Feature Generation through the Exploration of Domain Knowledge

Tiago Afonso
tiago.francisco.a@tecnico.ulisboa.pt
Instituto Superior Técnico - Universidade de Lisboa
Lisboa, Portugal

ABSTRACT

The diversification of areas where data science is present is leading to the need for more qualified scientists. To counteract this, research has shifted towards the automation of this workflow, namely with the development of frameworks for automated machine learning (AutoML). While these frameworks already bring great advancements in some aspects of the pipeline, the data preparation step continues to face great difficulties. This work proposes an algorithm that automates preparation steps and generates features using domain knowledge represented in entity-relationship diagrams, while also defining a set of operators that can be applied to distinct kinds of data. The work is validated with a case study composed of several datasets with ER models, showing improvements in model performance over existing AutoML tools such as auto-sklearn, while also having lower processing times.

KEYWORDS

feature engineering, feature generation, AutoML, domain knowledge, entity-relationship diagrams.

1 INTRODUCTION

Throughout the years, the amount of data that is collected and processed has increased exponentially. Treating data manually has become intractable, which is why development in machine learning (ML) has also increased massively. In this era of Big Data, the more data that can be processed by a system, the better the information that can be retrieved for it and the more robust it can get. Processes for turning raw data into functional knowledge have been defined and refined into what is known nowadays as *Data Science*.

This growth is making data science and machine learning expand in domains, as companies and industries race to use data-driven approaches to find the best insights. This is leading to companies not having enough data scientists that have the necessary amount of experience needed to deal with this amount of data [22]. To counteract this, research is inclining towards the automation of the data science pipeline, to be able to gather valuable insights without the need for human intervention. These AutoML tools can be a solution to the high demand and low supply of data scientists, and are already tackling important parts of the pipeline, such as model selection and hyper-parameter optimization.

However, the success of ML algorithms depends mainly on the quality of the data preparation performed on it. This is a crucial step of the pipeline which occupies 70% to 80% of the time spent on the knowledge discovery (KDD) process. Current AutoML solutions still lack quality in feature engineering and feature generation in particular, with simple or no solutions. This is mostly due to the frameworks staying domain agnostic, employing black-box approaches that work for a variety of data types and datasets, and therefore do not allow for the exploration of available domain knowledge, which can lead to some distrust among the data science community over AutoML methods [25].

Recognizing that harnessing domain knowledge improves the KDD process [1], we argue that representing knowledge and exploring it through automation tools to increase the information that can be extracted from datasets is beneficial. We propose a methodology for automating the data preparation step, including feature generation, by exploring domain knowledge expressed in an entity-relationship (ER) diagram. We also propose DANKFE, an algorithm that automatically generates variables from the diagram. The algorithm receives a diagram and a dataset, whose variables correspond to entities in the diagram and based on each relationship described, it generates a new variable, following the parameters imposed in the diagram. To ease the representation of domain knowledge, we also present a set of possible operations that can be applied with the algorithm.

The work will be validated both in its efficacy and efficiency, using a case study composed of several public datasets where an ER diagram was created for each, representing domain knowledge. The baseline, preprocessed and extended versions of each dataset were evaluated both in time and performance with the best classifiers trained over each version, with a number of ML models. All versions were also compared with a popular AutoML framework, auto-sklearn [7] over the original dataset. The proposed algorithm was also studied in its scalability. Results show an improvement both in performance and in computational cost, due to the generation of useful domain-specific features, compared to methods that do not use any domain knowledge.

This document is structured as follows: in section 2 we review related works and the necessary background. Section 3 states the problem and section 4 presents our solution. The work is then evaluated in section 5 and conclusions are drawn in section 6.

2 BACKGROUND

Knowledge Discovery in Databases (KDD) is the predecessor term to what is nowadays usually called *data science*, and it represents the entire process of extracting and using valuable information from raw data. The process begins with understanding the domain and defining a goal, followed by finding a dataset. The data is then cleaned and preprocessed and manipulated through *feature engineering*. After having the right variables to describe the data, a goal is then matched to the dataset, specifying the task to perform (classification, regression, clustering...), from which results a *model*. The process is then repeated iteratively, until the model is good

enough to be deployed into production, documented and further optimized.

Features are the *variables* describing the data, and in order to ensure that our models are able to achieve strong results, they are transformed, reduced and extended. *Feature selection* techniques remove redundant and irrelevant variables, avoiding an exceeding number of features, which can lead to overfitting and high variance [11]. The data can also be rearranged in some new space [17] through supervised or unsupervised methods, where the variables are combined or transformed from the original space to a new one. This is known as *feature extraction*. Variables can also be added to the original dataset, either by exploring or not domain knowledge. Unlike feature extraction, *feature generation* analyzes relations among features, augmenting the feature space [17].

Feature engineering is usually the most time-consuming step of the KDD process [26], since it requires human interaction and intuition to obtain the best results. This can be subjective, costly and limits the process's repeatability. To counteract this, there have been numerous works on automating feature generation, with or without the use of domain knowledge to improve induction. Without domain knowledge, feature generation can follow a *data-driven* approach, that only uses the input data for guidance, and works by applying operators to features (such as logarithm, exponential or extraction of parts of a value, e.g. year and month from a date), by combining features of the same data types through n-ary operators (such as sum and average), or making aggregations (for example count, max, min) [12]. *Hypothesis-driven* approaches use an induced hypothesis for ranking the new features accordingly. The majority of these methods use Decision Trees [19]. Other methods for feature generation without domain knowledge have been used, namely: hierarchical greedy search [15], neural networks [27], reinforcement learning [14],or genetic programming [18].

Several works have been published throughout the years researching the incorporation of domain knowledge into feature generation. This knowledge does not need to be complete, as it has been proven that fragmentary knowledge can still be applied for narrowing down the feature space [3]. These approaches range from using domain knowledge from experts [19], to the embedding of that knowledge into dedicated algorithms [20] or by exploring external knowledge representation formalisms. Some authors used a graph-based language for feature generation in linked data, by querying the relations inside the data. Those frameworks allow for extracting information from knowledge bases such as YAGO and DBPedia [2, 10]. There are also examples of feature generation regarding textual data [9]. Other approaches use already available knowledge repositories, such as ontologies, finding candidate terms that match the dataset to increase the feature space [8, 24].

The increased interest on automating the KDD process led to a huge increase in AutoML frameworks in recent years, which all have the goal of returning the best approach for a dataset with as little human intervention as possible [13]. In this context, Auto-sklearn [7] uses embeddings, clustering, matrix decomposition and one-hot encoding, as well as meta-features. Auto-Gluon [5] only uses simple data preprocessing techniques, as well as H2O [16]. TPOT [23] uses meta-features and polynomial combinations.

We can see that AutoML still has much room to improve, especially in feature generation based on domain knowledge. The inclusion of this knowledge into AutoML systems could help improve its adoption, seeing that giving data scientists the opportunity to incorporate domain knowledge into their workflow could lead to less skepticism around automated frameworks, as well as enabling them to retrieve more useful information.

3 PROBLEM STATEMENT

As seen before, the desire of exploring domain knowledge across the KDD process is not new, and has been pursued since its origins [1]. However, the different approaches proposed along time did not grasp enough quality to be generally adopted. We can distinguish two main approaches:

- Embedding the domain knowledge in the mining algorithms themselves. While these algorithms are undoubtedly more effective since the knowledge is directly linked with the algorithm, they suffer the downside of not generalizing well, requiring a different algorithm for each problem, making it difficult to adapt to new situations.
- Using general algorithms which explore external knowledge sources. This approach is much easier to generalize for multiple problems, but depends on the availability of knowledge bases and the need for them to be expressive enough to represent the domain expertise.

Since our problem is to incorporate the use of domain knowledge for automatic machine learning, which strives to be as general as possible (through the use of domain agnostic methods), the best approach is the latter, creating an algorithm capable of exploring external knowledge and using it to augment the feature space. Having chosen an approach, we now need to choose a way to represent knowledge. This knowledge will be represented as specific attributes and relationships among the concepts in the domain, which depending on their properties, can lead to the creation of new variables.

To our knowledge, ontologies are the most expressive of those formalisms, but their definition for each domain requires significant efforts as there needs to be a clear definition of how the axioms inside are matched to the dataset. Also, the lack of availability of semantically rich datasets means that there are not many ontologies ready to use. While this availability continues to be a mirage, a more available alternative is to make use of databases instead of knowledge bases.

Databases are indeed the most usual data sources, and they are usually designed through ER diagrams, formalized as relational schema *a posteriori*. ER diagrams have three main elements: rectangles used to define concepts, named *entities*, ellipses for *attributes* and diamonds for *relationships* among concepts. Since these diagrams represent the majority of the elements expressible through ontologies, we may say they are simplifications of those formalisms. As a matter of fact, ER diagrams are just not expressive enough to represent axioms. Nevertheless, they are frequently used in database design which guarantees their availability for a large number of situations, with plenty of experts which are able to design them.

Having studied the best approach for the algorithm and the method for representing the knowledge, we can now present the problem statement in this new context:

Given a dataset and a corresponding ER diagram, create a new dataset by transforming and extending the original one, through the exploration of the knowledge expressed in the diagram.

Before proceeding with the algorithm analysis, we need to specify how the ER diagram and the dataset are related to each other. Since a dataset is solely described by a set of variables, it needs to be able to represent those variables. The problem arises in choosing how to represent them, which can be done either through the entities themselves or through attributes that characterize the entities. This is known as the *reification problem* [21], and since we want to be able to manipulate the existing variables to create new ones, we choose to represent every variable as an entity, in order to reason and talk about them. In this manner, the ER diagrams have to represent all existing variables as entities, and since new variables result from the combination of existing ones, they have to be represented through relationships.

We are now ready to define an ER diagram in our context:

Definition 3.1. An ER diagram is a tuple $\mathcal{KB}$=($\mathcal{E}$, $\mathcal{R}$), where $\mathcal{E}$ is the set of `entities` and $\mathcal{R}$ is the set of `relationships` among the entities in $\mathcal{E}$.

Moreover,

Definition 3.2. Given an ER diagram, $\mathcal{KB} = (\mathcal{E}, \mathcal{R})$ as defined before, and a dataset $\mathcal{D}$ described by a set of d variables, $\mathcal{F} = \{v_1, ..., v_d\}$: $\forall v \in \mathcal{F}\ \exists e \in \mathcal{E}$: e corresponds to v.

In order to explore such diagrams, we translate them into JSON files, following a predefined structure. JSON is a standard text-based format for storing and transmitting structured data, used in plenty of web applications. Other formats could be used, including XML, RDFS and OWL, to name a few. Independently of the choice, the specification of the ER elements must follow a strict definition.

Each entity in an ER diagram is characterised by its *name*, used as an identifier, its *type* to help on determining the possible operations to perform over it and a *description* optionally used to clarify any additional information about the entity.

As relationships specify the new variables to generate, they have a more extensive definition. Each *relationship* in an ER diagram is characterised by its *name* again used as an identifier, but now also used for naming the new variable to generate, *inputs* for specifying the list of entities that make up the relationship, *operations* corresponding to the sequence of operations to perform over its inputs to generate the new variable and the *constraints* its inputs have to satisfy to make the generation possible. Optionally, it may include a *groupby* parameter specifying the variable along with an aggregation may be made and *condition* for specifying which records to aggregate.

3.1 Operations

With this schema, since the operation or operations used to generate a new variable are specified inside the diagram, the complication of understanding each operation arrives. In order to be possible for the algorithm to understand and generate variables with all imaginable operations, a decoder would be needed inside the algorithm, that would ideally translate the operation specified into a function. Unfortunately, this would add enormous processing time and complexity to the solution. To mitigate this, we propose a set of possible operations types that can be used to generate new variables. These types work as "relationship templates" that the algorithm is able to interpret and calculate, therefore creating new variables for the dataset.

These types of operations span a large range of possible operations, which can be specified in the ER diagram. We propose the following types:

- Decomposition operations: any operation over a single record, described by a single variable, that extracts some component from its value. Examples of these operations are the decomposition of a date into its components (*year*, *month*, *day*), decomposition of strings that follow certain patterns (*firstname*, *surname*), among others.
- Algebraic operations: any mathematical operation over a single record, described by one or more variables. Examples of such operations for a single variable are *absolute, square root*, *division, logarithm* for two variables, and *sum, product* for any number of variables.
- Mapping operations: any operation over a single record, that maps the value in one variable to another value, possibly from different types. Examples of these operations are mapping if a date is a *holiday* or which *weekday* it is. Comparing the value of a variable against some threshold or if it is equal/different from another value can also be considered a mapping operation.
- Aggregation operations: any operation to be applied over a set of records. Examples of these operations are *sum, average, max, stdev* applied over a set of records, that satisfy some imposed condition similar to the ones achieved with a *GROUPBY* clause in an *SQL* query.
- Composition operations: a sequence of operations to be applied one after the other, as a mathematical composition of functions. This allows for multiple different operations to be applied for the generation of a variable. An example is extracting the *nr_months* that have passed from two dates, first by subtracting the two dates, which may return the number of days between them, and taking that result, converting it into the number of months.

From these operation types, two different procedures can be distinguished - the generation of variables with or without aggregations, as specified by the *GROUPBY* parameter. We call *aggregation-based* generation the first one, and *record-based* generation the second approach. It is useful to differentiate both approaches since decomposition, algebraic and mapping operations do not require information about any other record besides the one where the operation is being applied, while aggregation operations require information about other records in which the aggregation is to be made.

Both procedures can be defined using the previous logic. Considering $\mathcal{D}$ to be a dataset, $\mathcal{F} = \{v_1, ..., v_d\}$ the set of d variables describing $\mathcal{D}$ and $\mathcal{KB}$=($\mathcal{E}$, $\mathcal{R}$) an ER diagram, as defined before.

Definition 3.3. Let $r = (\Theta, \Pi, \Psi, \emptyset, null)$ be a `record-based` relationship in $\mathcal{R}$, with $\Theta \subset \mathcal{E}$ the set of `input` variables, Π the sequence of `operations`, and Ψ the set of `constraints` to satisfy. The procedure generates a new variable v_r, and each record x =

$x_1 ... x_d$ in the dataset $\mathcal{D}$ becomes $x' = x_1 ... x_d, x_r$, with x_r filled as follows:

(1) if $\exists\theta \in \Theta\ \exists\psi \in \Psi$: $x_\theta \not\models \psi$, a *null* value is assigned to x_r;
(2) otherwise,
   (a) x_r becomes $\pi(x_{\theta_1} ... x_{\theta_k})$, where π is the last operation in Π and $(x_{\theta_1} ... x_{\theta_k})$ is the projection of x along each variable $\theta_i \in \Theta$;
   (b) if $|\Pi| > 1$ then x_r becomes $\pi_i(x_r)$ with π_i being each one of the i^th with $0 < i < j$, and j the number of operations in Π, from the $(j-1)^th$ to the first one.

It is also possible to define *aggregation-based* procedures in a similar way:

Definition 3.4. Let $r = (\Theta, \Pi, \Psi, \Delta, \phi)$ be a `aggregation-based` relationship in $\mathcal{R}$, with $\Theta \subset \mathcal{E}$ the set of `input` variables, Π the sequence of `operations`, Ψ the set of `constraints` to satisfy, Δ the set of variables to specify the `aggregation` and ϕ the `condition` to constraint the aggregation.
The procedure generates a new variable v_r, and each record $x = x_1 ... x_d$ in the dataset $\mathcal{D}$ becomes $x' = x_1 ... x_d, x_r$, with x_r filled as follows:

(1) if $\exists\theta \in \Theta\ \exists\psi \in \Psi$: $x_\theta \not\models \psi$, a *null* value is assigned to x_r;
(2) otherwise,
   (a) a temporary variable γ is created for storing the projections of each x along each variable $\theta_i \in \Theta$ $(x_{\theta_1} ... x_{\theta_k})$ for all records in $\mathcal{D}'$ satisfying the condition ϕ, and aggregated according to all variables $\delta \in \Delta$;
   (b) then x_r becomes $\pi(\gamma)$, where π is the last operation in Π;
   (c) if $|\Pi| > 1$ then x_r becomes $\pi_i(x_r)$ with π_i being each one of the i^th with $0 < i < j$, and j the number of operations in Π, from the $(j-1)^th$ to the first one.

In this way, variables that require aggregation operations can be generated via an *aggregation-based* procedure, and all other specified operations can be generated via a *record-based* procedure.

3.2 Illustration

In order to better understand the algorithm proposed, we can consider as example the knowledge base represented by the ER diagram represented in fig. 1.

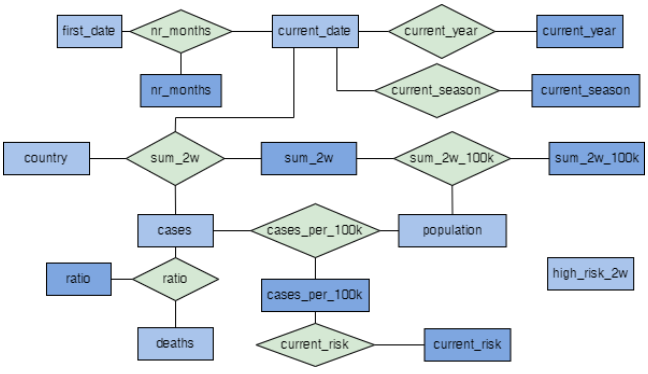

Figure 1: Example of the ER diagram for feature generation.

current_date	cases	deaths	country	population	first_date	high_risk_2w
2021/02/23	1032	63	PT	10295909	2020/03/03	TRUE
2022/02/14	20360	78	UK	10718565	2020/02/23	TRUE
2021/08/12	223	2	PL	37958138	2020/03/07	FALSE
2020/06/11	22	0	AT	8901064	2020/02/26	FALSE

Table 1: Illustration dataset, labeled by *high_risk_2w*

current_date	year	season	nr_months	ratio	cases_100k	current_risk	sum_2w	sum_2w_100k
2021/02/23	2021	winter	11	16.4	10.023	FALSE	33692	327.237
2022/02/14	2022	winter	24	261.0	189.951	TRUE	284573	2654.954
2021/08/12	2021	summer	17	111.5	0.587	FALSE	2453	6.462
2020/06/11	2020	spring	3	null	0.247	FALSE	471	5.292

Table 2: Generated variables, indexed by current_date.

Additionally, we also consider the data on table 1, which corresponds as the input dataset $\mathcal{D}$ to the algorithm, which is described by the set of variables $\mathcal{F}$={*current_date*, *cases*, *deaths*, *country*, *population*, *first_date*, *high_risk_2w*}.

We can see that all variables in $\mathcal{F}$ are represented in the ER diagram as entities (light blue rectangles). Besides the entities that map to each variable, there are also eight relationships (green diamonds) and eight additional entities (dark blue rectangles), which correspond to the variables that will be generated by the algorithm. Each relationship is linked to a set of entities, where the lighter ones correspond to the inputs, and the darker ones to the output (the variable that will be generated).

Table 2 summarizes the variables generated by our algorithm when applied to the data in $\mathcal{D}$, shown in the previous table, and using the ER diagram in fig. 1. First, we find *year* resulting from a `decomposition` operation, computed by extracting the year from *current_date*. Similarly, we have *season* that `maps` the *current_date* to the yearly season. `Algebraic` operations are illustrated through *ratio* (*deaths* divided by *cases*), *nr_months*, (difference in months between *current_date* and *first_date*) and *cases_100k* (number of *cases* divided by the *population*, multiplied by 100k). All of these variables only use *record-based* operations, since the new variables only depend on values from variables of that same record.

On the other side, *sum_2w* is an example of a variable resulting from an `aggregation` operation, resulting from summing the number of *cases* from the last two weeks for the country under analysis (the country for the record whose value is being filled).

Finally, *sum_2w_100k* and *current_risk* could be seen as `composition` if we had omitted the *sum_2w* and *cases_per_100k*, respectively. Actually, they are just a division by the population and a comparison to a threshold (120 cases per 100k), after computing those previous variables.

Now that the problem and a black-box view of the algorithm has been explained and illustrated, we can now explain how the proposed algorithm itself works and the benefits and trade-offs of each iteration.

4 DANKFE ALGORITHM

The DANKFE (DomAiN Knowledge based Feature Engineering) algorithm transforms the relationships between entities into new variables, when presented with an ER diagram and a dataset. The algorithm is presented in various versions that trade-off speed with versatility of operations that it the version is able to deal with.

4.1 DANKFE-I

The first version of the algorithm, DANKFE-I, is described in algorithm 1 and works as follows: the relationships are read from the ER model, and stored as a queue to be processed. The relationships are processed one by one, if the input variables for them are already available. If part of the input is not yet available (meaning that at least one of the input variables is not originally in the dataset and still in the queue to be processed), that relationship is sent to the end of the queue. If all the inputs are already available (have all been generated or are originally present in the input dataset), the list of operations specified in the diagram for that relationship is applied to any row in the dataset that satisfies the constraints imposed for the relationship. Whenever any row does not meet the constraints, a null value is imputed. When all rows are processed, the relationship is removed from the queue. The algorithm ends when the processing queue is empty.

Algorithm 1 DANKFE-I algorithm

```
procedure DANKFE-I(𝒟, ℱ, 𝒦ℬ)
    queue ← 𝒦ℬ['relations']
    while queue is not empty do
        current_relation ← pop(queue)
        inputs ← current_relation['inputs']
        constraint ← get_constraint(current_relation['constraint'])
        operations ← reverse(current_relation['operations'])
        if inputs ∈ ℱ then
            args ← 𝒟[inputs]
            for operation in operations do
                for row in args do
                    if satisfies(row, constraint) then
                        row ← operation(args)
                    else
                        row ← null
                    end if
                end for
            end for
        else
            queue ← append(queue)
        end if
    end while
end procedure
```

This version of the algorithm abides by definition 3.3, meaning it can perform *record-based* operations, since it processes the dataset one record at a time. If a relationship has multiple operations to be performed (composition operation), the algorithm applies each operation in the list of operations in reverse (similarly to a composition of operations) sequentially over the corresponding inputs (the values of the input variables defined in the relationship), returning the output value (assigned to the given record), processing the defined operations row by row.

The operations defined in section 3.1 are defined similarly to a Production Rule System (PRS), an if-then system which interprets the operation needed to generate a new value for a relationship, triggering the necessary action with the values of the inputs given to the algorithm from that relationship. For example, if the algorithm is processing a relationship found in the ER diagram that wants to create the *year* variable, where the input is *current_date* and the output is *year*, the algorithm's PRS senses if the operation defined in the relationship is the operation used to extract the year from the input from the list of operations, triggering the action that completes that calculation and returns only the year from the date, which is then written as the new value for the record being processed for the new variable. This process is then repeated for all rows that pass the possible constraint defined in the relationship, and in the end, the *year* variable is complete and part of the dataset.

Since there is no row dependence in *record-based* operations, meaning that these operations only require values of the row being processed by the algorithm, it can be done very efficiently using the Pandas function `apply` and using lambda functions in Python.

4.2 DANKFE-II

The second iteration of DANKFE extends the previous version by allowing the user to define *aggregation-based* operations, as described in definition 3.4. The algorithm is defined in algorithm 2.

Algorithm 2 DANKFE-II algorithm

```
procedure DANKFE-II(𝒟, ℱ, 𝒦ℬ)
    queue ← 𝒦ℬ.relations
    while queue is not empty do
        rel ← pop(queue)
        if rel.inputs ∉ ℱ then
            queue ← append(rel)
        else
            for row in 𝒟 do
                if satisfies(row, constraint) then
                    if rel.groupby exists then
                        row' ← get_rows(row, 𝒟[rel.inputs], rel.groupby, rel.condition)
                    else
                        row' ← row
                    end if
                    for operation in reverse(rel.operations) do
                        row' ← operation(row')
                    end for
                else
                    row' ← null
                end if
                𝒟' ← 𝒟' ∪ {row + row'}
            end for
        end if
    end while
    return 𝒟'
end procedure
```

DANKFE-II works in a similar fashion to DANKFE-I when generating features that only require record-based operations. If the relationship requires an aggregation operation (specifies a *groupby* parameter, then the rows that match the specified condition to perform the aggregation need to be collected. Only after this collection it is then possible to apply the list of operations which create the new values for that variable, for all the records that specify the

imposed constraints. The operations are applied as a composition of functions, beginning with the last one and sequentially applying the following ones. Whenever any row does not meet the constraints, a null value is imputed. When all rows are processed, the relationship is removed from the queue.

The algorithm's PRS interprets the operation needed to generate the value, but for aggregation-based operations it also has access to the collected rows, triggering the calculation and outputting the calculated value to new variable of that record. For example, if the algorithm is processing *avg_temperature_week_per_city*, with *current_date* and *temperature* as inputs and *city* as *groupby*, the algorithm collects the rows with a similar city to that record, but only the ones within one week. It checks that the operation is *average* and after the calculation, the value is written to the new variable in that record. The process is then repeated for all records that pass the possible constraint, until the variable *avg_temperature_week_per_city* is complete.

As stated before, this version of the algorithm is an extension of DANKFE-I, meaning it performs *record-based* operations in a similar fashion, using lambda functions in Python. This version of the DANKFE algorithm sacrifices some processing speed (since the process of collecting rows for generating variables that require *aggregation-based* operations takes longer time to run), but it adds the ability of performing these kinds of operations, which can greatly benefit the datasets depending on the available domain knowledge.

4.3 DANKFE-III

As seen before, feature engineering (and subsequently, feature generation), is only a part of a pipeline of operations that turn raw data into possibly important information, known as the KDD or data science process. AutoML frameworks that automate this pipeline are currently not spending much time or resources into augmenting datasets, even less with the use of domain knowledge, but rather spending more computation in other parts such as data preprocessing, model selection and hyper-parameter optimization.

Similarly to how these frameworks work, to be able to reach the most robust models, other parts of the KDD process can be coupled with the DANKFE algorithm to yield the best results. The algorithm can be encapsulated into a new function that deals with data preparation before and after generating the features, before running ML models.

Additionally, not all variables require domain knowledge to be generated. Simple record-based operations such as decomposition of dates or aggregation-based operations such as a descriptive statistic of a numeric variable (mean, median, maximum, minimum, etc) can be easily generated by checking in which existing variables we are able to apply these operations (easily accomplished by checking their *type* in the ER model) and by doing so, adding these new relationships to the ER model, thus creating new variables. For this to be possible, there needs to be a template where these relationships are described, with the necessary operations for creating this variables, leaving only the *input*, *output* and *groupby* parameters empty. An example of a variable template is in fig. 2 (left). This template relationship is similar to the relationships found in the ER models, but with those three parameters missing. The user can choose which automatic variables are to be created, and these will be added to the ER model with the empty parameters filled in.

The created pipeline can be seen in fig. 3. We can see that data preprocessing techniques can be applied before the generation of features with operations such as missing value imputation, dummification/discretization, label encoding, etc. Afterwards, automatic variables can be added to the ER model if required by the user. Then, the DANKFE algorithm runs (either the first or second version depending on whether the ER model has relationships that create variables which require *aggregation-based* operations), and finally additional processing can be applied to the data, such as scaling or balancing of variables, if chosen by the user.

To implement this, a configuration JSON file is required at the beginning, as well as the dataset and ER model, where the preparation techniques can be programmed per user choice. The configuration file can be seen in fig. 2 (right). The user can specify whether the system checks missing values, scaling, balancing and if automatic features such as decomposition of dates and aggregated summary of numeric variables (max, min, average, standard deviation and median) can be added (given a *groupby* variable). The configuration file changes the parameters in the template relationships and introduces these new relationships inside the ER model, being added to the domain knowledge, so that DANKFE can generate them. From the example in fig. 2 (right), the input dataset would be cleaned and checked for missing values, dates would be decomposed automatically and a total of 10 aggregation-based variables would be added (5 for the summary of *cases* per *country* and another 5 for the summary of *deaths* per *country*). Scaling and balancing would be performed after the features are generated.

```
"dates":
[{
  "name": "day",
  "type": "int",
  "operations": ["getDay"],
  "inputs": [],
  "output": "day",
  "groupby": "",
  "needsRows" : 0,
  "constraint": ""
},
```

```
{
 "checkMissingValues" : "auto",
 "checkScaling" : "zscore",
 "checkBalancing" : true,
 "generateDates" : true,
 "generateFiveSummary" : ["cases","deaths"],
 "groupby": ["country"]
}
```

Figure 2: Example of automatic variable template (left) and configuration file (right).

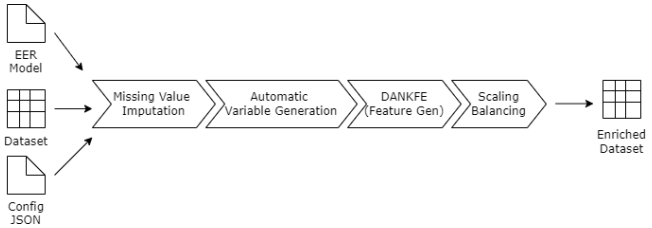

Figure 3: Data preparation and feature generation pipeline.

The configuration file allows the user to select some techniques for each data preparation step, depending on their suitability for the task. Missing values can be imputed using the median for numeric variables and the mode for symbolic variables. A label encoder can also be used to turn symbolic variables into ordinal ones. Scaling can be done either by *z-score* or *minmax*, and balancing is done via an hybrid approach that depends on the proportion of the data,

where it tries to balance the records of each class to 25000. This allows for the models to continue to reach strong results with a lower number of records in a more distributed fashion, which helps in some ML models.

While the DANKFE-III system does not change how the features are generated, it helps automate the entire data preparation step, which can be very beneficial for the later data mining stage. This way, some features can still be generated even with very little domain knowledge (only knowing which variables are dates or numerical), which still augments the feature space and possibly improve the amount of information that ML models can extract. It also facilitates data cleaning and preprocessing, ensuring that the enhanced datasets are ready for data mining.

5 CASE STUDY

In order to validate our proposal, we compared the quality of classification models trained over a case study composed of various datasets, both in their baseline and extended versions with the generated features from each algorithm. The datasets spread over several domains, and an ER diagram and target variable were created for each one. Covid-based datasets were explored through the ER diagram on fig. 1, with the others following similar reasoning. Besides performance, we studied the time spent training and predicting models as well as the time spent generating the variables, and how important they were for the models. Additionally, the scalability of the DANKFE algorithm was also studied, as well a comparison of all results with running the same case study through auto-sklearn [7], one of the most popular AutoML frameworks.

Since most datasets were unbalanced (except for DANKFE-III where preprocessing techniques were applied), AUC [6] was used as the leading metric. Since the goal of the solution is to improve AutoML frameworks with the use of domain knowledge and feature generation, the behavior of model evaluation was made similar to the behavior of AutoML frameworks. Several training techniques (Naive Bayes, KNN, Decision Trees, Random Forests and Gradient Boosting) were used to find the 10 best models trained over an equal number of random data partitions, to minimize variance.

For each ML model, the hyper-parameters were tuned using Grid Search, which explores every defined combination returning the best result. Since this process can be very time-consuming, if a model achieved a strong enough result with a combination of hyper-parameters (over 98% accuracy and 90% F1-Score), then it would return that specific combination. The 10 optimized models for each dataset were then averaged.

5.1 DANKFE-I Results

The DANKFE-I algorithm cannot exploit row dependencies, therefore it only works with record-based operations. The results obtained from the extended datasets were compared to the baseline where no features were generated, and no preprocessing was done on either version.

Results in fig. 4 (left) show the average AUC for all datasets, for each ML model. We can see that feature generation guided by domain knowledge improved the performance over the baseline. The models that have benefited the most are KNN and Gradient Boosting, with an increase in AUC in almost 5 percentage points

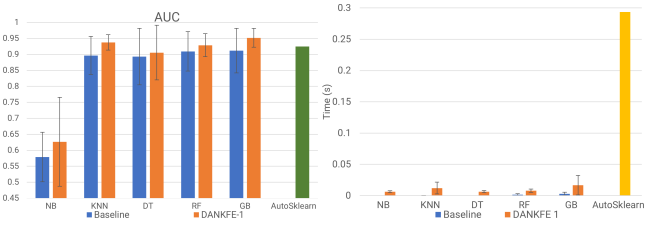

Figure 4: Quality of models (left) and processing times (right) for different machine learning algorithms.

(pp). It is also interesting to see that auto-sklearn achieves a result very similar to the original dataset without feature generation, and KNN and Gradient Boosting achieve in average results above this framework, by generating variables that require record-based operations.

Fig. 4 (right) shows the average time per record spent by each model, including the time spent running DANKFE-I for extending the dataset. We can see that the time spent in feature generation and model training has a small increase, which stays somewhat constant independently of the model used, at around 10 to 20 milliseconds per record, which is much lower than auto-sklearn, which takes the fixed time of one hour or around 300 milliseconds per record.

The efficacy of the DANKFE algorithm can also be evaluated by measuring the impact that the generated features had on the models. Fig. 5 shows the average feature importance for Decision Trees, Random Forests and Gradient Boosting. The generated features (blue) have made the most impact in COVID-based datasets, making up almost all the importance given by the algorithms, and they also benefited the other datasets, especially in Random Forests.

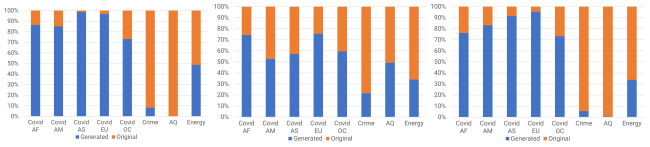

Figure 5: Average feature importance for original and generated variables for Decision Trees (left), Random Forests (middle) and Gradient Boosting (right).

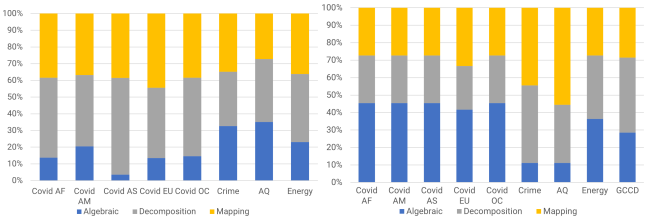

Figure 6: Time comparison (left) and amount of generated features (right) per type of operation.

The type of variable generated also influences the time spent, as seen on 6 (left). Since all record-based operations are performed using the *apply* function in the *Pandas* library, and they do not require any additional rows, their time is somewhat similar, only depending on the amount of operations performed per dataset. The

distribution of operations per dataset can be seen on fig. 6 (right). Decomposition and mapping operations take similar time to run, with algebraic operations either taking less or more time than the other two depending on the dataset.

5.2 DANKFE-II Results

The DANKFE-II algorithm, which introduces aggregation-based operations, was evaluated in the same manner as the previous version. Since it also does not feature any data preprocessing techniques on the dataset, the baseline was also not preprocessed.

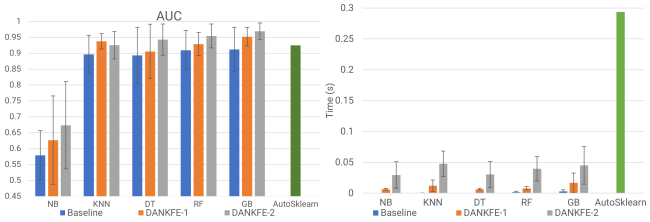

Figure 7: Quality of models (left) and processing times (right) for different machine learning algorithms.

As seen in fig. 7, the addition of aggregation-based operations leads to an overall increase in results for all algorithms except KNN. Naive Bayes and Decision Trees benefited the most, improving 5 pp. On average, model results with DANKFE-II surpass auto-sklearn, with a maximum difference of 4.6 pp. for Gradient Boosting.

In terms of time spent, features that require aggregation-based operations can take much longer to generate compared to ones that do not need them, due to the fact that they require rows to be collected and aggregated, which can take a longer time to run. Nevertheless, the time spent per record still remains somewhat constant independently of the algorithm used, taking approximately 15% of the time spent by auto-sklearn for the entire process (generation + model selection and optimization).

Aggregation-based operations have also caused a big impact in feature importance for the machine learning models. Fig. 8 shows that apart from the Crime dataset, the generated features were given the most importance by far, in Decision Trees, Random Forests and Gradient Boosting. The largest difference compared to the previous version of the algorithm can be seen in the AQ dataset, where aggregation-based operations became the most important for the models.

As stated before, aggregation-based operations take longer to run, depending on the amount of records needed for aggregation before the value for the new variable is calculated. Fig. 9 (left) confirms this. Even though only the AQ and GCCD datasets have the majority of generated features needing aggregation-based operations, as seen on fig. 9 (right), this type of operation is the one that takes longer to run.

5.3 DANKFE-III Results

DANKFE-III couples the feature generation algorithm inside a data preparation pipeline, which ensures the datasets not only benefit from being enriched with the use of domain knowledge, but also improve on possible pitfalls that can appear while data mining, such

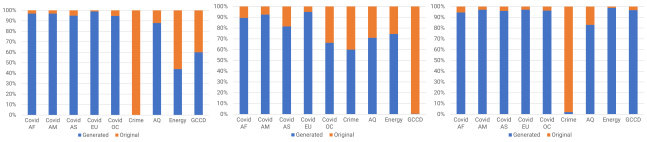

Figure 8: Average feature importance for original and generated variables for Decision Trees (left), Random Forests (middle) and Gradient Boosting (right).

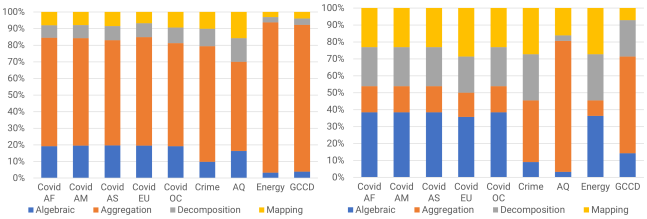

Figure 9: Time comparison (left) and amount of generated features (right) per type of operation.

as being unbalanced, having data with different scales, having missing values which require imputation, nominal data which requires encoding, etc. Some of these issues can decrease the performance of ML models, while others can even prevent them from running and need immediate fixing.

For fairer comparison, the baseline datasets were also preprocessed, using the same scaling and balancing techniques in order to study the impact of the feature generation algorithm alone.

Fig. 10 shows that adding data preprocessing techniques can greatly increase the performance of models, as seen in the preprocessed baseline (`Base + Prep`), which depending on the model can have better AUC than the results from running DANKFE-II. However, adding these techniques to DANKFE-II also results in more robust models, with higher scores than the preprocessed baseline for all models, especially in Naive Bayes with an increase of over 10 pp. Compared to auto-sklearn, DANKFE-III achieves a higher score in all models expect Naive Bayes, with the highest score being Gradient Boosting with an increase of 5.3 pp.

In terms of time spent per record, DANKFE-III has a small increase compared to the previous iteration. This is due to the larger amount of variables generated by automatic operations, such as decomposition of dates and summaries of numeric variables, as well as preprocessing techniques. Fig. 11 shows that these values stay constant independently of the model used, at around 50 milliseconds per record, which is a substantial decrease from the 300 milliseconds by auto-sklearn, while also achieving stronger results.

Similarly to DANKFE-II, all models give larger importance to generated features (blue) instead of the original ones (orange), except for Decision Trees and Gradient Boosting in the GCCD dataset and Random Forests in the AQ dataset, which can be seen in fig. 12.

Compared to the previous version, all datasets have more aggregation-based variables due to the automatic generation of the summary for numeric variables, which can be seen on fig. 13 (right). The larger amount of these variables contributes to a larger time spent in generating them, which can be seen in 13 (left).

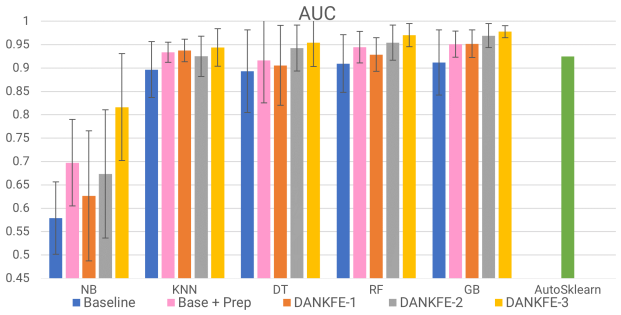

Figure 10: Quality of models (left) and processing times (right) for different machine learning algorithms.

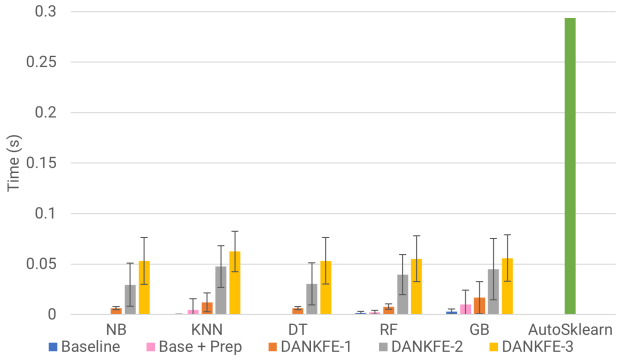

Figure 11: Quality of models (left) and processing times (right) for different machine learning algorithms.

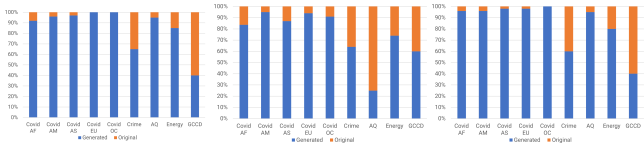

Figure 12: Average feature importance for original and generated variables for Decision Trees (left), Random Forests (middle) and Gradient Boosting (right).

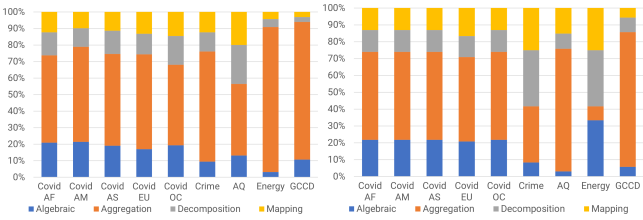

Figure 13: Time comparison (left) and amount of generated features (right) per type of operation.

Finally, to test how the DANKFE algorithm works in terms of scalability, different samples of a larger GCCD dataset (containing 135k records) were taken with different sizes, generating the same set of variables for each one.

Fig. 14 (left) shows the total amount of time spent on feature generation when varying the dataset size, including reading the original dataset and writing the extended version. It is clear that the algorithm presents a linear growth in time depending on the number of records. This behavior can be further detailed in fig. 14 (right), where we can see that compared to aggregation-based operations, record-based ones (algebraic, decomposition and mapping) take residual time. This evidences the trade-off between the different iterations of DANKFE, where processing speed (and therefore time to put a model into production, as per the KDD process) is sacrificed to give room for more functionality and therefore models with better performance, using potentially interesting variables that require aggregation-based operations.

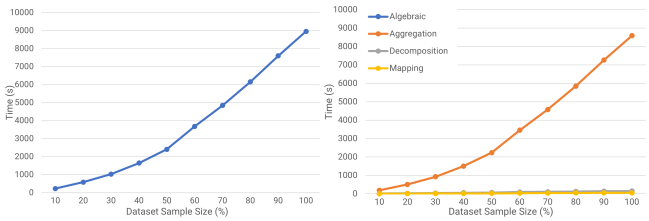

Figure 14: Scalability study: total time on variable generation (left) per types of variables generated (right).

6 CONCLUSION

In this era of Big Data, where the amount of data that needs to be processed on a daily basis is becoming intractable for the amount of data scientists that are qualified for a specific domain, the paradigm of data science is being shifted, where the focus is not on creating the best possible model for a specific problem, but rather to make machine learning more available to the general public, while keeping a good performance, in order to still make the most of the benefits that come from applying the KDD process, to the large amounts of data that companies require nowadays.

This paradigm is leading to the rapid development of AutoML frameworks, with black-box domain agnostic approaches that achieve good results without the need for expert data scientists (which can lead to some mistrust in such frameworks), but do not take any advantage of domain knowledge in the data preparation step, which is known to improve results and the amount of useful information retrievable from a system, since it is the most demanding part of the workflow.

Therefore, we proposed a system that extends datasets to increase the amount of information using domain knowledge stored in an ER model. Following the KDD process, we implemented an algorithm that is able to interpret the knowledge in the ER diagrams and extend a given dataset, using a set of operations to augment the feature space and possibly improve the performance of ML models trained with that dataset. The algorithm works for every domain as long as there is domain knowledge present in an ER model, which is widely used in the database community. Even if there is not much domain knowledge available, the DANKFE algorithm can be coupled to the rest of the data preparation and feature engineering phase of the data science workflow by automatically generating some potentially interesting variables, as well as cleaning, scaling and balancing the dataset.

For evaluation, a case study composed of several datasets of different domains was created, as well as a classification problem and an ER model with some domain knowledge for every dataset. Validating the algorithm in both efficacy and efficiency, results show that feature generation through the use of domain knowledge improves the quality of models, for all models tested (varying in simplicity). Not only does the score improve, but also the time spent in generation and training remains short, with DANKFE-III spending 50 milliseconds per record on average. Compared to a popular AutoML framework (auto-sklearn), the generation of features yielded better results except for Naive Bayes, and while taking at worse around 6x less time per record.

In conclusion, making use of domain knowledge could prove very beneficial for automatic machine learning methods which currently do not make use of it, since it is able to improve model performance with low added complexity. It also gives data scientists a way to continue to use their valuable domain knowledge in the implementation of their models, which could improve the trust in these systems.

While the DANKFE system shows strong results compared to a recent and widely used AutoML approach, it also has some limitations, as previously stated. The system is currently limited to a set of defined operations that while span a large amount of possibilities in terms of feature generation, it is not yet possible to generate every single kind of feature. For this to be possible, the DANKFE system would require a decoder that would translate automatically the operations inside the ER model into Python functions, before running them.

Even though it was not the main focus of our work, the data preparation surrounding the DANKFE algorithm could also be improved, by enlarging the amount of preprocessing operations that a user can choose to perform on the data, such as dummification, type transformations or other kinds of encoding. While results show that the generated features tend to be considered useful by ML models, it is also known that the presence of redundant or irrelevant features can hurt models, as well as add unnecessary temporal and spatial complexity. To mitigate this, techniques such as feature selection or extraction, which are part of feature engineering, could also be tested, ensuring that only relevant or important features are added to the input dataset.

While DANKFE runs much faster than an AutoML approach, it can still be further optimized, by generating multiple features at the same time, given that the features follow the same constraints. The current implementation of DANKFE only processes one feature at a time, which can be slow if the dataset is large and especially the generated features use aggregation-based variables. Lastly, other knowledge representations can be tested in the future, such as ontologies or extended ER models [4], which have the benefits of allowing axioms and inheritances, aggregations and compositions, respectively. While ER models are more publicly available, these other approaches could allow for more functionalities.

7 ACKNOWLEDGEMENTS

This work was supported by national funds by Fundação para a Ciência e Tecnologia (FCT) through project VizBig (PTDC/CCI-CIF/28939/2017).

REFERENCES

[1] Cláudia Antunes and Andreia Silva. 2014. New Trends in Knowledge Driven Data Mining.. In *ICEIS (1)*. 346–351.
[2] Weiwei Cheng, Gjergji Kasneci, Thore Graepel, David Stern, and Ralf Herbrich. 2011. Automated feature generation from structured knowledge. In *Proc. 20th ACM international conference on Information and knowledge management*. 1395–1404.
[3] Steve Donoho and Larry Rendell. 1998. Feature construction using fragmentary knowledge. In *Feature Extraction, Construction and Selection*. Springer, 273–288.
[4] Ramez Elmasri and Shamkant B. Navathe. 2000. *The Enhanced Entity–Relationship (EER) Model*. Addison-Wesley, 107–135.
[5] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505* (2020).
[6] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.
[7] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv preprint arXiv:2007.04074* (2020).
[8] Lior Friedman and Shaul Markovitch. 2018. Recursive feature generation for knowledge-based learning. *arXiv preprint arXiv:1802.00050* (2018).
[9] Evgeniy Gabrilovich and Shaul Markovitch. 2009. Wikipedia-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research* 34 (2009), 443–498.
[10] Sainyam Galhotra, Udayan Khurana, Oktie Hassanzadeh, Kavitha Srinivas, Horst Samulowitz, and Miao Qi. 2019. Automated feature enhancement for predictive modeling using external knowledge. In *2019 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1094–1097.
[11] Trevor Hastie, Jerome Friedman, and Robert Tisbshirani. 2017. *High-Dimensional Problems* (2 ed.). Springer, 649–650.
[12] Yuh-Jyh Hu and Dennis Kibler. 1996. Generation of attributes for learning algorithms. In *AAAI/IAAI, Vol. 1*. 806–811.
[13] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.
[14] Udayan Khurana and Horst Samulowitz. 2020. Autonomous Predictive Modeling via Reinforcement Learning. In *Proc. 29th ACM International Conference on Information & Knowledge Management*. 3285–3288.
[15] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. 2016. Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1304–1307.
[16] Erin LeDell and Sebastien Poirier. 2020. H2o automl: Scalable automatic machine learning. In *Proc. AutoML Workshop at ICML*, Vol. 2020.
[17] Huan Liu and Hiroshi Motoda. 1998. *Feature extraction, construction and selection: A data mining perspective*. Vol. 453. Springer Science & Business Media.
[18] Jianbin Ma and Xiaoying Gao. 2020. A filter-based feature construction and feature selection approach for classification using Genetic Programming. *Knowledge-Based Systems* 196 (2020), 105806.
[19] Shaul Markovitch and Dan Rosenstein. 2004. Feature Generation Using General Constructor Functions. *Machine Learning* 49 (2004), 59–98.
[20] Christopher J Matheus. 1991. The need for constructive induction. In *Machine Learning Proc. 1991*. Elsevier, 173–177.
[21] John McCarthy. 1987. Generality in artificial intelligence. *Commun. ACM* 30, 12 (1987), 1030–1035.
[22] Steven Miller and Debbie Hughes. 2017. The quant crunch: How the demand for data science skills is disrupting the job market. *Burning Glass Technologies* (2017).
[23] Randal S Olson and Jason H Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*. PMLR, 66–74.
[24] Alberto G Salguero, Javier Medina, Pablo Delatorre, and Macarena Espinilla. 2019. Methodology for improving classification accuracy using ontologies: application in the recognition of activities of daily living. *Journal of Ambient Intelligence and Humanized Computing* 10, 6 (2019), 2125–2142.
[25] Dakuo Wang, Justin D Weisz, Michael Muller, Parikshit Ram, Werner Geyer, Casey Dugan, Yla Tausczik, Horst Samulowitz, and Alexander Gray. 2019. Human-ai collaboration in data science: Exploring data scientists' perceptions of automated ai. *Proc. ACM on Human-Computer Interaction* 3, CSCW (2019), 1–24.
[26] Jonathan Waring, Charlotta Lindvall, and Renato Umeton. 2020. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine* 104 (2020), 101822.
[27] Yuexiang Xie, Zhen Wang, Yaliang Li, Bolin Ding, Nezihe Merve Gürel, Ce Zhang, Minlie Huang, Wei Lin, and Jingren Zhou. 2021. FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data. arXiv:2007.14573 [cs.LG]