obvious keywords related to a term. For evaluating results, the authors use the metrics *Relatedness* and *Non obviousness* and compute an approximation of the *Recall*. The three measures are also combined, two at a time, in an *F-score*. *Relatedness* and *Non obviousness* are binary variable, taking values $\{0, 1\}$. The former is evaluated by humans, while the latter is evaluated automatically, defining a hard rule to assess originality: if the suggestion does not contain the query keyword, part of it, or its variants sharing a common stem, than it is non-obvious.

Rusmevichientong and Williamson [12] proposed an algorithm that adaptively identifies the set of keywords to bid on based on historical performance (profit-to-cost ratio) in order to maximize total expected profits, given a fixed daily budget and unknown click-through probabilities. The latter can only be estimated by selecting the keyword and observing the actual click- through. This process can result in significant costs, yet may offer an opportunity to discover potentially profitable keywords. Given the daily budget constraint, it is necessary to balance the tradeoff between selecting keywords that seem to yield high average profits based on past performance, and selecting previously unused keywords in order to learn about their click-through probabilities. This is usually cast in the machine learning literature as balancing "exploitation" and "exploration".

In [13], the authors start from the observation that the search volume of queries exhibits a long-tailed distribution, meaning that a large number of terms with low search volume cumulatively make up a significant share of the total traffic. Hence, it could be possible to match the search volume of a popular (and so very expensive) keyword using many unpopular and cheap keywords, at a fraction of the cost. The authors propose a method that can be used by an advertiser to generate relevant keywords given his website. An extended dictionary of keywords is constructed by first crawling the webpages in this website and then expanding the set with search results from a search engine. In order to find relevant terms for a query term, the semantic similarity between terms in this dictionary is established. The similarity graph thus generated is traversed by a watershed algorithm that explores the neighborhood and generates novel suggestions for a seed keyword.

Chen et al. [14] propose a novel keywords' suggestion method that fully exploits the semantic knowledge among "concept hierarchy" (a hierarchically organized concept set) overcoming the existing models based on some statistical information (for example, the keyword co-occurrence). Given a keyword, it is first matched with some relevant concepts. Then the relevant concepts are used with their hierarchy to fertilize the meanings of the keywords. Finally, new keywords are suggested according to the concept information.

In [15], the criterion for choosing the set of keywords K is to pick the ones that searchers may use in their queries when looking for their products. Since it is impossible for the advertisers to identify all possible variations of keywords, they often rely on *broad matching* provided by search engines. The problem is that the same keyword could be associated, as a result of the broad match, with query q and query q', which have different values v_q and $v_{q'}$ for the advertiser.

In [16], Thomaidou and Vazirgiannis follow a similar method as in [13], extracting relevant terms consisting of two or three words from the advertiser's landing page and expanding the suggestions using search results snippets. The HTML content of each landing page is parsed, stopwords are removed, and the most significant tokens are extracted and ranked on the basis of the occurrences of each token in each type of HTML tag, which have different weights according to their importance. Single tokens are combined to form pairs and triplets of words, whose weighted co-occurrence score is greater than a fixed threshold. Moreover, an additional measure aside *Relatedness* and *Non obviousness* is suggested: *Specificity*. It quantifies how original are the suggested keywords with respect to the query one. The authors define a graded scale of 5 different levels for each metric and rely exclusively on human evaluators to score the results.

At the time of writing, the BBS problem is not addressed explicitly in the literature. Anyway the broker plays two roles simultaneously:

1. the advertiser (he purchases ad slots from the search engine),
2. the auctioneer (he sells ad slots to the advertisers).

Hence many results obtained in the BS can be extended to the BBS framework.

In the literature presented so far, the problem of the *optimal initial bid for new keywords* is embedded in the broader BS problem, which is solved relying on assumptions that often do not hold in the real world.

For example, in [2], the agent knows the prices that each competitor is going to pay for each position on every future auction for every keyword of intestest. In [4], the strategy is limited to work for a repeated auction on a single keyword and the agent is the only player who updates his bid for the next round, while the others merely repeat their previous bid. In [6], the optimal bidding policy is defined for a single ad campaign, while in the real world, advertisers manage multiple campaigns with multiple budgets simultaneously.

Instead, in the current thesis, the *optimal initial bid for new keywords* problem is tackled as a stand-alone one, following a pragmatic approach based on available data rather than looking

for a new model able to provide a theorethically sound solution, but pratically infeasible in a real world business framework. The same logic is followed also for the *new keywords suggestion* problem, where existing literature has developed both theorethical models and empirical tools. Our interest focuses on the second ones, which are used as a source of inspiration to define a new keywords extraction procedure.

1.3 Thesis Content

The objective of this work consists in optimizing the bidding strategy of a digital advertising broker in SSA. In detail, the contribution of the thesis is based on the development of two models, able to, respectively:

1. estimate the optimal initial bid for new keywords;
2. suggest profitable new keywords.

Both tools are designed to work in synergy with the CPC-Optimizer, already developed in the project perimeter by ACTOR.

The thesis is organized as follows. Part I addresses the *optimal initial bid estimation problem*. The collection and exploration of the available data, the explanation of the bids' time-series behavior, the preprocessing steps necessary to build the training dataset, are presented in Chapter 2.

In Chapter 3, the optimal initial bid is cast as a regression problem. First, some baseline models (multiple linear regression, Ridge, Lasso) are used and their prediction error is analyzed. Second, a Fully Connected Neural Network is trained to solve the same task. An empirical approximation of the *training error lower bound* is provided and the overall training procedure followed to fine tune the hyperparameters and check the soundness of different features, carefully explained. The higher network's complexity is justified by a significant drop in the prediction error, which is $0.15\$$. The results are counter-checked against the *negative test set*: the set of observations with negative profitability belonging to the same keywords used in the test set. The low error ($0.17\$$) highlighted the need to change the level of granularity at which the profitability is assessed: from *observation level* to *keyword level*.

In Chapter 4, an alternative dataset is built, selecting all observations belonging to keywords with positive average profit. The same training procedure and error analisys is repeated, leading to a similar prediction error ($0.16\$$). The prediction error in the *negative test set* is now almost

doubled ($0.30\$$), confirming the network's capacity to predict bids, for negative keywords, significantly different from the observed ones. Since the *training error lower bound* makes not possible to improve the model in terms of performance, the model is improved in terms of complexity. A new encoding technique, GloVe, reduces drastically the number of features preserving similar prediciton error ($0.17\$$).

Concluding remarks for Part I are reported in Chapter 5.

Part II tackles the *Profitable New keywords' suggestion problem*. In Chapter 6, the importance to develop a procedure able to suggest new keywords related to the query, non-obvious, with a good quantity and variety and, as far as possible, language independent, is explained, highlighting its positive impact on the broker's profits. In a nutshell, the procedure is based on two steps: *web scraping* and *keywords extraction*. Given the query, the former retrives the raw tex scraping the top URLs' webpages resulting from a Google search, using different sets of HTML tags. Based on which set is used, the depth to which the webpage is scraped varies, defining three different scrap levels (*super-light*, *light* and *full*). Then, *Keywords extraction* receives as input the cleaned text and identifies the keywords. An overview of the main types of keywords extractors models available in the literature is presented, highlighting pros and cons of each of them and motivating the chosen ones: *RAKE* and *YAKE!*. Before concluding the chapter, a *BERT Filter* is developed in order to increase the relevance of the keywords extracted by both models. It works by encoding the query keyword and each suggestion using "BERT-Large Uncased" model and then computing the cosine similarity between the query and each of the suggestions. This tool moves suggested keywords, very similar to the seed, on top positions, which could be positive (avoid non relevent suggestions) but even negative (increase obviousness of suggestions). This shows the need to develop a methodology to evaluate systematically the keywords extracted for all the possible combinations of scrap levels, models and number of keywords returned.

This is done in Chapter 7, where an overview of the available metrics is presented, highlighting pros and cons of each of them and selecting the most suitable ones for the problem at hand: *Relatedness@K*, *Non obviousness@K* and *F-score@K*. Then the selected metrics are used to evaluate the performances of all possible combinations of scrap levels, models and number of suggested keywords. Results are presented graphically and in tabular format, highlighting the most important observations and defining the best model for each scrap level. It turned out that a single winning model does not exist, but it also emerged that *YAKE!+BERT* is the model with the most robust performance across different scrap levels and number of suggestions.

Final observations for Part II are reported in Chapter 8.

Part I

Optimal Initial Bid for New Keywords

2

Dataset Design

The chapter introduces and explores the data available in order to estimate the *optimal initial bid* for new keywords. It defines what is a good observation (*ground truth*) according to business criteria, encodes the keywords using BERT model and reorganizes the data in a new dataset useful for training the model in the next chapter.
In detail, it is composed by four sections:

- Data Collection: the available data are listed and described.
- Data Exploration: relevant statistics are computed and empirical evidence is used to understand the data's behavior.
- Preprocessing Steps: the *ground truth* is defined and the data are filtered; the keywords are encoded.
- Dataset Construction: the final dataset is assembled.

2.1 Data Collection

The data available in the broker's IT system are heterogeneous in terms of time frequency (hourly, daily) and granularity (campaign, adgroup, keyword level). The choice about which data to use is not free but based on the broker's actual behavior and on the problem at hand:

- bids are updated with daily frequency;
- in order to build a model able to estimate the optimal initial bid at the keyword level, the finer granularity is required.

The features relevant for the analysis are listed in Table 2.1. They are clustered by data source, identified by name and, where needed, the corresponding formula and comment are provided. Features known *ex ante* (so also for a new keyword) are marked with an asterisk (*). These features are of primary importance since are the only ones which can be used to estimate the `current bid` for a new keyword. Instead, all other features are known *ex post*, after historical statistics have been collected. Anyway they are useful for discriminating positive versus negative `current bid` and so in order to identify the *groud truth*.

Table 2.1: Overview of the features used in the analysis

Source	Feature	Formula	Comment
GOOGLE	`date*`	-	This feature allows to order chronologically the observations for each keyword, which is needed to analyze the timeseries. The time period format is yyyy-mm-dd.
	`keyword ID*`	-	The numerical code needed to identify uniquely a keyword.
	`keyword name*`	-	The string representing the keyword
	`match type*`	-	The type of match allowed when a query on the search engine is done by a potential customer: Exact, Phrase or Broad.
	`first impression date`	-	The first time the broker's link associated with the keyword appears in the SERP. This feature can be used in place of the origin date, since it represents the point in time when the keyword starts receiving some visibility and so starts having some probability to be clicked.
	`impressions`	-	The number of times an ad has been displayed on the SERP.
	`current bid`	-	The bid associated with the keyword.
	`clicks in`	-	The number of times a potential customer clicks on the broker's incoming link, so landing on the broker's website.
	`clicks out`	-	The number of times a potential customer clicks on the broker's outgoing link, so landing on the website of an advertiser.

Table 2.1: Overview of the features used in the analysis *(Continued from previous page)*

Source	Feature	Formula	Comment
	ctr in	$\frac{\text{clicks in}}{\text{impressions}}$	The click-through-rate-in is the proportion of impressions which turn in a click-in.
	ctr out	$\frac{\text{clicks out}}{\text{clicks in}}$	The number of clicks-out for one clicks-in.
	revenue	-	The daily total revenue of the keyword.
	cost	-	The daily total cost of the keyword.
	observed profit	revenue−cost	The daily total profit of the keyword.
	profit-cost ratio	$\frac{\text{observed profit}}{\text{cost}}$	The amount of profit for one dollar of cost. It is known in performance analisys as the Return On Investment (ROI).
MIMIR	date*	-	This feature allows to order chronologically the observations for each keyword, which is needed to analyze the timeseries. The time period format is yyyy-mm-dd.
	keyword name*	-	The string representing the keyword
	suggested bid*	-	The bid suggested by GOOGLE for a new keyword.
	competition*	-	The level of competition on a scale from 0 to 1.
	search volume*	-	The daily volume of queries matching the keyword.

MIMIR is a service part of the broker's IT system which records, stores and provides a series of information (list of active keywords, list of proprietary web pages, the adgroup/campaign associated to each keyword, suggested bid, level of competition, etc.), provided by GOOGLE for any keyword. Hence, these data are always known *ex ante*.

In the market of digital advertisement, a `profit-cost ratio` ≥ -0.2 is considered acceptable: what matters is the overall profit of the adgroup to which the keyword belongs to; moreover even if a keyword has a mild negative profitability but generates good traffic volumes, it is awarded by Google with a good Quality Score, so it could get a higher position in the future SERP at a moderate cost. Quantify the traffic volume classifiable as "good" it's practically unfeasible. It depends on the keyword itself, the timeframe, the level of competition, etc. In practice a simple rule of thumb is used: every `profit-cost ratio` ≥ -0.2 is considered "positive". Anyway, even not considering the `profit-cost ratio` ≥ -0.2 a positive result, the CPC-Optimizer (Appendix A) is able to move effectively the `profit-cost ratio` above 0 if the latter is mildly negative (≥ -0.2).

A value of `clicks in` equal to zero for all the observations means that the sponsored link associated with the keyword has never been clicked. Hence `clicks out`, `revenue`, `cost` and `profit` are all zero. Such keyword has never run, so it cannot be judged as positive or negative. In this case the `profit-cost ratio` is also reported as zero in the IT system, even if mathematically is an undefined quantity. In other cases it could be actually zero. In order to distinguish the two cases, it's sufficient to check the value of `clicks in` feature.

For the above considerations, two conditions are required to select properly the timeseries:

1. `profit-cost ratio` ≥ -0.2;
2. `clicks in` > 0.

The analysis in Section 2.2 and 2.3 is based on the data with source GOOGLE and filtered in the following way: only timeseries of `current bid` with at least one observation satisfying both conditions are taken. This is equivalent to discard all timeseries that have never run and so have no performance to judge. The data from MIMIR are added in Section 2.4.

2.2 Data Exploration

The dataset used in this section and next one (Section 2.3) refers to a subset of new keywords used to test the CPC-Optimizer's effectiveness in adjusting the `current bid`. It spans a period

of approximately five weeks (4th February-15th March) and around 10 thousand keywords. For clarity, we refer to it as `new keyword dataset`. These data are useful for the following reasons:

1. data are available since the inception. This is not true for older keywords, since data are not conserved for the full history but only for the last few months;
2. the CPC-Optimizer is active and used to adjust automatically the bid for each keyword with a daily frequency.

The importance of both points will be clear in the following sections.

The current strategy, adopted in order to find the optimal bid for a new keyword, is called *bottom-up*: it consists in setting a very low bid (e.g. $\frac{1}{32}$ of the `suggested bid`) and then, if no clicks-in are recorded, increasing the bid by 10% until at least one click-in happens. This strategy is motivated by the following reasons:

1. the bid suggested by the search engine is not reliable;
2. there is no tool currently available able to estimate a good bid for a new keyword;
3. setting high bids generates unsustainable losses, since usually thousands new keywords are launched together.

Clearly, this strategy is not efficient, since many days of adjustments could be required to reach a bid sufficiently high to generate some traffic and additional days could be required to find a profitable bid value, if any. These considerations motivate the need to build a model able to suggest an enough good starting bid for a new keyword.

The *bottom-up* strategy should be visible in the timeseries of bids which should be composed by two phases:

- a 1° phase characterized by a stable positive trend (essentially a straight line with slope 0.1);
- a 2° phase where the bid is progressively adjusted and converges toward its optimal value.

Adjustments in both phases are automatically implemented through the CPC-Optimizer. Note that the 1° phase is non-stationary, due to the presence of a trend; the 2° phase is, or should become quickly, covariance stationary. The evidence of this behavior has to be checked against the data, since reality could be more complex and substantially different. For example, some

initial bids could be still too high, generating losses; the bid optimal value could not exist in practice, since it could change continuously and abruptly due to external factors (e.g. traffic volume, period of the year, etc), leading to a non-stationary 2° phase.

In order to check the behavior of thousands of timeseries it's needed some form of automation. The `ruptures` Python package is a useful tool in this sense. It provides a series of algorithms, with different search methods and cost functions, able to scan a timeseries and find the *change points* (CP), if any. If the timeseries exhibit actually two phases (1° phase non-stationary and 2° phase stationary), then it's reasonable to expect one and only one CP detected for each timeseries (or at least in the great majority of timeseries). Note that even if the number of CPs is equal to 1, this does not imply that the 1° phase is non-stationary and the 2° phase is stationary, but just that the timeseries shows two distinct phases. Hence, the existence of a single CP is a necessary but not sufficient condition to claim the existence of a non-stationary phase followed by a stationary one. So, if a number of CPs equal to 0 or greater than 1 is found, than it is a clear proof against the idea that exist two distinct phases. If the number of CPs is equal to 1, a graphical inspection it's needed to verify if it's reasonable or not to claim that the 1° part is non-stationary while the 2° part is stationary.

More in detail, `ruptures` is designed to detect the CPs, which are points where the timeseries changes its behavior, which means mainly: changes trend, changes variance and jumps. The basic idea is to segment the timeseries in a number of non overlapping timeframes in order to minimize a suitable cost function. The problem, which in literature is called *Change Point Detection*, has two settings:

1. number of CPs is unknown;
2. number of CPs is known

and each of the problems can be solved with exact or approximate methods. In this work, the goal is to verify if it's true that timeseries show two different phases and so just one CP. Hence, the number of CPs is unknown and the goal is to find if it is actually equal to one. It's possible to choose among different algorithms, but the chosen ones are based on window-sliding (`Window`) and binary segmentation (`Binseg`) for their simplicity. Both follow a sequential detection approach, meaning that they return a single CP estimate $\hat{t}^{(k)}$ at the k-th iteration and the sequential algorithm is run until an appropriate stopping criterion is met [17].

Both the algorithms have the following characteristics:

1. *problem formulation*: the timeseries $y = y_1, \ldots, y_T$ is assumed to be piecewise stationary, meaning that some characteristics of the process change abruptly at some unknown

instants $t_1^* < t_2^* < \cdots < t_K^*$. CPs detection consists in estimating the indexes t_k^*. In our setting, the numbers of CPs is unknown.

2. *criterion function*: CPs are chosen in order to find the best segmentation $\mathcal{T}$ (the set of CPs) able to minimize the criterion function $V(\mathcal{T}, y)$, which is defined as the sum of costs of all the segments that define the segmentation:

$$V(\mathcal{T}, y) = \sum_{k=0}^{K} c(y_{t_k^* \ldots t_{k+1}^*})$$

where $c(\cdot)$ is a cost function which measures goodness-of-fit of the model to each segment of the timeseries. The cost function $c(\cdot)$ is a measure of "homogeneity". Its choice encodes the type of changes that can be detected. Intuitively, $c(y_{a\ldots b})$ is expected to be low if the sub-signal $y_{a\ldots b}$ is "homogeneous" (meaning that it does not contain any change point), and large if the sub-signal $y_{a\ldots b}$ is "heterogeneous" (meaning that it contains one or more change points). The cost function we are going to use is the quadratic error loss (c_{L_2}) :

$$c_{L_2}(y_{a\ldots b}) = \sum_{t=a+1}^{b} \|y_t - \bar{y}_{a\ldots b}\|_2^2$$

where $\bar{y}_{a\ldots b}$ is the empirical mean of the sub-signal $y_{a\ldots b}$.

3. *objective function*:

$$F\Big(y, \mathcal{T}, pen(\mathcal{T})\Big) = V(\mathcal{T}) + pen(\mathcal{T})$$

The problem consists in minimizing the objective function with respect to the segmentation $\mathcal{T}$:

$$\min_{\mathcal{T}} \; V(\mathcal{T}) + pen(\mathcal{T})$$

4. *penalty*: when the number of CPs is unknown, a constraint is added in the form of a complexity penalty, which is an appropriate measure of the complexity of a segmentation $\mathcal{T}$ and is indicated as $pen(\mathcal{T})$. Its role is to balance the goodness-of-fit term $V(\mathcal{T}, y)$, otherwise it would be possible to minimize the objective function $F\Big(y, \mathcal{T}, pen(\mathcal{T})\Big)$ just increasing the number of change points. The choice of the complexity penalty is related to the amplitude of the changes to detect: with a too "small" penalty (compared to the goodness-of-fit), many CPs are detected, even those that are the result of noise. Conversely, with a too high penalty (compared to the goodness-of-fit), only the most significant changes (if any) are detected. Beyond the general formulation, the penalty used in this work is defined as:

$$pen_{BIC,L_2}(\beta) = \beta \cdot \sigma^2 \cdot log\, T$$

where σ is the standard deviation, T is the number of samples and β is a scale parameter. Note that the penalty is not a function of $\mathcal{T}$, but a constant. It will be used as a lower bound to decide if the CP t_k^* proposed at the k-th iteration leads to a drop in the criterion function (also called *gain* $G_{t_k^*}$) big enough to justify the addition of t_k^* to the set of CPs $\mathcal{T}$. Hence the gain is computed as:

$$G_{t_k^*} = V(\mathcal{T}, y) - V(\mathcal{T} \cup t_k^*, y)$$

and the CP is added to $\mathcal{T}$ if and only if $G_{t_k^*} > pen_{BIC,L_2}(\beta)$. It's important to try different values of β in order to find the penalty that empirically works enough well given the available data.

Now the two algorithms chosen are briefly presented in order to show the idea behind them.

2.2.1 Window-Sliding

It consists in computing the discrepancy between two adjacent windows that slide along the signal y. For a given cost function $c(\cdot)$, the discrepancy between two sub-signals is given by:

$$d(y_{a\ldots t}, y_{t\ldots b}) = c(y_{a\ldots b}) - c(y_{a\ldots t}) - c(y_{t\ldots b}) \quad \text{for} \quad (1 \leq a < t < b \leq T)$$

If the $y_{a\ldots b}$ is homogeneous, the cost function computed over two consecutive windows $c(y_{a\ldots b})$ would be very similar to the sum of the cost functions of each window $c(y_{a\ldots t}) + c(y_{t\ldots b})$. If the $y_{a\ldots b}$ is not homogeneous, the cost function computed over two consecutive windows $c(y_{a\ldots b})$ would be significantly greater than the sum of the cost functions of each window $c(y_{a\ldots t}) + c(y_{t\ldots b})$. A graphical representation is depicted in Figure 2.1. The Window-Sliding algorithm is presented in Algorithm 1. Note that it is important to choose the *window size* value which works empirically well on available data. The `PKSearch`(Z) is not illustrated for brevity, but it does the following:

1. sorts the score list Z in ascending order;
2. extracts the last element in Z (we call it Z_{max}) and compute the gain of the corresponding CP $\hat{t}_{Z_{max}}$:
$$G_{\hat{t}_{Z_{max}}} = V(\mathcal{T}, y) - V(\mathcal{T} \cup \hat{t}_{Z_{max}}, y);$$
3. stops if $G_{\hat{t}_{Z_{max}}} \leq pen_{BIC,L_2}(\beta)$.

Note that, if the number of CPs is known a priori to be N (not our case), the *stopping criterion* is simply $|L| = N$.

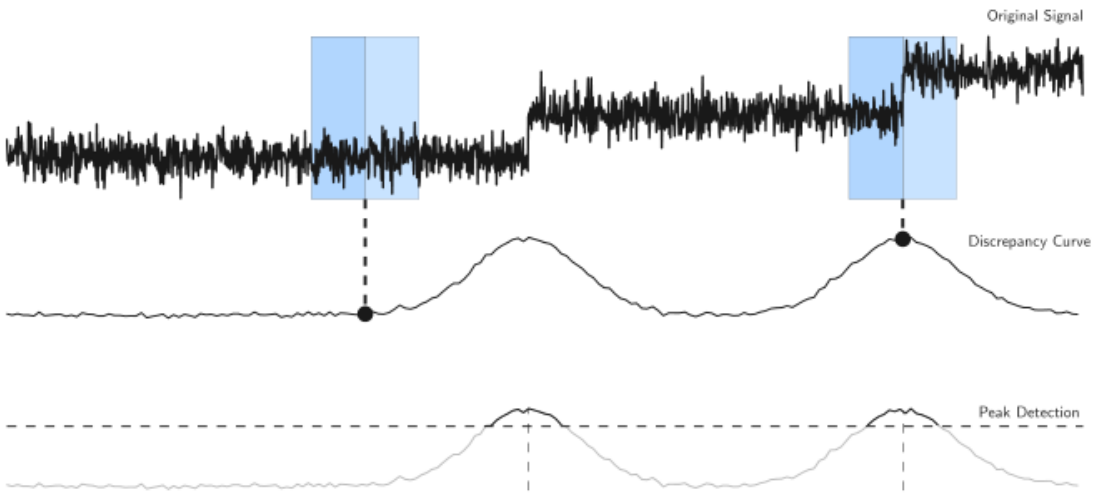


Figure 2.1: Window-Sliding.

Algorithm 1: Window-Sliding

Input: signal $\{y_t\}_{t=1}^{T}$, cost function $c(\cdot)$, half-window width w, peak search procedure `PKSearch`.

Initialize $Z \longleftarrow [0, 0, \dots]$ a T-long array filled with 0 `// Score list`

for $t = w, \dots, T - w$ **do**

$p \longleftarrow (t - w) \dots t$

$q \longleftarrow t \dots (t + w)$

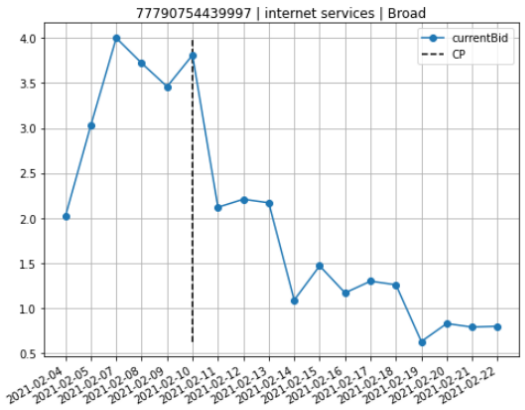
$r \longleftarrow (t - w) \dots (t + w)$

$Z[t] \longleftarrow c(y_r) - [c(y_p) + c(y_q)]$

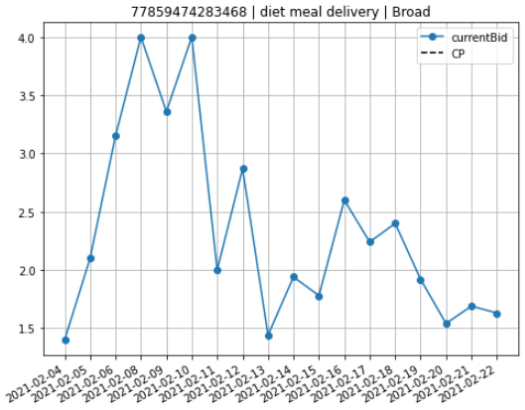
end

$Z[t] \longleftarrow$ `PKSearch`(Z) `// Peak Search Procedure`

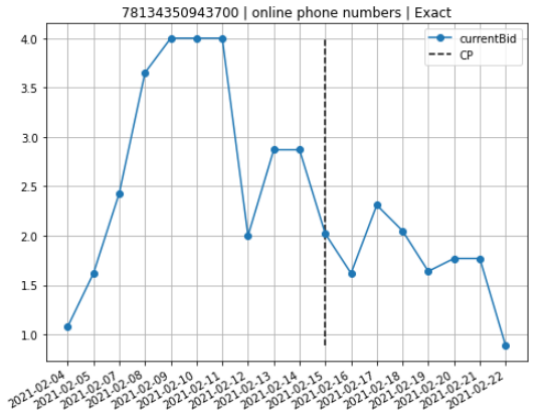
Output: set L of estimated breakpoint indexes.



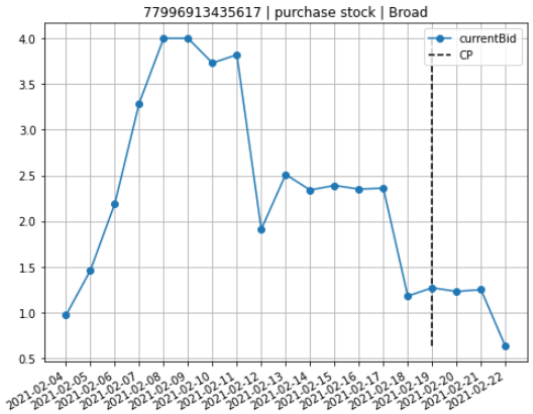
(a) Example n° 1



(b) Example n° 2



(c) Example n° 3



(d) Example n° 4

Figure 2.2: Examples of CPs detection using Window-Sliding.

Results

Figure 2.2 shows some timeseries and the corresponding CPs, detected using Window-Sliding algorithm. The goal is to illustrate the actual behavior of the timeseries in order to check if it is in line with the expected one described in Section 2.2. In all the plots, the initial `current bid` is set according to the *bottom-up* strategy and a positive trend is observed. What happens later it's not a simple convergence toward some long term optimal bid value. In Subfigure 2.2a, two phases are identified. In the first one, the `current bid` is increased until the upper bound is reached; in the second one, a strong negative trend brings the bid well below its initial value. In Subfigure 2.2b, no CPs are detected and so a unique phase is identified. In Subfigure 2.2c and 2.2d, similar considerations of Subfigure 2.2a are valid.

The values of hyperparameters $width$ and β have been selected using a heuristic approach:

for each value, the plot of timeseries with the corresponding CPs have been visually inspected in order to gauge the reliability of the proposed segmentation. This manual method is a compulsory choice given the absence of a metric usable to quantify the quality of the algorithm's output. Decreasing the parameter β, leads to an increase in the n° CPs, but not all these CPs are properly selected. In the dataset available the n° CP detected are 0 or 1 with frequency $\approx 37\%$ and $\approx 63\%$, respectively. Hence, in 63% of timeseries exist actually two phases, but many timeseries do not show a 2° phase stationary characterized by a long run optimal bid value.

In conclusion, the timeseries with:

- two phases;
- 1° phase non-stationary and 2° phase stationary

are a minority part. Data does not support the idea that timeseries should show a 1° phase non-stationary followed by a 2° phase stationary.

2.2.2 Binary Segmentation

It is a greedy sequential algorithm where the first CP, $\hat{t}^{(1)}$, is detected as:

$$\hat{t}^{(1)} = \underset{t_0<t<T}{\operatorname{argmin}} \underbrace{c(y_{t_0\ldots t}) + c(y_{t\ldots T})}_{V(\mathcal{T}=\{t\})}$$

This operation is "greedy", in the sense that it searches the CP that lowers the most the sum of costs (so at each iteration one and only one CP is returned). The signal is then split in two at the position of $\hat{t}^{(1)}$; the same operation is repeated on the resulting sub-signals until a stopping criterion is met. A graphical representation is depicted in Figure 2.3. The Binary Segmentation algorithm is presented in Algorithm 2. Note that:

- if the number of CPs is known a priori to be N (not our case), the *stopping criterion* is simply $|L| = N$;
- if the number of CPs is unknown, the *stopping criterion* is $G[\hat{i}] \leq pen_{BIC,L_2}(\beta)$. In this case the best CP $\hat{t}$ found at iteration k is associated with a gain $G[\hat{i}]$ too low to justify a more complex segmentation.

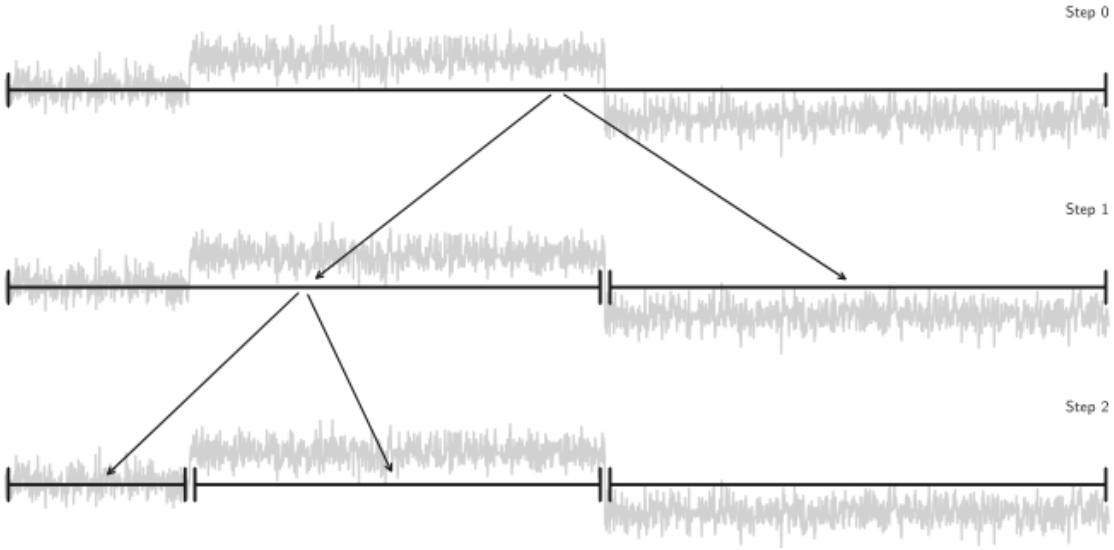


Figure 2.3: Binary Segmentation.

Algorithm 2: Binary Segmentation

Input: signal $\{y_t\}_{t=1}^{T}$, cost function $c(\cdot)$, stopping criterion.

Initialize $L \longleftarrow \{\}$ `// Estimated breakpoints`

repeat

  $k \longleftarrow |L|$ `// Number of breakpoints`

  $t_0 \longleftarrow 0$ and $t_{k+1} \longleftarrow T$

  if $k > 0$ **then**

    Denote by t_i $(i = 1, \ldots, k)$ the elements (in ascending order) of L; ie $L = \{t_1, \ldots, t_k\}$

  end

  Initialize G a $(k+1)$-long array `// List of gains`

  for $i = 0, \ldots k$ **do**

    $G[i] \longleftarrow c(y_{t_i \ldots t_{i+1}}) - \min_{t_i < t < t_{i+1}} [c(y_{t_i \ldots t}) + c(y_{t \ldots t_{i+1}})]$

  end

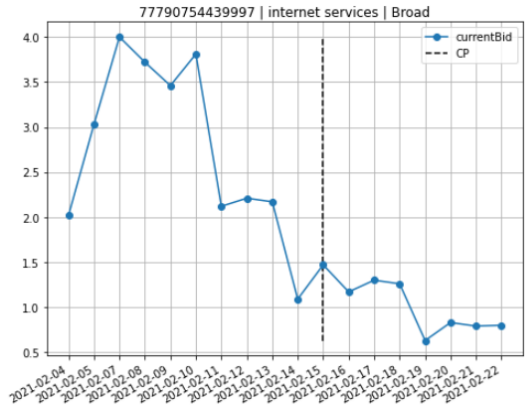
  $\hat{i} \longleftarrow \operatorname{argmax}_i G[i]$

  $\hat{t} \longleftarrow \operatorname{argmin}_{t_{\hat{i}} < t < t_{\hat{i}+1}} [c(y_{t_{\hat{i}} \ldots t}) + c(y_{t \ldots t_{\hat{i}+1}})]$

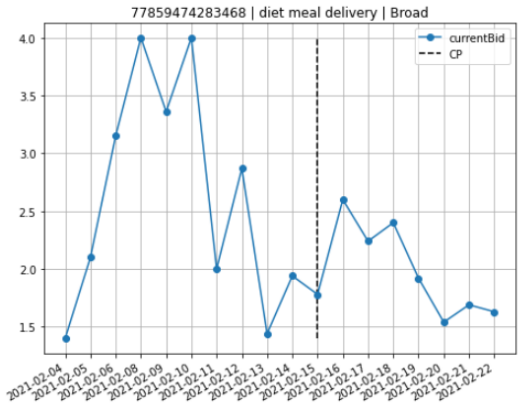
  $L \longleftarrow L \cup \{\hat{t}\}$

until *stopping criterion is met*

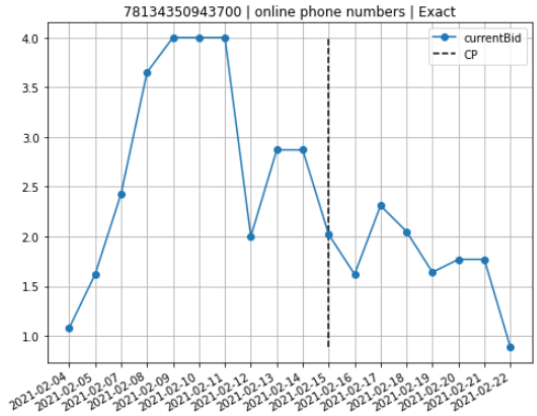
Output: set L of estimated breakpoint indexes.



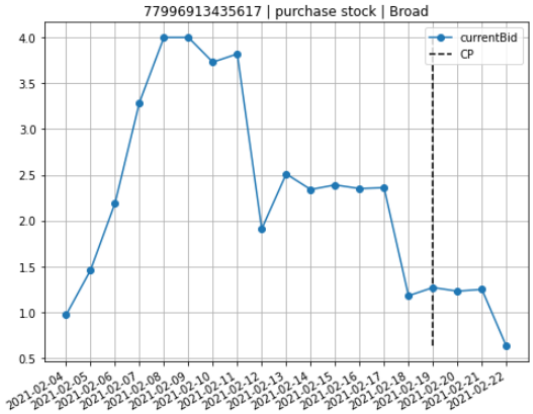
(a) Example n° 1



(b) Example n° 2



(c) Example n° 3



(d) Example n° 4

Figure 2.4: Examples of CPs detection using Binary Segmentation.

Results

Figure 2.4 shows some timeseries and the corresponding CPs, detected using Binary Segmentation algorithm. The goal is to illustrate the actual behavior of the timeseries in order to check if it is in line with the expected one described in Section 2.2. In all the plots, the initial `current bid` is set according to the *bottom-up* strategy and a positive trend is observed. What happens later it's not a simple convergence toward some long term optimal bid value. In Subfigure 2.4a, two phases are identified. In the first one, the `current bid` is increased until the upper bound is reached; in the second one, a strong negative trend brings the bid well below its initial value. Now, in Subfigure 2.4b, a CP is detected. In Subfigure 2.4c and 2.4d, similar considerations of Subfigure 2.4a are valid.

The values of hyperparameters $width$ and β have been selected using a heuristic approach: for each value, the plot of timeseries with the corresponding CPs have been visually inspected in order to gauge the reliability of the proposed segmentation. This manual method is a compulsory choice given the absence of a metric usable to quantify the quality of the algorithm's output. Decreasing the parameter β, leads to an increase in the n° CPs, but not all these CPs are properly selected. In the dataset available the n° CP detected are 0, 1, 2 and 3. In 82% of cases n° CP is 1. Again, this does not imply that the 1° phase is non-stationary and the 2° phase is stationary, but just that exist two phases.

In conclusion, the timeseries with:

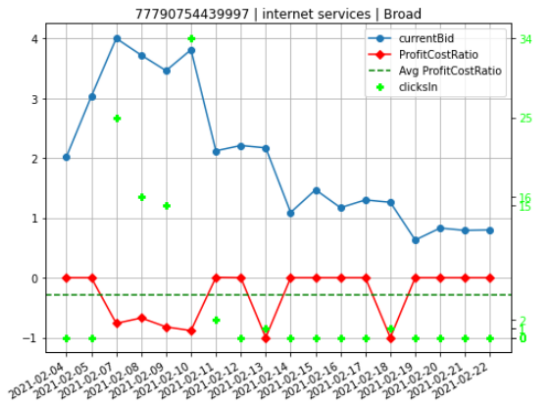
- two phases;
- 1° phase non-stationary and 2° phase stationary

are a minority part. Data does not support the idea that timeseries should show a 1° phase non-stationary followed by a 2° phase stationary.

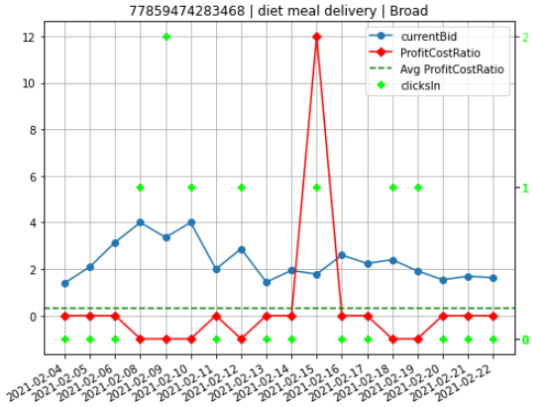
2.2.3 Timeseries' Behavior

The results in Section 2.2.1 and 2.2.2 have shown that the timeseries of the `current bid` do not follow in general the idea of a 1° phase non-stationary and a 2° phase stationary. Figure 2.5 shows some timeseries and the corresponding `profit-cost ratio` and `clicks in` (measured on the left and right vertical axis, respectively). The goal is to use these two metrics in order to explain the timeseries' behavior.

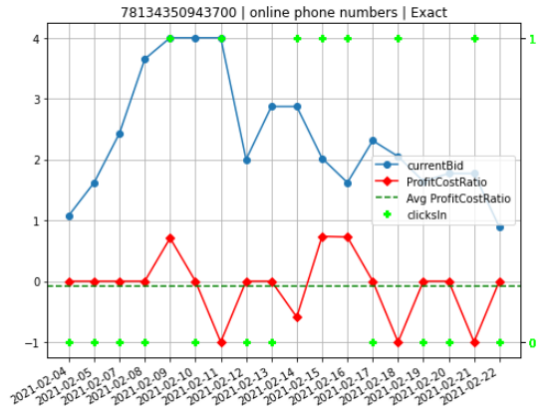
The above plots show that the `current bid` is determined by the `profit-cost ratio` value:



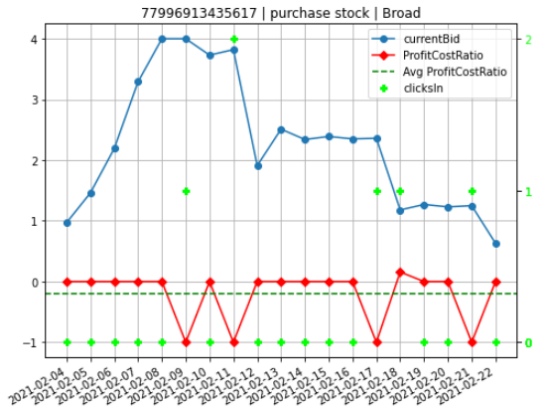
(a) Example n° 1



(b) Example n° 2



(c) Example n° 3



(d) Example n° 4

Figure 2.5: Examples of timeseries behavior based on `clicks in` and `profit-cost ratio`.

- a profit-cost ratio > 0 increases the current bid; the scenario is favorable, so more risk can be undertaken;
- a profit-cost ratio < 0 decreases the current bid; the scenario is not favorable, so more protection is needed.

In order to understand better how the CPC-Optimizer adjusts the current bid, consider Figure 2.5a: the initial current bid does not generate traffic (clicks in is zero) and so it is increased until some traffic is generated. But when this happens, the profit-cost ratio is negative, so the bid is reduced. This is not enough, since the profit-cost ratio remains negative. Hence, the current bid is further reduced (2021-02-11) until profit-cost ratio increases to zero. Later, it drops again to -1, and the current bid is decreased again, pushing up the profit-cost ratio and so on so forth.

This example shows an unsuccessful keyword: when the current bid generates high traffic volume (many clicks in), the profits are negative (and so the profit-cost ratio). The CPC-Optimizer tries to reduce the losses decreasing the bid, but this kills the traffic, leading to null profits.

A detailed explanation of the CPC-Optimizer is out of the scope of the current work, but the interested reader can find it in Appendix A.

2.3 Preprocessing Steps

2.3.1 Ground Truth Filtering

The result of the Section 2.2 has shown that is not in general true that the current bid time-series can be split in a 1° phase non-stationary and a 2° phase stationary. Hence, the best way to construct the dataset is without using any CP, but just keeping all observations with:

1. clicks in > 0;
2. profit-cost ratio ≥ -0.2.

These are all the observations with a current bid value able to generate traffic satisfying the minimum profitability threshold.

The observations remaining after the filtering of the new keyword dataset are just few thousands. This is not surprising, since the dataset covers only a limited subset of all the keywords and for few weeks. If these were the only data available, it would be a compulsory choice

to rely just on them. But there are many other (older) keywords' data for the last few months (January-April) stored. These data are about keywords which are:

- not new;
- handled manually, which means that the bid is not automatically adjusted using CPC-Optimizer.

Hence, it would have not been possible to carry out the analysis of Section 2.2 with these data. Anyway they contain many thousands of *ground truth* observations which can be exploited to develop a more accurate model. We refer to this dataset as `main dataset`.

2.3.2 BERT Encoding

Each keyword becomes an input of the model: it is converted in a vector (embedding) using Bidirectional Encoder Representation from Transformers (BERT) model [18] and the embedding is added to the set of features.

BERT is the first *pre-train*, *unsupervised*, *deeply bidirectional* language representation model.

Pre-train means that BERT has been trained as a general-purpose "language understanding" model, and not to solve a specific task (e.g. sequence classification, question answering, etc.). Hence, it can be fine-tuned (*training*) to solve various NLP tasks.

Unsupervised means that BERT was trained using only a plain text corpus, which is important because an enormous amount of plain text data is publicly available on the web in many languages.

Deeply bidirectional means that BERT builds a contextual language representation by jointly conditioning on both left and right context in all layers. This implies that the representation of each word is based on the other words in the sentence. This is a breakthrough with respect to context-free models such as `word2vec` or `Glove`, which generate a single "word embedding" representation for each word in the vocabulary, regardless of the context ("bank" would have the same representation in "bank deposit" and "river bank") and are unidirectional, where every token can only attend to previous tokens.

BERT has these appealing properties thanks to the way in which has been pre-trained. The pre-training is based on two unsupervised tasks:

1. "masked language model" (MLM). It randomly masks 15% of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Unlike left-to-right language model pre-training, the MLM objective

enables the representation to fuse the left and the right context, which allows to pretrain a deep bidirectional Transformer.

2. "Next sentence prediction". There are pairs of sentences (A,B), where 50% of the time B is the actual next sentence that follows A and 50% of the time it is a random sentence from the corpus.

The goal is to use BERT as a sentence (keyword) encoding service (e.g. mapping one *variable*-length sentence to one *fixed*-length numerical vector). A sentence is represented by BERT using a 4 dimensional tensor:

1. the number of layers (e.g. 24 layers for BERT-Large);
2. the number of batch (1 sentence);
3. the number of tokens (tokens in the sentence);
4. the number of hidden units (i.e. 1024 features for BERT-Large).

Note that each token is represented by multiple layers (e.g. 24), each with multiple features (e.g. 1024). First it's needed a way to represent a token using just one vector. A first idea could be to compute some aggregate statistics, like summing or averaging the layers of a token. Some hints come from the way in which BERT's layers capture different information:

- in the first layer there is no contextual information (e.g., the representation of the token "bank" is the same in "river bank" and "bank account");
- deeper and deeper layers capture more and more contextual information;
- approaching the final layer, however, layers start picking up information that is specific to BERT's pre-training tasks (MLM and NSP), so they could be biased toward the training target.

This implies the existence of a trade-off between better token representation and better pre-training score.

A pooling strategy that empirically works well consists in taking, for each token, only one layer (e.g. pick the 2^{nd} layer as the vector representing the token). So now each token is represented by one vector and the sentence is represented by the set of these vectors. Then the sentence can be represented as the average of these vectors. The pooling strategy described above can be implemented using the `bert-as-a-service` tool, developed by researcher Han Xiao, who proposed it [19].

The encoding has been applied to a sample of real keywords extracted from the data. Several embedding strategies have been tested using BERT-large and the cosine similarity for each pair of keywords (A,B) has been computed according to the formula:

$$similarity = \cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \cdot \|\vec{B}\|}.$$

Several pre-trained model are available, but only the largest ones have been considered here:

- BERT-Base, with 12 hidden layers and 768 hidden units;
- BERT-Large Uncased, with 24 hidden layers and 1024 hidden units;
- BERT-Large Uncased (Whole Word Masking), with 24 hidden layers and 1024 hidden units.

In the following, the analysis is based on the embeddings obtained from "BERT-Large Uncased (Whole Word Masking)", since it turned out to provide the best encodings of the keywords (examples using BERT-Base and BERT-Large Uncased are reported in Appendix B).

"BERT-Large Uncased (Whole Word Masking)" differs from "BERT-Large Uncased" in just one aspect: the masking. In fact the latter applies the masking at the token level; if a word is composed by several tokens, some of them could be masked and others not. The former applies the masking at the word level, hence the entire word is masked. This approach is more similar to how humans interpret a sentence, reading one word after the other and then understanding the overall meaning. This approach is also highly desiderable for the keywords encoding, where it's natural to work at the word level and not at the token level. The presence of a word rather than another one, can change completely the meaning (e.g. "apple pie", "apple pc").

Four examples, using different *pooling layers*, are reported in tables 2.2, 2.3, 2.4 and 2.5. To allow comparison, the same target keyword "pediatric nurse" is used and its cosine similarity with all the other keywords is computed. Then the keywords are ranked in decreasing order of cosine similarity. If the BERT encoding is effective, the most similar keywords should be considered similar also by a human reader.

keyword	cosine similarity
pediatric nurse	1.000000
asthma treatment medicine	0.627102
online nurse practitioner programs	0.626773
Nursing Masters Programs	0.621545
medical billing certificate	0.604025
medical assistant certificate	0.594548
treat lung cancer	0.586177
maryland health insurance	0.555291
medical coding online	0.543766
healthcare management master	0.538779
health insurance find	0.526645
non cell lung cancer	0.521676
puppy insurance	0.517043
Medicare Plan Supplement	0.516892
marketplace healthcare	0.514201
healthcare online	0.510604
credit counseling	0.509416
banyan treatment center	0.505204
medicare information	0.502602
emergency 24 hour dentist near me	0.502538

Table 2.2: Cosine similarity using BERT-Large Uncased (Whole Word Masking) encoding with pooling layer=0,1,2 and pooling strategy={REDUCE_MEAN}

keyword	cosine similarity
pediatric nurse	1.000000
asthma treatment medicine	0.769481
Nursing Masters Programs	0.763387
medical billing certificate	0.744568
medical assistant certificate	0.734013
treat lung cancer	0.716228
online nurse practitioner programs	0.710297
medical coding online	0.704952
healthcare management master	0.699064
maryland health insurance	0.696401
non cell lung cancer	0.674025
credit counseling	0.669154
Revenue Cycle Management Healthcare	0.659375
healthcare online	0.658066
Medicare Plan Supplement	0.656348
dental plan discounts	0.654299
insulin pumps for diabetics	0.651706
banyan treatment center	0.646709
accredited schools online	0.644007
medicare information	0.642167

Table 2.3: Cosine similarity using BERT-Large Uncased (Whole Word Masking) encoding with pooling layer=2 and pooling strategy={REDUCE_MEAN}

keyword	cosine similarity
pediatric nurse	1.000000
credit counseling	0.961464
treat lung cancer	0.954706
accredited schools online	0.952889
payroll services pricing	0.952495
Nursing Masters Programs	0.951862
non cell lung cancer	0.951662
marketplace healthcare	0.951197
Online University	0.950153
chicago community colleges	0.949997
Criminal Law Lawyer	0.949190
online school high	0.949077
Cheap Liability Insurance Coverage	0.948853
maryland health insurance	0.948727
grocery shopping online	0.948209
digital signage display	0.947832
Llc Virginia	0.945679
antivirus malware protection	0.945333
Lawyer Divorce	0.945221
buy stocks online	0.945148

Table 2.4: Cosine similarity using BERT-Large Uncased (Whole Word Masking) encoding with pooling layer = $-3, -2, -1$ and pooling strategy = {REDUCE_MEAN}

keyword	cosine similarity
pediatric nurse	1.000000
credit counseling	0.968099
treat lung cancer	0.964629
non cell lung cancer	0.961824
accredited schools online	0.961635
payroll services pricing	0.961507
marketplace healthcare	0.961011
Nursing Masters Programs	0.960975
chicago community colleges	0.960967
Online University	0.960620
Criminal Law Lawyer	0.960321
grocery shopping online	0.960085
Cheap Liability Insurance Coverage	0.959140
digital signage display	0.959032
online school high	0.958915
maryland health insurance	0.958653
Investment Management Companies	0.957985
Small Business Insurance Companies	0.957316
antivirus malware protection	0.956222
buy stocks online	0.955886

Table 2.5: Cosine similarity using BERT-Large Uncased (Whole Word Masking) encoding with pooling layer = -2 and pooling strategy = {REDUCE_MEAN}

The main conclusions are:

1. the embedding strategies which use shallow layers (Table 2.2 and 2.3) are the best ones and lead to very similar results: almost the same keywords are identified with almost the same ranking order. Considering that Table 2.3 uses one pooling layer (1024 hidden units) rather than three (3072 hidden units), it is preferable due to its parsimony;

2. the embedding strategies which use deep layers (Table 2.4 and 2.5) lead to a higher cosine similarity, which theoretically should imply a higher semantic similarity. But actually this is not the case, since the keywords are not similar to the target "pediatric nurse";

3. the results in the previous two points are surprisingly, since the first layers in the BERT model provide a representation which does not take much into account the context and so should be less able to detect semantic similar keywords than deeper layers.

Considering these observations, probably the context learned by BERT during the pre-training is not helpful, and even harmful. The reason could be that the text corpora used during the pre-training is quite different from the keywords used in digital advertisement. Hence, using the last layers (*pooling layer*$=-3,-2,-1$), which capture the context, leads to a lower accuracy than using the first layers (*pooling layer*$=0,1,2$). The first layers are not capturing much context, but at least can cluster together similar keywords.

According to these evidences, the keywords are going to be encoded using:

- pooling layer = 2;
- pooling strategy = `REDUCE MEAN`.

2.4 Dataset Construction

The `main dataset` is used in this section and in Chapter 3. It contains the observations filtered in Subsection 2.3.1 and encoded in Subsection 2.3.2. Now it can be joined with the dataset from MIMIR, according to the value of columns:

- `date`;
- `keyword name`.

Finally, all the columns unknown *ex ante* (so not marked with an asterisk in Table 2.1) are removed, except for `current bid`, which plays the role of dependent variable. Also the column `keyword ID` is removed, since each keyword is now identified by its encoding. Anyway the column `keyword name` is kept, since it will be useful in the next chapter to split the keywords in train, validation and test set. In this way, the final dataset contains only features known *ex ante* and observations classified as positive (Table 2.6).

In this framework, the estimation of the *optimal initial bid* for new keywords is cast as a regression problem. The limited number of observations and the different relationship between `current bid` and `suggested bid` for each keyword, make not easy to learn the relation between explanatory variables and dependent variable.

In order to learn such complex relationship, the most appropriate tool seems to be a Neural Network, which is going to be built in Chapter 3.

keyword encoding								dependent variable
f_1	$\cdots$	f_{1024}	keyword name	match type	suggested bid	competition	search volume	current bid
$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 2.6: Header of the final dataset.

3 Model Design

The chapter addresses the *regression problem* introduced in Chapter 2 using different models and analyzing the respective performances.
In detail, it is composed by three sections:

- Baseline Models: the natural starting models to solve a regression problem.
- Neural Network Model: the same problem is solved using a *Fully Connected Neural Network*, explaining the *training procedure* and the *prediction errors analysis*.
- Negative Test Set: a test set containing only observations with negative profitability is used to counter check the reliability of results.

The first two sections use the `main dataset` introduced in Section 2.4.
The feature `keyword name` is used to group the `main dataset` by keyword and split it at the *keyword level* (instead of the *observation level*). This ensures that all observations belonging to a keyword appear in just one of the training, validation and test sets. To allow comparability, the same splits are used for all models.

3.1 Baseline Models

The *regression problem* is tackled in the next three subsections using linear models. They assume a linear relationship between the dependent variable and each of the regressors. For complete-

ness of exposition, every model is briefly introduced and explained in its major characteristics, basing on [20].

These models are fast to train due to the existence of closed form solutions (except for *Lasso* regression) and easy to fine tune, since have zero or one hyperparameter. Once estimated, their performances represent a benchmark to evaluate the effectiveness of more complex models, like a neural network.

The set of features depicted in Table 2.6 are not all numeric, hence some preprocessing steps are necessary:

- `keyword name` is dropped since it is encoded by the features $f_1, \ldots, f_{1024}$;
- `match type` is One-Hot-Encoded.

The resulting dataset is used as input in each of the next models. The variable `current bid` plays the role of dependent variable.

It is worth noting that the main goal is *prediction* and not *inference*. This is because:

1. for the business profitability it's of primary importance to have a model able to predict an enough accurate `current bid`;
2. all regressors, except $f_1, \ldots, f_{1024}$ and `match type`, take exogenous values determined by the market and so are out of the broker's control.

3.1.1 Multiple Linear Regression

The basic model for multiple linear regression is:

$$\hat{\mathbf{y}} = \beta_0 + \sum_{j=1}^{p} \mathbf{x_j} \cdot \beta_j$$

where $\hat{\mathbf{y}}$ is the vector of estimates of the true output vector $\mathbf{y}$ and $\mathbf{x_j}$ is the vector containing values of regressor j for all observations in the dataset.

The β_j's are unknown parameters to estimate in order to minimize the residual sum of squares (RSS):

$$\begin{aligned} RSS(\boldsymbol{\beta}) &= \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2. \end{aligned}$$

Denoting by $\mathbf{X}$ the $N \times (p+1)$ matrix with each row an input vector (with a 1 in the first position) and each column a regressor, and denoting by $\mathbf{y}$ the vector of outputs, the RSS can be expressed in matrix form:

$$RSS(\boldsymbol{\beta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

Differentiating with respect to $\boldsymbol{\beta}$:

$$\begin{aligned} \frac{\partial RSS}{\partial \boldsymbol{\beta}} &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \\ \frac{\partial^2 RSS}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^T} &= 2\mathbf{X}^T\mathbf{X}. \end{aligned}$$

If $\mathbf{X}$ has full rank columns, as in this case, then $\mathbf{X}^T\mathbf{X}$ is positive definite. This means that the RSS is a convex function and the critical point is a minimum. Hence, it's sufficient to set the first derivative to zero and solve with respect to $\boldsymbol{\beta}$:

$$\begin{aligned} \mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) &= 0 \\ \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} &= 0 \\ \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} &= \mathbf{X}^T\mathbf{y} \\ \hat{\boldsymbol{\beta}} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned}$$

Results

In order to evaluate the prediction capacity of the multiple linear regression model, there are two main metrics:

1. *absolute error*;

2. *percentage error*.

The *absolute error* is measured in dollar ($) and so it has an easy interpretation, but it's not scale free. This means that it quantifies just the error's size, without any comparison with the size of the *ground truth*'s value.

The *percentage error* measures the error's size as a proportion of the *ground truth*'s value, so it's scale free. But it is not easy to gauge consciously: an error of few cents (e.g. $0.05$$) can lead to a high *percentage error* if the real value is low (e.g. $0.2$$).

From the business perspective, the *absolute error* is the preferred choice since what matters is the gap in $ between the prediction and the true value. This is true for two reasons:

1. a smaller gap can be corrected in a relative short amount of time;

2. the losses (or missing profits) associated with a small gap are lower given the same traffic volume.

So, reconsidering the previous example, if the correct bid is $0.20$$ and the predicted bid is $0.15$$, the error is 5 cents. The corresponding *percentage error* is 25%. Assume now that exists another keyword for which the correct bid is $4$$ and the predicted bid is $3$$. The *absolute error* is $1$$, so the time needed to adjust the bid will be longer and the losses (or missing profits) will be greater. But the *percentage error* is still 25%. In the first case the *percentage error* is acceptable, in the second case it is not. For this reason, in the remaining part of the chapter only the *absolute error* is considered to evaluate results.

Firstly, the capacity of the multiple linear regression output to "follow" the true output is graphically inspected in Figure 3.1. This allows to verify that the regression is not just returning a dummy value, like the average, for all the observations.

The *average absolute error* in *test set* is equal to $0.26$$ with the *average* `current bid` equal to $0.63$$. To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure 3.2.

The box-plot shows that 25% of errors are lower than 10 cents and 50% of errors are lower than 21 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 36 cents. There is another approximately 14% of cases where the error is between 36 and 50 cents. Very large errors are limited to the last 10% of data.

Considering the simplicity of the model, results are acceptable.

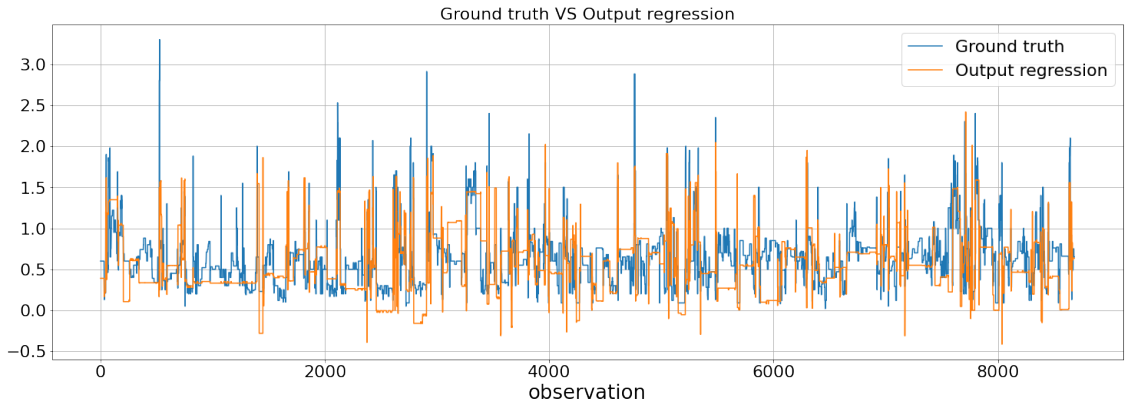
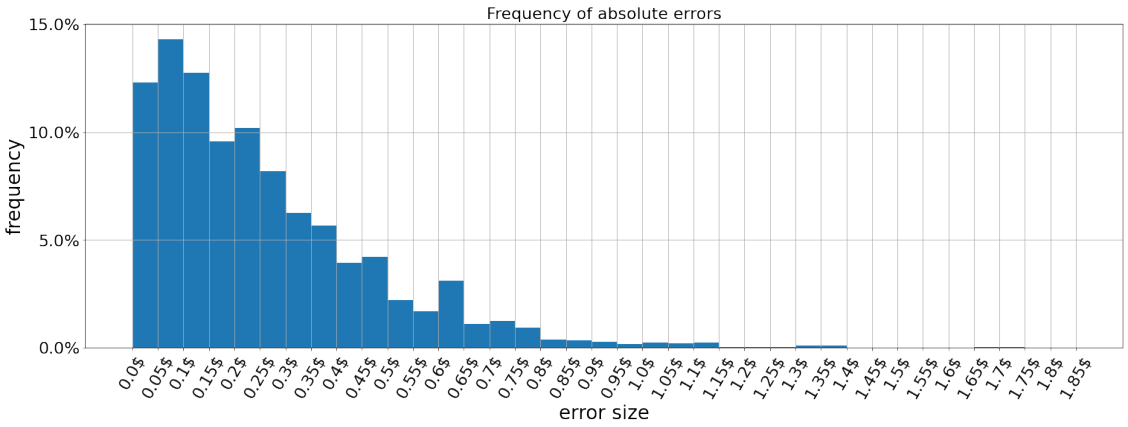
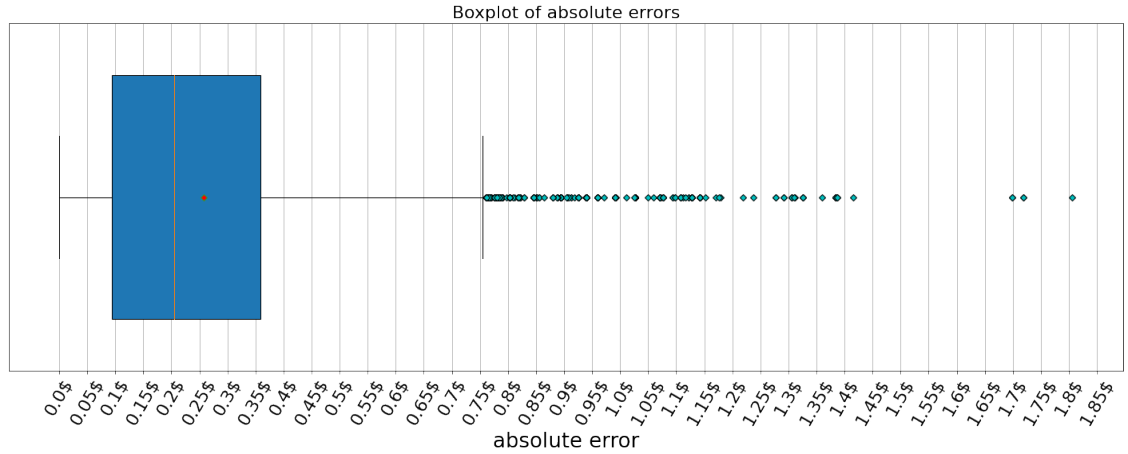


Figure 3.1: Ground truth VS Output Regression.



(a) Regression's Absolute Errors distribution.



(b) Regression's Absolute Errors box-plot.

Figure 3.2: Regression's Absolute Error quantile distribution.

3.1.2 Ridge Regression

The model is the same introduced in Subsection 3.1.1. What changes is the objective function:

$$\begin{aligned}\hat{\boldsymbol{\beta}}^{ridge} &= \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \\ &= \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}.\end{aligned}$$

The change consists in the introduction of a penalty hyperparameter λ that controls the amount of shrinkage: higher values of λ lead to greater shrinkage. In fact, λ can be interpreted as the weight, or the importance, of the penalty.

Denoting by $\mathbf{X}$ the $N \times (p+1)$ matrix with each row an input vector (with a 1 in the first position) and each column a regressor, and denoting by $\mathbf{y}$ the vector of outputs, the objective function can be expressed in matrix form:

$$RSS(\boldsymbol{\beta}, \boldsymbol{\lambda}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}$$

Differentiating with respect to $\boldsymbol{\beta}$:

$$\begin{aligned}\frac{\partial RSS}{\partial \boldsymbol{\beta}} &= -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + 2\lambda\boldsymbol{\beta} \\ \frac{\partial^2 RSS}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} &= 2\mathbf{X}^T\mathbf{X} + 2\lambda\mathbf{I}\end{aligned}$$

where $\mathbf{I}$ is a $(p+1) \times (p+1)$ identity matrix. If $\mathbf{X}$ has full rank columns, as in this case, then $\mathbf{X}^T\mathbf{X}$ is positive definite and $2\lambda\mathbf{I}$ is also positive definite. Their sum is still positive definite. This means that the RSS is a convex function and the critical point is a minimum. Hence, it's sufficient to set the first derivative to zero and solve with respect to $\boldsymbol{\beta}$:

$$\begin{aligned}-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} + 2\lambda\boldsymbol{\beta} &= 0 \\ \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} + \lambda\boldsymbol{\beta} &= \mathbf{X}^T\mathbf{y} \\ (\mathbf{X}^T\mathbf{X} + I\lambda)\boldsymbol{\beta} &= \mathbf{X}^T\mathbf{y} \\ \hat{\boldsymbol{\beta}}^{ridge} &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}.\end{aligned}$$

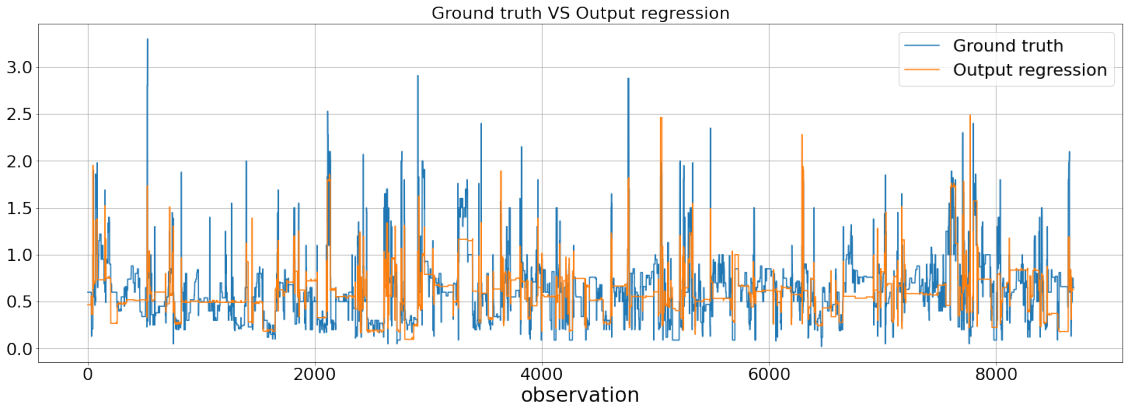


Figure 3.3: Ground truth VS Output Ridge regression

Results

The Ridge Regression model requires the value of the hyperparameter λ. The term *hyperparameter* means that it is not determined by the model itself through the training, but it has to be specified externally. Different values of λ allow to reach different minima for the objective function. The optimal lambda (λ^*) is the one which leads to the smallest possible value of the objective function.

In order to find λ^*, a *grid search* procedure together with a *cross validation* (CV) is applied. First, a list of λ values is identified; then, for each of them, the training set is split in five folds and, in turn, one is used as validation set and the remaining ones as training set. For each λ, the average value of the objective function through the five splits is computed and used as the metric to determine λ^*.

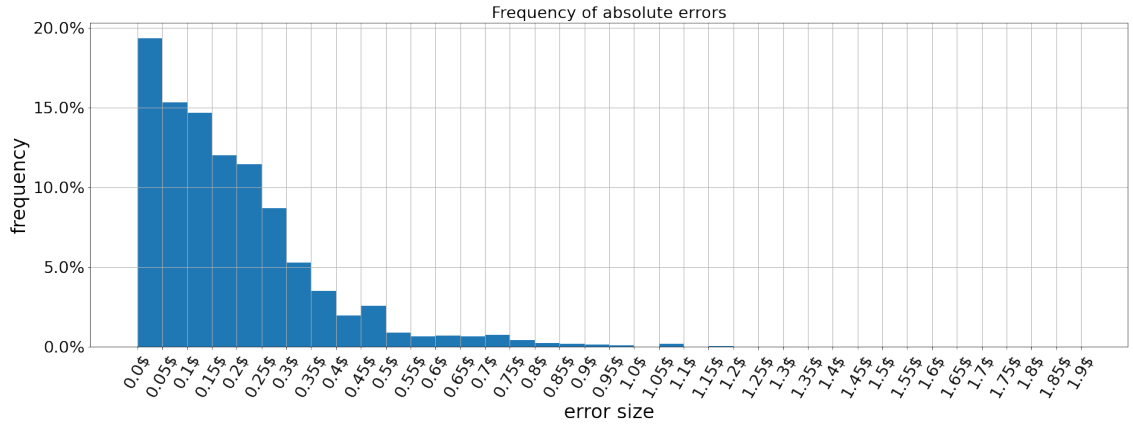
The whole procedure is repeated a second time selecting a finer grid of λ values around the best value found before. The λ^* is equal to 1750.

The capacity of the ridge regression's output to "follow" the true output is graphically inspected in Figure 3.3. This allows to verify that the regression is not just returning a dummy value, like the average, for all the observations.

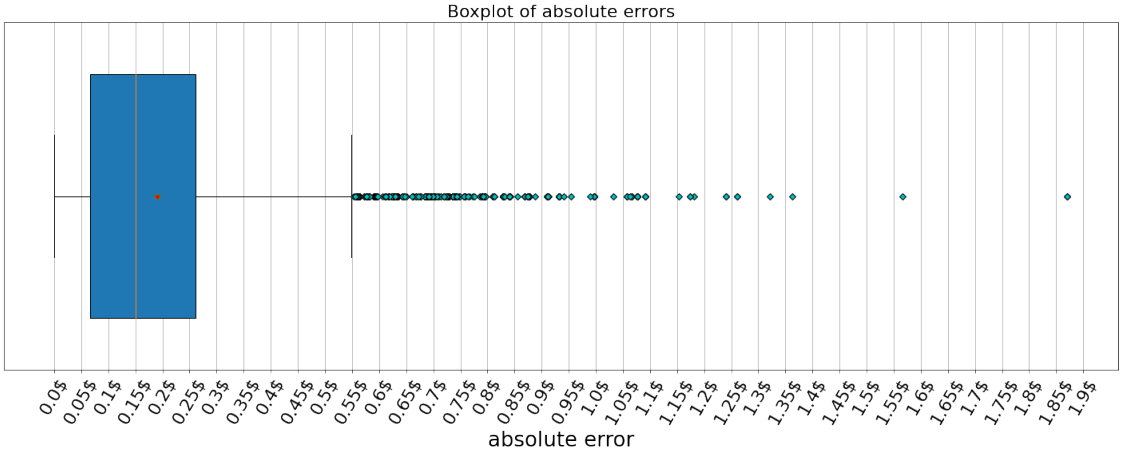
The *average absolute error* in *test set* is equal to $0.19\$$ with the *average* `current bid` equal to $0.63\$$. This is a significant drop with respect to the $0.26\$$ achieved using multiple linear regression.

To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure 3.4.

The box-plot shows that 25% of errors are lower than 7 cents and 50% of errors are lower



(a) Ridge's Absolute Errors distribution.



(b) Ridge's Absolute Errors box-plot.

Figure 3.4: Ridge's Absolute Error quantile distribution.

than 15 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 27 cents. There is another approximately 15% of cases where the error is between 27 and 50 cents. Errors above 50 cents are limited to the last 10% of data. Considering the simplicity of the model, results are already good.

3.1.3 Lasso Regression

The model is the same introduced in Subsection 3.1.1. What changes is again the objective function:

$$\begin{aligned}\hat{\boldsymbol{\beta}}^{lasso} &= \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \\ &= \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}.\end{aligned}$$

The change, with respect to the ridge regression, consists in the substitution of the L_2 penalty ($\sum_{j=1}^{p} \beta_j^2$) with the L_1 penalty ($\sum_{j=1}^{p} |\beta_j|$). Using this constraint there is no closed form expression for $\boldsymbol{\beta}$. Anyway exist algorithms able to compute efficiently $\boldsymbol{\beta}$.

Results

The Lasso Regression model requires the value of the hyperparameter λ. In order to find λ^*, the same *grid search* procedure together with the *cross validation* (CV) used for ridge regression, is applied here. First, a list of λ values is identified; then, for each of them, the training set is split in five folds and, in turn, one is used as validation set and the remaining ones as training set. For each λ, the average value of the objective function through the five splits is computed and used as the metric to determine λ^*.

The whole procedure is repeated a second time selecting a finer grid of λ values around the best value found before. The λ^* is equal to 0.0019.

The capacity of the lasso regression's output to "follow" the true output is graphically inspected in Figure 3.5. This allows to verify that the regression is not just returning a dummy value, like the average, for all the observations.

The *average absolute error* in *test set* is equal to 0.19$ with the *average* `current bid` equal to 0.63$. This is a significant drop with respect to the 0.26$ achieved using multiple linear regression and equal to the ridge result.

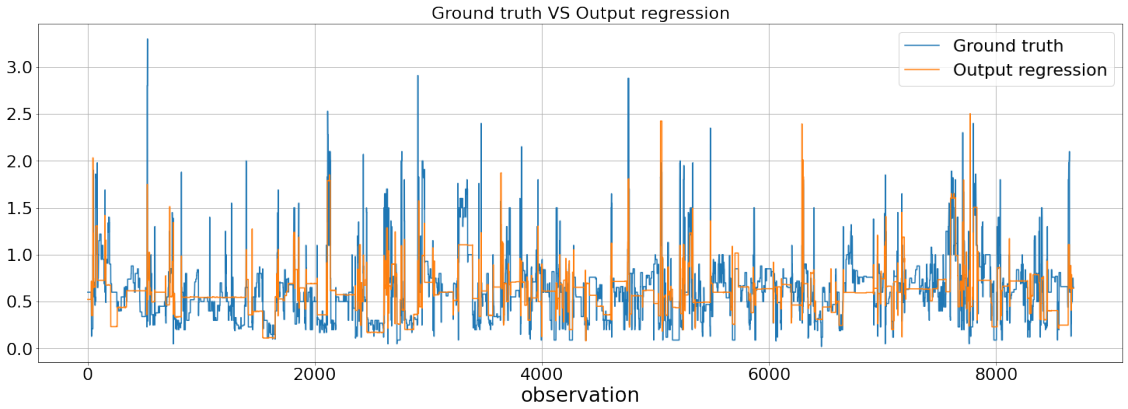


Figure 3.5: Ground truth VS Output Lasso regression.

To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure 3.6.

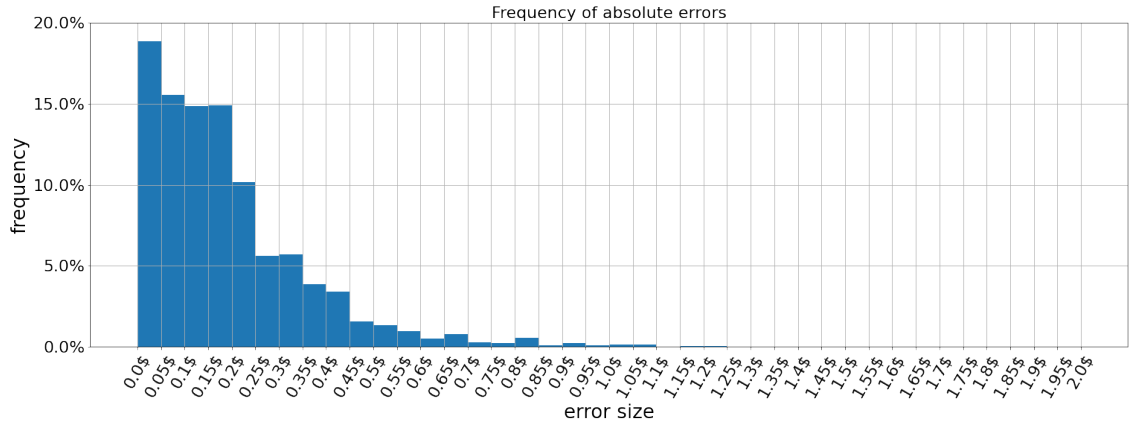
The box-plot shows that 25% of errors are lower than 7 cents and 50% of errors are lower than 15 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 26 cents. There is another approximately 20% of cases where the error is between 26 and 50 cents. Errors above 50 cents are limited to the last 5% of data. The major difference with respect to the ridge regression's result is in the right tail: now only 5% of error is above 50 cents. Considering the simplicity of the model, results are already good.

3.2 Neural Network Model

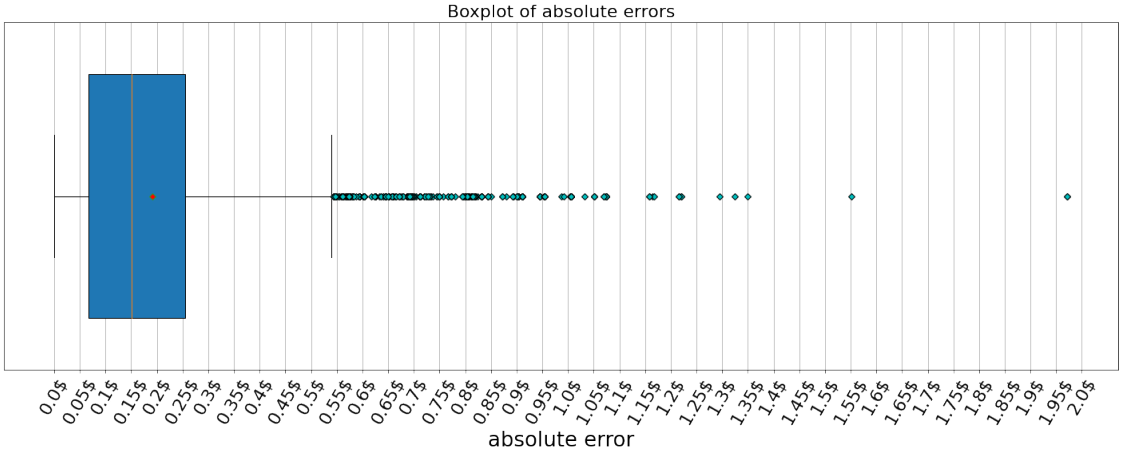
The goal is to train a neural network to approximate the unknown function:

$$f : \mathbb{R}^p \to \mathbb{R}$$
$$\mathbf{x} \to f(\mathbf{x})$$
$$\text{network}(\mathbf{x}) \approx y$$

producing the output $f(\mathbf{x})$ given the input vector $\mathbf{x}$ as close as possible to the ground truth y.



(a) Lasso's Absolute Errors distribution.



(b) Lasso's Absolute Errors box-plot.

Figure 3.6: Lasso's Absolute Error quantile distribution.

3.2.1 Training Procedure

Strategy

The training of a neural network requires to fine tune several hyperparameters. The goal is to find the combination of them which allows to minimize the prediction error.

First, the `main dataset` is split in:

- *full training set*;
- *test set*.

Then, the *full training set* is further split in:

- *training set*;
- *validation set*.

The size of the `main dataset` is big enough to expect similar results from different splits in training and validation sets. Hence, in first place, no cross-validation (CV) is used. The best hyperparameters are found basing on a unique validation set and their goodness checked against the test set. Finally, in order to be sure that the model selected is robust to different splits, CV is applied to the full training set. It is split in five folds and, in turn, one is selected as validation set while the remaining ones are used as training set. The validation error in each split is computed.

Two cases are possible:

1. validation errors in various splits are very similar. This means that the splits are very similar (due to the dataset size) and so there is no added value in using CV. So results of the training without CV are confirmed.
2. Validation errors in various splits are different. This means that the splits are different and so the results of the training without CV are only valid for a specific split. The procedure to find the best hyperparameters has to be repeated using CV.

Exist various techniques, usable with or without CV, in order to find the best hyperparameters. The most known are *grid search* and *random search*. Such techniques are simple to apply but computationally very expensive. In fact, for each combination of hyperparameters a new model is trained. Considering the high number of hyperparameters, a blind grid/random

search is not an efficient strategy to select the best model and it can become rapidly computationally unfeasible.

A different approach is used here. The best value for each hyperparameter is set *sequentially*: first, the optimal values for the most important hyperparameters are found; then, keeping the latter fixed, the optimal values for the hyperparameters of secondary importance are selected.

Does not exist a rank of importance for every hyperparameter, but it's possible to identify a macro-order. First comes the network's architecture: *number of layers* and *number of units*. If the network's size is too small, any other hyperparameters can only provide little help in reducing the error. Then the *activation function* has to be properly chosen basing on the task at hand and to avoid issue like gradient saturation. Once the network is able to learn through epochs, the *learning rate* can be fine-tuned, or even better, dynamically adjusted to allow a finer grain learning. If the learning curves show presence of overfitting, the network's complexity has to be shrunk using regularization techniques, like *dropout* and *weight decay*.

The sequential procedure allows to explore the space of the possible hyperparameters' combination in an efficient way. Trying all of them, like in grid search, is computationally unfeasible; trying randomly some combinations, like in random search, could result in a model too far from the best achievable one.

Then, this *training strategy* is repeated for different *loss functions*:

- Mean Squared Error (MSE). Errors are squared, which implies that larger mistakes are penalized more than linearly as error increases. This pushes the network to avoid large mistakes even at the cost of more frequent small/medium mistakes.

- Mean Absolute Error (MAE). Errors are taken in absolute value, which implies that larger mistakes are penalized linearly as error increases. This pushes the network to be more tolerant toward large mistakes than using MSE. Since the ground truth is measured in dollars, the MAE has also an intuitive interpretation: it is the average prediction error measured in dollar.

- Mean Logarithmic Error (MLE). It is computed as:

$$MLE = \frac{1}{N}\sum_{y=1}^{N} \log(|output - target| + 1)$$

  Errors are first taken in absolute value, to ensure that the logarithm is always defined, and then 1 is added, to ensure that the minimium value of the argument is 1. In this way, if the absolute loss is 0, also the logarithmic loss will be 0. If the absolute loss is > 0, then also the MLE will be > 0, but the MLE will increase less than linearly as the error

increases. Hence, larger mistakes are penalized less than linearly as error increases. This pushes the network to be more tolerant toward large mistakes than using MAE and, of course, MSE.

Finally, the robustness of the best model to different splits is checked. The *full training set* is split in 5 folds and, for each of them, the validation loss is computed. If the validation losses are similar, then the model is robust to different splits.

Training Error Lower Bound

The goal is to determine the *minimum achievable training error* through an empirical analysis of the training set. Having an idea, even approximate, about the order of magnitude of the training error, is a counter check for the soundness of results obtained during the training procedure. In particular, it ensures that the model found is actually enough complex to reach the highest possible overfitting in the training set. This is the optimal starting point to apply *regularization* techniques, in order to get the lowest *validation loss*.

The key idea, which leads the analysis, is the following: a neural network, no matter how complex it is, cannot reduce the training error below a certain threshold if identical inputs (the terms *set of features* and *observations* are also used) are associated to different outputs (`current bid`).

In order to check this, the training set is scanned and identical observations are grouped together. Take one of such groups as an example. If it contains all observations with the same output, then the *proportion of identical outputs* is 100% for this group.

If it contains two different outputs, then there is a certain proportion, e.g. 0.9, of observations associated to an output and another 0.1 of observations associated to a different output. In this case the highest proportion is considered, and so it's possible to say that 0.9 of observations are associated with the same output.

If it contains 10 different outputs, then there is a certain proportion, e.g. 0.1, of observations associated to each of the 10 different outputs. In this case, as always, the highest proportion is considered, and so it's possible to say that 0.1 of observations are associated with the same output.

This allows to verify not only the presence of different outputs associated to the same input, but also to quantify the uncertainty. In fact, having the highest proportion of identical outputs equal to 0.9, means that the neural network can learn a strong relationship. Having the highest proportion of identical outputs equal to 0.1, means that the neural network cannot learn any relationship.

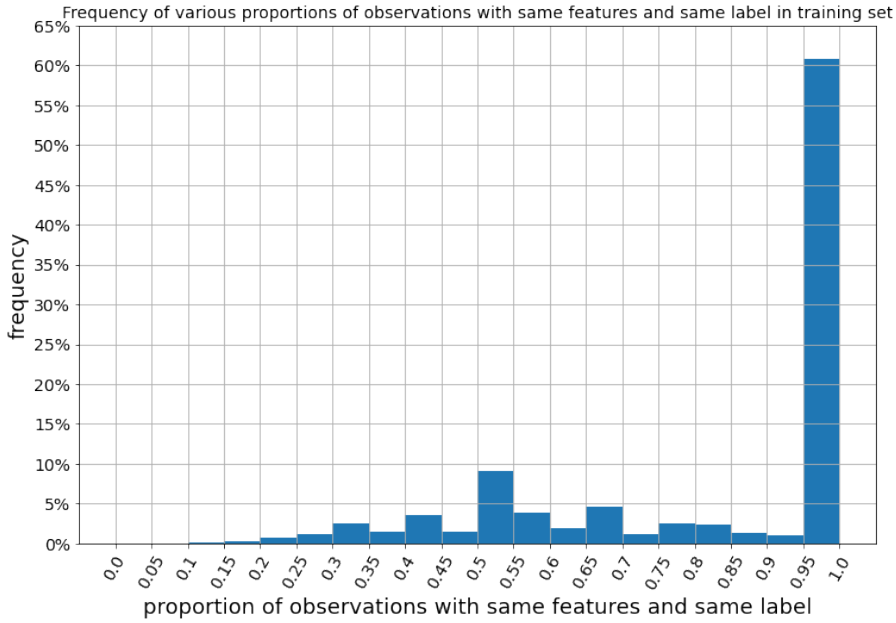


Figure 3.7: Distribution of *proportion of the same observations* in the training set.

Repeating this analysis for every group and storing the corresponding proportions in a list, it's possible to draw a histogram in order to have an idea of the proportions' distribution in the entire training set. For example, in Figure 3.7, approximately 60% of groups have a unique (or more that 95% of times) output. As another example, 9% of groups have half of observations associated to the same output while the other half is associated to different output/s.

Moreover, for each group j, the weighted average of outputs ($\overline{output}_j$) is computed and its distance from the output of each observation i in the same group is calculated:

$$error_i = |\overline{output}_j - output_i|$$

Storing these values for each observation and for each group, allows to compute the average error ($\overline{error}$) in the training set, which is a reasonable approximation of the *minimum achievable training error*. It is shown as a red dotted line at $0.068\$$, together with the entire $error$ distribution in Figure 3.8.

In fact, if, by assumption, all groups contains a unique output, the $error$ in each group is 0 and so also the average training error is 0. This is the case of a network which is perfectly fitting the training set just learning it by heart.

If, more realistically, different groups contain different proportions of identical outputs, the

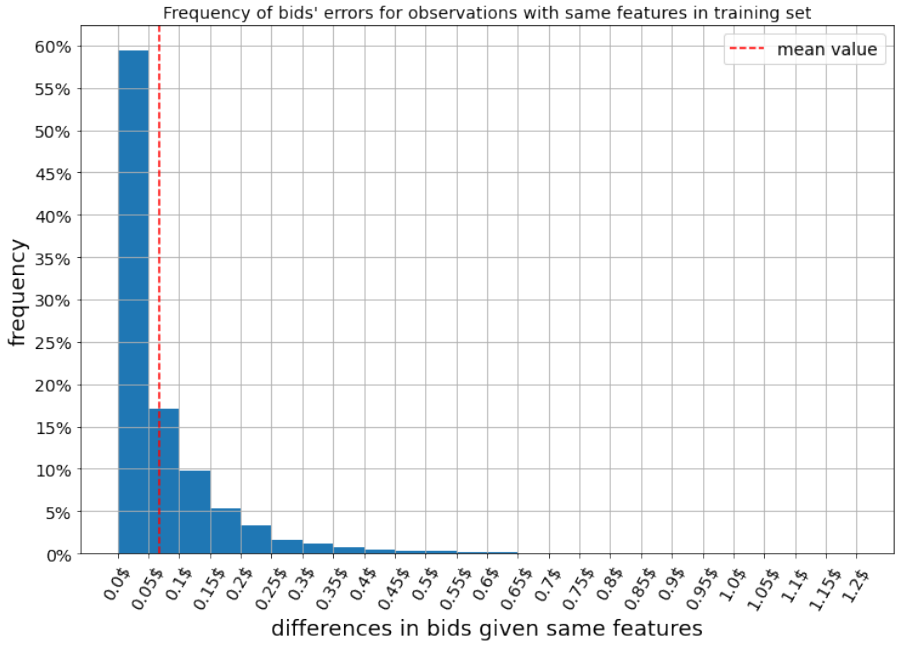


Figure 3.8: Distribution of *bids' errors* in the training set.

$error$ is greater than 0, but anyway it's not possible to do better. In all the groups where are present different outputs, the network is forced to learn a unique output to provide as estimate. In order to do so, the network learns a sort of weighted average of outputs.

Best Model Selection

The training strategy presented before is enriched by a series of "experiments" on the features. The first part of the strategy is the same: the optimal *number of layers* and *number of units* is found and the most suited *activation function* is selected. In this case, it is the Rectified Linear Units (ReLU). The network's architecture is depicted in Figure 3.9. In order to increase readability, the various hidden layers are represented with same length, but the actual size is reported for each layer in the pedix of the last hidden unit (e.g. H_{4096} is the last unit in the 1° hidden layer). The activation function is also depicted, inside each hidden unit, using a standard symbol for ReLU. The training and validation losses reach a plateau around 0.16 and 0.20, respectively, as shown in Figure 3.10. Note that to allow a better representation, the vertical axis is in logarithmic scale.

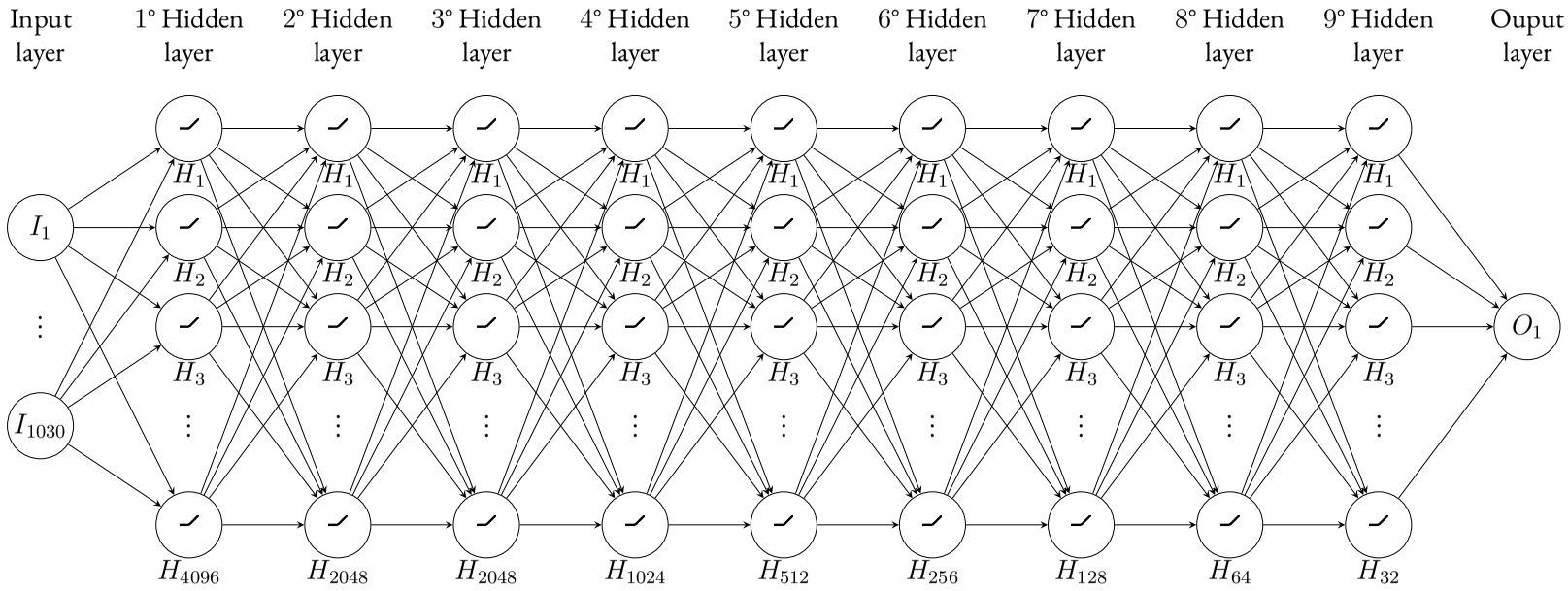


Figure 3.9: Fully Connected Neural Network Architecture.

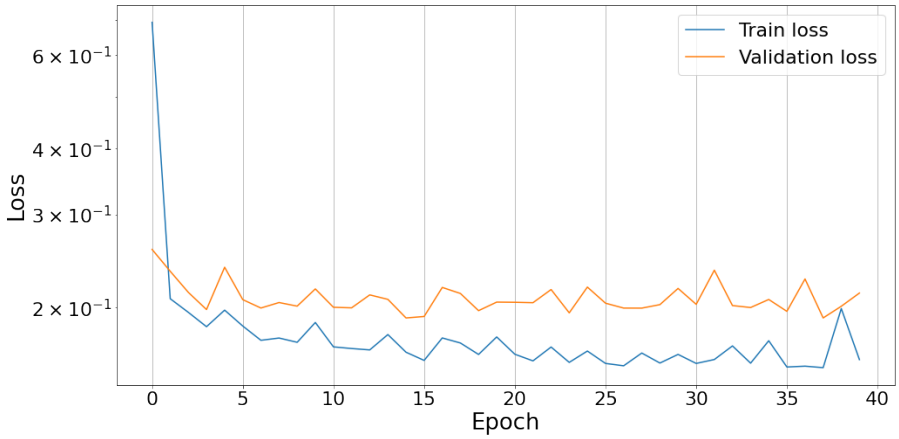


Figure 3.10: Training Loss VS Validation Loss.

Before completing the fine-tuning, some experiments are carried out in order to assess the soundness of the model. In each of them, the same split in training set and validation set is used, to allow full comparability.

First, the contribution of the keywords encoding ($f_1, \ldots, f_{1024}$) in the prediction of ground truth (`current bid`), is checked. In order to do so, the encoding features are removed from the training and validation set. The training and validation losses increases to 0.24 and 0.28, respectively. This proves, as expected, that the encoding plays a role in prediction.

A specular experiment consists in dropping all the features but the encoding ones. Now the goal is to verify if the non-encoding features provide a positive contribution in the prediction accuracy. This check is motivated by the high asymmetry in the cardinality of encoding and non-encoding features. The former are 1024, while the latter are just 6 (`match type broad`, `match type exact`, `match type phrase`, `suggested bid`, `competition` and `search volume`). So it's needed to verify that the neural network is not just learning a relationship between the encoding features and the output. Results show a reduction of the validation loss to 0.19: the network reduces the prediction error using only the encoding features. This behavior is unexpected, since at least some features, like `suggested bid`, should have a strong relation with the output. The key point is that this result does not mean that all the non-encoding features are useless: some of them could be helpful. In fact, dropping all the non-encoding features, only their aggregate impact on the learning capacity of network has been assessed, and it is negative. The next experiments will drop in turn different non-encoding features to check if they are all really useless or not.

The feature `search volume` is removed: the training error halves to 0.08 (very close to the

training error *lower bound* of Subsection 3.2.1) and the validation error reduces to 0.185. This means that this feature is not helpful for the prediction of the dependent variable and it's a source of noise. The strong reduction in the training loss and the wider gap with the validation loss, suggests using some regularization technique to improve the generalization power of the model. The hope is to find a compromise such that the validation loss decreases at the cost of a higher training loss. Simultaneously also the learning rate is dynamically adjusted: the validation loss of each epoch is recorded and, if it is not lower than the previous five validation losses, the learning rate is reduced by 10 times. This ensures a finer grain learning as training proceeds and reduces oscillations in learning curves. The dynamically adjusted learning rate, together with dropout equal to 0.3, gets the best results: training error increases to 0.10, but validation error decreases to 0.169.

In the next experiment, the same procedure is followed, but dropping, together with `search volume`, the features `match type broad`, `match type exact` and `match type phrase`. Different dropouts and weight decays are tried, but the validation loss increases in every trial. This means that the features representing the match types are useful in the learning task.

Finally, in turn are dropped only the features `competition` and `suggested bid`. Both turns out to be useful since the validation loss increases in every trial.

The full procedure is repeated for the *loss functions* MSE, MAE and MLE. In the available data, the most suited one is the MAE. The others lead to a higher validation loss.

To recap, the best model found has:

- features $f_1, \ldots, f_{1024}$, `match type broad`, `match type exact`, `match type phrase`, `suggested bid`, `competition`;
- adjusting learning rate;
- dropout probability equal to 0.3 and no weight decay;
- loss function MAE.

Results on Full Training Set

The best model is trained over the *full training set* and tested on *test set*. Results are shown in Figure 3.11. The test loss reaches a minimum value around 0.15, then starts some overfitting. This means that the model predicts a $\widehat{current\ bid}$ which is, on average, 15 cents of dollar far

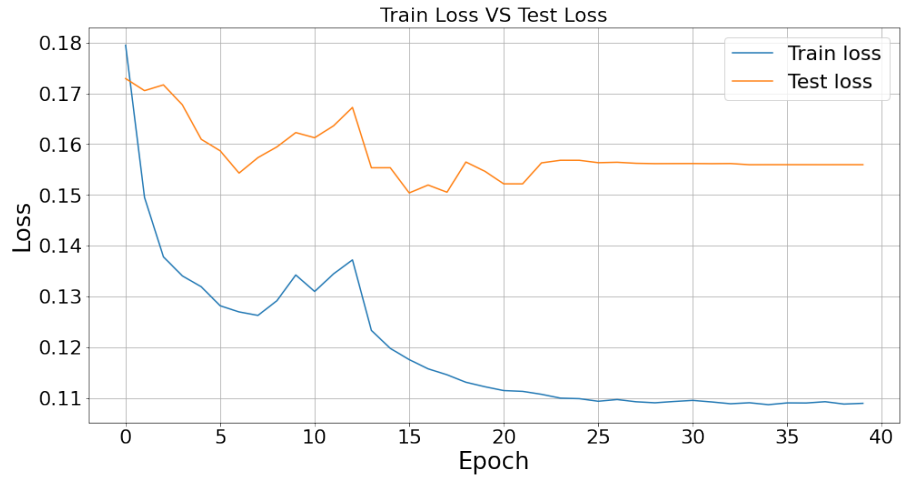


Figure 3.11: Training Loss VS Validation Loss in the *full training set*.

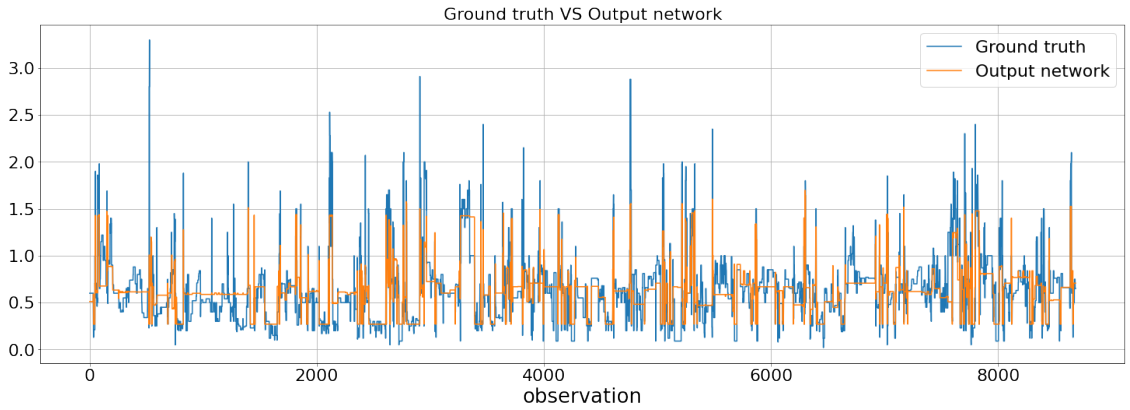


Figure 3.12: Ground truth VS Output Neural Network in *test set*

away from the true one. The *average* `current bid` in *test set* is equal to $0.63\$$. The capacity of the neural network's output to "follow" the true output is graphically inspected in Figure 3.12. The graph gives a raw idea of the fitting, but a quantitative assessment is needed. It is performed in the next subsection.

The CV confirms the robustness of the model (Figure 3.13). Different splits show similar behavior in train and validation loss. For example:

- split 1, the validation loss starts around ≈ 0.177 and drops to 0.172;
- split 2, the validation loss starts ≈ 0.18 and drops to ≈ 0.177;
- split 3, the validation loss starts ≈ 0.177 and drops to ≈ 0.16;

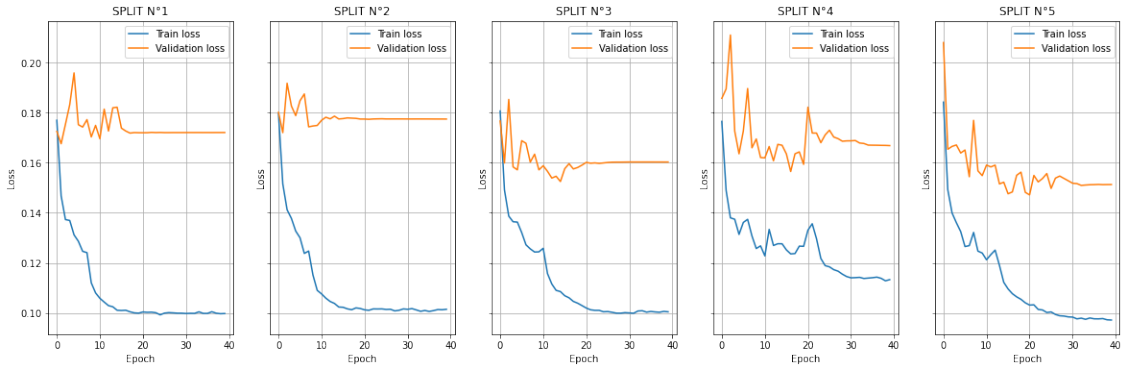


Figure 3.13: CV Neural Network

- split 4, the validation loss starts ≈ 0.185 and drops to ≈ 0.167;
- split 5 has an easier validation set. The validation loss starts at ≈ 0.21, then drop around 0.151.

Moreover, all splits show some overfitting, with the smallest validation loss reached within the first 20 epochs and in the range $[0.152, 0.172]$. The exception is split 5 which has an easier validation set and so its smallest value is around 0.147. Hence, in all splits the best validation loss is pretty the same. As expected, given the size of the dataset, different splits are similar.

3.2.2 Prediction Errors Analysis

The *average absolute error* is equal to $0.15\$$. This is a significant drop with respect to the $0.19\$$ achieved using Ridge and Lasso regression.

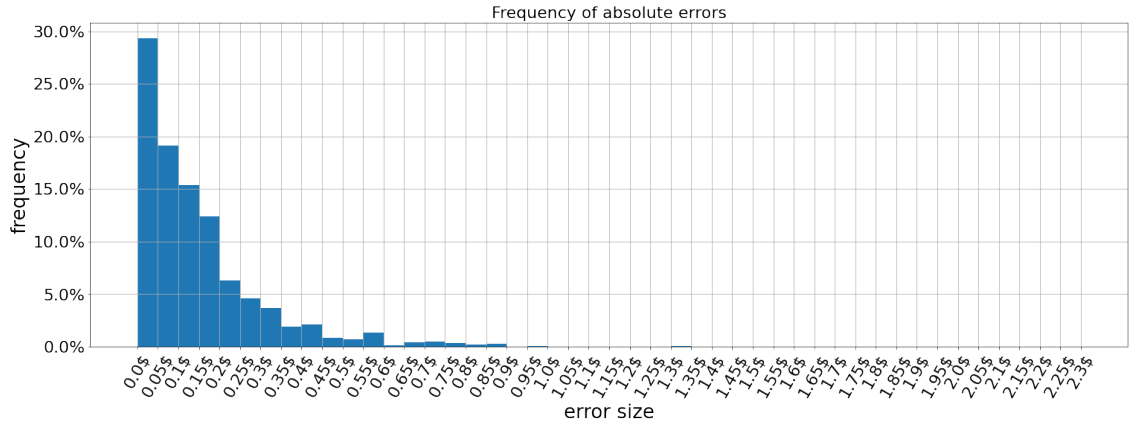
To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure 3.14.

The box-plot shows that 25% of errors are lower than 5 cents and 50% of errors are lower than 11 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 20 cents. There is another approximately 18% of cases where the error is between 20 and 45 cents. Errors above 45 cents are limited to the last 7% of data.

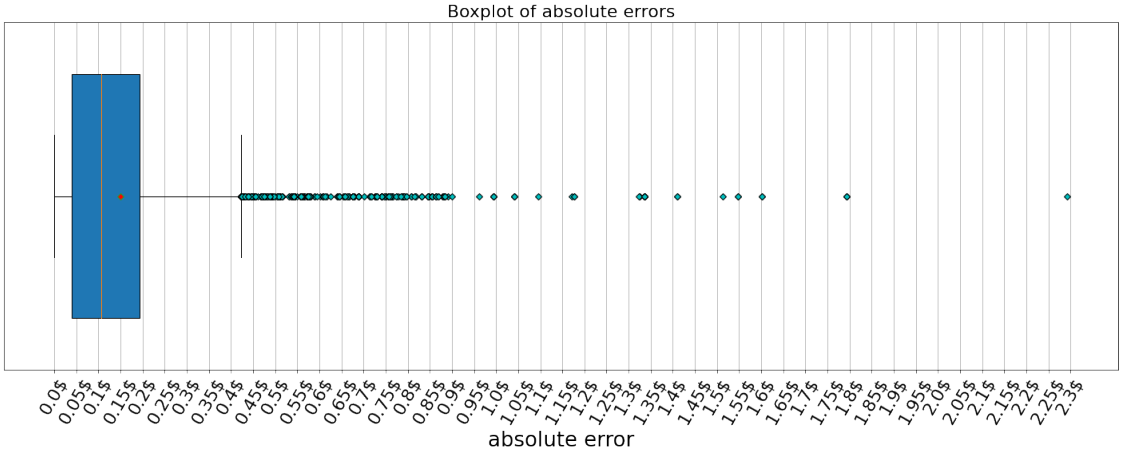
3.3 Negative Test Set

The reliability of results is counter checked picking observations:

- belonging to the same keywords used in *test set*;



(a) Neural Network's Absolute Errors distribution.



(b) Neural Network's Absolute Errors box-plot.

Figure 3.14: Neural Network's Absolute Error quantile distribution.

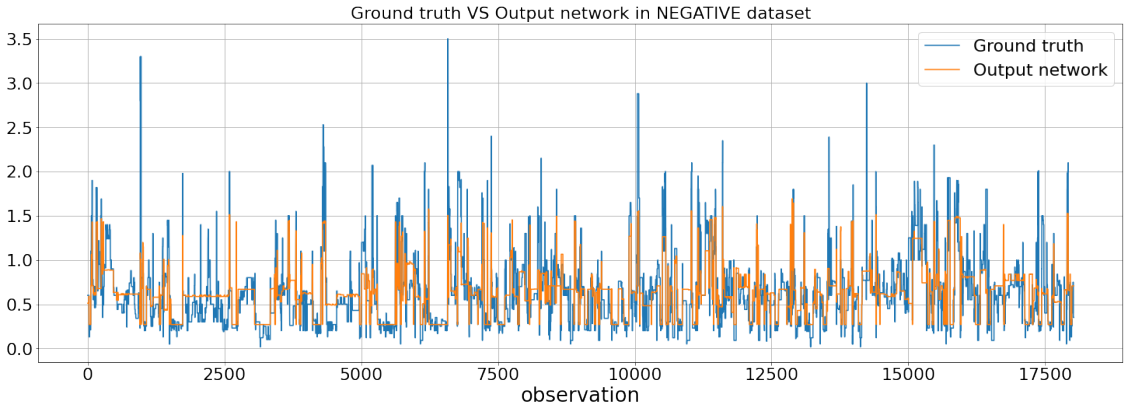


Figure 3.15: Ground truth VS Output Neural Network in *negative test set*

- with negative profitability (`profit-cost ratio` < -0.2).

Such observations constitute the *negative test set*. It allows a *ceteris paribus* comparison with the *test set*. In fact the only difference lies on the `profit-cost ratio`'s values.

The idea is that, since the network has learned the relationship between the set of features and the output for observations with positive profitability, the average error in the *negative test set* should be much higher than the one observed in the *test set*.

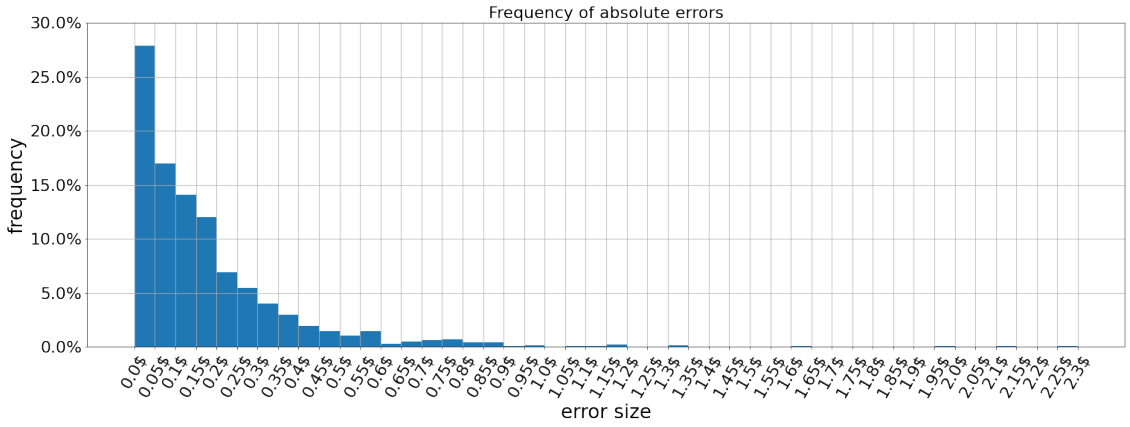
3.3.1 Results

The capacity of the neural network's output to "follow" the true output of *negative test set* is graphically inspected in Figure 3.15.

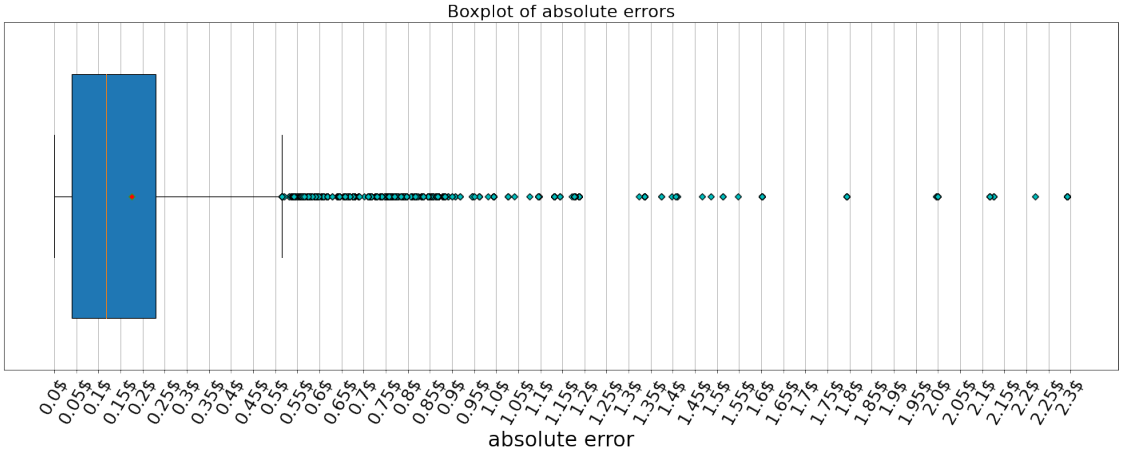
The *average absolute error* in the *negative test set* is equal to $0.17\$$ with the *average* `current bid` equal to $0.62\$$. This is a small increase with respect to the $0.15\$$ achieved in the *test set*. The result is unexpected and needs further analysis in order to be explained. This is going to be done in a while.

In the meantime, to get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure 3.16.

The error distribution and boxplot show similar results to the ones obtained in the *test set*. 25% of errors are lower than 5 cents and 50% of errors are lower than 12 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 27 cents. There is another approximately 15% of cases where the error is between 24 and 45 cents. Errors above 45 cents are limited to the last 10% of data.



(a) Neural Network's Absolute Errors distribution in *negative test set*.



(b) Neural Network's Absolute Errors box-plot.

Figure 3.16: Neural Network's Absolute Error quantile distribution.

3.3.2 Error Explanation

The goal of this subsection is to explain why the average prediction error of the neural network (the same it's true for any other model) in the *negative test set* is only slightly greater than the error in the *test set*.

In order to do so, it's necessary to go back to the original data and build a new dataset organized as follows:

1. both positive observations (`clicks in` > 0 and `profit-cost ratio` ≥ -0.2) and negative observations (`clicks in` > 0 and `profit-cost ratio` < -0.2) are taken;
2. observations are grouped at the keyword level.

This allows to get, for each keyword, all the observations "judgeable" (as positive or negative). An example is reported in Table 3.1. The column `profit-cost ratio` contains positive observations, highlighted in green, and negative observations, highlighted in red. The column `current bid` contains the actual bid submitted by the advertiser. In this case it is always equal to $1.0\$$.

date	keyword name	keyword ID	match type	device	clicks in	current bid	profit-cost ratio
2021-03-08	0 purchase credit cards	1039595542	broad	desktop	3	1.0	−0.33
2021-03-09	0 purchase credit cards	1039595542	broad	desktop	3	1.0	0.02
2021-03-10	0 purchase credit cards	1039595542	broad	desktop	8	1.0	−1.00
2021-03-11	0 purchase credit cards	1039595542	broad	desktop	1	1.0	−1.00
2021-03-12	0 purchase credit cards	1039595542	broad	desktop	4	1.0	−1.00
2021-03-13	0 purchase credit cards	1039595542	broad	desktop	1	1.0	3.63
2021-03-15	0 purchase credit cards	1039595542	broad	desktop	1	1.0	5.70
2021-03-16	0 purchase credit cards	1039595542	broad	desktop	1	1.0	6.96
2021-03-18	0 purchase credit cards	1039595542	broad	desktop	1	1.0	−1.00
2021-03-23	0 purchase credit cards	1039595542	broad	desktop	2	1.0	−1.00

Table 3.1: Positive and negative observations for a sample keyword.

The interesting aspect is that the same `current bid` can generate positive profit in one day and negative profit in another one. This behavior is in line with the analysis carried out in Subsection 2.2.3. In fact, for a given keyword, the effectiveness of a `current bid`'s value to generate positive profit depends not just on the bid itself, but also on the market conditions. Different market conditions can turn a profitable bid in an unprofitable one (or viceversa).

It's important to notice that this behavior it's independent by the use of the CPC-Optimizer. It can increase the frequency with which the bids are updated and reduce the proportion of negative observations, but anyway the presence of positive and negative profits associated to the same (or similar) bid's value, will remain. This is because the `current bid` is adjusted mainly as reaction to a negative profit: first the bid is profitable and so not (or a little) adjusted, then it becomes non-profitable and so a significant adjustment is needed. This process implies that a similar bid's value is associated with positive and negative profits.

When the *trained* network (on `main dataset`, hence with positive observations) receives as input the features from the *negative test set*, it will produce an estimated $\widehat{current\ bid}$ based on what learned on positive observations. The point is that this $\widehat{current\ bid}$ is also the true `current bid` for many observations in the *negative test set*. Hence, the average error in the *negative test set* will be just a little higher.

In order to quantify the level at which the above behavior affect the *test set*, for each pair {`keyword name`, `current bid`} contained in the *test set*, the "proportion of positive observations" is computed and the corresponding distribution plotted (Figure 3.17).

For example, in Table 3.1 the pair {0 purchase credit cards, 1.0} counts 10 observations, of which 4 are positive. Hence, the "proportion of positive observations" for this pair is equal to 0.4.

The distribution of the "proportion of positive observations" conveys the following message:

- $\approx$ 30% of {`keyword name`, `current bid`} contain 0% of positive observations. In other words, contains only negative observations.
- $\approx$ 8% of {`keyword name`, `current bid`} contain 100% of positive observations. In other words, contains only positive observations.
- The remaining 62% of {`keyword name`, `current bid`} contain positive and negative observations. This implies that the same bid is associated to both positive and negative profit.

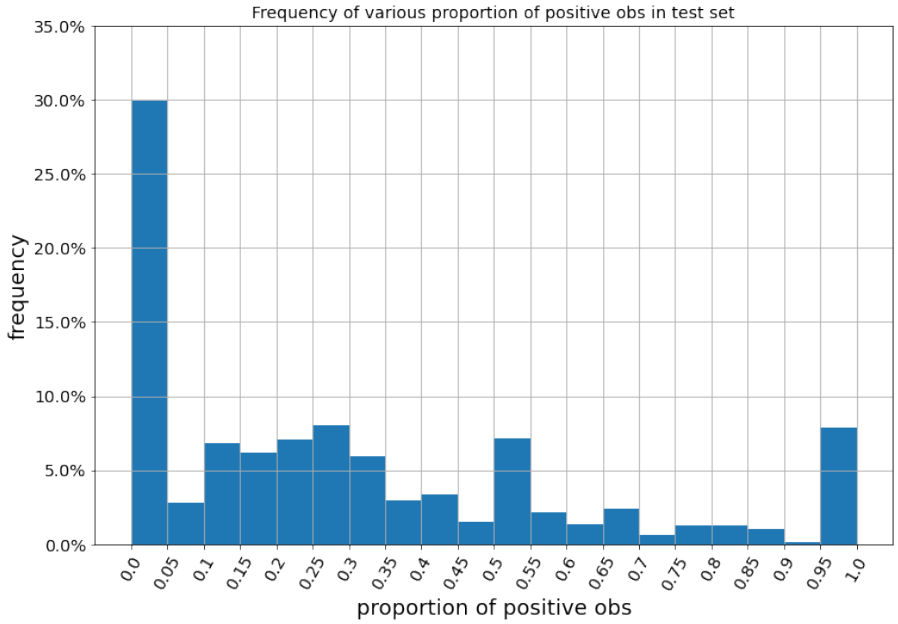


Figure 3.17: Proportion of positive observations in *test set*

This evidence makes not reliable to judge the profitability at the *observation level*. It appears more sound to assess the profitability at the *keyword level*. This means to gauge as positive or negative, not the single observation, but the whole set of observations belonging to the same keyword. For example, in Table 3.1, the keyword "0 purchase credit cards" is positive, since it's average `profit-cost ratio` is positive.

Changing the granularity level at which profitability is judged, implies to change the way in which the positive observations are selected. Hence, the training set has to be rebuilt and the training procedure redone. This is the task of the next chapter.

4

Alternative Datasets Design

The chapter tackles the problem emerged in Subsection 3.3.2: assessing the profitability at the *observation level* is not meaningful, hence the assessment is shifted at the *keyword level*. In order to do so, a new dataset is built selecting *positive keywords* instead of *positive observations*. This dataset is referred as `main dataset keyword level`.

In detail, the chapter is composed by four sections:

- Datasets Construction: the new datasets are built.
- Training Error Lower Bound: an estimation of the minimum achievable training error for the new datasets is provided.
- Neural Network Model with BERT Encoding: keywords' names are encoded as done until now, using BERT. The same training procedure described in Section 3.2.1 is followed and the network's outcome is checked against *test set* and *negative test set*.
- Neural Network Model with GloVe Encoding: keywords' names are encoded using GloVe (Global Vectors). The basic idea behind this new encoding technique is presented. The same training procedure described in Section 3.2.1 is followed and the network's outcome is checked against *test set* and *negative test set*.

4.1 Datasets Construction

An intermediate dataset is built from the original data as follows:

1. positive observations (`clicks in` > 0 and `profit-cost ratio` ≥ -0.2) and negative observations (`clicks in` > 0 and `profit-cost ratio` < -0.2) are taken;
2. observations are grouped at the keyword level.

This allows to get, for each keyword, all the observations "judgeable" (as positive or negative). All observations not "judgeable" are removed from the intermediate dataset.

The next step is to define when a keyword is *positive* and when it isn't. All keywords with average `profit-cost ratio` ≥ -0.2 are classified as *positive*, otherwise as *negative*. All negative keywords are removed from the intermediate dataset.

Then, for each encoding technique (BERT and GloVe), a new dataset is built starting from the intermediate one:

1. each keyword name is converted in a vector;
2. the encoding is added to the set of features of the intermediate dataset;
3. the MIMIR dataset is joined according to the value of columns `date` and `keyword name`;
4. all the columns unknown *ex ante* are removed, except for `current bid`, which plays the role of dependent variable. Also the column `keyword ID` is removed, since each keyword is now identified by its encoding. In this way, the dataset contains only features known *ex ante* and observations belonging to *positive* keywords. An overview of the set of features when BERT encoding is used is shown in Table 2.6. The only difference in case of GloVe is that the encoding features are $f_1, \ldots, f_{50}$.
5. `match type` is One-Hot-Encoded.

These are exactly the same steps applied for the `main dataset` in previous chapters.

The resulting dataset is called:

- `main dataset keyword level (BERT Encoding)`, in case of BERT encoding;
- `main dataset keyword level (GloVe Encoding)`, in case of GloVe encoding.

The term `keyword level` indicates that the profitability is assessed looking at all judgeable observations belonging to a keyword; the term `(BERT Encoding)` or `(GloVe Encoding)`, is necessary to avoid confusion about the encoding used.

4.2 Training Error Lower Bound

The training set is scanned and identical observations are grouped together. Take one of such groups as an example. If it contains all observations with the same output, then the *proportion of identical outputs* is 100% for this group.

If it contains two different outputs, then there is a certain proportion, e.g. 0.9, of observations associated to an output and another 0.1 of observations associated to a different output. In this case the highest proportion is considered, and so it's possible to say that 0.9 of observations are associated with the same output.

If it contains 10 different outputs, then there is a certain proportion, e.g. 0.1, of observations associated to each of the 10 different outputs. In this case, as always, the highest proportion is considered, and so it's possible to say that 0.1 of observations are associated with the same output.

This allows to verify not only the presence of different outputs associated to the same input, but also to quantify the uncertainty. In fact, having the highest proportion of identical outputs equal to 0.9, means that the neural network can learn a strong relationship. Having the highest proportion of identical outputs equal to 0.1 means that the neural network cannot learn any relationship.

Repeating this analysis for every group and storing the corresponding proportions in a list, it's possible to draw a histogram in order to have an idea of the proportions' distribution in the entire training set. For example, in Figure 4.1, approximately 48% of groups have a unique (or more that 95% of times) output. As another example, almost 10% of groups have half of observations associated to the same output while the other half is associated to different output/s.

Moreover, for each group j, the weighted average of outputs ($\overline{output}_j$) is computed and its distance from the output of each observation i in the same group is calculated:

$$error_i = |\overline{output}_j - output_i|$$

Storing these values for each observation and for each group, allows to compute the average error ($\overline{error}$) in the training set, which is a reasonable approximation of the *minimum achievable training error*. It is shown as a red dotted line at $0.064\$$, together with the entire $error$ distribution in Figure 4.2.

In fact, if, by assumption, all groups contains a unique output, the $error$ in each group is 0

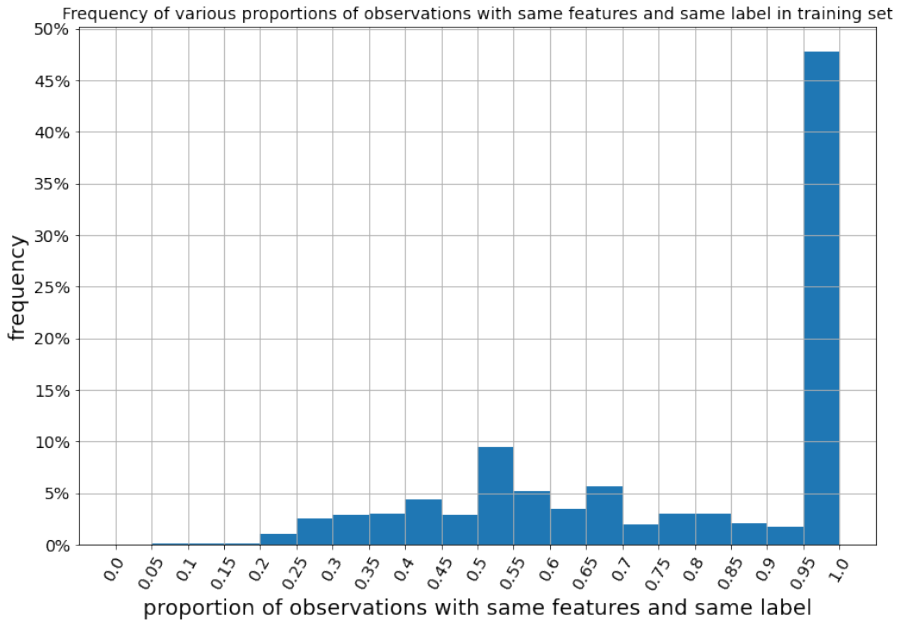


Figure 4.1: Distribution of *proportion of the same observations* in the training set.

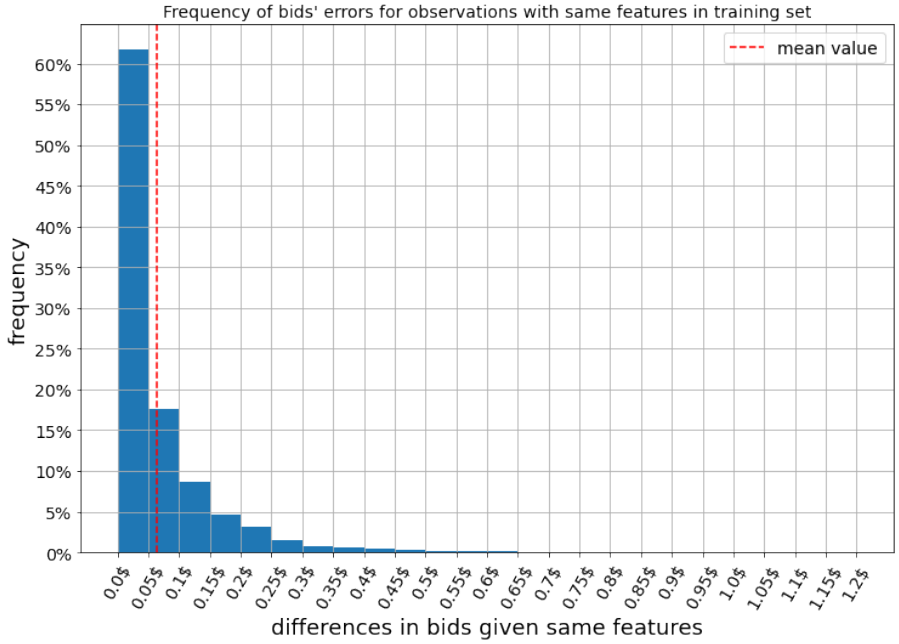


Figure 4.2: Distribution of *bids' errors* in the training set.

and so also the average training error is 0. This is the case of a network which is perfectly fitting the training set just learning it by heart.

If, more realistically, different groups contain different proportions of identical outputs, the $error$ is greater than 0, but anyway it's not possible to do better. In all the groups where are present different outputs, the network is forced to learn a unique output to provide as estimate. In order to do so, the network learns a sort of weighted average of outputs.

Note that the above results are independent by the encoding technique used, so are valid for BERT and GloVe.

4.3 Neural Network Model with BERT Encoding

Similarly to what done with the `main dataset`, the feature `keyword name` is used to group `main dataset keyword level (BERT Encoding)` by keyword and split it at the *keyword level* (instead of the *observation level*). This ensures that all observations belonging to a keyword appear in just one of the training, validation and test sets.

The same training procedure described in Section 3.2.1 is used. It is not repeated here for brevity and to avoid redundancy. What is reported here is:

- the best model selected;
- the results on full training set.

4.3.1 Best Model Selection

The best model found has:

- same architecture depicted in Figure 3.9;
- features $f_1, \ldots, f_{1024}$, `match type broad`, `match type exact`, `match type phrase`, `suggested bid`, `competition`;
- adjusting learning rate;
- dropout probability equal to 0.2 and no weight decay;
- loss function MAE.

Figure 4.3: Training Loss VS Validation Loss in the *full training set*.

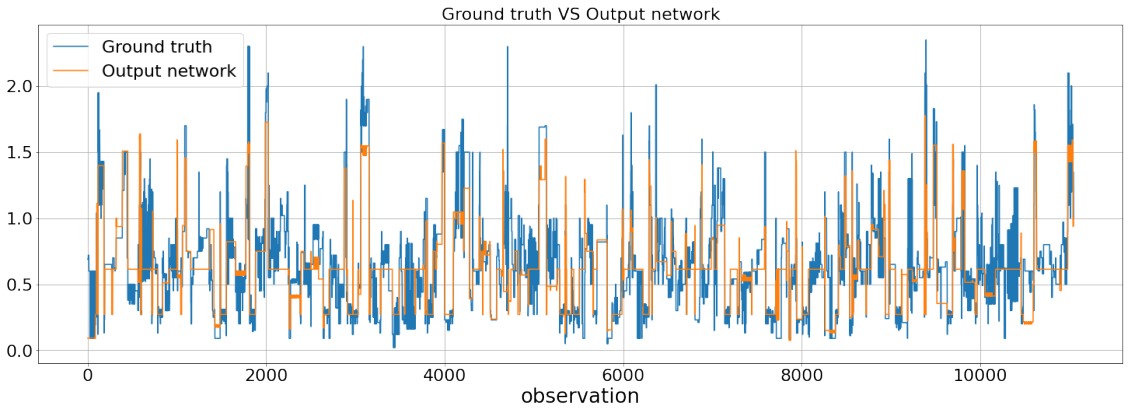


Figure 4.4: Ground truth VS Output Neural Network in *test set*

4.3.2 Results on Full Training Set

The average absolute error achieved by the baseline models (multiple linear regression, Ridge and Lasso) are $0.46\$$, $0.20\$$ and $0.20\$$, respectively. They can be found in Appendix C.

The best model is trained over the *full training set* and tested on the *test set*. Results are shown in Figure 4.3. The test loss reaches a minimum value around 0.16, then starts some overfitting. This means that the model predicts a $\widehat{current\ bid}$ which is, on average, 16 cents of dollar far away from the true one. The *average* `current bid` in the *test set* is equal to $0.66\$$. The capacity of the neural network's output to "follow" the true output is graphically inspected in Figure 4.4. The graph gives a raw idea of the fitting, but a quantitative assessment is needed.

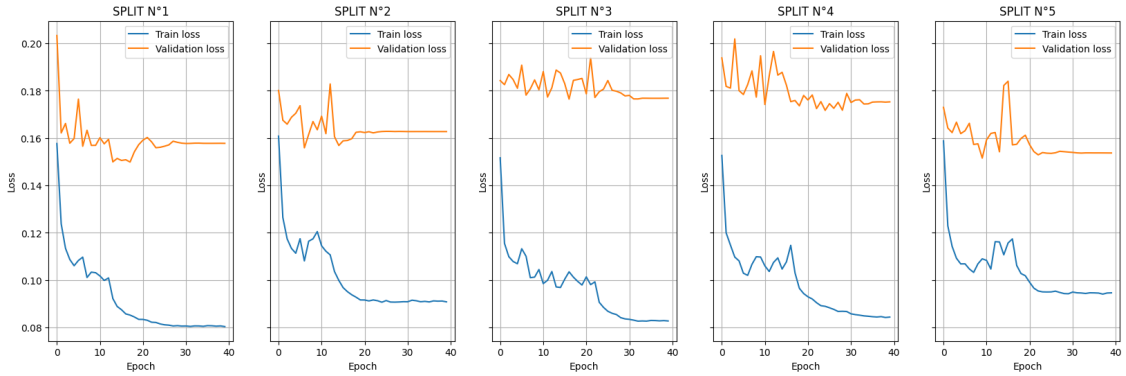


Figure 4.5: CV Neural Network

It is performed in the next subsection.

The CV confirms the robustness of the model (Figure 4.5). Different splits show similar behavior in train and validation loss. For example:

- split 1, the validation loss starts around ≈ 0.203 and drops to 0.157;
- split 2, the validation loss starts ≈ 0.18 and drops to ≈ 0.163;
- split 3, the validation loss starts ≈ 0.184 and drops to ≈ 0.177;
- split 4, the validation loss starts ≈ 0.194 and drops to ≈ 0.175;
- split 5, the validation loss starts ≈ 0.172 and drops to ≈ 0.154.

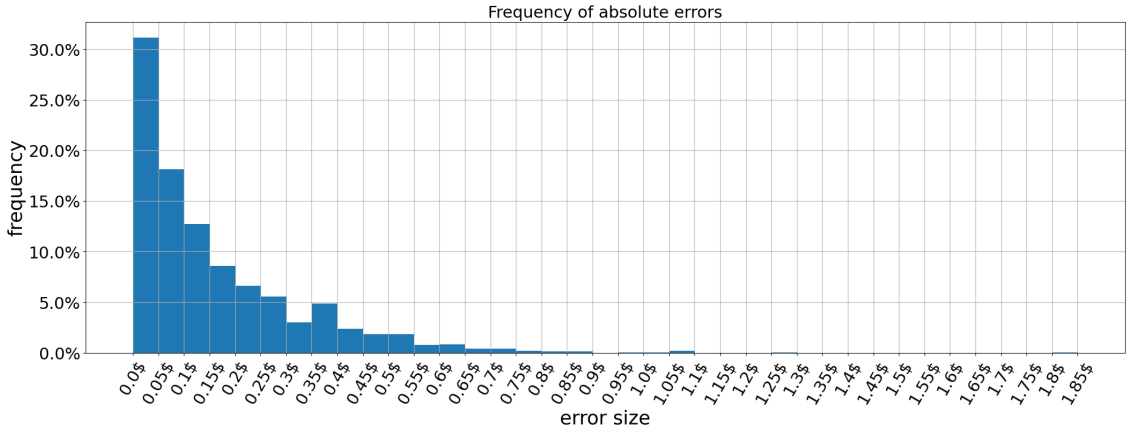
Moreover, all splits show some overfitting, with the smallest validation loss reached within the first 20 epochs and in the range $[0.150, 0.172]$. Hence, in all splits the best validation loss is pretty the same. As expected, given the size of the dataset, different splits are similar.

4.3.3 Prediction Errors Analysis

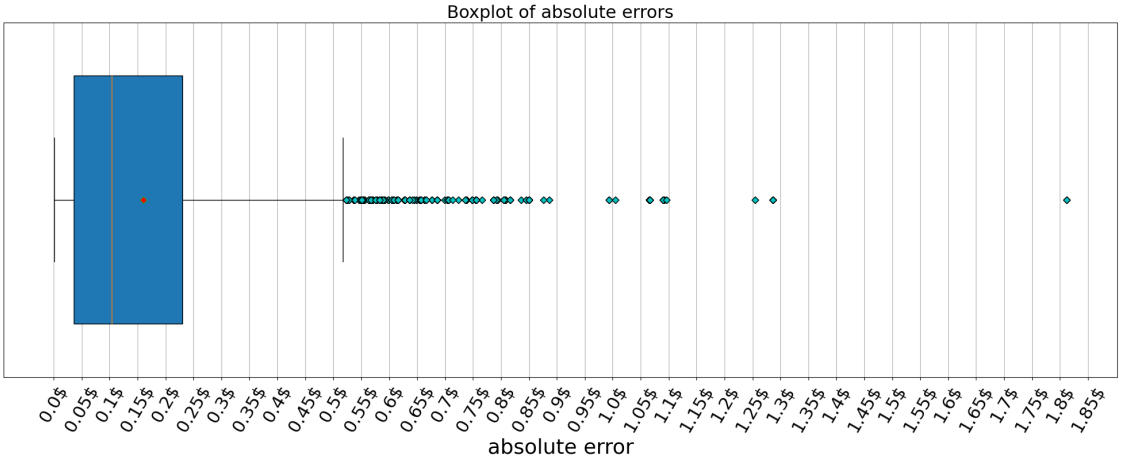
The *average absolute error* is equal to $0.16\$$. This is a significant drop with respect to the $0.20\$$ achieved using Ridge and Lasso regression.

To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure 4.6.

The box-plot shows that 25% of errors are lower than 5 cents and 50% of errors are lower than 11 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 24 cents. There is another approximately 15% of cases where the error is between 25 and 45 cents. Errors above 45 cents are limited to the last 10% of data.



(a) Neural Network's Absolute Errors distribution.



(b) Neural Network's Absolute Errors box-plot.

Figure 4.6: Neural Network's Absolute Error quantile distribution.

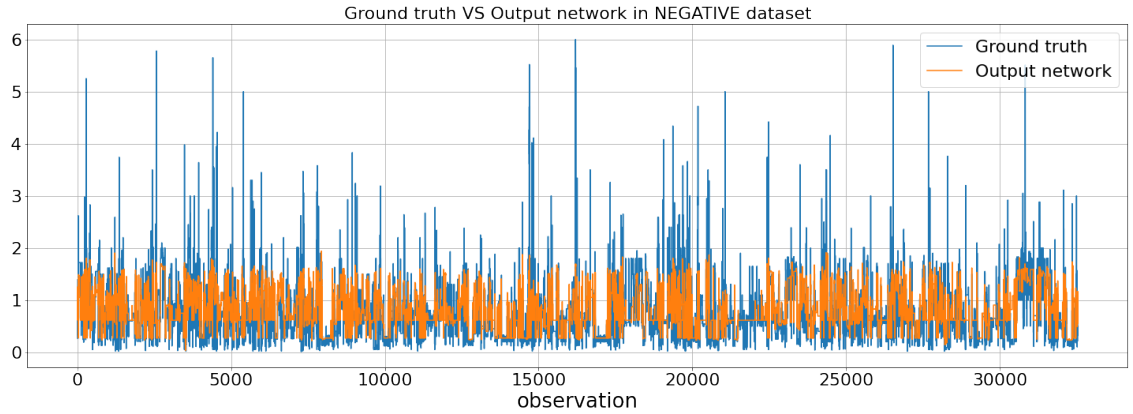


Figure 4.7: Ground truth VS Output Neural Network in *negative test set*

4.3.4 Negative Test Set

The reliability of results is counter checked picking observations belonging to the *negative* keywords. They have not been used in the training procedure, so are unknown observations for the neural network. Such observations constitute the *negative test set*.

The idea is that, since the network has learnt the relationship between the set of features and the output for observations belonging to positive keywords, the average error in *negative test set* should be much higher than the one observed in the *test set*.

The capacity of the neural network's output to "follow" the true output of the *negative test set* is graphically inspected in Figure 4.7.

The *average absolute error* in the *negative test set* is equal to $0.30\$$, almost the double with respect to the $0.16\$$ achieved in the *test set*. Negative keywords seem to be associated with higher average bids: the *average* `current bid` is equal to $0.74\$$.

In order to check that the higher *average absolute error* is not just generated by some exceptionally high bids, it is recomputed using only negative keywords with `current bid` $\leq 2.0\$$, which is the maximum value of bids observed for the positive keywords.

The capacity of the neural network's output to "follow" the true output of the *negative test set* is graphically inspected in Figure 4.8.

The *average absolute error* is equal to $0.28\$$. This confirms that the neural network's predictions are, as expected, significantly different from the `current bid` observed in the *negative test set*.

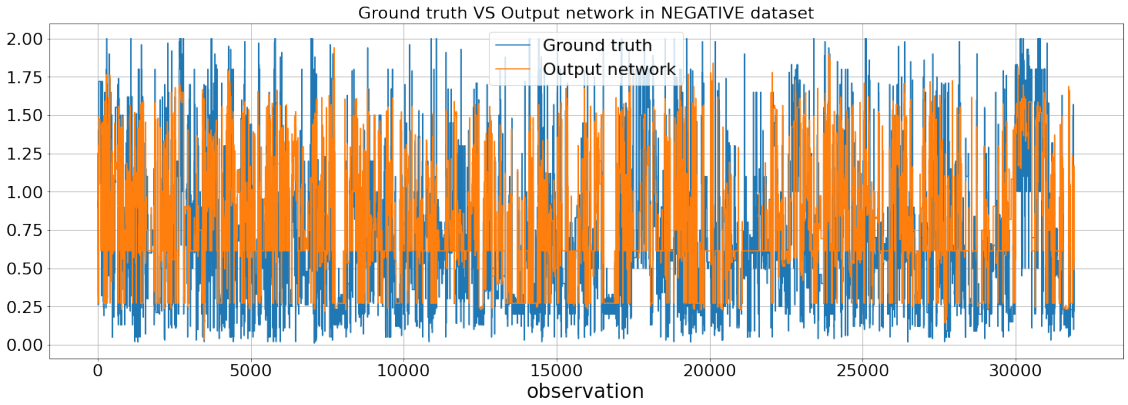


Figure 4.8: Ground truth VS Output Neural Network in *negative test set*

4.4 Neural Network Model with GloVe Encoding

The *training error lower bound* has set a limit to the learning capacity of the neural network. Hence, it's not possible to improve the model reducing further the error. But it's possible to try to improve the model differently: reducing its complexity preserving (almost) the same performances.

The wider margin of complexity reduction is on the encoding technique used. BERT is the state of the art in Natural Language Processing, but even in its simpler version, it counts 128 encoding features in each of the two layers. Exists other encoding techniques requiring many less features. Of course there is a trade-off between encoding complexity and quality of the representation.

At first attempt, one of the encoding techniques which requires, in absolute, the least number of features is tried: GloVe with 50 dimensions.It is able to encode any token/word in a vector with dimension equal to 50. This will reduce by almost 20 times the number of features, hence a new suitable model architecture and hyperparameters are needed.

If the test error will be similar to the one achieved with BERT encoding (0.16$), then GloVe will prove to be enough good to the data available. At the same time, the greater complexity of BERT will be not justified by a significantly better performance. In this case, the preferred model will be the neural network with GloVe encoding.

If the test error will be significantly greater than the one achieved with BERT, than encoding techniques more complex than GloVe will be tried.

Before going further, it's necessary to introduce what GloVe model is and the main idea behind it. The next subsection takes care of this.

4.4.1 GloVe

Global Vectors (GloVe) is an unsupervised learning algorithm for obtaining vector representations for words [21]. *Global* refers to the fact that training is performed on word-word co-occurrence statistics computed on the whole corpus.

Beside GloVe, exist two main model families for learning word vectors:

1. global matrix factorization methods, such as latent semantic analysis (LSA) (Deerwester et al., 1990). In the latter, the matrix is of "term-document" type, i.e., the rows correspond to words or terms, and the columns correspond to different documents in the corpus;

2. local context window methods, such as the skip-gram model of Mikolov et al. (2013c).

Both families suffer significant drawbacks. Methods like LSA efficiently leverage statistical information, but they do relatively poorly on the word analogy task, indicating a sub-optimal vector space structure. Methods like skip-gram may do better on the analogy task, but they poorly utilize the statistics of the corpus since they train on separate local context windows instead of on global co-occurrence counts.

GloVe overcomes both these issues.

For example, *man* may be regarded as similar to *woman* in that both words describe human beings; on the other hand, the two words are often considered opposite since they highlight a primary axis along which humans differ from one another.

In order to capture in a quantitative way the nuance necessary to distinguish *man* from *woman*, it is necessary for a model to associate more than a single number to the word pair. A natural and simple candidate for an enlarged set of discriminative numbers is the vector difference between the two word vectors. GloVe is designed in order that such vector differences capture as much as possible the meaning specified by the juxtaposition of two words.

The underlying concept that distinguishes *man* from *woman*, i.e. sex or gender, may be equivalently specified by various other word pairs, such as *king* and *queen* or *brother* and *sister*. To state this observation mathematically, we might expect that the vector differences:

- $man - woman$;
- $king - queen$;
- $brother - sister$

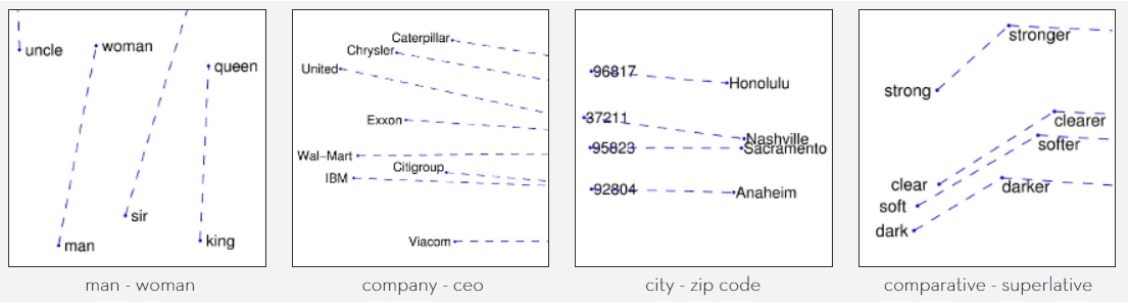


Figure 4.9: Examples of linear structures discovered by GloVe.

might all be roughly equal. This property and other interesting patterns can be observed in Figure 4.9.

The GloVe model is trained only on the non-zero entries of a global word-word co-occurrence matrix (rather than on the entire sparse matrix or on individual context windows in a large corpus), which tabulates how frequently words co-occur with one another in a given corpus. Populating this matrix requires a single pass through the entire corpus to collect the statistics. Here the training set is the combination of Gigaword5+Wikipedia2014 for a total of 6 billion tokens. Each term in the dataset is tokenized and lowercased; then a vocabulary of the 400,000 most frequent words is built.

These terms are used to compute the matrix of word-word co-occurrence counts, denoted by $\mathbf{X}$, whose entry $\mathbf{X}_{ik}$ tabulates the number of times word k occurs in the context of word i. The number of words in the context of a word i, $\mathbf{X}_i$, is determined by a decreasing weighting function, so that any word k that are d words apart contribute $1/d$ to the total count of $\mathbf{X}_i$. This is one way to account for the fact that very distant word pairs are expected to contain less relevant information about the words' relationship to one another. Let $\mathbf{X}_i = \sum_k \mathbf{X}_{ik}$ be the number of times any word appears in the context of word i. Finally, let $P_{ik} = P(k|i) = \frac{\mathbf{X}_{ik}}{\mathbf{X}_i}$ be the probability that word k appears in the context of word i.

The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning. For example, consider the co-occurrence probabilities for target words $i = ice$ and $j = steam$ with various probe words from the vocabulary. The relationship of these words can be examined by studying the ratio of their co-occurrence probabilities with various probe words, k. For words k related to *ice* but not *steam*, say $k = solid$, we expect the ratio P_{ik}/P_{jk} will be large.

Some actual probabilities from a 6 billion word corpus are shown in Table 4.1:

As one might expect, *ice* co-occurs more frequently with *solid* than it does with *gas*, whereas

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Table 4.1: Examples of co-occurrence probabilities discovered by GloVe.

steam co-occurs more frequently with *gas* than it does with *solid*. Both words co-occur with their shared property *water* frequently, and both co-occur with the unrelated word *fashion* infrequently. Only in the ratio of probabilities does noise from non-discriminative words like *water* and *fashion* cancel out, so that large values (much greater than 1) correlate well with properties specific to *ice*, and small values (much less than 1) correlate well with properties specific to *steam*. In this way, the ratio of probabilities encodes some crude form of meaning associated with the abstract concept of thermodynamic phase.

The above argument suggests that the appropriate starting point for word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves. The most general model takes the form:

$$F(w_i, w_j, w_k) = \frac{P_{ik}}{P_{jk}}$$

where $w \in \mathbb{R}^d$. The number of possibilities for F is vast, but by enforcing a few desiderata it's possible to get a unique choice. We would like F to encode the information present in the ratio P_{ik}/P_{jk} in the word vector space. The most natural way to do this is with vector differences:

$$F(w_i - w_j, w_k) = \frac{P_{ik}}{P_{jk}}$$

The final model takes the form of a *weighted least squares regression*:

$$J = \sum_{i,k=1}^{V} f\Big(\mathbf{X}_{ik}\Big)\Big(w_i^T w_k + b_i + b_k - \log \mathbf{X}_{ik}\Big)^2$$

where V is the vocabulary's size, $f(\mathbf{X}_{ik})$ is the weighting function, b_i and b_k are bias terms for words i and k.

The evaluation method is based on *word analogies*: it probes the finer structure of the word vector space by examining not the scalar distance between word vectors, but rather their various

dimensions of difference. For example, the analogy "king is to queen as man is to woman" should be encoded in the vector space by the vector equation:

$$king - queen = man - woman.$$

The semantic statement is: "king is to queen as man is to _". To correctly complete the statement, the model should uniquely identify the missing term, with only an exact correspondence counted as a correct match. The answer consists in the vector $woman$ closest to $man + queen - king$ according to the cosine similarity.

Similarly to what done with the `main dataset`, the feature `keyword name` is used to group `main dataset keyword level (GloVe Encoding)` by keyword and split it at the *keyword level* (instead of the *observation level*). This ensures that all observations belonging to a keyword appear in just one of the training, validation and test sets.

The same training procedure described in Section 3.2.1 is used. It is not repeated here for brevity and to avoid redundancy. What is reported here is:

- the best model selected;
- the results on full training set.

4.4.2 Best Model Selection

The best model found has:

- architecture depicted in Figure 4.10;
- features $f_1, \ldots, f_{50}$, `match type broad`, `match type exact`, `match type phrase`, `suggested bid`, `competition`;
- adjusting learning rate;
- dropout probability equal to 0.1 and weight decay equal to $1e-4$;
- loss function MAE.

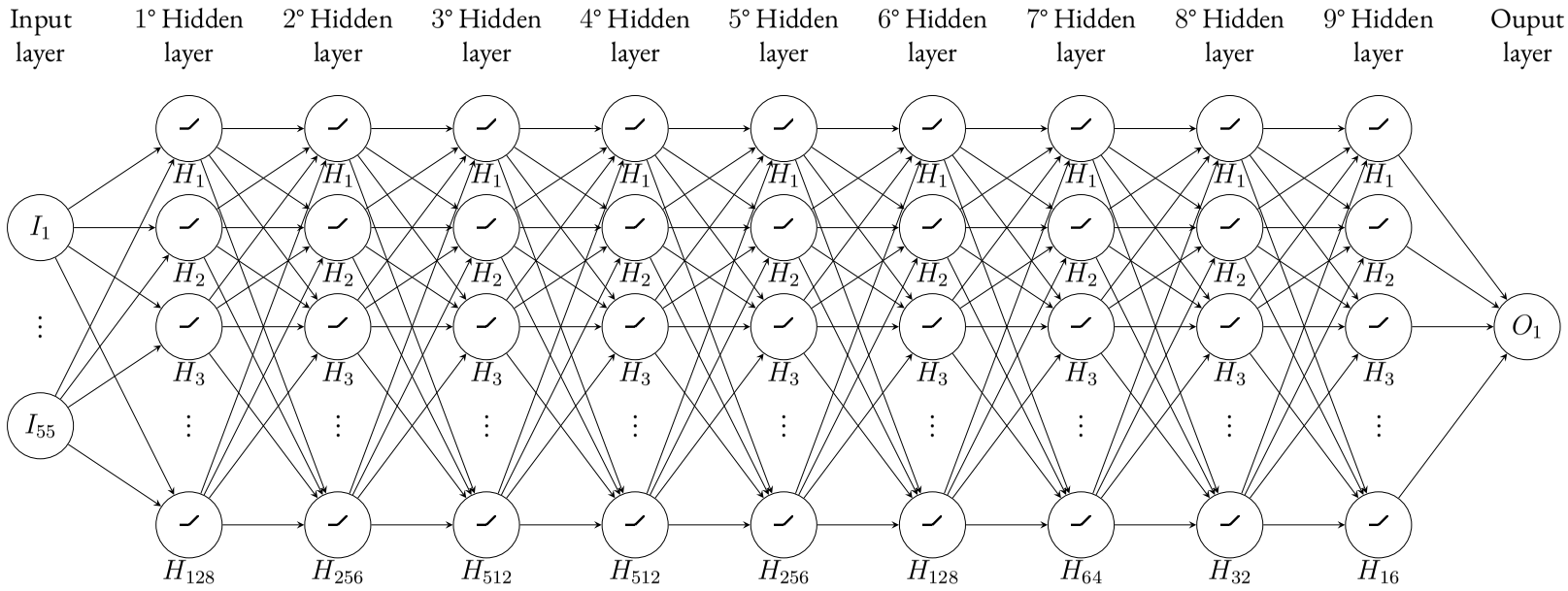


Figure 4.10: Fully Connected Neural Network Architecture with GloVe encoding.

Figure 4.11: Training Loss VS Validation Loss in the *full training set*.

4.4.3 Results on Full Training Set

The average absolute error achieved by the baseline models (multiple linear regression, Ridge and Lasso) is $0.22\$$ for all models. They can be found in Appendix C.

The best model is trained over the *full training set* and tested on the *test set*. Results are shown in Figure 4.11. The test loss reaches a minimum value around 0.17. This means that the model predicts a $\widehat{current\ bid}$ which is, on average, 17 cents of dollar far away from the true one. The *average* `current bid` in *test set* is equal to $0.66\$$. The capacity of the neural network's output to "follow" the true output is graphically inspected in Figure 4.12. The graph gives a raw idea of the fitting, but a quantitative assessment is needed. It is performed in the next subsection.

The CV confirms the robustness of the model (Figure 4.13). Different splits show similar behavior in train and validation loss. For example:

- split 1, the validation loss starts around ≈ 0.165 and drops to 0.16;
- split 2, the validation loss starts ≈ 0.174 and drops to ≈ 0.172;
- split 3, the validation loss starts ≈ 0.181 and drops to ≈ 0.169;
- split 4, the validation loss starts ≈ 0.212 and drops to ≈ 0.175;
- split 5, the validation loss starts ≈ 0.181 and drops to ≈ 0.159.

Moreover, all splits show some overfitting, with the smallest validation loss reached within the first 20 epochs and in the range $[0.154, 0.168]$. Hence, in all splits the best validation loss is pretty the same. As expected, given the size of the dataset, different splits are similar.

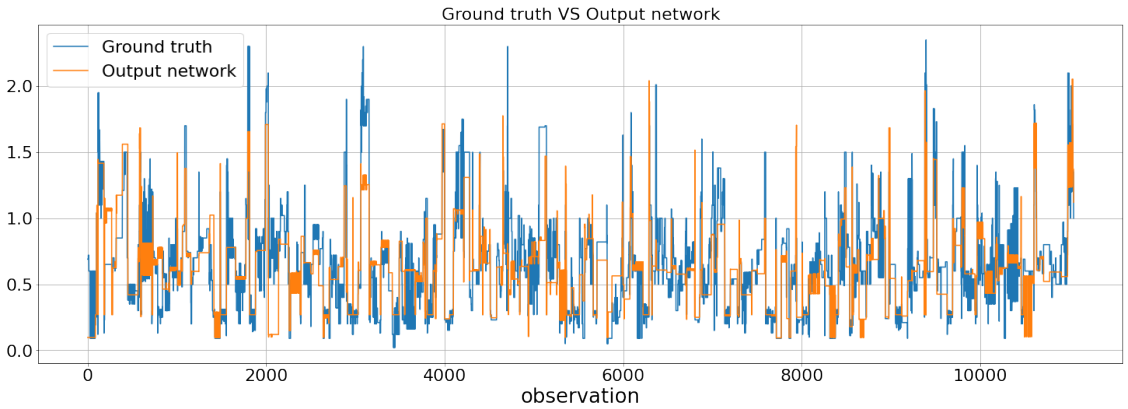


Figure 4.12: Ground truth VS Output Neural Network in *test set*

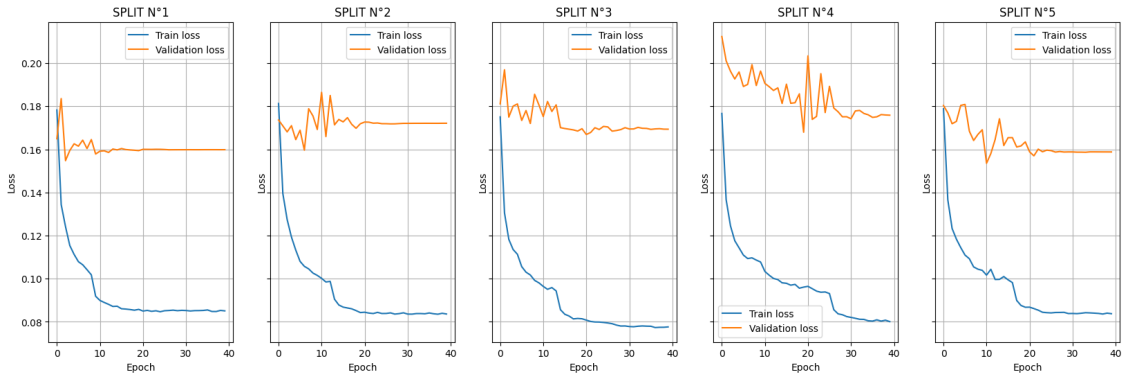


Figure 4.13: CV Neural Network

4.4.4 Prediction Errors Analysis

The *average absolute error* is equal to $0.17\$$. This is a significant drop with respect to the $0.22\$$ achieved using multiple linear, Ridge and Lasso regression.

To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure 4.14.

The box-plot shows that 25% of errors are lower than 5 cents and 50% of errors are lower than 11 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 25 cents. There is another approximately 17% of cases where the error is between 25 and 45 cents. Errors above 45 cents are limited to the last 8% of data.

4.4.5 Negative Test Set

The reliability of results is counter checked picking observations belonging to the *negative* keywords. They have not been used in the training procedure, so are unknown observations for the neural network. Such observations constitute the *negative test set*.

The idea is that, since the network has learnt the relationship between the set of features and the output for observations belonging to positive keywords, the average error in the *negative test set* should be much higher than the one observed in the *test set*.

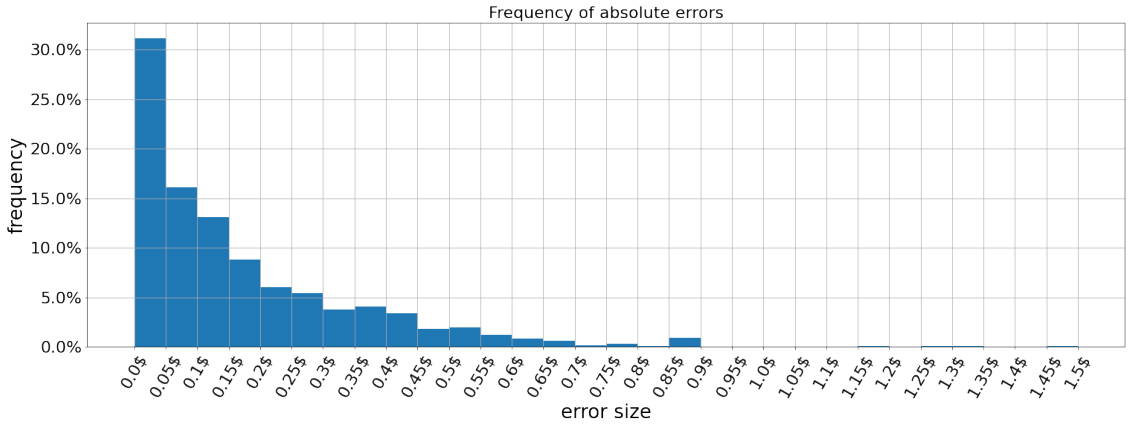
The capacity of the neural network's output to "follow" the true output of the *negative test set* is graphically inspected in Figure 4.15.

The *average absolute error* in *negative test set* is equal to $0.32\$$, almost the double with respect to the $0.17\$$ achieved in *test set*. Negative keywords seem to be associated with higher average bids: the *average* `current bid` is equal to $0.74\$$.

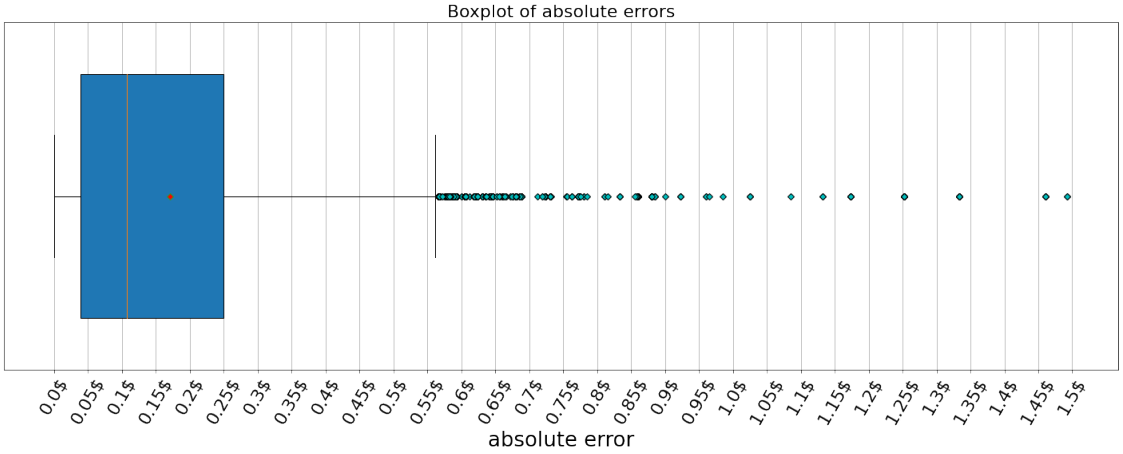
In order to check that the higher *average absolute error* is not just generated by some exceptionally high bids, it is recomputed using only negative keywords with `current bid` $\leq 2.0\$$.

The capacity of the neural network's output to "follow" the true output of the *negative test set* is graphically inspected in Figure 4.16.

The *average absolute error* is equal to $0.30\$$. This confirms that the neural network's predictions are, as expected, significantly different from the `current bid` observed in the *negative test set*.



(a) Neural Network's Absolute Errors distribution.



(b) Neural Network's Absolute Errors box-plot.

Figure 4.14: Neural Network's Absolute Error quantile distribution.

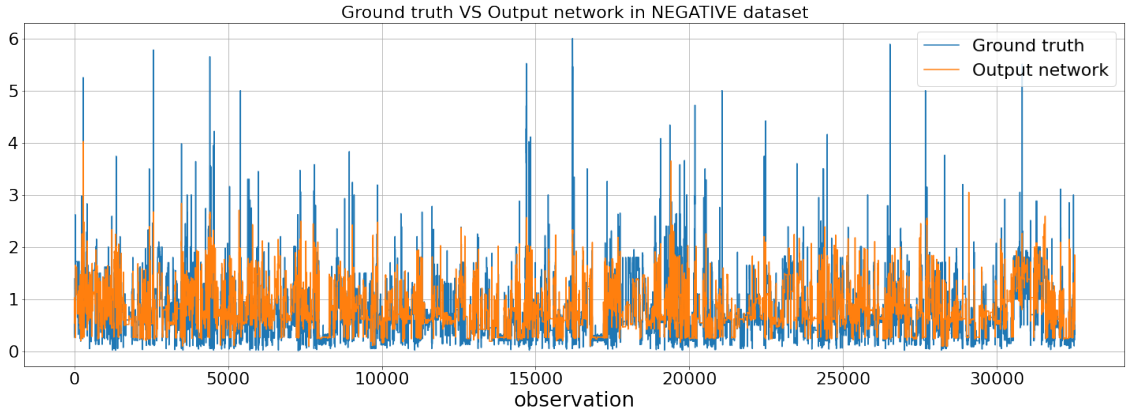


Figure 4.15: Ground truth VS Output Neural Network in *negative test set*

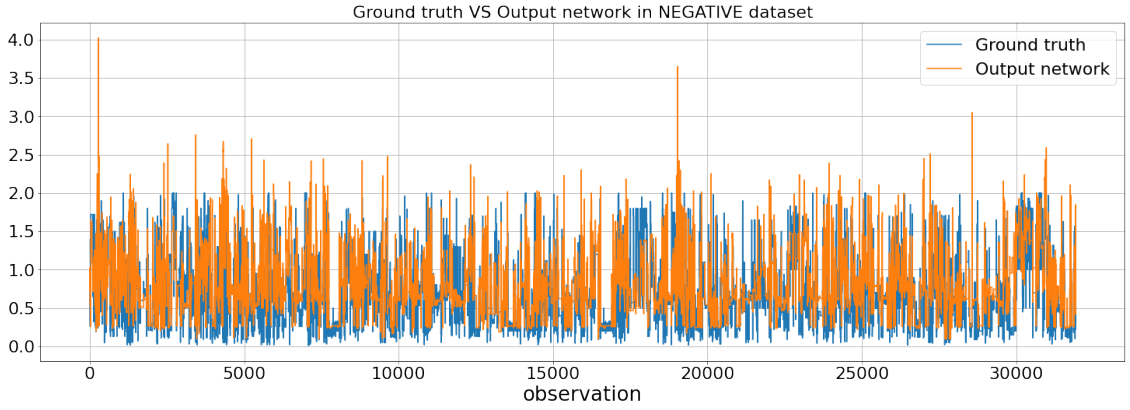


Figure 4.16: Ground truth VS Output Neural Network in *negative test set*

5
Conclusion

The correct estimation of the *optimal initial bid for a new keyword* is crucial for the broker's profitability: if the bid is too low, the advertisement appears in a low rank position, with a low probability to be clicked; if the bid is too high, the advertisement appears in a top position but at a too high cost, generating a lower or even negative profit.

The problem has been tackled *empirically*, using the most recent data available at the time of writing, covering 4 months (January-April 2021) of daily observations involving more than 10 thousands keywords.

The most important features have been presented and the actual behavior of the bids' time-series checked against the expected one. The analysis has been possible thanks to the use of algorithms, like "Window-Sliding" and "Binary Segmentation", able to detect *change points* (CP) in a timeseries. It turned out to be meaningless the distinction of different phases, since does not exist any *long run* bid. The primary driver of the bid's changes is its profitability, which can vary abruptly depending on market conditions.

Hence, the observations with a bid value able to generate traffic satisfying a minimum profitability threshold have been selected, discarding the others.

Following the idea to make the keywords' names an input of the model, each keyword has been encoded in a numerical vector using the pre-trained BERT model. It comes in different sizes and flavors; moreover, different layers and pooling strategy can be chosen for the encoding. A series of empirical tests have been carried out to find the best encoding for the keywords' names in the dataset.

The dataset has been completed joining for each keyword a set of features known *ex ante*, coming from MIMIR.

In this framework, the estimation of the *optimal initial bid* has been cast as a regression problem. It was first solved using simple models: multiple linear regression, Ridge regression and Lasso regression. They are fast to train and have zero or one hyperparameter to fine tune. Their performances have been used as a benchmark to evaluate the effectiveness of a more complex model: Fully Connected Neural Network (FCNN).

The training procedure has been organized as follows. First, the dataset has been split in training, validation and test set at the *keyword level*. In other words, the split has not involved single observations, but single keywords. This was needed to avoid the spread of observations belonging to the same keyword over the training, validation and test set, which would have artificially shrunk the validation and test error. As useful indication about the maximum learning capacity achievable, an estimate of the *training error lower bound* has been computed. The best value of each hyperparameter has been set sequentially: first the optimal values for the most important hyperparameters have been found; then, keeping the latter fixed, the optimal values for the hyperparameters of secondary importance have been found. A series of "experiments" have been applied on the set of features in order to understand their relative importance in the prediction of the output. The entire procedure has been repeated using different loss functions (MSE, MAE, MLE). The MAE turned out to perform best. The robustness of the best model to different splits has been checked using a 5 folds CV. Finally, it has been retrained over the full training set and tested in the test set.

The MAE in the test set was equal to $0.15\$$. Results justify the use of a FCNN: the error drops significantly: $\approx 40\%$ lower than multiple linear regression and $\approx 20\%$ lower than Ridge and Lasso regression.

Results have been counter checked against the *negative test set*: a dataset containing observations belonging to the same keywords used in the test set, but with negative profitability. The idea is that the network should forecast a bid estimate learnt on the observations with positive profitability, and so significantly different from the one observed in the *negative test set*, leading to a higher average error. This counter check highlighted a problem: the average error was just slightly higher ($0.17\$$) in the *negative test set*. The explanation has been found noticing that the same pair {keyword name; bid} is associated to positive and negative profit for the majority of keywords. This is a structural characteristic of the dataset: the bid is adjusted mainly as a reaction to a negative profit; first the bid is profitable and so not adjusted, then it becomes non-profitable and so an adjustment is needed. The process implies that a similar bid's value

is associated with positive and negative profits. This aspect was noticed during the analysis of the timeseries' behavior, but its actual impact not.

This evidence made not reliable to judge the profitability at the *observation level*. So it has been assessed at the *keyword level*: only the observations belonging to *positive* keywords have been used. In this way, the observations selected have been changed, requiring to repeat the entire training procedure.

The MAE in the test set was $0.16\$$. The *negative test set* has been used to counter check results: its MAE was equal to $0.30\$$. This result confirmed the significant difference between the bids' estimates proposed by the model and the bids observed for negative keywords.

Finally, an alternative encoding technique has been tried: GloVe. The *training error lower bound* proved the impossibility to reduce the error below a certain threshold. Anyway was possible to try preserving a similar error with a simpler model. In order to do so, GloVe with 50 dimensions was used. The number of features shrank by almost 20 times, but the test error increased by just 1 cent to $0.17\$$. For this evidence, the best model found, considering complexity and performances, was the FCNN with GloVe encoding.

Some important final remarks:

- for each keyword, the bid estimated by the model ($\hat{y}$) is an average of the various bids observed given the same set of input features. This implies that the $\hat{y}$ is optimal in the sense that it's a good starting value for that keyword, and not in the sense that it is for sure able to generate a positive profit immediately. In fact, the market conditions can require to increase or decrease such initial bid. But the key point is that such initial bid will be enough close to a profitable one, allowing a quick adjustment that can take some days rather than some weeks.

- The limited size of adjustments in the bids' values foster the effectiveness of the CPC-Optimizer. The synergy between the model's capacity to estimate good initial bid and the CPC-Optimizer to adjust it on the right direction and right amount, should increase the overall average broker's profits.

- The choice to select the model with the best trade-off between performance and complexity it's a matter of efficiency. In fact, the model, in the business context in which is going to be applied, has to be updated at a regular frequency, to incorporate the last available data of existing and new keywords.

- It's reasonable to state that some keywords present a seasonality component (e.g. keyword *Christmas gift*, *swimsuit*, *ski boots*, etc.). It would have been interesting to identify such keywords and then estimate their optimal initial bid taking into account the season. But the data available does not allow to do it due to the limited time interval available.

- The use of a *Fully Connected Neural Network* architecture is the most natural choice given the dataset's structure (each row is an observation, each column a feature) and the type of problem (regression). Anyway a future development could consist in solving the same problem with a *Convolutional Neural Network* (CNN). This architecture has been used mainly in tasks such as image recognition, image classification and computer vision [22, 23]. Considering that an image is represented as a matrix of data with each numerical entry corresponding to the color's intensity in a given pixel, almost any $2D$ data structure is potentially suited to become an input of a CNN. This observation has extended the use of CNN to panel data of almost any type, like data from inertial motion sensors, showing state-of-the-art performances in human activity recognition classification tasks [24, 25, 26]. CNN has also been applied for regression problems [27]. In this thesis, the dataset could be segmented in portion with a fixed number of rows, and each portion could become a sample to feed the CNN. Anyway a criticity that hurdles a direct application of this idea is that the dataset is actually not a *panel* data. In fact, it is not a continuous set of features observed through time, but an aggregation of discontinuous timeseries (one for each keyword). From the way in which the dataset has been assembled, all *judgeable* observations (Section 4.1) belonging to the same keywords occupy contiguous rows. The point is that all judgeable observations are usually a small subset of all available ones, creating temporal gaps between them. Hence, the idea to use a CNN with a $2D$ kernel size in order to capture the correlations through time, is not guaranteed to work at all. It could even be possible to use a $1D$ kernel size (to capture correlation through features in a point in time) and apply a *Recurrent Neural Network* (RNN) in next layers to capture correlation through time. The effectiveness of these architectures should be empirically tested in order to claim something about their utility.

Part II

Profitable New Keywords Suggestion

6

Web Scraping and Keywords Extraction

The chapter addresses the *new keywords' suggestion problem* in SSA, presenting a system for multiword keyword recommendations. The problem has been already tackled by the *New Keyword Generation Module* developed by ACTOR in the project perimeter. Here a complementary procedure is used, designed to work in synergy with the existing module, improving the quantity, quality and variety of proposed keywords. The chapter explains the process followed to retrieve information from webpages, to clean it and to extract keywords with and without the help of a *BERT-filter*.
In detail, it is composed by two sections:

- Web Scraping: for each keyword, raw text is retrieved from top URLs returned by a Google query.
- Keywords Extraction: among the rich corpora of *keywords' extraction algorithms*, two of them (RAKE and YAKE!) are selected, according to their suitability to the task and to the overall business framework. Both are presented and explained.

6.1 Web Scraping

At the time of writing, the suggestion of new keywords relies on the *New Keyword Generation Module*. It extracts keywords directly from the landing pages associated to each ad. From

the pages, the most significant tokens are extracted and ranked on the basis of the occurrences of each token in each type of HTML tag, which have different weights according to their importance. Single tokens are combined to form couples and triples of words, whose weighted co-occurrence score is greater than a fixed threshold.

The great advantage of this approach is that the generated keywords are tailored on the ad landing page, ensuring the highest possible matching between the page's contents and the keywords. Moreover, if the page changes, new keywords can be extracted again in order to be in line with the new page's contents.

The potential drawback is that a low quality in page's contents is directly reflected in a low quality of the suggested keywords.

The motivations behind the development of a new tool for keywords' suggestion are:

1. reduce the reliability on the ad landing page;
2. increase the quantity, quality and variety of proposed keywords.

Both points try to suggest non-obvious yet relevant keywords, which are economically more viable. Bidding on many non-obvious low traffic keywords, the combined traffic from them can add up to the level produced by a popular keyword, at a fraction of the cost. Moreover, the traffic received is targeted better and will typically result in a better convertion rate. It is important to find out new alternative keywords, relevant to the base query, but non-obvious in nature, so that little competition is faced from other advertisers. The challenge lies in not only finding relevant keywords, but also in finding many of them.

The tool is designed to work on top of the *New Keyword Generation Module*. The keywords extracted by the latter on a given ad landing page (referred as *query/seed keywords*) become the input to feed the former. Each given seed keyword is entered as a query into Google search engine and the top n URLs (`n_urls`) are considered. Among them, could be, or could not be, present the ad landing page' URL.

In turn, all URLs retrieved for a query keyword are parsed. The corresponding web pages are scraped using HTML tags. Here comes an important choice: which HTML tags to take among all possible ones. A crucial observation is that the web scraping has to be effective on an undefinetely large pletora of websites. This implies the impossibility to retrieve the exact information of interest, since it would require to know a priori the website's structure (HTML tags and classes' names) where the information is stored. For this reason, the web scraping can be carried on only in *general* terms, just taking all text belonging to a selected HTML tag. This

	Scrap level		
	Super-Light	**Light**	**Full**
HTML tags	`<title>`	`<title>`	`<title>`
	`<h1>`	`<h1>`	`<h1>`
		`<h2>`	`<h2>`
		`<ul>`	`<ul>`
		`<ol>`	`<ol>`
			`<p>`

Table 6.1: Web scraping's depths.

could lead to retrieve information meaningful in the context of the webpage (and so present on it), but not related semantically with the query keyword.

The obvious idea is to use the most common HTML tags. In particular, it's of interest to investigate the effect of different *text richness* on the suggested keywords. In fact, depending on the HTML tags used, it's possible to retrieve a tiner or broader amount of text. For example, using `<title>` and `<h1>` tags, the text extracted tends to be short, general and close to the seed keyword. Using `<p>`, `<ul>` and `<ol>` tags, the text tends to be long, specific and not always related to the seed keyword. With these considerations in mind, three different *depths* of web scraping are used: *super-light*, *light* and *full*. They are reported in Table 6.1.

An example of raw text retrieved with *super-light* web scraping with query keyword "*houses for sale Miami*" is reported here:

Raw text retrived by super-light web scraping.

```
Pardon Our Interruption | Pardon Our Interruption... | Please contact Customer Service at (800)
 878-4166\n\x03or unblockrequest@realtor.com with any issues. Please include the Reference ID
shown above. |  |  | Miami, FL Real Estate & Homes For Sale | Trulia | Miami, FL Homes For Sale
 & Real Estate | Miami, FL Luxury Real Estate - Homes for Sale | 190 homes for sale in Miami |
 |  | Miami, FL Real Estate & Homes for Sale | RE/MAX | Miami, FL Real Estate and Homes for
Sale | Miami Real Estate | Find Houses & Homes for Sale in Miami, FL | Miami Homes for Sale |
Miami, FL Homes For Sale | Real Estate by Homes.com | 19,374 Homes For Sale in Miami, FL |
ERROR: The request could not be satisfied | 403 ERROR | Savills | Property for sale in Miami,
Florida, United States of America | 687 Properties for sale in Miami | Access to this page has
been denied. | Please verify you are a human
```

For each URL associated to a query keyword, the parts of text retrieved from different HTML tags are joined by "|" symbol to form a unique new broader text. Then, the text extracted from each URL is joined with the texts extracted from the other URLs of the same query keyword, again by the "|" symbol. In this specific example, each part of text can be a `<title>` or a `<h1>`

tag. Note that not all websites allow web scraping, hence the number of URLs returned for each keyword has to be enough high. Here, and in all future results, it's set `n_urls` $= 10$. This value covers the standard number of results returned by Google in the first page. Considering that users seldom go beyond the first four/five results and that it's unprobable that all websites do not allow web scraping, this choice seems reasonable.

The next step consists in cleaning the raw text. Whatever *keyword extraction* algorithm will be used, the punctuation is crucial: it allows to split the text in sentences and sub-sentences (*chunks*). The "`|`" delimits different content displayed on the website, which could terminate with or without a punctuation mark. Moreover, other special characters could be present (`\n`, `\@`, `+`, etc.). For these reasons:

- the "`|`" symbol is replaced by "`.`" symbol. In other words, each HTML tag content becomes a sentence in the cleaned text;
- the `\n` is replaced by an empty space `' '`;
- only alpha-numerical, punctuation and empty space characters (`A-Za-z0-9,.!?:;' '`) are kept.

Continuing with the previous example, the cleaned text looks like:

Cleaned text.

```
Pardon Our Interruption. Pardon Our Interruption. Please contact Customer Service at 800
8784166 or unblockrequestrealtor.com with any issues. Please include the Reference ID shown
above.. Miami, FL Real Estate Homes For Sale. Trulia. Miami, FL Homes For Sale Real Estate.
Miami, FL Luxury Real Estate Homes for Sale. 190 homes for sale in Miami. Miami, FL Real Estate
 Homes for Sale. REMAX. Miami, FL Real Estate and Homes for Sale. Miami Real Estate. Find
Houses Homes for Sale in Miami, FL. Miami Homes for Sale. Miami, FL Homes For Sale. Real Estate
 by Homes.com. 19,374 Homes For Sale in Miami, FL. ERROR: The request could not be satisfied.
403 ERROR. Savills. Property for sale in Miami, Florida, United States of America. 687
Properties for sale in Miami. Access to this page has been denied. Please verify you are a
human
```

The first lines are garbage text, just the information that web scraping is not allowed. But from the third line, some interesting alternative to the seed keyword can be seen at first glance ("FL Luxury Real Estate Homes", "Miami Real Estate", "Property for sale in Miami", etc.).

6.2 Keywords Extraction

Once the clean text coming from the desired HTML tags is available, automatic keywords extraction methods are needed. They are frequently used to capture the fundamental idea of a document by a list of most relevant *terms*, able to provide the user with a kind of "summary" of the document.

Keywords extraction are of interest to people who need to become acquainted with a given topic. For instance, a journalist looking for information on the U.S.A. president election would find useful a summary consisting of keywords automatically created to explore existing information. It would also be beneficial to researchers who wish to have a quick glimpse of a given article or, as here, for advertisers looking for the most relevant keywords of a web page.

A broad set of keywords extraction algorithms have been developed by the research community. An overview of the most important ones, classified by type, methodology and name is presented in [28] and schematically depicted in Table 6.2.

Supervised models, like *Keyphrase Extraction Algorithm* (KEA), make use of discriminant features and of machine learning algorithms to learn the distinction between relevant and non-relevant keywords. The main hurdle to their concrete use is the need of a rather long training process and the demand of large manually annotated collections of documents in order to capture the nature of a language. For example KEA requires, for each document in the training corpus, the list of manually assigned keywords in order to learn which features'values allow to minimize the training error [29]. Then, optimal value of model's hyperparameters should be chosen according to the validation error, or even better, according to a Cross-Validation procedure. And the entire process crucially depends on the training corpus used, so it's language and domain specific [30].

In order to overcome the need of human labeled training sets, unsupervised models have been developed over the last few years. They can be broadly classified in two methodologies: *Graph-based* and *Statistical*.

Graph-based methods represent the text using a graph. They rely on *Part Of Speech* (POS) tagging system, which consists in assigning a tag (noun, verb, adjective, preposition, adverb, conjunction) to each word in a document. The selection of the employed tag depends on the language and specific domain of application. Given a sequence of words, the output is a list of pairs (word, tag), where there may exist more tags for a given word and the POS tagger task is to solve these ambiguities by selecting the most appropriate tag given the word context.

Type	Methodology	Name	Language Dependence			External Corpus
			Stopword List	POS	Stemming	
Supervised	Binary	KEA			✓	✓
Unsupervised	Graph-based	TextRank		✓		
		SingleRank	✓	✓		
		TopicRank	✓	✓		
		TopicalPageRank	✓	✓	✓	✓
		PositionRank	✓	✓		
		MultipartiteRank	✓	✓	✓	
		ExpandRank	✓	✓		✓
	Statistical	TF.IDF	✓			✓
		KP-Miner	✓			
		RAKE	✓			
		YAKE!	✓			

Table 6.2: Overview main keywords extraction algorithms.

The level of granularity can go well beyond using a richer POS tagging system. For example `45 Penn Treebank` and `87 Brown corpus` use, respectively 45 and 87 different POS tags. In any case, the POS tag provides crucial information to determine the role of the word itself and of the words close to it in the sentence. For example, if the word is a personal pronoun (i.e. I, you, he/she, etc.), it has a higher probability to be followed by a verb; instead, if it is a possessive pronoun (i.e. my, your, his/her, etc.), it has a higher probability to be followed by a noun.

Some graph-based models rely on a *Stemming* procedure, which reduces inflected or derived words to their word stem, base or root form. Others rely on *External Corpus*, like dictionaries, providing useful relations among words, as synonyms (alternative words with same meaning); antinomyns (words with opposite meaning); hypernyms (words with a more general meaning that include as a specific case the target word) and hyponyms (words with a more specific meaning that are included as a specific case of the target word). These resources enpower the capacity to "understand" the text, but depend clearly on the language and the domain.

TextRank model [31] builds an undirected graph where nodes are terms filtered using POS tags, and edges (connections) are set between nodes that co-occur within a window of n terms. A ranking algorithm is then run on top of this graph, and keywords are sorted by decreasing order. *SingleRank* and *ExpandRank* [32] are variations of *TextRank*, with the latter enriched by terms of the k-nearest neighboring documents.

TopicRank model is a generalization of TextRank, where nodes instead of terms, are topic clusters of single and composed expressions.

An exhaustive overview of the various graph-based methods is out of the scope of the current exposition, so the interested reader can consult [28] and the references herein. The take-home-message is that all graph-based methods rely on a POS tagging system, which is language and domain dependent. This implies the impossibility to apply these models directly to new languages or new domains.

In the SSA business framework, a desirable feature for a keyword extraction model is the possibility to be easily extended to new languages and domains. This capacity would help the expansion in new country markets, regardless the existence or not, of pre-built POS tags systems, stemming and external corpus. For example, an advertising company active in the US market, could apply the keyword extraction tool even in non-English speaking countries, like Spain, Italy, France, Germany and many others.

This crucial feature is owned by the *Statistical* models, which are able to extract keywords using uniquely the input text and the *Stopword List* (except for *TF.IDF* which requires a col-

lection of documents). Note that a complete language independence is not possible using the models currently available, since the stopword list is a requiered input of each of them, and it is clearly language dependent. Anyway building a stopword list is much easier than building a POS tags system or a stemming. The latters requiere to consider ideally all the words existing in the language's vocabulary. The former requires to pick a small subset of frequently used words which do not convey any particular meaning.

Among the *Statistical* models, *TF.IDF* [33] is not considered since it requires access to a large corpus, which is not available in this application framework. *KP-Miner* [34] is a feasible choice, but is anyway excluded in favor of two models which are gaining popularity for their simplicity and effectiveness: *RAKE* and *YAKE!*. Both are going to be presented in the next two subsections.

6.2.1 RAKE

RAKE stands for *Rapid Automatic Keyword Extraction* [35]: an unsupervised model for extracting keywords from individual documents. It does not requiere labeled data for training (as supervised models do), and it is almost independent by external-corpora, language or domain (as unsupervised graph-based models do). In fact, the only language dependency is a list of stopwords. Anyway building a stopword list is a relatively easy task, since it does not involve all the words in the vocabulary but a small portion of them.

RAKE is based on the observation that keywords frequently contain multiple words but rarely contain standard punctuation or stop words, such as the function words *and*, *the*, and *of*, or other words with minimal lexical meaning. This reasoning is based on the expectation that such words are too frequently and broadly used to aid users in their analyses or search tasks.

Words that do carry meaning within a document are described as content bearing and are often referred to as *content words*.

The input parameters for RAKE comprise a list of stop words, a set of phrase delimiters, and a set of word delimiters. RAKE uses stop words and phrase delimiters to partition the document text into candidate keywords, which are sequences of content words as they occur in the text.

Co-occurrences of words within these candidate keywords are meaningful and allow to identify word co-occurrence without the application of an arbitrarily sized sliding window. Word associations are thus measured in a manner that automatically adapts to the style and content of the text, enabling adaptive and fine-grained measurement of word co-occurrences that will

be used to score candidate keywords.

It works in three steps [36], described in the following.

Step 1: Text Pre-Processing

Convert all text to lower case (i.e.: "Google" → "google" or "GOOGLE" → "google").

Then split it into an array of words (tokens) by the specified word delimiters (space, comma, dot etc.).
For example, the input text:

Input text.

```
Google quietly rolled out a new way for android users to listen to podcasts and subscribe to
shows they like, and it already works on your phone. Podcast production company pacific content
 got the exclusive on it. This text is taken from Google news.
```

becomes:

Partitioned text.

```
['google','quietly,'rolled,'out,'a','new,'way','for','android','users','to','listen','to','
podcasts','and','subscribe','to','shows','they','like',',','and','it','already','works','on','
your','phone','.','podcast','production','company','pacific','content','got','the','exclusive
','on','it','.','this','text','is','taken','from','google','news','.']
```

Step 2: Candidate Keywords Generation

The array is then split into sequences of contiguous words by phrase delimiters and stop word positions. Words within a sequence are assigned the same position in the text and together are considered a candidate keyword, as in Table 6.3, where stopwords and punctuation are reported in red.

Table 6.3: Sentence splitting using RAKE.

Split by delimiters	Split by stop word	Candidate Keyword
google	google	[google, quietly, rolled]
quietly	quietly	
rolled	rolled	

Table 6.3: Sentence splitting using RAKE *(Continued from previous page)*

Split by delimiters	Split by stop word	Candidate Keyword
out		
a		
new		
way		
for		
android	android	[android, users]
users	users	
to		
listen	listen	[listen]
to		
podcasts	podcasts	[podcasts]
and		
subscribe	subscribe	[subscribe]
to		
shows	shows	[shows]
they		
like		
,		
and		
it		
already		
works	works	[works]
on		
your		
phone	phone	[phone]
.		
podcast	podcast	[podcast, production, company, pacific, content]
production	production	
company	company	
pacific	pacific	

Table 6.3: Sentence splitting using RAKE *(Continued from previous page)*

Split by delimiters	Split by stop word	Candidate Keyword
content	content	
got		
the		
exclusive	exclusive	[exclusive]
on		
it		
.		
this		
text	text	[text]
is		
taken		
from		
google	google	[google, news]
news	news	
.		

Step 3: Candidate Keywords' Score

A score is computed for each *candidate keyword* in the following way. First, the co-occurrence graph (a term document matrix with one extra count of each word coming in a phrase) is computed. Continuing with the previous example, the results is shown in Table 6.4.

The keywords extracted in *step 2* are split in single words and each unique word becomes an entry of a row and column of the co-occurrences graph (i.e. "google" appears two times in the keywords, but a unique row and column is reserved to it). On the main diagonal is shown the number of times each word appears in the *candidate keywords*. Out of the main diagonal is reported (when different from 0) the number of times two words appear in the same keyword.

Two metrics are needed for each word: *word frequency* ($freq_w$) and *word degree* (deg_w). The $freq_w$ counts how many times a particular word w appears among all candidate keywords. The deg_w counts how many times w appears among all candidate keywords ($freq_w$) plus how many words appear in the same *candidate keyword/s* containing w. $freq_w$ favors words that occur frequently regardless of the number of words with which they co-occur. deg_w favors words that occur often and in longer candidate keywords.

	google	quietly	rolled	android	users	listen	podcast	subscribe	shows	works	phone	podcast	production	company	pacific	content	exclusive	text	news
google	2	1	1																
quietly	1	1	1																
rolled	1	1	1																
android				1	1														
users				1	1														
listen						1													
podcast							1												
subscribe								1											
shows									1										
works										1									
phone											1								
podcast												1	1	1	1	1			
production												1	1	1	1	1			
company												1	1	1	1	1			
pacific												1	1	1	1	1			
content												1	1	1	1	1			
exclusive																	1		
text																		1	
news																			1

Table 6.4: RAKE co-occurence graph.

	google	**quietly**	**rolled**
deg_w	5	3	3
$freq_w$	2	1	1
$score_w = \frac{deg_w}{freq_w}$	2.5	3	3

Table 6.5: Degree, Frequency and Score metrics for the first keyword.

The *word score* is calculated as the ratio:

$$score_w = \frac{deg_w}{freq_w}$$

The idea is to penalize the deg_w as $freq_w$ increases: in fact it's easier to have a higher degree for a word appearing with high frequency. In other words, $score_w$ favors words that predominantly occur in longer *candidate keywords*. This is the best metric for keyword suggestion in SSA, because longer keywords are usually:

- more specific, leading to a more targeted traffic with an higher conversion rate;
- less common, that is, are cheaper, since are exposed to less competition among advertisers.

Both observations lead to higher profit for the advertiser.

The *word score* is computed for each word in a *candidate keyword* (KW) and the individual scores are summed to provide the *candidate keyword score* ($score_{KW}$):

$$score_{KW} = \sum_{w \in KW} score_w.$$

As an example, the Degree, Frequency and Score metrics are reported for the *candidate keyword* "google quietly rolled" in Table 6.5.
The resulting score for the first *candidate keyword* ("google quietly rolled") is:

$$score_{KW} = \frac{5}{2} + \frac{3}{1} + \frac{3}{1} = 8.5$$

The score for all the *candidate keywords* in Table 6.3 is shown in Table 6.6.

Continuing the example started in Section 6.1, from the cleaned text:

keyword	score
podcast production company pacific content	25.0
google quietly rolled	8.5
google news	4.5
android users	4.0
exclusive	1.0
works	1.0
phone	1.0
text	1.0
podcasts	1.0
subscribe	1.0
listen	1.0
shows	1.0

Table 6.6: Score for all keywords.

Cleaned text.

```
Pardon Our Interruption. Pardon Our Interruption. Please contact Customer Service at 800
8784166 or unblockrequestrealtor.com with any issues. Please include the Reference ID shown
above.. Miami, FL Real Estate Homes For Sale. Trulia. Miami, FL Homes For Sale Real Estate.
Miami, FL Luxury Real Estate Homes for Sale. 190 homes for sale in Miami. Miami, FL Real Estate
 Homes for Sale. REMAX. Miami, FL Real Estate and Homes for Sale. Miami Real Estate. Find
Houses Homes for Sale in Miami, FL. Miami Homes for Sale. Miami, FL Homes For Sale. Real Estate
 by Homes.com. 19,374 Homes For Sale in Miami, FL. ERROR: The request could not be satisfied.
403 ERROR. Savills. Property for sale in Miami, Florida, United States of America. 687
Properties for sale in Miami. Access to this page has been denied. Please verify you are a
human
```

the most relevant keywords are extracted by RAKE using the Python package `python-rake` [37]. The resulting keywords are reported in Table 6.7, sorted by decreasing importance.

Final remarks:

1. RAKE does not have hyperparameters to be empirically set up;
2. sometimes RAKE results could be not accurate if relevant keywords contains stopwords and appear once in the document. For example, "new" is listed in RAKE's stopword list. This means that neither "New York" nor "New Zealand" can be ever a keyword. But if RAKE finds pairs of keywords that adjoin one another at least twice in the same document and in the same order, then a new candidate keyword is created as a combination of those keywords and their interior stop words. Because adjoining keywords must occur twice in the same order within the document, their extraction is more common on texts that are longer than short abstracts.

Ranking	Keyword Extracted
1	fl luxury real estate homes
2	fl real estate homes
3	fl real estate
4	contact customer service
5	reference id shown
6	find houses homes
7	miami real estate
8	sale real estate
9	real estate
10	fl homes

Table 6.7: Keywords extracted by RAKE.

6.2.2 YAKE!

Yet Another Keyword Extractor! (YAKE!) [38, 28] is an unsupervised algorithm able to select the most important keywords using the text statistical features extracted from single documents. It does not need to be trained on a particular set of documents and it does not depend on labeled data for training (as supervised models do), external-corpora, language or domain (as unsupervised graph-based models do).

YAKE! is composed by five main steps:

1. text-pre-processing,
2. features extraction,
3. individual term score,
4. candidate keywords generation,
5. data deduplication and ranking.

Each of them is going to be explained in the following.

Step 1: Text Pre-Processing

Given a text, the algorithm divides it into sentences using the *segtok* rule-based sentence segmenter [39], able to splits Indo-European texts into sentences following a given predefined pattern. It does not blindly rely on punctuation, allowing a higher quality split (i.e. "Mr. Smith" would be considered a single sentence).

Each sentence is then divided in chunks (whenever punctuation is found) and split into words. Each word is then converted into its lowercase form.

The result of this pre-processing stage is a list of sentences, where each sentence is divided into chunks formed by words.

Step 2: Features Extraction

A set of five features able to capture the characteristics of each individual term is defined. They are: (1) *Casing*; (2) *Word Position*; (3) *Word Frequency*; (4) *Word Relatedness to Context*; (5) *Word Different Sentence*.

Particular attention has to be made about features' interpretation: *Word Position* and *Word Relatedness to Context* are designed, as will become clear in the remaining, to assign a lower value to *important* terms;

Casing is designed to assign a higher value to *important* terms;

Word Frequency and *Word Different Sentence* have not an obvious interpretation in terms of word w importance, but have to be assessed in relation to *Word Relatedness to Context*.

Casing (W_{Case})

Word starting with a capital letter (excluding ones at the beginning of sentences) or acronyms (all letters of the word are capital) are considered more important, since this is what often happens in any text. The weight is computed as:

$$W_{case} = \frac{\max\{TF(U(w)), TF(A(w))\}}{\log_2(TF(w))}$$

where $TF(U(w))$ is the number of times the candidate word w starts with an uppercase letter; $TF(A(w))$ is the number of times the candidate word w is marked as an acronym and $TF(w)$ is the frequency of w. Thus, the more often the candidate term occurs with a capital letter, the more important it is considered.

Word Position ($W_{Position}$)

The underlying hypothesis is that early parts of the text tend to contain a high rate of relevant keywords. The formula is:

$$W_{Position} = \log_2(log_2(2 + Median(Sen_w)))$$

where Sen_w indicates the positions in the text of the set of sentences where the word w occurs and $Median$ is the median function. The result is an increasing function, where values tend to increase smoothly as words are positioned at the end of the document, meaning that the more often a word occurs at the beginning of a document the less its $W_{Position}$ value. Instead, words positioned more often at the end of the document (likely less relevant) will be given a higher $W_{Positional}$ value. Note that a value of 2, is considered in the equation to guarantee that $W_{Position} > 0$.

Word Frequency (W_{Freq})
It indicates the frequency of the word w within the document. It reflects the belief that higher the frequency, more important the word is. To prevent a bias towards high-frequency in long documents the TF value of a word w is divided by the mean of the frequencies ($MeanTF$) plus one time their standard deviation (σ). The goal is to score all those words that are above the mean of the terms balanced by the degree of dispersion given by the standard deviation:

$$W_{Freq} = \frac{TF(w)}{MeanTF + 1 \cdot \sigma}.$$

This feature itself has not an obvious interpretation in terms of word w importance: a particular word of no importance (name, adjective, verb, preposition, etc.) could appear just one time, having a very low W_{Freq}. A stopword could appear in almost every sentence, having a high W_{Freq}. So unimportant words could have low or high values of W_{Freq}. The same reasoning applies to important word in analogous way. This feature has to be interpreted in relation with W_{Rel}.

Word Relatedness To Context (W_{Rel})
It quantifies the extent to which a word resembles the characteristics of a stopword. To compute this measure, the number of different terms that occur in a window of size m to the left (and right) side of the candidate word are considered (m is a hyperparameter to be set experimentally). Higher the number of different terms that co-occur with the candidate word (on both sides), the more meaningless the candidate word is likely to be. W_{Rel} is defined as:

$$W_{Rel} = 1 + (DL + DR) \cdot \frac{TF(w)}{MaxTF} + PL + PR$$

where:

$\mathbf{m=2}$	$\mathbf{m=1}$	$\mathbf{w}$	$\mathbf{m=1}$	$\mathbf{m=2}$
of	the	relations	that	connect
	semantic	relations		
that	this	relations	supervene	solely

Table 6.8: Word relatedness to context table.

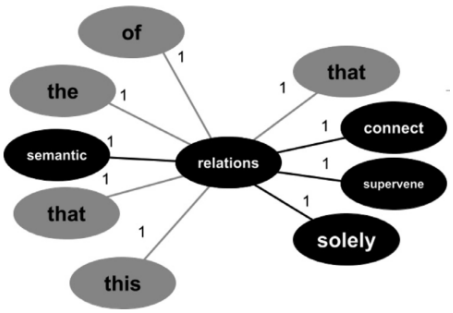


Figure 6.1: Word relatedness to context plot.

- $DL[DR] = \frac{|A_{w,t}|}{\sum_{k \in A_{w,t}} CoOccur_{w,k}}$ with $|A_{w,t}|$ the number of different terms that occurs on the left [right] side of a candidate word w within a given predefined window, with regard to the number of k terms that it co-occurs with.
- $\frac{TF(w)}{MaxTF}$ is the term frequency of the candidate word divided by the maximum term frequency ($MaxTF$) among all candidate words that occur in the document, in order to penalize candidate words that occur with high frequency (as stopwords do).
- $PL[PR] = \frac{|A_{w,t}|}{MaxTF}$ is the ratio between the number of different terms that co-occur with the candidate word (on the left [right] hand side) and the $MaxTF$.

The more insignificant the candidate word is, the higher the score of this feature will be. Thus, stopwords-like terms will easily obtain higher scores.

An example is needed to clarify how the various formula's components are computed. In Table 6.8 are shown the different terms that co-occur with the candidate term "relations" within a window of size $m = 2$. In Figure 6.1, the number on the edges is the frequency that the candidate term "relations" co-occurs with the term in the corresponding oval. Gray color indicates stopwords.
In detail:

- $DL = \frac{|A_{w,t}|}{\sum_{k \in A_{w,t}} CoOccur_{w,k}} = \frac{5}{5}$, since all 5 terms on Table 6.8 (highlighted in green) are different (so $|A_{w,t}| = 5$) and each of them co-occurs 1 time with "relations" (so $\sum_{k \in A_{w,t}} CoOccur_{w,k} = 5$). Analogous reasoning holds for $DR = 4/4$.
- $\frac{TF(w)}{MaxTF}$ and $PL[PR]$ are easy to understand.

Word Different Sentence ($W_{DifSentence}$)
It represents the proportion of sentences containing the word w, computed as:

$$W_{DifSentence} = \frac{SF(w)}{\#Sentences}$$

where $SF(w)$ is the number of sentences where w appears, and $\#Sentences$ is the total number of sentences in the text. This feature itself has not an obvious interpretation in terms of word w importance: a particular word of no importance (name, adjective, verb, preposition, etc.) could appear in just one sentence, having a low $W_{DifSentence}$. A stopword could appear in almost every sentence, having a $W_{DifSentence}$ close to one. So unimportant words could have low or high values of $W_{DifSentence}$. The same reasoning applies to important word in analogous way. This feature has to be interpreted in relation with W_{Rel}.

Step 3: Individual Term Score

The five features are combined into a single score $S(w)$. The smaller the value $S(w)$, the more important the word w is:

$$S(w) = \frac{W_{Rel} \cdot W_{Pos}}{W_{Case} + \frac{W_{Freq}}{W_{Rel}} + \frac{W_{DifSentence}}{W_{Rel}}}.$$

The W_{Rel} and the W_{Pos} are directly proportional to the word's score, so both features appear at the numerator. W_{Freq} and $W_{DifSentence}$ are offset by W_{Rel} in order to assign a high weight to words that appear frequently and appear in many sentences (likely indicative of their importance) as long as the word is relevant (W_{Rel} is low). Indeed, some words may occur plenty of times and in many sentences and yet be useless (stopwords or similar). Both previous ratios take higher values more important is the word, hence are inversely proportional to the way $S(w)$ is interpreted. For this reason are placed at the denominator. W_{Case} is also inversely proportional to $S(w)$, so it appears in the denominator.

Step 4: Candidate keywords generation

In order to generate the candidate keywords, a sliding window of size n is considered, generating a contiguous sequence of terms ranging from 1-gram to n-gram (where n is a hyperparameter to be set experimentally).

Each candidate keyword will be composed by words belonging simultaneously to:

- same sliding window;
- same sentence;
- same chunk.

This avoids having, for example, a candidate keyword including the end of a sentence and the beginning of the next one. In addition, keywords beginning or ending with a stopword will not be considered. Digits are also not considered, as they are rarely part of a keyword. No conditions are set in respect to the minimum frequency or sentence frequency that a candidate keyword must have. This means that a keyword can be considered as significant/insignificant with either one occurrence or with multiple occurrences.

Each candidate keyword will then be assigned a final $S(kw)$, such that the smaller the score the more meaningful the keyword will be:

$$S(kw) = \frac{\prod_{w \in kw} S(w)}{TF(kw) \cdot (1 + \sum_{w \in kw} S(w))}$$

where $S(kw)$ is the score of a candidate keyword with a maximum size of n terms, determined by multiplying (in the numerator) the score of the first term of the candidate keyword by the subsequent scores of the remaining terms, such that the smaller this multiplication the more meaningful the keyword will be. This is divided by the sum of the $S(w)$ scores in order to average out with respect to the length of the keyword, such that longer candidate keywords are not benefited/harmed just because they have a higher length. The result is further divided by the term frequency of the keyword, $TF(kw)$, to penalize less frequent candidates.

Step 5: Data Deduplication And Ranking

The final step in determining suitable candidate keywords is to eliminate similar candidates. For this, there are three available metrics: *Sequence Matcher*, *Levenshtein* and *Jaro-Winkler* distances. All of them measure the similarity between two strings. Among the strings considered similar (two strings are considered similar if their distance is above a given threshold θ, a hyperparameter to be set experimentally) the one that has the lowest $S(kw)$ score is kept. The final list of keywords and corresponding scores are sorted according to their relevance and returned as output, concluding the algorithm.

Hyperparameters Set Up

Empirical trials on a collection of 20 different datasets show that the following hyperparameters choices works well [28]:

- window-size W_{Rel} $(m) = 1$;
- window-size $kw\ generator$ $(n) = 3$;
- distance metric = Sequence Matcher;
- $\theta = 0.9$

In our data, the *query keywords* available have an average length equal to 3.4. Considering that more specific keywords tend to be composed by more words and are often less used (and so cheaper), it's advisable to choose a higher value for the hyperparameter n. A reasonable choice is a slightly higher value, able to generate longer keywords without exceeding in length. A good compromise is $n = 5$.

Continuing the example started in Section 6.1, from the cleaned text:

Cleaned text.

```
Pardon Our Interruption. Pardon Our Interruption. Please contact Customer Service at 800
8784166 or unblockrequestrealtor.com with any issues. Please include the Reference ID shown
above.. Miami, FL Real Estate Homes For Sale. Trulia. Miami, FL Homes For Sale Real Estate.
Miami, FL Luxury Real Estate Homes for Sale. 190 homes for sale in Miami. Miami, FL Real Estate
 Homes for Sale. REMAX. Miami, FL Real Estate and Homes for Sale. Miami Real Estate. Find
Houses Homes for Sale in Miami, FL. Miami Homes for Sale. Miami, FL Homes For Sale. Real Estate
 by Homes.com. 19,374 Homes For Sale in Miami, FL. ERROR: The request could not be satisfied.
403 ERROR. Savills. Property for sale in Miami, Florida, United States of America. 687
Properties for sale in Miami. Access to this page has been denied. Please verify you are a
human
```

the most relevant keywords are extracted by YAKE! using the Python package `yake`([40] and [41]). The resulting keywords are reported in Table 6.11, sorted by decreasing importance.

6.2.3 BERT Filter

The *BERT Filter* is designed as an additional tool working on top of RAKE and YAKE!. It takes as input the keywords returned by the keyword extractor algorithm for each *query keyword*, and applies the following operations:

1. it encodes the keywords using BERT-Large Uncased (Whole Word Masking), with 24 hidden layers and 1024 hidden units, pooling layer = 2 and pooling strategy = `REDUCE MEAN`. These settings come from the analysis in Subsection 2.3.2, to which the reader is directed for a complete explanation;

2. it computes the cosine similarity between the *query keyword* and each of the suggested keywords;

3. it sorts the suggested keywords in descending order of cosine similarity with the *query keyword*.

In this way, suggested keywords more similar to the query are moved on top positions of the ranking. On one side, this filtering avoids suggesting *garbage* keywords on top positions, like "pardon our interruption" in position 1 of Table 6.11. On the other side, the *non obviousness* of suggestions could be negatively affected, as appears clear comparing the top three suggestions for RAKE and RAKE+BERT in Table 6.9 and 6.10, respectively.

This implies that the use of *BERT filter* should not be done automatically, but depending on the extraction algorithm used and on the scrap level selected. In the introductory example shown here, the text for the query "houses for sale Miami" is retrieved using *super-light* scrap level (`<title>` and `<h1>` HTML tags). This text tends to be close to the query by definition, hence generating keywords with a good *relatedness* with the seed keyword. In this context, the *BERT filter* seems to provide, overall, no particular improvements.

But changing the number of keywords extracted and/or the scrap level, the situation could change. For example, a deeper scrap level is expected to retrieve a broader text with a higher heterogeneity of words. Not all text will be closely related to the query, since a website contains much supplementary information, useful for the user but not always directly related to the seed keyword. In this case, a keyword extraction algorithm will extract also keywords low related to the query, which could be ranked in top positions. Here comes in place the utility of *BERT filter*.

It appears clear from now the need to develop a methodology to evaluate systematically the keywords extracted for all the possible combinations of scrap level, model and number of keywords returned. This task is going to be addressed in the next chapter.

Ranking	Keyword Extracted
1	fl luxury real estate homes
2	fl real estate homes
3	fl real estate
4	contact customer service
5	reference id shown
6	find houses homes
7	miami real estate
8	sale real estate
9	real estate
10	fl homes

Table 6.9: Keywords extracted by RAKE.

Ranking	Keyword Extracted
1	miami homes
2	sale
3	sale real estate
4	miami real estate
5	374 homes
6	find houses homes
7	fl luxury real estate homes
8	fl real estate homes
9	florida
10	fl real estate

Table 6.10: Keywords extracted by RAKE+BERT.

Ranking	Keyword Extracted
1	pardon our interruption
2	homes for sale
3	miami homes for sale
4	real estate
5	homes for sale in miami
6	miami real estate
7	sale
8	miami
9	luxury real estate
10	homes

Table 6.11: Keywords extracted by YAKE!.

Ranking	Keyword Extracted
1	miami homes for sale
2	homes for sale in miami
3	property for sale in miami
4	properties for sale in miami
5	homes for sale
6	sale in miami
7	estate and home for sale
8	property for sale
9	properties for sale
10	miami homes

Table 6.12: Keywords extracted by YAKE!+BERT.

7

Evaluation Methodology and Results Analysis

The chapter tackles the evaluation task introduced in Subsection 6.2.3: it defines the metrics and the procedure to follow in order to get a quantitative evaluation of the results for the different scrap levels, keywords extractors models and number of suggestions returned.

In detail, the chapter is composed by three sections:

- Evaluation Methodology: an overview of the available metrics is presented, highlighting pros and cons of each of them, selecting the most suitable ones for the problem at hand and defining the overall evaluation procedure.

- Results Analysis: results for the selected metrics are shown and commented, for each combination of scrap level, keywords extractors models and number of suggestions returned.

- Illustrative Example: a subset of 5 keywords is used to present some empirical results.

7.1 Evaluation Methodology

7.1.1 Available Metrics

In order to assess the reliability of a model it's necessary to define a methodology to evaluate its results. In the context of keywords extraction, different metrics and procedures are adopted by the research community. It's worth noting that keywords extraction is fundamentally a *ranking* task rather than a *classification* task, where higher ranking's positions are reserved to more relevant keywords.

Traditional classification tasks just predict which class a sample belongs to and therefore, do not consider any form of ranking during evaluation. In this context, the most widely used metrics are *Precision* (a.k.a. *Relatedness*), *Recall*, *F-score*.

Keywords extraction, on the other hand, requires *Rank-Aware* evaluation metrics. An easy modification of the above metrics able to include the ranking positions, consists in computing them only on the top K results. For this reason, the new metrics are called *Precision@K*, *Recall@K* and *F-score@K*. Anyway, within the top K positions, the rank is still irrelevant for the computation of each measure.

More advanced metrics have been proposed in the literature [42]: *Mean Reciprocal Rank* (MRR), *n-Discounted Cumulative Gain* ($nDCG$) and *Mean Average Precision* (MAP). Anyway not all of them are suited for the problem at hand.

For example, MRR only considers the single highest ranked relevant item. If the model returns a relevant keyword in the third highest position, then MRR only cares about it. It doesn't take into account other relevant keywords ranked below position 3. This feature comes from the definition on MRR:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

where Q is a sample of query keywords and $rank_i$ refers to the rank position of the first relevant keyword for the i-th query keyword. Hence, this metric is not suited when multiple keywords are returned.

$nDCG$ relies on a numerical scale of relevance (i.e. $3 =$ highly relevant, $2 =$ medium relevant, $1 =$ low relevant, $0 =$ irrelevant) in order to assign a final relevance score. Given a sorted list

(in descending order of relevance) of K suggested keywords, first the DCG is computed:

$$DCG_{i,K} = \sum_{k=1}^{K} \frac{rel_k}{\log_2(k+1)}$$

where i is the query keyword and rel_k is the graded relevance of suggested keyword at position k. The denominator penalizes the relevance as ranking position moves away from the top. Then, a second quantity is needed:

$$IDCG_{i,K} = \sum_{k=1}^{K} \frac{rel_k}{\log_2(k+1)}$$

where I stands for "Ideal", since the ranking of the keywords is the "ideal" one, produced by a human evaluator. The final relevance score for the list of keywords returned as response to the query keyword i is the ratio:

$$nDCG_{i,K} = \frac{DCG_{i,K}}{IDCG_{i,K}}$$

which by definition takes value in the interval $[0, 1]$. The overall relevance score for a sample of query keywords Q is:

$$nDCG_{Q,K} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} nDCG_{i,K}$$

The quality of this metric depends crucially on two elements:

1. the possibility to assign a numerical scale of relevance to each suggested keyword;
2. the capacity to define an ideal ranking.

Both points tend to increase the difficulty of the evaluation. Considering that often is impossible to automatize the evaluation procedure, the task has to be assigned to human evaluators. The higher difficulty introduces a higher level of subjectivity. Of course *human evaluation* is subjective by its own nature, but finer the grain of judgement required, higher the room for subjectivity. The problem gets worse when the scale increases: higher the number of suggested keywords, richer their variety, which implies a less obvious (and so more subjective) solution to point 1. and 2..

MAP considers the extracted list of suggested keywords as an ordered list. In other words, it evaluates considering the ranks at which relevant keywords are extracted. The goal is to penalize more, irrelevant keywords placed higher in the ranking and gradually less, keywords placed

lower in the ranking. Given a query keyword i, for each *relevant* suggested keyword k in the ordered list, the precision of the list till that relevant keyword is computed. For example, if the first relevant suggested keyword is in second position, the precision is computed considering just the first two positions and would be $\frac{0+1}{2} = 0.5$; if the second relevant suggested keyword is in fifth position, the precision is computed considering the first five positions and would be $\frac{0+1+0+0+1}{5} = 0.4$. Then, the scores obtained for each relevant keyword are averaged. Continuing the example, if there are only two relevant keywords in the whole list, the MAP_i would be $\frac{0.5+0.4}{2} = 0.45$. Finally, the same procedure has to be repeated for each keyword j in the sample of query keywords Q, producing MAP_j. The overall MAP for the whole set of query keywords is:

$$MAP_Q = \frac{1}{|Q|} \sum_{j=1}^{|Q|} MAP_j.$$

This metric works really well when there are only binary ratings (relevant/irrelevant).

Other possible metrics come from research. In [11], Joshi and Motwani use the metrics *Relatedness* and *Non obviousness* and compute an approximation of the *Recall*. Then, the three measures are combined, two at time, in an *F-score*. *Relatedness* and *Non obviousness* are binary variable, taking values $\{0, 1\}$. The former is evaluated by humans, while the latter is evaluated automatically, defining a hard rule to assess originality: if the suggestion does not contain the query keyword, part of it, or its variants sharing a common stem, than it's non-obvious.

In [16], Thomaidou and Vazirgiannis suggest an additional measure aside *Relatedness* and *Non obviousness*: *Specificity*. It quantifies how original are the suggested keywords with respect to the query one. The authors define a graded scale of 5 different levels for each metric and rely exclusively on human evaluators to score the results.

7.1.2 Best Metrics selection

The new keywords suggestions' procedure presented in Chapter 6 is designed to return multiple keywords, hence MRR is not a suited metric.

The query keywords are composed, on average, by 3.4 terms, meaning that are characterized by a higher level of *lexical richness* with respect to short queries. As result, also the suggestions will tend to be longer and having many variants. The effect is a higher difficulty in assigning, each suggested keywords, to the most appropriate level in the graded scale of each metric. So $nDCG$ has also to be excluded from the eligible metrics. Also, methods like the one followed in [16] cannot be easily adopted.

On the other side, metrics and procedures adopting binary values reduce the complexity of evaluation task at a feasible level. Relevant suggested keywords receive a score equal to 1, not relevant equal to 0. The same logic is applied for other metrics. In this way the room for subjectivity is reduced and the evaluation task greatly simplified. The MAP respects this criterium, so it's a potential good candidate metric.

Anyway in the current work, considering:

- the high number of keywords under evaluation,
- the three different scrap levels (*super-light*, *light* and *full*),
- the four different models (*RAKE*, *RAKE+BERT*, *YAKE!*, *YAKE!+BERT*),
- the five different n° suggested keywords ($10, 20, 30, 40, 50$),
- the human evaluation,

the choice is fallen on *Relatedness@K*, *Non obviousness@K* and *F-score@K*, following a similar evaluation procedure as the one described in [11], but with some differences. *Relatedness@K* and *Non obviousness@K* still take binary values, but the *Recall* is not used and the human evaluation is extended also to *Non obviousness@K*.

The first modification is motivated by the objective difficulty in computing the *Recall*. It is the proportion of relevant keywords that are retrieved, out of all relevant keywords available. The problem with determining exact recall is that the total number of relevant keywords is unknown. Joshi and Motwani approximate this quantity as the size of the union of relevant results from all techniques used. Since it is anyway an approximate value and is not the primary metric of interest in the business framework (are relatedness and non-obviousness), a natural choice is not using this measure at all.

The second modification is also motivated by the difficulty in defining an automatic rule able to discriminate in a reliable way between related/non-related and non-obvious/obvious keywords suggestions. The authors'rule works well for keywords composed by a single word, but it's not reliable for longer keyword. For example, retaking the query keyword "houses for sale Miami", the suggestions:

- "fl luxury real estate homes",
- "fl real estate homes",
- "miami real estate",

- "properties for sale in miami",

are all considered *obvious* since contain part of the query or a common stem. But it's reasonable to state that, at the human perspective, the suggestions are *non obvious*. For this reason, the whole evaluation procedure is carried on by human evaluators. The unique objective rules used in order to assess the *relatedness* and *non obviousness* are, respectively:

1. when the suggested keyword is a copy of the query (even with a different order of words), then it is for sure *related* (i.e. "homes for sale in miami", "miami homes for sale", etc.);
2. when the suggested keyword is a subset of the query, then it is for sure *obvious* (i.e. "miami homes", "sale in miami", "homes for sale", etc.).

It's important to notice that:

- *non obviousness* and *relatedness* are independent evaluation metrics. Hence, when a suggestion is *obvious*, then it could be *related* or not (i.e. "miami homes" could be even related, "homes for sale" for sure not, since it's too broad to be used as a valid alternative to the query);
- for all the suggestions to which the objective rules are not applicable, the evaluation relies entirely on human interpretation. This implies a certain degree of subjectivity, which anyway could be offset using different human evaluators.

7.2 Results Analysis

A list of 3306 unique query keywords is obtained from the same dataset used to train, validate and test the model in Part I. The evaluation methodology described in Section 7.1 is not applicable to the entire list, since it would take too much time to be completed. A feasible alternative consists in selecting a well representative sample of keywords from the whole list. In order to be sure about its representativity, a set of 100 keywords is manually selected from different thematic areas, promoting several products and services. The main categories are:

- health care services,
- internet providers,
- job search,

- gps devices,
- lawyers,
- postal services,
- phones and mobile services,
- education,
- vacation packages.

The choice to take 100 keywords is based on two motivations:

1. the number is high enough to represent well the keywords,
2. it is still enough low to allow a manual evaluation.

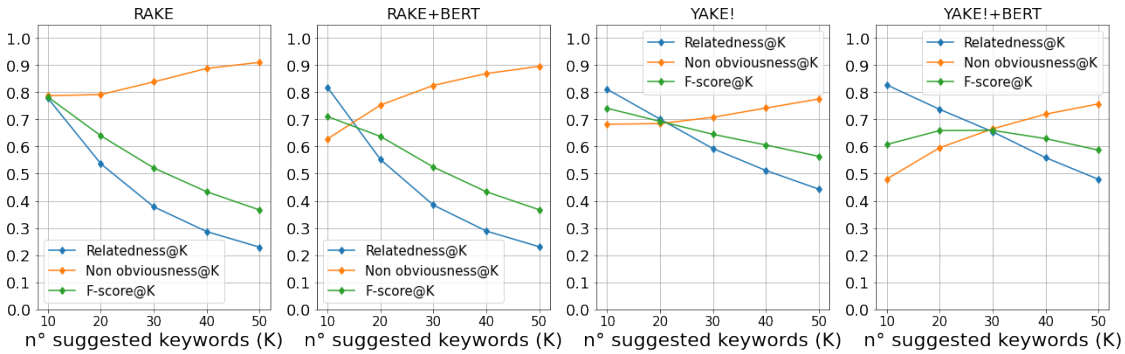
As pointed out in Subsection 7.1.2, there are three different scrap levels, each of them with four different models. For every model, five different number of suggested keywords (K) are extracted. All the possible combinations can be figured out as $3 \cdot 4 \cdot 5 = 60$ tables with 100 rows (one for each query keywords) and a number of columns equal to the number of suggestions required.

This representation reflects the way in which results are stored: for each scrap level, an `.xlsx` file is created, containing for each model the results (*Relatedness@K*, *Non obviousness@K* and *F-score@K*) for five different number of suggestions ($10, 20, 30, 40, 50$), for a total of $4 \cdot 5 = 20$ sheets per file.

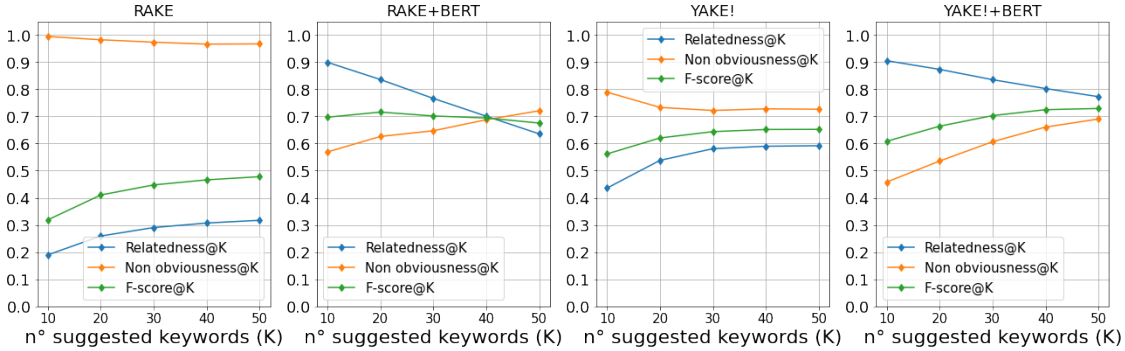
All results are shown in Figure 7.1

Comments on Figure 7.1a:

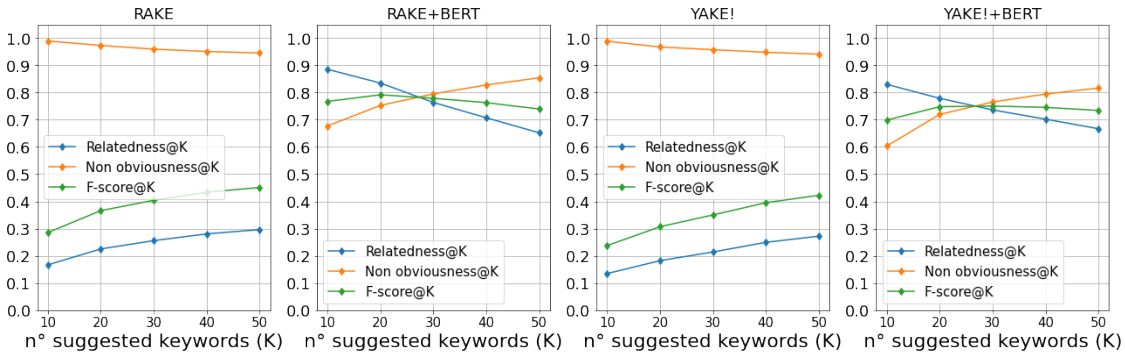
1. taking only `<title>` and `<h1>` tags, the text contains more terms closely related to the *query keywords* and tends to exhibit a lower lexical variability. This implies that RAKE and YAKE! discover keywords close to the *seed keyword* (receiving high score) or quite different (receiving low score). When BERT filter is applied to these keywords, the ones closed to the query keywords receive high similarity while the ones quite different receive low similarity. So BERT filter tends to award keywords similar to the query, despite their originality. Looking at the plots (RAKE vs RAKE+BERT and YAKE! vs YAKE!+BERT), *Relatedness@K* is pretty similar with and without BERT filter for each n° suggested keywords (K). *Non obviousness@K* is ≈ 0.20 lower with BERT filter. These two observations put together imply that BERT filter has replaced in top positions some relevant and non-obvious keywords with some relevant and obvious ones. Hence, the application of BERT filter is not suggested using *super-light* scrap level since the "originality" of suggestions decreases while the *Relatedness@K* does not increase.



(a) *Super-Light* web scrap.



(b) *Light* web scrap.



(c) *Full* web scrap.

Figure 7.1: *Relatedness@K*, *Non obviousness@K* and *F-score@K* for different scrap levels, models and n° suggested keywords.

2. Looking at the plots RAKE vs YAKE!, YAKE! offers higher *Relatedness@K* at the price of a slightly lower *Non obviousness@K*. The overall tradeoff is favourable to YAKE!, since its *F-score@K* is higher for every n° suggested keywords, except 10. For 10 suggested keywords, RAKE has a significantly higher *Non obviousness@K* and an *F-score@K* ≈ 0.05 higher. Anyway it's just a little better performance, which does not change the overall superior shown by YAKE!.

3. Putting together points 1. and 2., the best model using *super-light* scrap level is YAKE!.

Comments on Figure 7.1b:

1. taking `<title>`, `<h1>`, `<h2>`, `<ul>` and `<ol>` tags, the text contains a higher proportion of terms not closely related to the query keywords and tends to exhibit a richer lexical variability. This implies that RAKE and YAKE! discover keywords which are less close to the *seed keyword*. When BERT filter is applied to these keywords, the ones closed to the query keywords receive high similarity while the ones quite different receive low similarity. So BERT filter tends to award keywords similar to the query, despite their originality. Looking at the plots (RAKE vs RAKE+BERT and YAKE! vs YAKE!+BERT), *Relatedness@K* improves significantly with BERT filter for each n° suggested keywords (K). *Non obviousness@K* is also significantly lower with BERT filter. These two observations put together imply that BERT filter has replaced in top positions some non-relevant and non-obvious keywords with some relevant and obvious ones. Hence, the application of BERT filter is suggested using *light* scrap since the "originality" of suggestions decreases less than the increase in the *Relatedness@K*. In general, RAKE seems to benefit more by the BERT filter, while YAKE! benefits less, and the marginal benefits of BERT filter reduces as the n° suggested keywords increases (i.e. YAKE! with 50 keywords is just slightly worse than YAKE!+BERT with the same number of suggestions).

2. Looking at the plots RAKE vs YAKE!, YAKE! offers higher *Relatedness@K* at the price of a lower *Non obviousness@K*. Anyway the overall tradeoff is favourable to YAKE!, since its *F-score@K* is ≈ 0.2 higher for every n° suggested keywords.

3. Putting together points 1. and 2., the best model using *light* scrap is:

    - for $K = \{10, 20\}$: RAKE+BERT, since its *F-score@K* is ≈ 0.1 and ≈ 0.06 higher than YAKE!+BERT, respectively;
    - for $K = \{30, 40\}$: RAKE+BERT and YAKE!+BERT are different for less than 0.05, so are almost equivalent alternatives;
    - for $K = 50$: YAKE!+BERT is ≈ 0.05 higher than RAKE+BERT.

Comments on Figure 7.1c:

1. taking `<title>`, `<h1>`, `<h2>`, `<ul>`, `<ol>` and `<p>` tags, the text contains an even higher proportion of terms not closely related to the query keywords and tends to exhibit an even richer lexical variability. This implies that RAKE and YAKE! discover keywords which are less close to the *seed keyword*. When BERT filter is applied to these keywords, the ones closed to the query keywords receive high similarity while the ones quite different receive low similarity. So BERT filter tends to award keywords similar to the query, despite their originality. Looking at the plots (RAKE vs RAKE+BERT and YAKE! vs YAKE!+BERT), *Relatedness@K* improves significantly with BERT filter for each n° suggested keywords (K). *Non obviousness@K* is also significantly lower with BERT filter. These two observations put together imply that BERT filter has replaced in top positions some non-relevant and non-obvious keywords with some relevant and obvious ones. Hence, the application of BERT filter is suggested using *full* scrap level since the "originality" of suggestions decreases less than the increase in the *Relatedness@K*. In general, RAKE and YAKE! show very similar performances and seem to benefit in the same way by the application of BERT filter.

2. Looking at the plots RAKE vs YAKE!, RAKE offers a slightly higher *Relatedness@K* at the price of an identic *Non obviousness@K*. Hence, the overall tradeoff is favourable to RAKE, since its *F-score@K* is ≈ 0.05 higher for every n° suggested keywords.

3. Putting together points 1. and 2., the best model using *full* scrap level is RAKE+BERT. It shows a slightly higher *Relatedness@K* than YAKE!+BERT for low n° suggested keywords and a higher *Non obviousness@K* for any number of suggestions. Its *F-score@K* is always greater except for $K = 50$, where F-scores are almost the same.

Having an idea of which model works better for every scrap level, it's time to find the best model in absolute. The model with the highest *Relatedness@K*, *Non obviousness@K* and hence, *F-score@K*, for almost all K's values it's RAKE+BERT with *full* scrap level, closely followed by YAKE!+BERT. Both models have an initial *Relatedness@K* ($K = 10$) well above 0.80, which declines until ≈ 0.65 for $K = 50$. The *Non obviousness@K* starts above 0.60 and increases over 0.8. Exist combinations of models and scrap levels with a better *Relatedness@K* (i.e. YAKE!+BERT with *light* scrap level) or with a better *Non obviousness@K* (i.e. RAKE with *super-light* scrap level; RAKE with *light* scrap level; RAKE and YAKE! with *full* scrap level). But no one of them has both metrics as high as RAKE+BERT and YAKE!+BERT with *full* scrap level.

It's worth noticing that YAKE! shows a clear superiority with respect to RAKE for *super-light* and *light* scrap levels, probably determined by the far more complex keywords extraction procedure (Chapter 6, Subsection 6.2.2). Anyway when the text's size increases excessively (*full* scrap

level), YAKE!'s complexity becomes a double edge sword, producing a lower *Relatedness@K* and *F-score@K*, for all K values, than the simpler RAKE algorithm.

The importance of the BERT filter increases with the text's size and with the simplicity of the keywords extractor algorithm. A simpler extractor, like RAKE, benefits by the BERT filter from the *light* scrap level onward; a more complex extractor, like YAKE!, benefits significantly by the BERT filter for deeper scrap levels.

These results confirm what claimed at the end of Chapter 6: the best model and the effectiveness of the BERT filter depend on the scrap level and n° suggested keywords. It's anyway possible to identify a model which works pretty well always, being the best or close to the best in every case: YAKE!+BERT. Its performances are:

- close to the best model (YAKE!) for *super-light* scrap level;
- the best, together with RAKE+BERT, for *light* scrap level;
- close to the best model (RAKE+BERT) for *full* scrap level.

Hence, as a rule of thumb, it's possible to use YAKE!+BERT for any scrap level and number of suggestions.

For completeness, the exact values of each metric for all scrap levels, models, and n° suggested keywords are reported in Tables 7.2, 7.3, and 7.4.

Recalling that RAKE and YAKE! have been chosen, respectively, for their simplicity and effectiveness (Chapter 6, Section 6.2), it's expectable to find a computational time for the latter of several order of magnitude greater than the former. In order to quantify the running time of both algorithms, the text retrieved for each query is provided to the keywords extractors and the time needed to return the suggested keywords is measured. Using the representative set of 100 keywords introduced ealier, 100 running times are computed for both RAKE and YAKE!. Moreover, for each query, exist three different retrieved texts, one for each scrap level.

To get a synthetic and robust measure, the average value of the running time for each scrap level and each model is computed. Results are reported in Table 7.1. For example, YAKE!'s computational time it's $\approx 400\times$ the one required by RAKE using *super-light* scrap level. Interestingly, YAKE! scales better than RAKE with the text's size: for *light* and *full* scrap levels, YAKE! running time is $\approx 6\times$ and $\approx 8\times$ bigger, while RAKE running time is $\approx 30\times$ and $\approx 174\times$ bigger. The size reported in the table refers to the dictionary containg all the retrieved text for all query keywords for a certain scrap level.

		Size (KB)	Running time (seconds)	
			RAKE	YAKE!
Scrap level	*Super-Light*	105	0.005	1.983
	Light	2915	0.151	12.193
	Full	9354	0.870	16.281

Table 7.1: Average running time per query keyword.

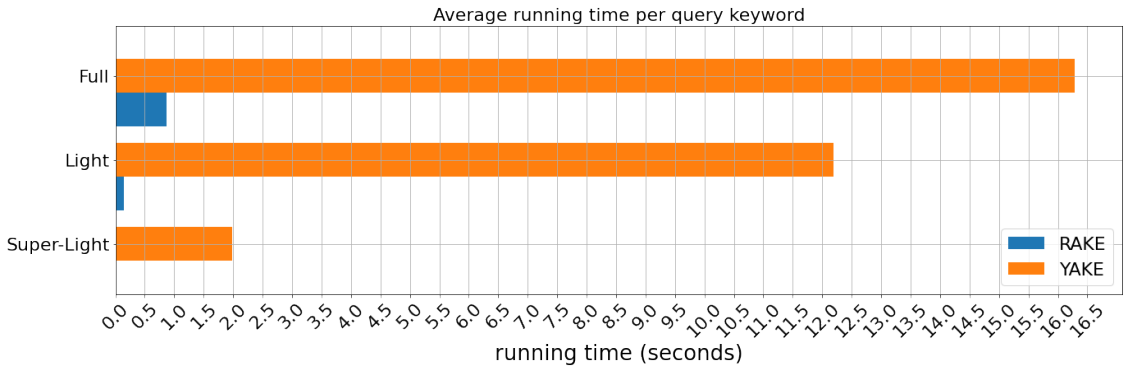


Figure 7.2: Average running time per query keyword.

The average running time is also depicted in Figure 7.2. Note that RAKE's average running time per keyword, when *super-light* scrap level is used, is so short (0.005 seconds) that the horizontal blue bar is invisible.

7.3 Illustrative Example

The goal of this section is to show some results to the reader for various models and scrap levels. In order to keep the presentation short and avoid and endless collection of tables, only an illustrative example of five query keywords is reported, together with the first 10 suggestions.

Note that not all combinations of models and scrap levels are shown. For *super-light* web scrap, all four models' results are reported, allowing the reader to get a rough idea about how the suggestions vary for different keywords extractor techniques (Tables 7.5, 7.6, 7.7, 7.8).

Then, for the model with the best average performances emerged in Section 7.2 (YAKE!+BERT), results for the remaining scrap levels (*light* and *full*) are shown (Tables 7.9 and 7.10). In this way, the reader can have an intuitive idea of the scrap level's effect on suggested keywords.

Being aware of the difficulty dictated by the reduced number of query keywords and sug-

RAKE

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.78	0.54	0.38	0.29	0.23
Non obviousness@K	0.79	0.79	0.84	0.89	0.91
F-score@K	0.78	0.64	0.52	0.43	0.37

RAKE + BERT

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.82	0.55	0.39	0.29	0.23
Non obviousness@K	0.63	0.75	0.83	0.87	0.90
F-score@K	0.71	0.64	0.53	0.43	0.37

YAKE!

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.81	0.70	0.59	0.51	0.44
Non obviousness@K	0.68	0.68	0.71	0.74	0.78
F-score@K	0.74	0.69	0.64	0.61	0.56

YAKE! + BERT

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.83	0.74	0.65	0.56	0.48
Non obviousness@K	0.48	0.60	0.67	0.72	0.76
F-score@K	0.61	0.66	0.66	0.63	0.59

Table 7.2: *Relatedness@K*, *Non obviousness@K* and *F-score@K* for *Super-Light* web scrap.

RAKE

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.19	0.26	0.29	0.31	0.32
Non obviousness@K	0.99	0.98	0.97	0.97	0.97
F-score@K	0.32	0.41	0.45	0.47	0.48

RAKE + BERT

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.90	0.84	0.77	0.70	0.64
Non obviousness@K	0.57	0.63	0.65	0.69	0.72
F-score@K	0.70	0.72	0.70	0.69	0.68

YAKE!

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.44	0.54	0.58	0.59	0.59
Non obviousness@K	0.79	0.73	0.72	0.73	0.73
F-score@K	0.56	0.62	0.64	0.65	0.65

YAKE! + BERT

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.91	0.87	0.84	0.80	0.77
Non obviousness@K	0.46	0.54	0.61	0.66	0.69
F-score@K	0.61	0.66	0.70	0.72	0.73

Table 7.3: *Relatedness@K*, *Non obviousness@K* and *F-score@K* for *Light* web scrap.

RAKE

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.17	0.23	0.26	0.28	0.30
Non obviousness@K	0.99	0.97	0.96	0.95	0.95
F-score@K	0.29	0.37	0.40	0.43	0.45

RAKE + BERT

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.89	0.83	0.76	0.71	0.65
Non obviousness@K	0.68	0.75	0.79	0.83	0.85
F-score@K	0.77	0.79	0.78	0.76	0.74

YAKE!

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.14	0.18	0.21	0.25	0.27
Non obviousness@K	0.99	0.97	0.96	0.95	0.94
F-score@K	0.24	0.31	0.35	0.40	0.42

YAKE! + BERT

	n° suggested keywords (K)				
	10	20	30	40	50
Relatedness@K	0.83	0.78	0.74	0.70	0.67
Non obviousness@K	0.60	0.72	0.76	0.79	0.82
F-score@K	0.70	0.75	0.75	0.74	0.73

Table 7.4: *Relatedness@K*, *Non obviousness@K* and *F-score@K* for *Full* web scrap.

gestions reported here, the hope is that the reader can anyway find confirmation of (at least some of) the comments provided in Section 7.2 and find them reasonable. In particular, given the first 10 suggested keywords, it's possible to verify that their average *Relatedness@K* and *Non obviousness@K* are in line with the values reported in Figure 7.1a and on the leftmost graph in Figure 7.1b and 7.1c for $K = 10$.

Related suggestions are written in green and *non-obvious* suggestions have a purple background. From Section 7.1, it's useful to recall that the unique objective rules used in order to assess the *relatedness* and *non-obviousness* of a suggestion are:

1. when the suggested keyword is a copy of the query (even with a different order of words), then it is for sure *related* (i.e. "rent office space");

2. when the suggested keyword is a subset of the query, then it is for sure *obvious* (i.e. "office space").

Query keyword	suggested keyword									
	n° 1	n° 2	n° 3	n° 4	n° 5	n° 6	n° 7	n° 8	n° 9	n° 10
small office space rent	find startup small business offices	sublets owner spec suites	small office spaces	taiwan furnished offices	rent flexible office	small office space	rent office space	office space	serviced offices	hybrid workplace
car games	car games awesome racing	free online racing games	race car rush	car racing games	car racing apps	car games play	car racing	car games	car 2 apps	race car
office security guard	security guard office building jobs	improve workplace security	office security guard	hugh dane wikipedia	security guard	office security	security guards	hugh dane	office wiki	hollywood reporter
mothers day card picture	mothers day photo upload cards	pictures royaltyfree images istock	mothers day photo cards	mothers day greeting cards	mothers day card pictures	mothers day card images	mothers day cards	270 mothers day cards	340 happy mothers day	happy mothers day
spanish lessons	free online spanish classes	free online spanish lessons	free online classes	3 months language hacking	free online lessons	start speaking spanish	learn spanish online	learn spanish duolingo	bbc learn spanish	free spanish lessons

Table 7.5: Illustrative example of RAKE with *Super-Light* web scrap.

Query keyword	suggested keyword									
	n° 1	n° 2	n° 3	n° 4	n° 5	n° 6	n° 7	n° 8	n° 9	n° 10
small office space rent	small office space	rent office space	small office spaces	rent flexible office	office space	find startup small business offices	taiwan furnished offices	australia spacely	rent taipei	rent
car games	car games	car racing games	car games play	car games awesome racing	car	games	car racing	car racing apps	car 2	race car
office security guard	office security guard	office security	security guard	security guard office building jobs	security guards	office	improve workplace security	office wiki	hollywood reporter	employment
mothers day card picture	mothers day card pictures	mothers day card images	mothers day photo cards	mothers day cards	270 mothers day cards	mothers day photo upload cards	mothers day	340 happy mothers day	mothers day greeting cards	happy mothers day
spanish lessons	spanish lessons	free spanish lessons	lessons	learn spanish	free online spanish lessons	bbc learn spanish	spanish courses	spanish experiment	learn spanish duolingo	learn spanish online

Table 7.6: Illustrative example of RAKE+BERT with *Super-Light* web scrap.

Query keyword	suggested keyword									
	n° 1	n° 2	n° 3	n° 4	n° 5	n° 6	n° 7	n° 8	n° 9	n° 10
small office space rent	space for rent in taiwan	space for rent in australia	space for rent in taipei	office space for rent taipei	rent office space	small office spaces for rent	office space	search for small office space	small office space	office space for rent
car games	play in the car	apps on google play	play car games online	free online racing games	car racing games	awesome racing games	play car games	car games	apps on google	car games online
office security guard	guards for your office	security guards for your office	security guard office building jobs	guard office building jobs	security guard office building	office building jobs	office security guard	guard office building	office security guard services	security guard office
mothers day card picture	mothers day card stock photos	mothers day card images	mother day photo cards designs	personalized mother day photo cards	mothers day card stock	mother day photo upload cards	mothers day card pictures	mother day photo cards	day card stock photos	compare personalized mother day photo
spanish lessons	world best way to learn	free online spanish language lessons	free online spanish language	online spanish language lessons	free online spanish lessons	free online spanish	free online spanish classes	learn spanish with free online	learn spanish online	learn spanish

Table 7.7: Illustrative example of YAKE with *Super-Light* web scrap.

Query keyword	suggested keyword									
	n° 1	n° 2	n° 3	n° 4	n° 5	n° 6	n° 7	n° 8	n° 9	n° 10
small office space rent	small office space	small office spaces for rent	rent office space	office space for rent	search for small office space	office space for rent taipei	small business offices to rent	small office	space for rent	rent office
car games	car games	car racing games	play car games	car	car games online	games	racing in car	car racing	play car games online	racing games
office security guard	office security guard	security guard office	security guard office building	office security guard services	guard office	office security	security guard	security guard office building jobs	security guard services	guard office building
mothers day card picture	mothers day card pictures	mothers day card images	mothers day card	day card pictures	mother day photo cards	mothers day card stock photos	day photo cards	mother day photo cards designs	day card images	mothers day
spanish lessons	spanish lessons	spanish language lessons	free spanish lessons	spanish lessons with audio	lessons	online spanish lessons	online spanish language lessons	learn spanish	online spanish lessons with audio	free online spanish language lessons

Table 7.8: Illustrative example of YAKE+BERT with *Super-Light* web scrap.

Query keyword	suggested keyword									
	n° 1	n° 2	n° 3	n° 4	n° 5	n° 6	n° 7	n° 8	n° 9	n° 10
small office space rent	rent a small office space	small office spaces for rent	office space for rent	small office spaces	private office space for rent	airpark small office spaces	hills office space for rent	beautiful office space for lease	unique small office spaces	office space for lease
car games	car games	car racing games	car racing games category	car quiz games	play car games	police car games	car puzzle games	motorcycle racing games	car	car games online
office security guard	security guard office	security guard office building	office building security guards	guard office	office security	security guard	security guard office building jobs	security guard services for office	office building security	protection security guard
mothers day card picture	mothers day card pictures	mothers day card images	photo mothers day cards	mothers day card	mothers day cards	day card pictures	photo mothers day	mothers day card flowers	day card template mothers	day card template mothers
spanish lessons	spanish lessons	spanish video lessons	video spanish lessons	spanish language lessons	free spanish lessons	lesson spanish	beginner spanish lessons	spanish lessons with audio	insta spanish lessons	free video spanish lessons

Table 7.9: Illustrative example of YAKE+BERT with *Light* web scrap.

Query keyword	suggested keyword									
	n° 1	n° 2	n° 3	n° 4	n° 5	n° 6	n° 7	n° 8	n° 9	n° 10
small office space rent	rent flexible office	rent flexible office from cowork-ing	flexible office	flexible office from cowork-ing	office	accessing officelist-com	office from cowork-ing	rent	browser before accessing officelist-com	officelistcom
car games	car games	car game	car racing games	stunt car games	enabled car games	car racing games category	top car games	car quiz games	gear for car racing games	play car games
office security guard	security guard office	security guard office building	office building security guards	guard of-fice	office security	security guard	security guard office building jobs	security guard services for office	office building security	protection security guard
mothers day card picture	mothers day card pictures	mothers day card images	mothers day cards	mothers day card flowers	day card template mothers	day card template mothers	card template mothers day	card template mothers day	mother day photo cards	mothers day card stock photos
spanish lessons	spanish lessons	spanish video lessons	video spanish lessons	spanish language lessons	free spanish lessons	spanish audio lessons	beginner spanish lessons	spanish courses have lessons	spanish lessons know that learning	free video spanish lessons

Table 7.10: Illustrative example of YAKE+BERT with *Full* web scrap.

8 Conclusion

The suggestion of *relevant* and *non-obvious* new keywords is of main importance for the broker's profitability: bidding on many non-obvious low traffic keywords, the combined traffic of them can add up to the level produced by a popular (and so very expensive) keyword, but at a fraction of the cost. Moreover, the traffic received is targeted better and will typically result in a better convertion rate. These observations motivated the need to find out new alternative keywords, relevant to the base query, but non-obvious in nature, so that little competition was faced from other advertisers.

The problem has been addressed keeping in mind also other important requirements:

1. reduce the reliability on the ad landing page;
2. suggest keywords with good quality, quantity and variety;
3. be (as far as possible) language independent.

All these considerations have led to the development of a procedure based on two main steps: *web scraping* and *keywords extraction*. The starting point was a *query* or *seed keyword* for which the user wanted to find some meaningful alternative suggestions.

Web scraping allowed to retrive the raw text associated to each query directly from the top URLs resulting from a Google search. In detail, given the seed keyword, the top URLs' web pages have been scraped using different sets of HTML tags. Based on which set was used, the depth to which the web page was scraped varied, defining three different levels (*super-light*, *light*

and *full*). The texts coming from all URLs have been joined in a unique one, which has been cleaned from special characters.

Keywords extraction received as input the cleaned text and identified the keywords (single or multiple words). An overview of the main types of keywords extractors models available in the literature have been presented, highlighting pros and cons of each of them and motivating the chosen ones: *RAKE* and *YAKE!*.

RAKE has been chosen for its simplicity, fastness, and almost language independence. Moreover, it represented a good benchmark to evaluate more complex models. Here came in play *YAKE!*, which has been chosen as a state-of-the-art model for its performances in keywords extractions tasks (and language independence). The theory underlying both models has been presented together with some examples.

A *BERT Filter* has been developed in order to increase the relevance of the keywords extracted by both models. It worked by encoding the query keyword and each suggestion using "BERT-Large Uncased model" and then computing the cosine similarity between the query and each of the suggestions. This tool moved suggested keywords, very similar to the seed, on top positions, which could be positive (avoid non relevent suggestions) but even negative (increase obviousness of suggestions).

This showed the need to develop a methodology to evaluate systematically the keywords extracted for all the possible combinations of scrap levels, models and number of keywords returned. In order to do so, an overview of the available metrics has been presented, highlighting pros and cons of each of them and selecting the most suitable ones for the problem at hand: *Relatedness@K*, *Non obviousness@K* and *F-score@K*.

Then the selected metrics have been used to evaluate the performances of all possible combinations of scrap levels, models and n° suggested keywords. Results have been presented graphically and in tabular format, highlighting the most important observations and defining the best model for each scrap level. As expected, it turned out the non-existence of a model always superior to others, but emerged also that YAKE!+BERT was the model with more robust performances through different scrap levels and number of suggestions.

Finally, a subset of 5 keywords was used to present some empirical results.

Some important final remarks:

- the new keywords' suggestion procedure has been designed to work both autonomously and in synergy with the *New Keyword Generation Module*, previously developed by ACTOR;

- RAKE does not have hyperparameters to set; YAKE! counts four main hyperparameters (Subsection 6.2.2), which values have been set following the analysis provided by authors in their reference paper [28]. With more time and human resources, could have been possible to fine tune such hyperparameters on the specific features of the text scrapped from webpages, which it's expected to be significantly different from a homogeneous body of text written by a human (like a report, an abstract, a thesis, etc.).

- An alternative metric to evaluate "relevance" and "non-obviousness" could have been MAP (Subsection 7.1.1). It would have required to store for each set of suggested keywords, not only the aggregate number of *related* and *non-obvious* suggestions, but also the judgement (non-related/obvious $= 0$, related/non-obvious $= 1$) for each suggested keyword. It would have taken too much time, hence simpler metrics (*Relatedness@K*, *Non obviousness@K*) have been used. The use of MAP could be deferred as a future development to check further the robustness of results.

Part III

Supplementary Material

A

CPC-Optimizer

The CPC-optimizer is a tool, developed by ACTOR in the business project perimeter, able to dynamically adjust the bids for existing keywords. It empirically works well, providing a valuable help in the resolution of part of the BBS problem. Anyway it is not able to propose an initial bid for new keywords, which is important in order to maximize profits. In fact, an optimal (or quasi-optimal) initial bid allows to avoid wide bid adjustments when a new keyword is launched. Such adjustments are economically costly, either in terms of missing revenue (underbidding) and of excessive costs (overbidding).

To better understand the context, the CPC-optimizer is going to be briefly presented in the following. Note that only the main idea is explained, while the low level details are voluntarily omitted, in order to protect ACTOR's intellectual property on this tool and its client, who paid for it. The CPC Optimizer has to determine, for each keyword, a bid suggestion value in order to increase profits. Increasing or decreasing the bid decision is not obvious at all. Increasing the bid of some keywords possibly leads to more clicks and more revenues, even though an excessive increase could result in paying too much with respect to the induced revenues. Similarly, decreasing the bid of other keywords could balance a current losing trend and in some case also increment profit, if previously the bid was set to a too high value compared to the revenues.

The result is a local search oriented online algorithm, that tries to get closer to the optimum configuration at each step. Obviously, there not exist a configuration that is better than the others forever. The environment characteristics, like users'interest and platforms'auctions contes-

tants, change day by day. What this method does is trying to detect, and thus follow or invert, current trends. The more these trends are stable, the more accurate the suggestions are.

In detail, there are two main steps:

1. bid proposal;
2. bid validation.

A.1 Bid Proposal

The input data for the CPC-optimizer is the clustering of keywords at the ad-group level. The keywords are clustered along two dimensions which represent the profitability: profit (p) and Return on Investment (r). So each keyword i can be visualized as a point $(p_i; r_i)$ in the plane with p on horizontal axis and r on vertical axis. Each cluster k has a centroid indicated as $(\overline{p_k}; \overline{r_k})$. An example table is reported:

Cluster	$\bar{p}$	$\bar{r}$
0	6654.84	1.11
1	2963.41	0.76
2	719.35	0.69
3	0	0
4	−19.80	−3.97
5	−115.92	−0.33

Table A.1: keywords clustered by profitability.

The sum of the positive $\bar{p}$ $\left(\sum p^+\right)$ and the sum of negative $\bar{p}$ $\left(\sum p^-\right)$ are computed. The same is done also for the $\bar{r}$, obtaining $\sum r^+$ and $\sum r^-$.

$\sum p^+$	$\sum p^0$	$\sum p^-$
10337.60	0	−135.72

Table A.2: Sum of positive and negative profits.

$\sum r^+$	$\sum r^0$	$\sum r^-$
2.56	0	-4.30

Table A.3: Sum of positive and negative return on Investment.

Then two cases are distinguished.

A.1.1 Case 1: $\sum p^+ \geq \sum p^-$

In this case the ratio $\sum p^+ / \sum p^- > 1$, indicating that the profitability scenario is favorable and more risk could be undertaken. For each keyword i the formula to determine the new bid is:

$$BID_i^{new} = \min\Big(\beta; BID_i + (M_1 \cdot w_1 + M_2 \cdot w_2 + M_3 \cdot w_3) \cdot \alpha \cdot BID_i\Big)$$

with:

- β the max bid value
- BID_i the current bid
- M_1 represents the incidence of the p_i on the total positive or negative profit.
- M_2 represents the incidence of the r_i on the total positive or negative ratio.
- M_3 represents the risk factor.
- $w_1, w_2, w_3 \geq 0$ represent the contribution of each metric M, with $\sum_j w_j = 1$.
- $\alpha = \begin{cases} \alpha_+, & \text{if } M_1 \cdot w_1 + M_2 \cdot w_2 + M_3 \cdot w_3 \geq 0 \\ \alpha_-, & \text{if } M_1 \cdot w_1 + M_2 \cdot w_2 + M_3 \cdot w_3 < 0 \end{cases}$

  α_+ represents the maximum increment percentage while α_- represents the maximum decrement percentage.

A.1.2 Case 2: $\sum p^+ < \sum p^-$

In this case the ratio $\sum p^- / \sum p^+ > 1$, indicating that the profitability scenario is negative and more protection could be undertaken. For each keyword i the formula to determine the new bid is still:

$$BID_i^{new} = \min\Big(\beta; BID_i + (M_1 \cdot w_1 + M_2 \cdot w_2 + M_3 \cdot w_3) \cdot \alpha \cdot BID_i\Big)$$

with:

- β, BID_i, M_1 and M_2 defined as in Case 1
- M_3 represents the risk factor.
- w_1, w_2, w_3 represent the contribution of each metric M, with $w_3 < 0$ to ensure the M_3 behaves as a protection factor.

Note that in both cases, $\beta, w_1, w_2, w_3, \alpha_+$ and α_- are hyperparameters to be fine-tuned on the available data.

A.2 Bid Approval

The bid proposal (BID_i^{new}) is validated using the variation in the profits ($\Delta p_i^{(t)} = p_i^{(t)} - p_i^{(t-1)}$) recorded in the previous timeframe. If the bid proposal of the previous time led to $\Delta p_i^{(t)} > 0$ then it was correct and so it's meaningful to confirm its variation direction (increase or decrease) even in the current bid proposal. If the bid proposal of previous time led to $\Delta p_i^{(t)} < 0$ then it was wrong and so it's meaningful to change its variation direction (increase or decrease) in the current bid proposal. The equation A.1.1 determines the BID_i^{new}, from which is possible to compute the associated bid variation $\Delta BID_i = BID_i^{new} - BID_i$. The bid approval just takes the bid variation value and confirms its direction or change it.

The CPC-optimizer procedure described above clearly requires the existence of historical observations. In fact BID_i^{new} is computed using metrics M_1, M_2 and M_3, which need records about profit p and ROI r at the keyword's and cluster's level. This explains why CPC-optimizer is not able to propose an initial bid for new keywords.

B

BERT Encoding

The example used in Subsection 2.3.2 is expanded, showing the results from "BERT-Base Uncased" and "BERT-Large Uncased". To allow comparison, the same strategies and the same target keyword "pediatric nurse" are used and its cosine similarity with all the other keywords is computed. Then the keywords are ranked in decreasing order of cosine similarity. If the BERT encoding is effective, the most similar keywords should be considered similar also by a human reader.

keyword	cosine similarity
pediatric nurse	1.000000
online nurse practitioner programs	0.730416
asthma treatment medicine	0.700486
treat lung cancer	0.648124
medical assistant certificate	0.612843
medical billing certificate	0.585164
Nursing Masters Programs	0.581949
donate children	0.577971
diet food delivery	0.558943
credit counseling	0.554224
health insurance find	0.552560
medical coding online	0.549473
birth control pill	0.544094
medicare information	0.537684
maryland health insurance	0.533820
insulin pumps for diabetics	0.532031
Family Insurance	0.528918
Lawyer Divorce	0.525006
Criminal Law Lawyer	0.522903
Medicare Plan Supplement	0.521980

Table B.1: Cosine similarity using BERT-Base Uncased encoding with pooling layer=0,1,2 and pooling strategy={REDUCE MEAN}

keyword	cosine similarity
pediatric nurse	1.000000
online nurse practitioner programs	0.767196
asthma treatment medicine	0.745355
treat lung cancer	0.703880
medical billing certificate	0.663241
medical assistant certificate	0.654847
credit counseling	0.645460
medicare information	0.644451
Criminal Law Lawyer	0.641803
birth control pill	0.640108
Nursing Masters Programs	0.638956
diet food delivery	0.632093
donate children	0.630807
vacation rental service	0.626829
health insurance find	0.625261
medical coding online	0.623044
vehicle donation programs	0.611841
banyan treatment center	0.611063
dental plan discounts	0.610930
Family Insurance	0.609444

Table B.2: Cosine similarity using BERT-Base Uncased encoding with pooling layer=2 and pooling strategy={REDUCE MEAN}

keyword	cosine similarity
pediatric nurse	1.000000
asthma treatment medicine	0.7080124
catholic charities car donation	0.702969
unlimited satellite internet	0.681546
best bank accounts	0.656119
Small Business Insurance Companies	0.645487
cost of bed bug extermination	0.631689
3d printing quote	0.627134
car loan auto	0.624982
car loan auto	0.624982
Family Insurance	0.623834
cheap internet	0.619958
psychic reading	0.619656
unlimited plans cell	0.616703
insurance auto quote	0.615741
banks with free checking	0.610272
banks with free checking	0.610272
10 best charities to donate to	0.607796
ethical hacking course	0.605017
vacation rental service	0.604246

Table B.3: Cosine similarity using BERT-Base Uncased encoding with pooling layer=$-3,-2,-1$ and pooling strategy={REDUCE MEAN}

keyword	cosine similarity
pediatric nurse	1.000000
asthma treatment medicine	0.736651
catholic charities car donation	0.734746
unlimited satellite internet	0.707598
best bank accounts	0.698270
cost of bed bug extermination	0.679451
Small Business Insurance Companies	0.672396
3d printing quote	0.671720
insurance auto quote	0.665925
car loan auto	0.664138
car loan auto	0.664138
Family Insurance	0.662067
sell my motor home	0.662022
vacation rental service	0.661001
banks with free checking	0.655724
banks with free checking	0.655724
cheap internet	0.654772
psychic reading	0.653736
unlimited plans cell	0.653280
10 best charities to donate to	0.652314

Table B.4: Cosine similarity using BERT-Base Uncased encoding with pooling layer = -2 and pooling strategy = {REDUCE MEAN}

keyword	cosine similarity
pediatric nurse	1.000000
online nurse practitioner programs	0.742176
asthma treatment medicine	0.706052
Nursing Masters Programs	0.650388
medical assistant certificate	0.623150
medical billing certificate	0.611977
treat lung cancer	0.610750
maryland health insurance	0.595861
emergency 24 hour dentist near me	0.595595
healthcare management master	0.594843
medical coding online	0.580702
Revenue Cycle Management Healthcare	0.578937
non cell lung cancer	0.576651
marketplace healthcare	0.569450
Hvac Service Software	0.561025
health insurance find	0.560748
insulin pumps for diabetics	0.557860
donate children	0.555634
vehicle donation programs	0.549034
credit counseling	0.548361

Table B.5: Cosine similarity using BERT-Large Uncased encoding with pooling layer=0,1,2 and pooling strategy={REDUCE MEAN}

keyword	cosine similarity
pediatric nurse	1.000000
online nurse practitioner programs	0.839324
asthma treatment medicine	0.808621
Nursing Masters Programs	0.787815
medical assistant certificate	0.760266
medical billing certificate	0.754390
medical coding online	0.742451
treat lung cancer	0.739876
healthcare management master	0.736500
marketplace healthcare	0.729050
Revenue Cycle Management Healthcare	0.720867
maryland health insurance	0.720808
Hvac Service Software	0.718313
emergency 24 hour dentist near me	0.717708
Medicare Plan Supplement	0.715805
Discrimination Attorneys	0.715565
non cell lung cancer	0.707970
vehicle donation programs	0.705279
healthcare online	0.702710
Small Business Phone Service Providers	0.698940

Table B.6: Cosine similarity using BERT-Large Uncased encoding with pooling layer=2 and pooling strategy={REDUCE MEAN}

keyword	cosine similarity
pediatric nurse	1.000000
digital signage display	0.887898
Cheaper Auto Insurance	0.856732
Cloud Consulting	0.838356
bugs treatment	0.815097
Debt Credit Card	0.811776
online nurse practitioner programs	0.810771
Cyber Security News	0.803846
roof inspectors	0.797389
car loan auto	0.795332
car loan auto	0.795332
medical billing certificate	0.786446
barricades	0.785396
moving long distance	0.776120
Automobile Insurance Companies	0.775680
Lawyer Divorce	0.757866
business camera	0.756651
best tax attorneys	0.755589
servpro cleaning	0.732765
student bank accounts	0.731955

Table B.7: Cosine similarity using BERT-Large Uncased encoding with pooling layer=$-3,-2,-1$ and pooling strategy={REDUCE MEAN}

keyword	cosine similarity
pediatric nurse	1.000000
digital signage display	0.955910
Cheaper Auto Insurance	0.948115
Cloud Consulting	0.922213
bugs treatment	0.921089
barricades	0.915580
roof inspectors	0.913015
Cyber Security News	0.912776
Best Credit Card Deal	0.905895
car loan auto	0.895823
car loan auto	0.895823
medical billing certificate	0.894522
online nurse practitioner programs	0.894248
Debt Credit Card	0.891260
moving long distance	0.884843
best tax attorneys	0.881754
Automobile Insurance Companies	0.881015
servpro cleaning	0.872291
business camera	0.869385
student bank accounts	0.859880

Table B.8: Cosine similarity using BERT-Large Uncased encoding with pooling layer = -2 and pooling strategy = {REDUCE MEAN}

C Baseline Models Results

The results for the baseline models:

- multiple linear regression,
- Ridge regression,
- Lasso regression,

are reported for the datasets:

- `main dataset keyword level (BERT Encoding)`,
- `main dataset keyword level (GloVe Encoding)`.

C.1 BERT Encoding

C.1.1 Multiple Linear Regression

Firstly, the capacity of the multiple linear regression output to "follow" the true output is graphically inspected in Figure C.1. This allows to verify that the regression is not just returning a dummy value, like the average, for all the observations.

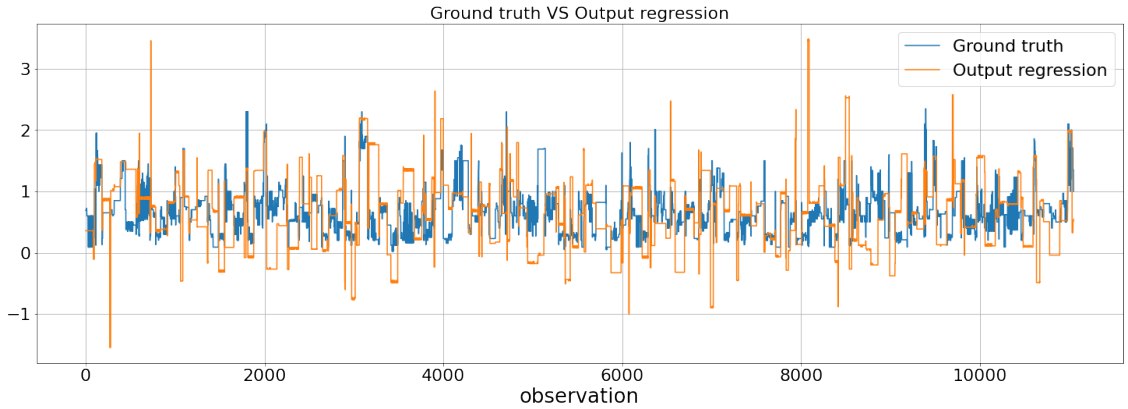
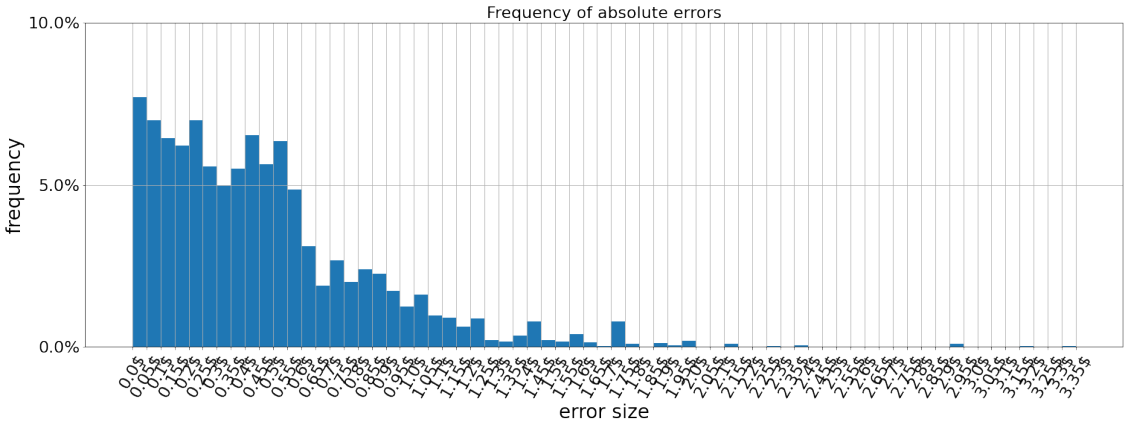
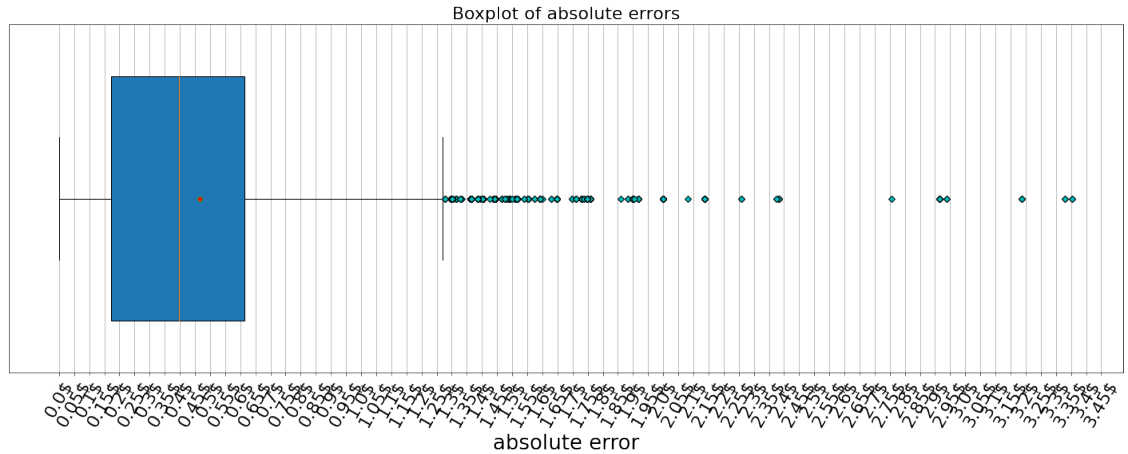


Figure C.1: Ground truth VS Output Regression.



(a) Regression's Absolute Errors distribution.



(b) Regression's Absolute Errors box-plot.

Figure C.2: Regression's Absolute Error quantile distribution.

The *average absolute error* is equal to $0.46\$$. To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure C.2.

The box-plot shows that 25% of errors are lower than 18 cents and 50% of errors are lower than 40 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 62 cents. There is another approximately 15% of cases where the error is between 63 and 90 cents. Very large errors are limited to the last 10% of data.

C.1.2 Ridge Regression

The Ridge Regression model requires the value of the hyperparameter λ. The term *hyperparameter* means that it is not determined by the model itself through the training, but it has to be specified externally. Different values of λ allow to reach different minima for the objective function. The optimal lambda (λ^*) is the one which leads to the smallest possible value of the objective function.

In order to find λ^* a *grid search* procedure together with a *cross validation* (CV) is applied. First, a list of λ values is identified; then, for each of them, the training set is split in five folds and, in turn, one is used as validation set and the remaining ones as training set. For each λ, the average value of the objective function through the five splits is computed and used as the metric to determine λ^*.

The whole procedure is repeated a second time selecting a finer grid of λ values around the best value found before. The λ^* is equal to 4850.

The capacity of the Ridge regression's output to "follow" the true output is graphically inspected in Figure C.3. This allows to verify that the regression is not just returning a dummy value, like the average, for all the observations.

The *average absolute error* is equal to $0.20\$$. This is a significant drop with respect to the $0.46\$$ achieved using multiple linear regression.

To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure C.4.

The box-plot shows that 25% of errors are lower than 7 cents and 50% of errors are lower than 15 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 28 cents. There is another approximately 15% of cases where the error is between 29 and 50 cents. Errors above 50 cents are limited to the last 10% of data. Considering the simplicity of the model, results are already good.

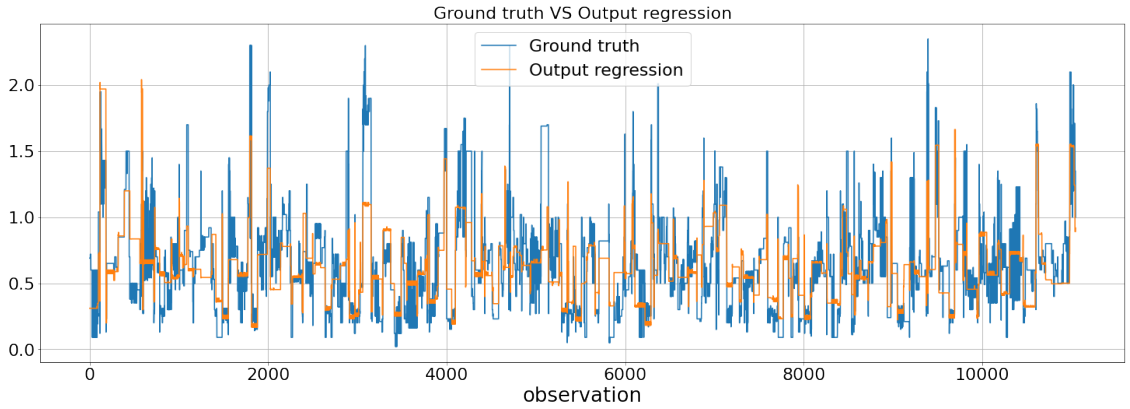
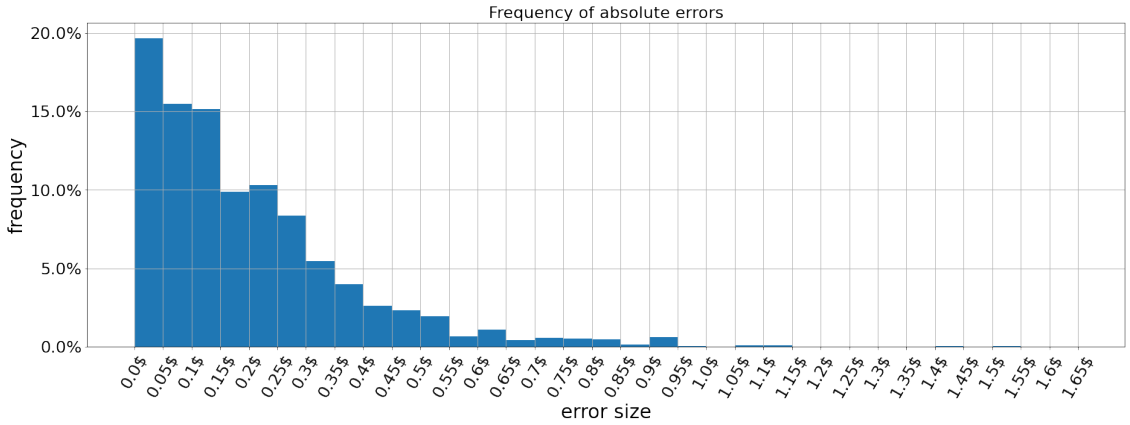
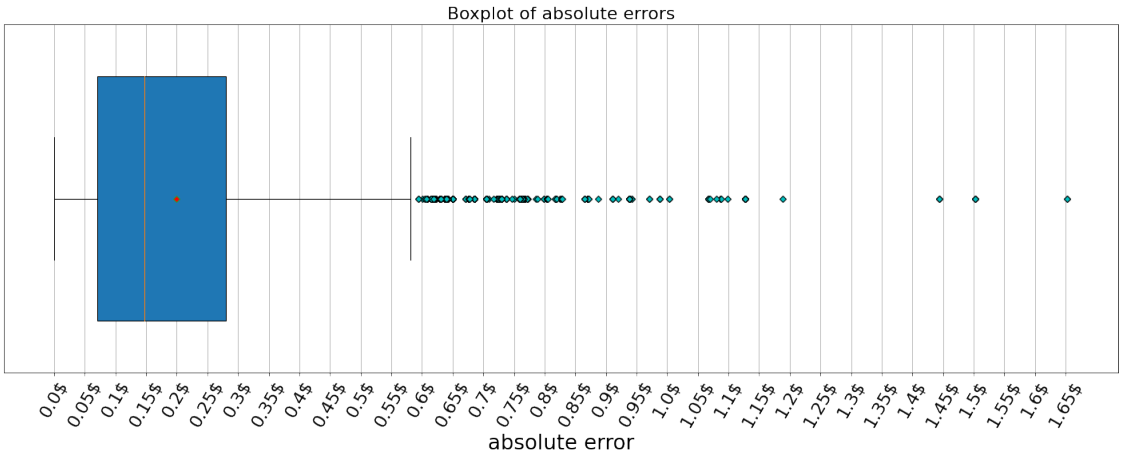


Figure C.3: Ground truth VS Output Ridge regression



(a) Ridge's Absolute Errors distribution.



(b) Ridge's Absolute Errors box-plot.

Figure C.4: Ridge's Absolute Error quantile distribution.

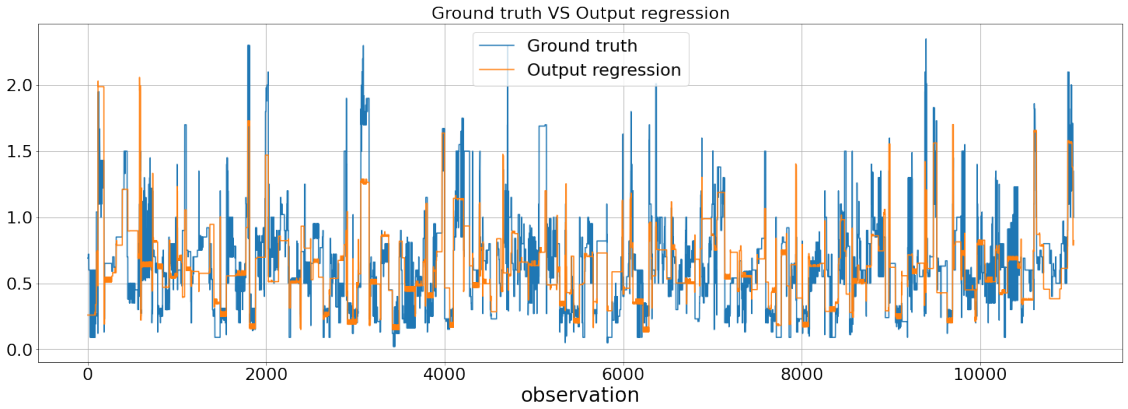


Figure C.5: Ground truth VS Output Lasso regression.

C.1.3 Lasso Regression

The Lasso Regression model requires the value of the hyperparameter λ. In order to find λ^*, the same *grid search* procedure together with the *cross validation* (CV) used for Ridge regression is applied here. First, a list of λ values is identified; then, for each of them, the training set is split in five folds and, in turn, one is used as validation set and the remaining ones as training set. For each λ, the average value of the objective function through the five splits is computed and used as the metric to determine λ^*.

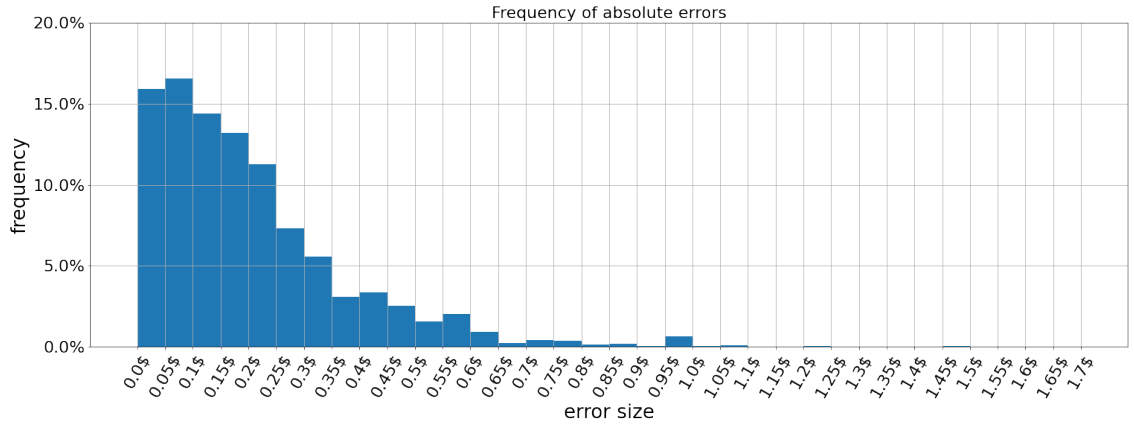
The whole procedure is repeated a second time selecting a finer grid of λ values around the best value found before. The λ^* is equal to 0.0010.

The capacity of the Lasso regression's output to "follow" the true output is graphically inspected in Figure C.5. This allows to verify that the regression is not just returning a dummy value, like the average, for all the observations.

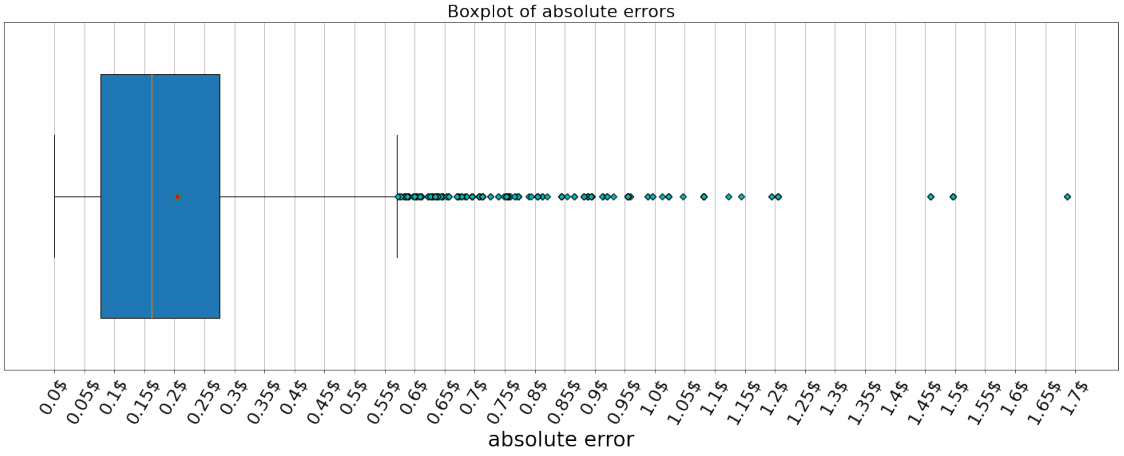
The *average absolute error* is equal to 0.20\$. This is a significant drop with respect to the 0.46\$ achieved using multiple linear regression and equal to the Ridge result.

To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure C.6.

The box-plot shows that 25% of errors are lower than 7 cents and 50% of errors are lower than 16 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 28 cents. There is another approximately 15% of cases where the error is between 29 and 50 cents. Errors above 50 cents are limited to the last 10% of data.



(a) Lasso's Absolute Errors distribution.



(b) Lasso's Absolute Errors box-plot.

Figure C.6: Lasso's Absolute Error quantile distribution.

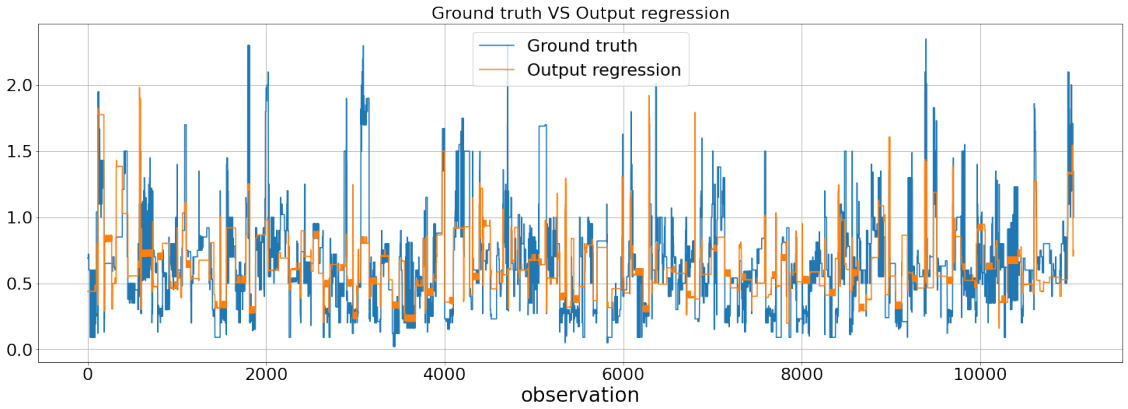


Figure C.7: Ground truth VS Output Regression.

C.2 GloVe Encoding

C.2.1 Multiple Linear Regression

Firstly, the capacity of the multiple linear regression output to "follow" the true output is graphically inspected in Figure C.7. This allows to verify that the regression is not just returning a dummy value, like the average, for all the observations.

The *average absolute error* is equal to $0.22\$$. To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure C.8.

The box-plot shows that 25% of errors are lower than 8 cents and 50% of errors are lower than 17 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 29 cents. There is another approximately 15% of cases where the error is between 30 and 50 cents. Very large errors are limited to the last 10% of data.

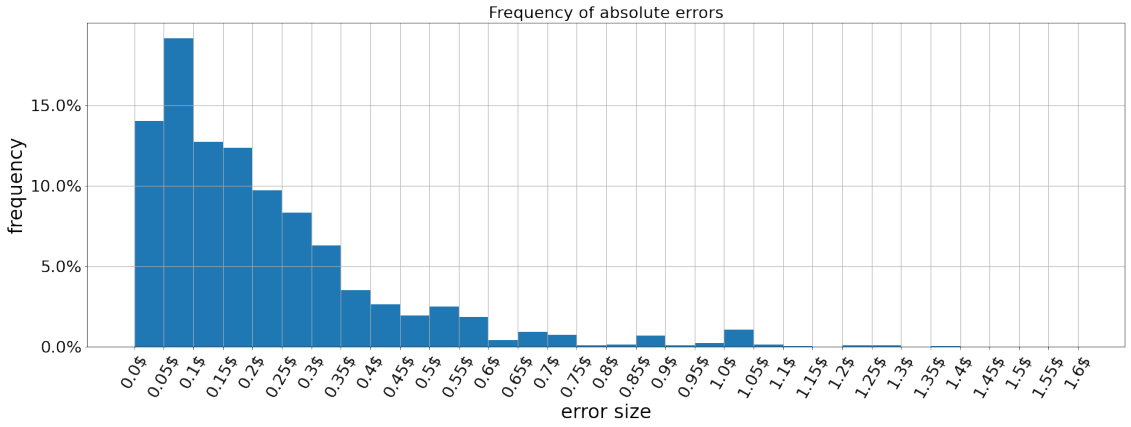
C.2.2 Ridge Regression

The same procedure described in Subsection C.1.2 is followed to find the optimal value of hyperparameter λ. The λ^* is equal to 1400.

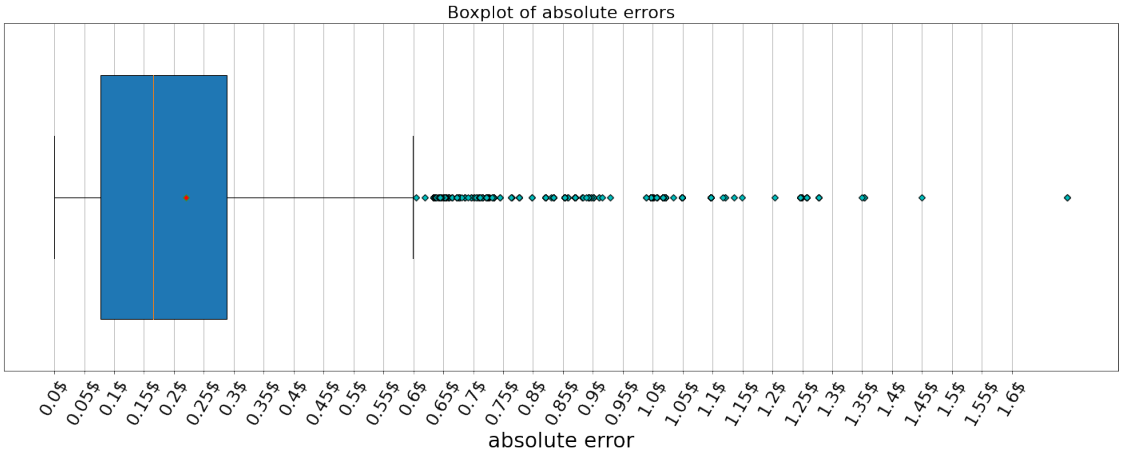
The capacity of the Ridge regression's output to "follow" the true output is graphically inspected in Figure C.9. This allows to verify that the regression is not just returning a dummy value, like the average, for all the observations.

The *average absolute error* is equal to $0.22\$$.

To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the



(a) Regression's Absolute Errors distribution.



(b) Regression's Absolute Errors box-plot.

Figure C.8: Regression's Absolute Error quantile distribution.

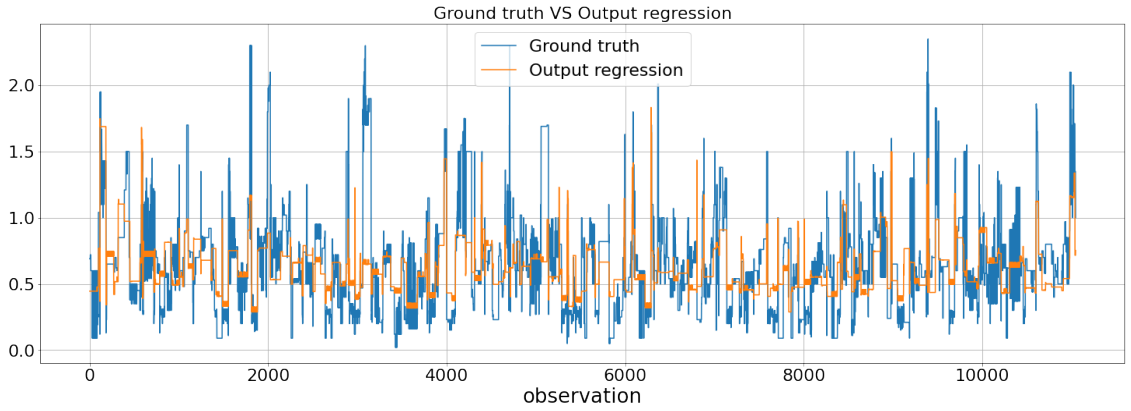
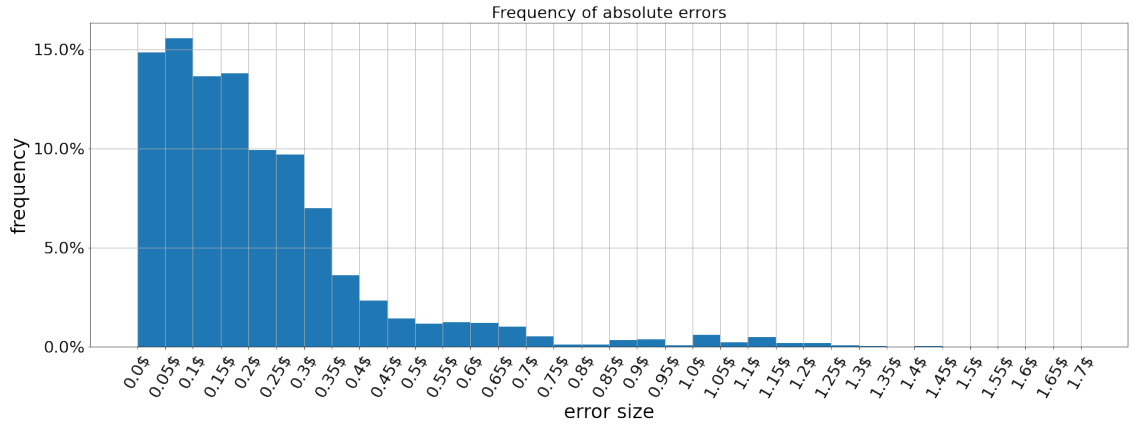
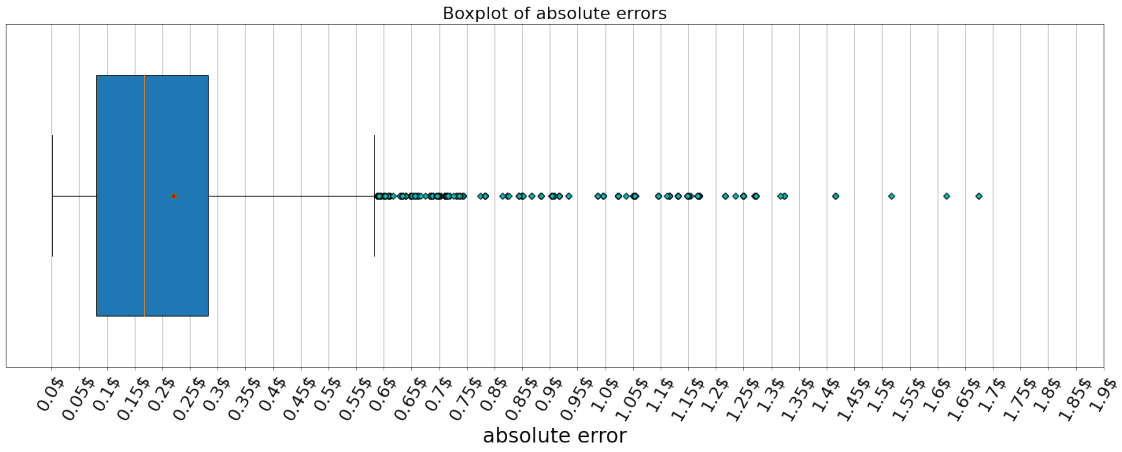


Figure C.9: Ground truth VS Output Ridge regression



(a) Ridge's Absolute Errors distribution.



(b) Ridge's Absolute Errors box-plot.

Figure C.10: Ridge's Absolute Error quantile distribution.

corresponding box-plot are depicted in Figure C.10.

The box-plot shows that 25% of errors are lower than 8 cents and 50% of errors are lower than 17 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 29 cents. There is another approximately 14% of cases where the error is between 30 and 50 cents. Errors above 50 cents are limited to the last 11% of data. Considering the simplicity of the model, results are already good.

C.2.3 Lasso Regression

The same procedure described in Subsection C.1.3 is followed to find the optimal value of hyperparameter λ. The λ^* is equal to 0.0009.

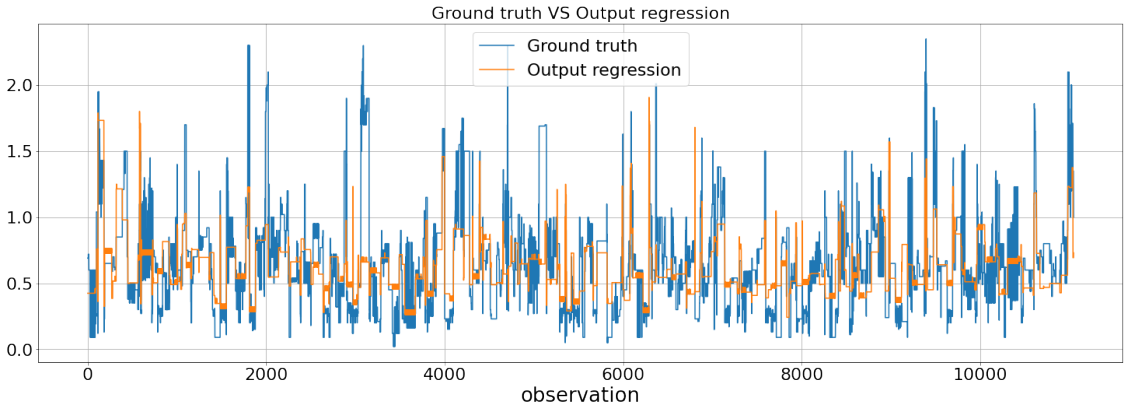


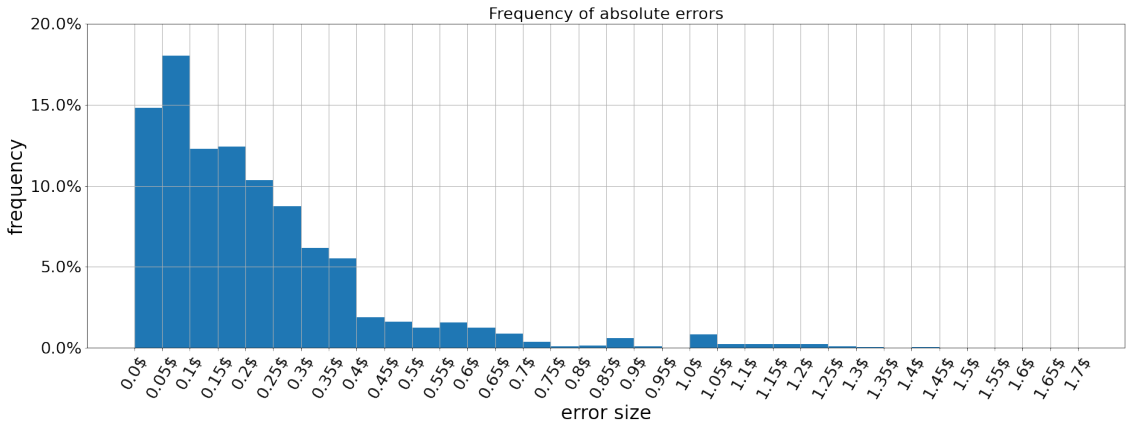
Figure C.11: Ground truth VS Output Lasso regression.

The capacity of the Lasso regression's output to "follow" the true output is graphically inspected in Figure C.11. This allows to verify that the regression is not just returning a dummy value, like the average, for all the observations.

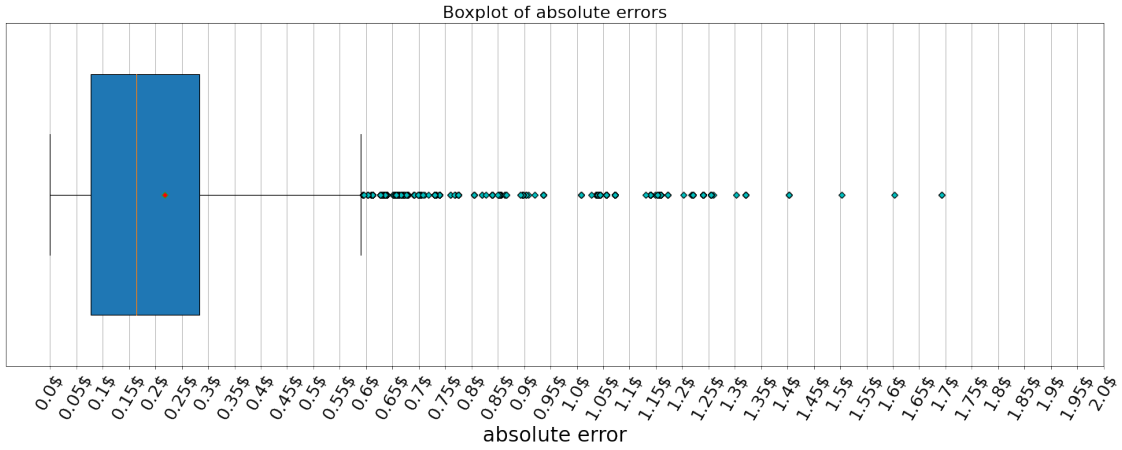
The *average absolute error* is equal to 0.22$.

To get a better feeling for the errors' behavior, the distribution of *absolute errors* and the corresponding box-plot are depicted in Figure C.12.

The box-plot shows that 25% of errors are lower than 8 cents and 50% of errors are lower than 16 cents. The *average absolute error* is also shown as a red diamond. 75% of errors are below 29 cents. There is another approximately 12% of cases where the error is between 30 and 50 cents. Errors above 50 cents are limited to the last 13% of data.



(a) Lasso's Absolute Errors distribution.



(b) Lasso's Absolute Errors box-plot.

Figure C.12: Lasso's Absolute Error quantile distribution.

References

[1] B. Jansen and T. Mullen, "Sponsored search: an overview of the concept, history, and technology," *Int. J. Electronic Business*, vol. 6, no. 2, pp. 114–131, 2008.

[2] B. Kitts and B. Leblanc, "Optimal bidding on keyword auctions," *Electronic Markets*, vol. 14, no. 3, pp. 186–201, 2004.

[3] B. Edelman, M. Ostrovsky, and S. M., "Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords," *The American Economic Review*, vol. 97, no. 1, pp. 242–259, 2007.

[4] M. Cary, A. Das, B. Edelman, I. Giotis, K. Heimerl, A. Karlin, C. Mathieu, and M. Schwarz, "Greedy bidding strategies for keyword auctions," in *Proceedings of the Eighth Annual Conference on Electronic Commerce*, San Diego, California USA, Jun. 2007.

[5] K. Amin, M. Kearns, P. Key, and A. Schwaighofer, "Budget optimization for sponsored search: Censored learning in mdps," in *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, USA, Aug. 2012.

[6] D. Lee, P. Zioło, W. Han, and W. B. Powell, "Optimal online learning in bidding for sponsored search auctions," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, USA, Nov. 2017.

[7] W. Zhang, Y. Zhang, B. Gao, Y. Yu, X. Yuan, and T. Liu, "Joint optimization of bid and budget allocation in sponsored search," in *Proceedings of the 18th SIGKDD International Conference on Knowledge Discovery and Data Mining*, USA, 2012.

[8] W. Shen, B. Peng, H. Liu, M. Zhang, R. Quin, Y. Hong, Z. Guo, Z. Ding, P. Lu, and T. P., "Reinforcement mechanism design: With applications to dynamic pricing in sponsored search auctions," in *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, New York, USA, Feb. 2020.

[9] Wordtracker tool. [Online]. Available: https://www.wordtracker.com/

[10] Google. Google ads. [Online]. Available: https://ads.google.com/home/

[11] A. Joshi and R. Motwani, "Keyword generation for search engine advertising," in *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)*. IEEE, 2006, pp. 490–496.

[12] P. Rusmevichientong and D. P. Williamson, "An adaptive algorithm for selecting profitable keywords for search-based advertising services," in *Proceedings of the 7th ACM Conference on Electronic Commerce*, 2006, pp. 260–269.

[13] V. Abhishek and K. Hosanagar, "Keyword generation for search engine advertising using semantic similarity between terms," in *Proceedings of the ninth international conference on Electronic commerce*, 2007, pp. 89–94.

[14] Y. Chen, G. Xue, and Y. Yu, "Advertising keyword suggestion based on concept hierarchy," in *Proceedings of the 18th International Conference on Web Search and Web Data Mining*, Palo Alto, USA, 2008.

[15] E. Even Dar, Y. Mansour, V. Mirrokni, S. Muthukrishnan, and U. Nadav, "Bid optimization for broad match ad auctions," in *Proceedings of the 18th International World Wide Web Conference*, Madrid, Spain, Apr. 2009.

[16] S. Thomaidou and M. Vazirgiannis, "Multiword keyword recommendation system for online advertising," in *2011 International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 2011, pp. 423–427.

[17] C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," *Signal Processing*, vol. 167, 2020.

[18] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[19] H. Xiao. bert-as-service. [Online]. Available: https://github.com/hanxiao/bert-as-service

[20] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer, 2008.

[21] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[23] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[24] M. Gadaleta and M. Rossi, "Idnet: Smartphone-based gait recognition with convolutional neural networks," *Pattern Recognition*, vol. 74, pp. 25–37, 2018.

[25] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang, "Convolutional neural networks for human activity recognition using mobile sensors," in *6th international conference on mobile computing, applications and services*. IEEE, 2014, pp. 197–205.

[26] N. Y. Hammerla, S. Halloran, and T. Plötz, "Deep, convolutional, and recurrent models for human activity recognition using wearables," *arXiv preprint arXiv:1604.08880*, 2016.

[27] S. Khalil, C. Amrit, T. Koch, and E. Dugundji, "Forecasting public transport ridership: Management of information systems using cnn and lstm architectures," *Procedia Computer Science*, vol. 184, pp. 283–290, 2021.

[28] R. Campos, V. Mangaravite, A. Pasquali, A. Jorge, C. Nunes, and A. Jatowt, "Yake! keyword extraction from single documents using multiple local features," *Information Sciences*, vol. 509, pp. 257–289, 2020.

[29] I. Witten, E. F. Paynter, C. Gutwin, and C. Nevill-Manning. Keyphrase extraction algorithm. [Online]. Available: http://community.nzdl.org/kea/

[30] I. Witten, G. W. Paynter, E. Frank, C. Gutwin, and C. Nevill-Manning, "Kea: Practical automated keyphrase extraction," in *Design and Usability of Digital Libraries: Case Studies in the Asia Pacific*. IGI global, 2005, pp. 129–152.

[31] R. Mihalcea and P. Tarau, "Textrank: Bringing order into text," in *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004, pp. 404–411.

[32] X. Wan and J. Xiao, "Single document keyphrase extraction using neighborhood knowledge." in *AAAI*, vol. 8, 2008, pp. 855–860.

[33] J. K. Sparck, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 5, pp. 111–121, 1972.

[34] S. El-Beltagy and A. Rafea, "Kp-miner: A keyphrase extraction system for english and arabic documents," *Information systems*, vol. 34, no. 1, pp. 132–144, 2009.

[35] M. Berry and J. Kogan, *Text mining: applications and theory*. Springer, 2011.

[36] A. Naskar. Automatic keyword extraction using rake in python. [Online]. Available: https://thinkinfi.com/automatic-keyword-extraction-using-rake-in-python/

[37] F. Von Feilitzsch. python-rake. [Online]. Available: https://github.com/fabianvf/python-rake

[38] R. Campos, V. Mangaravite, A. Pasquali, A. M. Jorge, C. Nunes, and A. Jatowt, "A text feature based automatic keyword extraction method for single documents," in *European conference on information retrieval*. Springer, 2018, pp. 684–691.

[39] F. Leitner. Sentence segmentation and word tokenization. [Online]. Available: https://pypi.org/project/segtok/

[40] A. Pasquali and V. Douzi. Yet another keyword extractor (yake). [Online]. Available: https://pypi.org/project/yake/

[41] ——. Yet another keyword extractor (yake). [Online]. Available: https://github.com/LIAAD/yake

[42] I. Shrivastava. Exploring different keyword extractors - evaluation metrics and strategies. [Online]. Available: https://medium.com/gumgum-tech/exploring-different-keyword-extractors-evaluation-metrics-and-strategies-ef874d336773

Acknowledgments

I would like to thank Prof. Michele Rossi for his role of supervisor in the thesis. Special thanks go to my company tutor, Dr. Marco Boresta, for his time and patience in the supervision of this work and for all his useful suggestions. Thanks also to Dr. Tommaso Colombo and Dr. Alessandro Pinzuti, for the helpful technical advices provided along the way. Moreover, I would like to thank my family who supported and encouraged me during this new academic career.

Giuliano Squarcina
University of Padova
2nd August 2021