FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Meta Feature classification and insertion on a Synthetic dataset generation process

João Carlos Fonseca Pina de Lemos

Master in Informatics and Computing Engineering

Supervisor: Daniel Augusto Gama de Castro Silva

Second Supervisor: Leonardo Silva Ferreira

June 29, 2022

Meta Feature classification and insertion on a Synthetic dataset generation process

João Carlos Fonseca Pina de Lemos

Master in Informatics and Computing Engineering

June 29, 2022

Abstract

Machine learning (ML) models can perform a vast amount of tasks. Recognition of images and speech, statistical arbitrage (useful in finance), performing medical diagnostics through the analysis of medical exams, and performing predictive analysis are skills in their repertoire. The ML models must be trained using extensive data to accomplish these tasks.

Valid data is usually either scarce, expensive, or not readily available. Scrapping tools are commonly used to circumvent the scarceness of data, but their quality is questionable. A small dataset is also insufficient to produce an accurate prediction model. Even in cases where a large amount of data is available, the it's use may infringe privacy policies (a common problem when entering the medical field). These problems represent an accumulation of problematic data that leads to incorrect results. Synthetic datasets have a proven track record of improving ML effectiveness by providing a better dataset with fewer data imbalances than its "organic" bredren.

The work developed in this study is focused on implementing improvements on an already existing Ticket-based Synthetic DataSet Generator. Although synthetic, the new datasets should be as realistic as possible. Each dataset has its group of meta features - its fingerprint. When an existing dataset is available, the generator should analyse it and its meta features extracted, then used as parameters for the generation process. This generation should also be available if no original dataset is provided by making a simple selection of parameters. Despite being referenced in many studies, meta features are not uniformly described. Nevertheless, some studies agree on lists of meta features, and those arranged lists will be explored. Besides extracting parameters from existing datasets' meta features, it is also essential to analyse the parameter-parameter relationship. The quality of the synthetic dataset is also a factor of concern: cases of missing data, imbalances, anomalies and overfitting can easily happen during the generation process, and as such, the number of outliers must be realistic. Outliers should exist in a dataset as they contain useful information. The inclusion of customisation features, percentage of outliers and enhancements to the current generator will also be considered.

The workflow will start with the analysis and definition of a set of meta features, passing to the creation of parameters in the generator corresponding to that same data features. The quality assurance process will be the final step in the practical development of this study.

Keywords: Synthetic Dataset, Meta Features, Machine Learning

"The story so far:
In the beginning the Universe was created.
This has made a lot of people very angry and been widely regarded as a bad move."

Douglas Adams, The Restaurant at the End of the Universe

Contents

List of Figures

List of Tables

Abbreviations

DT	Decision Tree
GUI	Graphical User Interface
ML	Machine Learning
MLM	Machine Learning Model
PCA	Principal Component Analysis
SD	Synthetic Dataset

Chapter 1

Introduction

Dataset is the name given to a collection of related, discrete items of associated data. This data may be accessed apiece, in combination or handled as a complete entity [Akhter et al. (2019)]. The data is typically obtained from historical observations. However, good data is time-consuming, complicated or expensive to acquire [Kar et al. (2019)]. Synthetic datasets created by a dataset generator are used to combat that problem, creating data with similar or better results tham organic datasets [Anantrasirichai et al. (2019); Arvanitis et al. (2021)]. SNOOKER is one such generator.

SNOOKER: A DataSet GeNeratOr fOr HelpdesK SERvices is a software program developed initially by Leonardo Ferreira (Second Supervisor) for his doctorate dissertation. As the name implies, this software is responsible for generating datasets for helpdesks. The generator can forge realistic ticket-based datasets to train a machine learning model (MLM). However, several improvements can be made to the current program, and its output should be analysed to extract the dataset's characteristics. To decide on further steps, we will explore the concept of dataset generation further. In the next section, a historical overview and the motivation for the development of this thesis will be explored.

1.1 Context and Motivation

Machine learning (ML) is a technique of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the presumption that systems can learn from data, identify patterns and make decisions with minimal human intervention [Wong (2021)]. It is, in part, based on a model of brain cell interaction. The term was coined by Donald Olding Hebb in 1949.

> "When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell." - Hebb (1949)

Evolution resulted in machines capable of processing data using a structural model of the Real World as the basis. These machines used mathematical models and algorithms and ended up not learning but, in fact, merely following instructions [Cohen (2021)]. The next step was to create instructions that allow the computer to learn from experience, using the previous model. The machine must be able to extract its own rules from large amounts of data and use those rules for classification and prediction.

Dataset is a term used to describe a collection of data. Data is usually represented in tabular form, making datasets collections of one or more tables. Table columns symbolise a certain variable, and rows relate to the conveyed values of the dataset in question [Priyanka et al. (2016)]. The dataset lists values for each variable for each data set member. Datasets can also consist of a collection of documents or files [Snijders et al. (2012)]. The description of dataset is consensual among the scientific community that uses data as a basis for research activities [Renear et al. (2010)]. As data differs, datasets also differ, having different characteristics. Several studies were made to examine these characteristics and recognise patterns [Reif et al. (2014)]. This topic is explored in further detail in Chapter 2.

Scrapping and collection tools are commonly used to circumvent the scarceness of data, but their quality is questionable. Challenges still arise when more large-scale datasets are collected. Small datasets are not as effective in producing accurate prediction models [Lateh et al. (2017)]. Even in cases where a large amount of data is available, their use may infringe privacy policies (a common problem when entering the medical field as Dahmen and Cook (2019) described in their studies). These problems represent an accumulation of problematic data that leads to incorrect results [Sethi and Kantardzic (2018)]. Early works on Synthetic datasets (SDs) were used to fill in missing values in surveys [Rubin (1986)]. Nowadays SDs have a proven track record of improving ML effectiveness by providing a better dataset with fewer data imbalances than its "organic" bredren [Anantrasirichai et al. (2019); Arvanitis et al. (2021)].

Currently, SDs are used in various fields, ranging from the previously mentioned medical to many others. Anantrasirichai et al. (2019) used SDs in order to train a Machine Learning Model (MLM) to pinpoint surface deformation with a strong statistical link to a volcanic eruption. Yu et al. (2011) looked at increasing social media connection by training a MLM with SDs to group personally and geographically close users, increasing the potential for real-world relationships to flourish. Sethi et al. (2017) and Hamsapriya et al. (2011) used them to detect Spam messaging in SMS and email, respectively. Kumar et al. (2018) looked at predicting stock market trends using SD trained MLM. As we can see, the applications of SDs are numerous, making SDs an excellent alternative to real data.

While the study of dataset generation is growing (Fig. 1.1), there are still many ways to study, explore and improve the process of SD generation, be it theoretical (through the definition of a set of core meta feature lists) or practical (by looking at generation and evaluation processes). Through the development of this study, we strive to add something to the field.

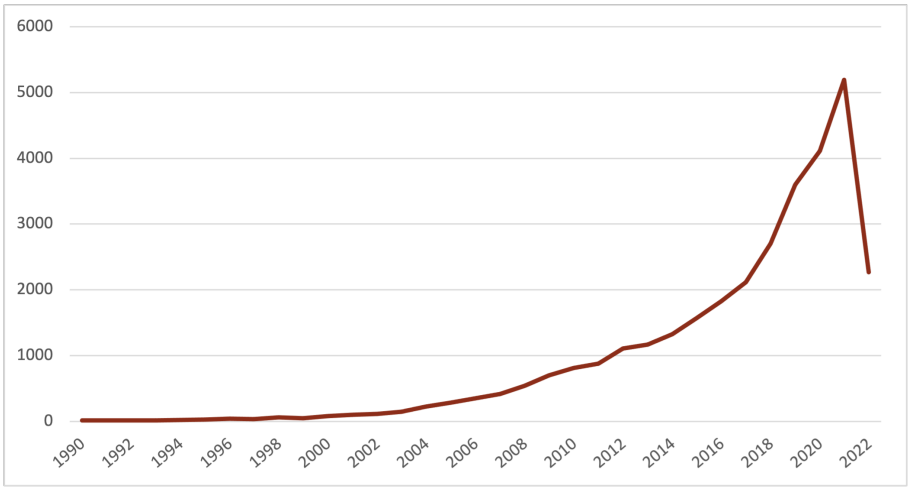


Figure 1.1: Number of SD papers published on Scopus from 1990 to February 2022. Data was collected by searching papers using "Machine Learning" as keywords and applying a yearly filter

1.2 Objectives

The work developed in this study is related to enhancing an already existing Ticket-based SD Generator for Helpdesk purposes. We will look at the output dataset and study it, extracting its meta features. We will also explore the possibility of integration of meta features inside the generation process. The aim of this study is both theoretical and practical and will try to answer a series of questions:

1. How is the current generator working?
   The first step during the development process is a dive into the heart of the generator.

2. What selection of meta features should be calculated from the data in the synthetic dataset?
   This task started theoretically; a list of meta features was made, and selected meta features were included in a meta feature extraction module.

3. What generation methods can we use?
   Collection of generation algorithms followed by analysis and application of the most suitable process to the generator.

4. Can we include specific values of meta features inside the generation process?
   Evaluation of methods to cause certain values of specific meta features inside the generation process. We also study the utility of datasets that go through such methods.

5. What features can be applied to the generator?
   Analysis of the generator and development of new features and improvements.

In chapter 4 we will take a look at development methodologies and planning.

1.3 Document Structure

In this chapter, we looked at the context, motivation, and application of SDs. We also described the objectives of exploring and enhancing the current Ticket-based Synthetic DataSet Generator.

Chapter 2 will take a look at the characteristics of datasets, the meta features. We will analyze how the inclusion of meta feature selection during the generation process can enhance the creation of a more suitable dataset for the ML process. In Chapter 3 we will explore the process of generating a synthetic dataset by first investigating some commonly used algorithms and second by reviewing how other SDs generation projects performed their generation undertakings. We will also take a look at methods for evaluating the quality of a dataset. These Chapters are part of the state-of-the-art review.

In Chapter 4 the planning for the development of the process will be probed. Close examination of needed tasks will take place. Expected results and validations will also be analyzed. Due to the nature of the study, we will also present a risk analysis in this chapter. The current state of our generator will be studied in Section 4.1. We also provide a simplistic outline of SNOOKER and SNOOKER's output.

Chapter 5 will focus on the meta feature classification of synthetic datasets. We will analyse and present the methodology needed to develop this feature.

A critique of the concept of meat feature inclusion during generation is made in Chapter 6.

Finaly in Chapter 7 we display our final remarks.

Chapter 2

Meta Features

Datasets are data collections; as data differs, datasets also change, having different characteristics. The 'meta' prefix is of Greek origin and means 'among,' 'after,' 'beside' or 'with,' and is often used to identify something that provides information about something else, naming the collection of those characteristics inside the datasets meta-features [Binder and Wirth (2015)].

Several authors have studied meta-features [Reif et al. (2014)]. However, a uniform description of meta-features does not exist. Many studies diverge on selecting characteristics to construct their meta-feature definition [Rivolli et al. (2019)]. The reproduction and comparison of said studies are therefore complex and abstract. A meta-feature should not be too difficult to calculate [Castiello et al. (2005)].

Meta-features are helpful when looking at MLMs as they depict properties of the data, which are predictive for ML algorithms' performance [Santos et al. (2021)]. By choosing datasets with a particular assortment of meta-features, more valuable data will be analyzed and processed, reducing the use of irrelevant and redundant data.

Several studies use simple datasets or only focus on purely mathematical distribution-based datasets. While mathematical meta-features are valuable [Kanda et al. (2016)], even if datasets with those exact meta-features had positive results when applied to the corresponding studies, the same does not apply to a generator. A generator containing only a selection of simple meta-features would not be helpful for most cases.

However, analysis of several studies revealed some, even if small, agreeance on some datasets commonly used. It led to the extrapolation of three families: general, statistic-based, and information theoretic-based meta-features [Reif et al. (2012); Wang et al. (2014)]. General meta-features can be directly observable from datasets, statistical uses mathematical distributions to describe data distribution, and information-theoretic makes use of data entropy. Some other studies also included the model-based (usually based on properties of decision trees) and landmarking (use the performance of fast and straightforward learning algorithms to characterize datasets) families [Rivolli et al. (2019); Filchenkov and Pendryak (2015)]. The following sections present further analysis of these and other meta-feature families.

2.1 General Meta-Features

Table 2.1: General Meta-Features

Acronym	Meaning
nrAttr	number of attributes
nrBin	number of binary attributes
nrCat	number of categorical attributes
nrNum	number of numeric attributes
numToCat	proportion of numerical to categorical attributes
catToNum	proportion of categorical to numerical attributes
nrInst	number of instances
nrClass	number of classes
attrToInst	proportion of number of attributes to number of instances
instToAttr	proportion of number of instances to number of attributes
classToAttr	number of classes per attribute
instToClass	number or instances per class
freqClass	frequency of instances in each class
nrAttrMissing	number of missing attributes
nrInstMissing	number of missing instances
nrMissing	total missing number

General meta-features are directly observable from the dataset, representing basic information and being the most explicit set in terms of computation cost. To a certain extent, they are devised to estimate the complexity of the problem related to that dataset [Bilalli et al. (2017)].

The measures suggested in Table 2.1 represent concepts connected to the number of predictive attributes, instances, target classes, and missing values. These measures are relevant to illustrating a dataset's main aspects, delivering knowledge that can support the selection of a dataset for a particular learning task; most of these measurements are self-explanatory.

It should be explained that the *nrAttrMissing* and *nrInstMissing*, although looking similar, represent different aspects of the same problem. *nrAttrMissing* represents the number of empty columns in a dataset, counting the currently missing attributes. *nrInstMissing* on the other hand, represents the number of entries (rows) that are empty.

While most meta-features are synonymous with a straightforward examination, others can provide additional connotation: *AttrToInst* helps analysing the dimension of the dataset and *inst-ToAttri* illustrates the absence of data; a diminutive value of *instToAtrrib* can support the detection of overfitting issues on a MLM; *freqClass* values allow the extraction of suitable measures, such as the standard deviation of the class distribution and *missing-type* values help evaluate the quality of the dataset [Lindner and Studer (1999)].

2.2 Statistical Meta-Features

Table 2.2: Statistical Meta-Features

Acronym	Meaning
cor	correlation
cov	covariance
nrCorAttr	proportion of highly correlated attribute pairs
gMean	geometric mean
hMean	harmonic mean
tMean	trimmed mean
mean	-
median	-
mad	median absolute deviation
max	maximum
min	minimum
range	-
iqRange	interquartile range
kurtosis	-
sd	standard deviation
skewness	-
var	variance
nrNorm	normality of the attributes
nrOutliers	number of attributes that contain outliers
canCor	canonical correlations
nrDisc	number of discriminant values
sdRatio	homogeneity of covariances
wLambda	Wilks lambda

Statistical meta-features extract details about the performance of statistical algorithms or data distribution [Michie et al. (1994)]. They are the most extensive and the most diversified group of meta-features and represent attribute statistics of a dataset [Bilalli et al. (2017)]. Statistical measures are deterministic and support only numerical attributes. Datasets that hold categorical data must be either partially dismissed or edited and transformed to numerical values [Rivolli et al. (2022)]. Table 2.2 presents a list of statistical meta-features.

Most statistical features are extracted per attribute individually. The measures of central tendency (a single value that attempts to describe a set of data by identifying the central position within that set of data [Weisberg and Weisberg (1992)]) consist of the *mean* (and its variations) and the *median*.

Correlation (*cor*) and covariance (*cov*) mirror the interdependence of the attributes. Elevated values are associated with an increased grade of redundancy in the data.

Measures of dispersion depict the spread of the data. Those measures are represented by kurtosis, range, standard deviation (sd), the interquartile range (iqRange), maximum (max), median absolute deviation (mad), minimum (min), skewness and variance (var). The values of *Kurtosis* and *skewness* are appropriate for grasping the normalcy of the data [Vanschoren (2010)].

The nrNorm value echoes the count of normally distributed attributes in the dataset. nrOutliers, on the other hand, measure the number of visible outliers. Both values can impact the behaviour of MLMs algorithms.

The discriminant statistical measures categorise observations into non-overlapping sets [McLachlan (2005)]. They are represented by *nrDisc*, *sdRatio* and *wLambda*. These meta features present some specifics regarding the datasets and are exclusively used in classification assignments.

2.3 Information-Theoretic Meta-Features

Table 2.3: Information-Theoretic Meta-Features

Acronym	Meaning
attrEnt	entropy of the predictive attributes
classEnt	entropy of the target values
eqNumAttr	equivalent number of attributes
jointEnt	joint entropy
mutInf	mutual information
nsRatio	noise signal ratio

Information-theoretic meta-features, presented in Table 2.3, illustrate the quantity of information in the data. Most of them are limited to expressing classification problems. In fact, from the meta-features presented only the entropy of the predictive attributes (attrEnt) can be used in other tasks.

Information-theoretic meta-features are robust, deterministic and directly computed. Semantically, they represent the variability and redundancy of the predictive attributes to represent the classes [Rivolli et al. (2018)].

The entropy of the predictive attributes (*attrEnt*) represents the average uncertainty of the predicative attributes. This meta-feature outlines the capacity for class discrimination in the dataset. The entropy of the target values (*classEnt*) does the same regarding the class attributes [Segrera et al. (2008)] and is used to represent the amount of data needed to specify one class.

The joint entropy meta-feature (*jointEnt*) displays the proximate significance of the attributes in the mather of represention. On the other hand mutual information meta-features (*mutInf*) represents the common information in the attributes, correlating to the degree of interdependance.

Lastly, the equivalent number of attributes (*eqNumAttr*) echoes the lowest number of attributes necessary to represent the target attribute. At the same time, the noise signal ratio (*nsRatio*) refers to the ratio of irrelevant data in the dataset.

2.4 Model-Based Meta-Features

Table 2.4: Model-Based Meta-Features

Acronym	Meaning
leaves	number of leaves
leavesBranch	number of distinct paths
leavesCorrob	support described in the proportion of training instances to the leaf
leavesHomo	distribution of the leaves in the tree
leavesPerClass	proportion of leaves to the classes
nodes	number of nodes
nodesPerAttr	proportion of nodes per attribute
nodesPerInst	proportion of nodes per instance
nodesPerLevel	number of nodes per level
nodesRepeated	number of repeated nodes
treeDepth	depth of each node and leaf
treeImbalance	degree of imbalance in the tree
treeShape	shape of the tree
varImportance	importance of each attribute

Meta-features based in models consist of information extracted from a predictive learning model -usually a Decision Tree (DT) model- that characterises the dataset by the complexity of such model. The number of leaves, nodes and shape of the tree are factors that represent the complexity. Table 2.4 shows the Decision Tree model meta-features.

Leaf-based attributes measure the orthogonal complexity of the decision surface. Some measures result in a value for each leaf, and those measures are the number of distinct paths (*leavesBranch*), the support described in the proportion of training instances to the leaf (*leavesCorrob*) and the distribution of the leaves in the tree (*leavesHomo*). The ratio of leaves to the classes (*leavesPerClass*) represents the classes complexity.

Node-related features extract information about the balance of the tree to describe the discriminatory power of attributes. The proportion of nodes per attribute (*nodesPerAttr*) and nodes per instance (*nodesPerInst*) result in singular values. The number of nodes per level (*nodesPerLevel*) describes how many nodes are present in each level, and the number of repeated nodes (*nodesRepeated*) represents the number of nodes associated with each attribute used for the model. These last two meta-features have each node at its maximum value.

The estimates based on the tree size extract details around the leaves and nodes to depict the data complexity. The tree depth (*treeDepth*) illustrates the depth of each node and leaf. The tree imbalance (*treeImbalance*) describes the degree of imbalance in the tree. The shape of the tree (*treeShape*) symbolises the entropy of the probabilities to randomly reach a specific leaf in a tree from each one of the nodes.

The importance of each attribute (*varImportance*) represents the amount of information present in the attributes before a node split operation.

All the meta-features presented in this section are helpful when exclusively looking at DT models. Meta-features related to k-Nearest Neighbour (kNN) and Perceptron neural networks exist but were not further explored when producing this investigation.

2.5 Landmarking Meta-Features

Table 2.5: Landmarking Meta-Features

Acronym	Algorithm
bestNode	Best Node
eliteNN	Elite Nearest Neighbor
linearDiscr	Linear Discriminant
naiveBayes	Naive Bayes
oneNN	One Nearest Neighbor
randomNode	Randomnode
worstNode	Worst Node

Landmarking uses the execution of a collection of simple and swift algorithms to characterise datasets. In this section we demonstrate some commonly used algorithms that are used as landmarking meta-features: *bestNode, eliteNN, linearDiscr, naiveBayes, oneNN, randomNode* and *worstNode*, to those algorithms we give the name of landmarkers. They are also presented in Table 2.5. Moreover, it should be noted that the performance of any algorithm can be used as a landmarking meta-feature.

The performance of a DT model can be measured using single attributes to initialise the model. The landmarkers used in this case are *bestNode*, *randomNode* and *worstNode*. In the first case, *bestNode* employs the most informative attribute to initialise the model. The *randomNode* algorithm utilises a random value of a node. Lastly, *worstNode* makes use of the least informative attribute to initialise the model.

The elite-Nearest Neighbor (*eliteNN*) landmarker uses the kNN algorithm were k is equal to one (making it a 1NN) and results from the k-NN model using a subset of the most informative attributes in the dataset. In contrast, the one-Nearest Neighbor (*oneNN*) results from a similar learning model induced with all attributes in the dataset Kramer (2013).

The Linear Discriminant (*linearDiscr*) and the Naive Bayes (*naiveBayes*) algorithms use all attributes to induce the learning models. The first technique finds the best linear combination of predictive attributes able to maximise the separability between the classes. The second technique is based on the Bayes' theorem and calculates, for each feature, the probability of an instance to belong to each class.

Besides the ones described before, relative and subsampling landmarkings also exit. On relative landmarkings, a pairwise comparison is used instead of the algorithm's performance. In this case, the meta-feature indicates the winner, the difference between them, or the two performances' ratio. Subsampling landmarking works by applying the algorithms mentioned before to the original dataset's smaller set (a subsample).

2.6 Other Meta-Features

Presented before were the most common and useful meta-features described multiple times in literature. However, many others exist and can be helpful in particular scenarios [Rivolli et al. (2022)]. Some of these meta-feature groups include *clustering and distance*, *complexity*, *data distribution*, *case-based*, *concept-based*, *structural information* and *time-based measures*.

2.6.1 Clustering Meta-Features

Table 2.6: Clustering Meta-Features

Acronym	Meaning
compactness	how close the clusters are
connectivity	local densities
nrClusters	number of clusters
nre	normalised relative entropy
purityRatio	ratio of clusters that contain instances related to each class
sizeDist	allocation of the clusters

Clustering measures (Table 2.6) characterise the space, splitting it into clusters. The clustering partitions can be defined by three aspects: the distance between the instances, their density or a particular data distribution. These partitions are evaluated by both clustering algorithms and measures used to calculate the distance between instances. Predictive attributes are used to calculate these measures as they are commonly used in unsupervised problems.

Clustering meta-features include: *compactness*, *connectivity*, *nrClusters*, *nre*, *purityRatio* and *sizeDist*.

The quality of the partitions can be obtained by looking at different validations. The compactness and connectivity are excellent examples. *Compactness* calculates how close the clusters are, with lower values denoting tighter groups. *Connectivity* demonstrates local densities by calculating the infringements of the nearest-neighbour relationship between instances in different partitions.

Given the data partition produced by a clustering algorithm, *nrCluster* represents the number of clusters. When a clustering algorithm is used and the clusters defined dynamically, this simple informative measure becomes more valuable to calculate, which doesn't happen in cases where the number of clusters becomes an input parameter defined by the user.

The distribution of instances among the clusters is analysed by the normalised relative entropy (*nre*). This value indicates how uniform the instances are distributed. Values close to zero reveal well-distributed clusters.

The value of *purityRatio* looks at the instances' classes to evaluate the partitions. The purityRation is calculated for each class and captures the ratio of clusters that contain instances related to the respective class. Datasets with high values are more complex than those with low values.

The measure *sizeDist* grabs the allocation of the clusters based on the instances' frequency. As glimpsed in Ler et al. (2018), a distribution skewed to the right reveals a complex dataset.

One of the main obstacles in using these meta features is their high computational complexity, which restricts their use. Additionally, they allow a wide range of choices, with different impacts in the value returned. In spite of being able to provide a good characterization, clustering measures are under-explored in the meta feature field.

2.6.1.1 Complexity Meta Features

Ho and Basu (2002) proposed *Complexity*-based meta-features and used them to capture the underlying difficulty of classification tasks. They analyse things like class overlapping, the density of manifolds and the shape of decision boundaries, among other aspects of the dataset. The meta-features presented below were first extrapolated by Rivolli et al. (2022) that divided them into five types of measures: feature-based, linearity, neighbourhood, network, and dimensionality measures. A list of this type of meta-features can be found in Table 2.7.

Feature overlapping measures illustrate how informative the predictive attributes are. There are five types of measures of this kind: maximum Fisher's discriminant ratio, directional-vector maximum Fisher's discriminant ratio, the volume of the overlapping region, maximum individual feature efficiency and collective feature efficiency. The complexity is low if at least one predictive attribute can separate the classes.

Linearity measures quantify whether the classes are linearly split. They incorporate the sum of the error distance by linear programming, the error rate of the linear classifier and the non-linearity of a linear classifier.

Neighbourhood measures explore the vicinities of singular examples and capture class overlapping and the shape of the decision boundary. These measures include the fractions of Borderline Points, the ratio of intra/extra class nearest neighbour distance, the error rate of the nearest neighbour classifier, the non-Linearity of the nearest neighbour classifier, the fraction of hyperspheres covering data, and the local set average cardinality.

The network measures convert a dataset into a graph and pull structural and statistical information. Each example from the dataset corresponds to a node, while undirected edges connect examples and are weighted by their distances. These measures contain the average density of the network and Hub score.

Lastly, the dimensionality measures assess data sparsity according to the number of instances comparative to the predictive attributes of the dataset. The measures include the average number

Table 2.7: Complexity Meta-Features

Meta-feature	Type
maximum Fisher's discriminant ratio	feature-based
directional-vector maximum Fisher's discriminant ratio	feature-based
volume of the overlapping region	feature-based
maximum individual feature efficiency	feature-based
collective feature efficiency	feature-based
sum of the error distance by linear programming	linearity
error rate of linear classifier	linearity
non-linearity of a linear classifier	linearity
fractions of Borderline Points	neighbourhood
ratio of intra/extra class nearest neighbour distance	neighbourhood
error rate of the nearest neighbour classifier	neighbourhood
non-Linearity of the nearest neighbour classifier	neighbourhood
the fraction of hyperspheres covering data	neighbourhood
local set average cardinality	neighbourhood
average density	network
hub score	network
average number of points per dimension	dimensionality
the average number of points per PCA dimension	dimensionality
the ratio of the PCA	dimensionality
dimension to the original dimension	dimensionality

of points per dimension, the average number of points per Principal Component Analysis (PCA) dimension, the ratio of the PCA and dimension to the original dimension.

Complexity meta-features are beneficial when looking at imbalanced datasets. Although valuable, their high computational requirements make them challenging to use. Lorena et al. (2019) performed a further review of these meta-features.

2.6.2 Data Distribution Meta-Features

Data distribution measures assess how the data is distributed in the predictive attribute space.

One of these measures is the concentration coefficient. This coefficient can be applied per pair of attributes and for each attribute and the class. It represents the association strength between each pair of attributes and the predictive and target attributes.

Two more meta-features exist in this group: the proportion of principal components that explain a specific dataset variance, used for capturing the redundancy of predictive attributes, and the sparsity, used for indicating the variance in the values of the attributes.

2.6.3 Case-based Meta-Features

Case-based measures examine the dataset, comparing its instances among themselves. The objective is to identify properties that might make the learning process more complex [Köpf and Iglezakis (2002)] such as inconsistency, incoherence and uniqueness making the meta-features included in this group *consistencyRatio*, *uniquenessRatio* and *incoherenceRatio*.

The *consistencyRatio* quantifies the balance of replicated instances with different targets, where zero is an ideal value.

The *uniquenessRatio* is an abstraction of consistencyRatio, using only the predictive attributes.

The *incoherenceRatio* meta-feature calculates the proportion of instances that do not overlap with other instances in a predefined number of attributes. Values nearest to 1 are favoured in a dataset, indicating the scattered of the instances.

2.6.4 Concept-based Meta-Features

The concept-based measures characterize the irregularity of the input-output distribution [Perez and Rendell (1996)]. "An irregular distribution is observed when neighbouring instances have distinct target values" (cited from Muñoz et al. (2018)). Meta features in this group include the *weighted distance*, *cohesiveness* and *concept variation*.

The *weighted distance* displays how dense or sparse the distribution of the instances is. The *cohesiveness* gauges the density of the example distribution. Lastly the *concept variation* is defined by the cohesiveness average of all possible instances in the input space.

2.6.5 Structural Information Meta-Features

Structural information meta-features are helpful when it comes to analysing similarity between datasets [Wang et al. (2015)]. They characterise binary item sets to apprehend the allocation of values of both single attributes (*oneItemset*) and pairs of attributes (*twoItemset*).

The value of *oneItemset* captures each individual's attributes, whereas *twoItemset* displays possible correlations concerning pairs of attributes.

2.6.6 Time-based Meta-Features

Time-based measures use elapsed time to characterize the datasets [Reif et al. (2011)]. It should be noted that these meta features are very hardware dependent as a difference in computational power can change the results.

2.7 Conclusion

In this chapter, we took a look at various sets of commonly used meta features, creating a collected list of features that can be usefull in analyzing datasets. The main objective was to collect and present a list of meta-features to be used later during the development phase of this study.

Chapter 3

Generation

Valid data is usually either scarce, expensive, or not readily available Kar et al. (2019). Public availability of the datasets is crucial for scientific and development processes. The use of public or scrapped data can unfortunatly cause privacy problems. Synthetic datasets that conserve the utility while safeguarding the privacy of the data owners stands as a viable alternative.

Synthetic data can have several forms. They can be textual, tabular or media-based. Artificial text-based data is very commonly used in sentence formation and automated voice-over (ex. amazon uses artificial text data to bootstrap Alexa's language options Kollar et al. (2018)). An example of its usage can be seen in figure 3.1. Tabular data is the most common type of synthetic data. Its objective is to mimic real-life data stored in tables. An example of artificial tabular data is the one used by Toutanova et al. (2016) to create a synthetic dataset focused on language by hand. Lastly, we have synthetic media-based data. This data can be synthetic video, image, or sound. Media is rendered with properties close-enough to real-life data. This similarity allows using the synthetic media as a drop-in replacement for the original data.

There are three types of artificially generated data. It is possible to subsample it from a synthetically generated population. This kind of generation is known as fully synthetic dataset generation. Theoretically, fully synthetic datasets provide 100% guarantee against the disclosure of the value of sensitive attribute Rubin (1993). Using an already existing dataset to generate a new dataset containing the original characteristics while artificially generating data used to replace sensitive values is also possible. This generation method is known as partially synthetic dataset generation. In order to protect sensitive information, an imputer decides to alter the values of a set of attributes for a subset of data points. This way, the actual values that contain a high risk of revelation are replaced Dandekar et al. (2018). We can also generate data using both natural and synthetic data. To this kind, we give the name of hybrid synthetic data. For each random record of actual data, we chose a close record in the synthetic data and then both are combined to form hybrid data. It has the advantages of both fully and partially synthetic data. Therefore, it provides proper privacy preservation with high utility compared to the other two. Its generation requires extra steps, and as such, this type of synthetic data requires more memory and processing time.

The goal of dataset generation is to create realistic a synthetic data generation method that is

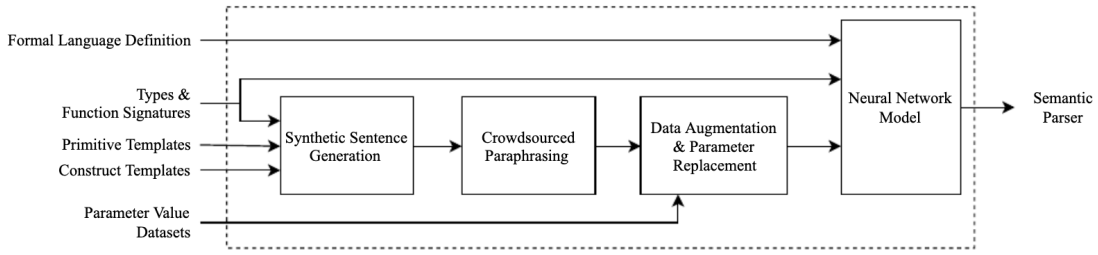


Figure 3.1: Overview of the Genie Semantic Parser Generator, a toolkit for creating a semantic parser for new virtual assistant capabilities. Sentences are synthetically generated Campagna et al. (2019)

modular, repeatable, and automated. The synthetic data should be realistic enough that its measurements and performance are similar (and as such predictive) to real-world data measurements and performance Boggs et al. (2014). One example of integration of a generator can be seen in Fig 3.1.

In order to generate synthetic data, the a statistical distribution of data is the method typically used. This distribution is made for a set of samples from directly measured data. New values are created in the same format as the actual data.

Drechsler (2010) presented a procedure for synthetic dataset generation: "We define an order of the attributes that will be synthesised; values of the first attribute are synthesised by training the dataset synthesiser on the original dataset". With this method attributes are then generated later use already synthetic data as basis.

In the following sections we will take a look at a series of dataset generation methodologies.

3.1 Sampling from Independent marginals

The Independent marginals method consists of sampling from the observed marginal distributions of each variable. Marginal probability is the name given to the probability of an event irrespective of the outcome of another variable.

The marginal distribution of a group of arbitrary variables is the probability distribution of the variables included in the group. It gives the likelihoods of varied values of the variables in the group without reference to the values of the other variables. This distribution differs from a conditional distribution, which presents the probabilities taking into account the values of the other variables Trumpler and Weaver (1953).

The independent marginals method approach has its advantages, being computationally efficient and having the possibility of estimating marginal distributions for different variables in parallel. However, it does not capture statistical dependencies across variables. Therefore, the generated synthetic data may fail to capture the structure of the original data.

3.2 Bayesian network

Bayesian networks are probabilistic graphical models. In these models each node represents a random variable. The edges between the nodes represent probabilistic dependencies among the corresponding random variables. The graph structure and conditional probability distributions are inferred from the actual data.

The learning process in Bayesian networks is comprised of two steps. The first one consists of learning a directed graph from the data. This graph expresses all the pairwise conditional dependencies (or lack thereof) among the variables. The second step is estimating the conditional probability tables.

The graph deduced from the actual data contains the conditional reliance among the variables. This graph also provides a visual representation of the variables' relationships. By sampling from the inferred Bayesian network, we can generate synthetic data.

3.3 Mixture of product of multinomials

A multinomial experiment is a statistical experiment with a series of characteristics. The experiment consists of a set amount of repeatable trials, each with a set number of outcomes and a constant probability for each outcome. These trials are independent, meaning that a previous trial does not influence future trials.

Any multivariate categorical data distribution can be expressed as a mixture of product of multinomials (MPoM) Dunson and Xing (2009).

The multinomial model over term frequencies is defined as in the Enquation (3.1).

$$P(\tilde{x}|\tilde{y}) = \frac{l!}{\prod_i x_i} \prod_{i=1}^{n} y_i^{x_i} \tag{3.1}$$

where x_i is the number of times word i occurs, y_i is the mean parameter for word i, and $l = \sum_i x_i$ is the document length Rennie (2005).

Theoretical guarantees exist regarding the flexibility of this methodology to model any multivariate categorical data. However, this process can be very time-consuming in problems with large datasets.

3.4 Generative adversarial networks

In the generative adversarial networks (GANs), two neural networks are trained jointly, competing. While the first network focuses on creating a realistic artificial dataset, the second tries to distinguish between natural and artificial data from the first dataset. Due to the competition, each network pushes the other one, leading to better results.

GANs have been successful in generating more complex data, being capable of producing synthetic images and text Mirza and Osindero (2014). They are, however, incapable of producing

categorical datasets, which happens as GANs are unable of computing "gradients latent categorical variables for training vis backpropagation" - in Goncalves et al. (2020a). GANs do not require strict probabilistic models to perform their generation tasks. Therefore they are more flexible than the models previously mentioned. They can also deal with mixed data types.

When a GAN has many parameters, proper choice of multiple tuning ones (hyper-parameters) is difficult and time-consuming. GANs are also known to be challenging to train as the process of solving the associated min-max optimization problem can be very unstable, a problem that can be circumvented if the variation Wasserstein GAN is applied Arjovsky et al. (2017).

3.5 Multiple imputation

Multiple imputation is a method where several distinct possible datasets are created and the final dataset results from combining those datasets. This way, this approach is an optimal one to use in order to solve problems of missing data.

The generation process starts with the creation of several copies of the dataset. These copies will have their missing values replaced by plausible imputed user values according to their original distribution.

The next set is the analysis of the created datasets. In this step, each dataset is analysed individually. This step is followed by combining these datasets in the proper output synthetical dataset.

3.6 Linear Regression

Linear regression seeks to model the association between two variables by using a linear equation to the observed data. One variable is considered an explanatory variable, and the other is considered a dependent variable.

We can describe a linear regression line using the formula: $Y = a + bX$. This formula's X is the explanatory variable, and Y is the dependent variable. The line slope is b, and a is the intercept (the value of y when x = 0).

In order to synthesise every attribute Y_i in a dataset, we learn the parameters of the regression model using the dataset with attributes in Y_{-i}. We generate values for Y_i by sampling from a Gaussian distribution with constant variance and the mean as determined parameters of regression.

3.7 Decision Tree

Reiter (2005) proposed a generation technique that uses classification and regression tree and applied it to the generation of a partially syntetic dataset. The main components of a decision tree model are nodes and branches, and the most critical steps in building a model are splitting, stopping, and pruning Song and Ying (2015).

Day	Outlook	Temp.	Humidity	Wind	Play Tennis
T1	Sunny	Hot	High	Weak	No
T2	Sunny	Hot	High	Strong	No
T3	Overcast	Hot	High	Weak	Yes
T4	Rain	Mild	High	Weak	Yes
T5	Rain	Cool	Normal	Weak	Yes
T6	Rain	Cool	Normal	Strong	No
T7	Overcast	Cool	Normal	Weak	Yes
T8	Sunny	Mild	High	Weak	No
T9	Sunny	Cool	Normal	Weak	Yes
T10	Rain	Mild	Normal	Strong	Yes

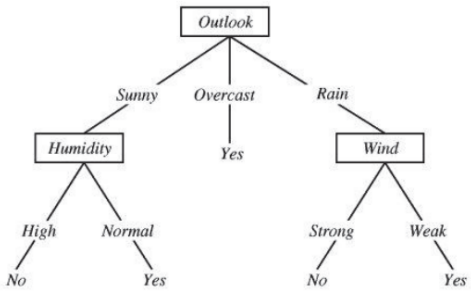


Figure 3.2: Creation of a Decision Tree (bellow) using a dataset (above). This was used for a ML model to decide to play tennis Gavankar and Sawarkar (2017).

The process starts with building a decision tree using the values of the available attributes in the dataset $Y - i$. To synthesise the value of an attribute Yi for a data point, we trace down the tree using the known attributes of the datapoint until reaching the leaf node. Kirchner et al. (2004) used this method and applied it to swine production and concluded that "the decision tree algorithm can detect relationships and patterns in real swine breeding datasets". Fig. 3.2 depicts a decision tree model created form a tabluar dataset.

3.8 Random Forecast

Random Forest is a method introduced by Breiman (2001). As its name implies, a random forest consists of many individual decision trees that operate as a group. Each particular tree in the random forest releases a class prediction, voting occurs for each class. The class with the most votes becomes our model's prediction.

In order to synthesise values for a distinct point Yi, a fixed number of decision trees on random samples of training dataset $Y - i$ are trained. For a categorical attribute, the assemblage of results from the decision tree forms a multinomial distribution. For a continuous attribute, Caiola and Reiter (2010) proposes the use of a kernel density estimator over the results from decision trees using sample values from the estimator.

3.9 Neural Network

Neural Networks Abdi et al. (1999) are at the core of deep learning algorithms and are inspired by the human brain, simulating how bodily neurons signal to one another.

Artificial neural networks (ANNs) incorporate node layers possessing an input layer, one or more hidden layers, and an output layer. Each node represents a neuron that connects to another and has an associated weight and target value (threshold). If the signal sent individual node is above the specified target value, that node is activated. Data is then sent to the next layer of the network.

Neural networks depend on training data to learn and improve accuracy over time. Nevertheless, once these learning algorithms are fine-tuned for precision, they are potent tools in computer science and artificial intelligence, allowing us to categorise and cluster data at a high rate.

3.10 Evaluation

One of the primary intents of enhancing a dataset generator is to increase the quality of the produced synthetic datasets. In order to evaluate the quality of a dataset, some methodologies are required.

The majority of evaluation functions use a scoring scheme. Let us suppose that D1 and D2 are datasets and $E(x)$ is an evaluation function, and thus $E(D1)$ and $E(D2)$ generate some scores. If $E(D1) > E(D2)$, we can expect that dataset D1 will yield better training/testing accuracy than D2.

Dash and Liu (1997) grouped evaluation functions into five categories: distance measures, information measures, dependency measures, consistency measures, and classifier error rate measures. Distance measures are the most popular and include separability, divergence, and discrimination measures. These measures were further analysed by Parthalain et al. (2007).

In the following sections, we will look at some evaluation approaches and points of interest when it comes to dataset evaluation.

3.10.1 Distance Methods

In this section, we summarize several distance measures for dataset evaluation. These measures are designed for feature evaluation, but they can be adopted for dataset evaluation. The goal of distance measurement is to calculate the distances among categories in a dataset.

3.10.1.1 Euclidean distance

The Euclidean distance between the two points $p(p_1, p_2, p_3, \ldots, p_n)$ and $q(q_1, q_2, q_3, \ldots, q_n)$ is defined as follows:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} \tag{3.2}$$

Various equations can define Euclidean distance measure (EDM) for quantifying distance between two categories. One possible representation is:

$$E(A,B) = d(c_A, c_B) - (r_A + r_B) \tag{3.3}$$

In this equation c_A and c_B are the center points of categories A and B. r_A and r_B are the radii of A and B. To calculate these values we can use the formula:

$$r_A = \frac{1}{m} \sum_{i=1}^{m} d(c_A, p_i) \tag{3.4}$$

Here m is the total number of points p_i and is a point in category A.

3.10.1.2 Manhattan distance

The Manhattan distance is similar to the Euclidian distance, but the distance between points p_i and q_i is calculated via a different equation.

$$m(p,q) = \sum_{i=1}^{n} |p_i - q_i| \tag{3.5}$$

Manhattan distance measure (MDM) can be defined as follows:

$$M(A,B) = m(c_A, c_B) - (r_A + r_B) \tag{3.6}$$

3.10.1.3 Hausdorff distance

In mathematics, the Hausdorff distance, or Hausdorff metric, also called Pompeiu–Hausdorff distance Birsan and Tiba (2005). It measures how far two subsets of a metric space are from each other. It turns the set of non-empty compact subsets of a metric space into a metric space in its own right. It is named after Felix Hausdorff and Dimitrie Pompeiu.

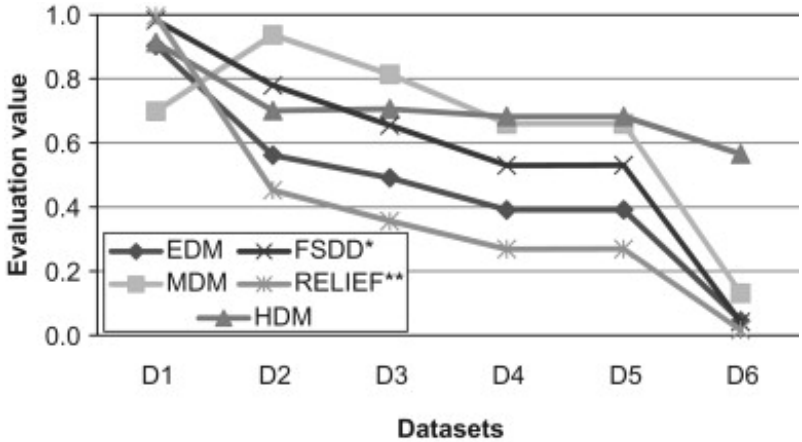
$$\forall x_1 \in A, D(x_1, B) = min_{x_2 \in B}\{d(x_1, x_2)\} \tag{3.7}$$

$$h(A,B) = max_{x_1 \in A}\{D(x_1, B)\} \tag{3.8}$$

$$H(A,B) = max\{h(A,B), h(B,A)\} \tag{3.9}$$

These previous 3 equations display the needed steps for calculating the Hausdorff distance between set A and set B.

$D(x_1, B)$ is the distance between a point x_1 in sets A and B, and $d(x_1, x_2)$ means the distance between the two points x_1 and x_2. The directed function $h(A,B)$ refers to the distance between set A and set B, and $H(A,B)$ is also the distance between sets A and B. The Hausdorff Distance Measure (HDM) uses $H(A,B)$ to calculate the distance among categories of a dataset.



Measure	D_1	D_2	D_3	D_4	D_5	D_6
EDM	0.9028	0.5622	0.4917	0.3913	0.3918	0.0463
MDM	0.6988	0.9366	0.8140	0.6600	0.6600	0.1320
HDM	0.9130	0.7009	0.7061	0.6829	0.6829	0.5660
FSDD*	0.9827	0.7794	0.6544	0.5297	0.5306	0.0421
RELIEF**	0.9959	0.4525	0.3566	0.2695	0.2695	0.0165

(*β=2,**we run 200 iterations for experiment)

Figure 3.3: Evaluation of distance metrics for dataset classification Oh (2011)

3.10.2 Comparison

The measures presented before were analysed (alongside other metrics) by Oh (2011). They performed a series of analysis tests on 6 different datasets. The quality of this sets can be describe as follows: $D_1 > D_2 > D_3 > D_4 > D_5 > D_6$. The results can be seen in Fig. 3.3

The relevant data to our study is the one related to the EDM (Euclidean distance measure), MDM (Manhattan distance measure) and HDM (Hausdorff Distance Measure). As we know that the datasets are numbered by highest quality to lowest quality, we can see that EDM proved more correct tham the others, with MDM having some problems with quantifying the quality of the 1st dataset and HDM performing the worst, being unable to identify major diferences betewenn the qualities if the datasets 2 to 5.

The other two methods displayed are used in feature selection.

3.11 Common dataset problems

Other metrics exist that should be analysed when asserting the quality of a dataset. The usual suspects are overfitting, underfitting, missing data and data imbalances. In the following sections, we will look at these problems.

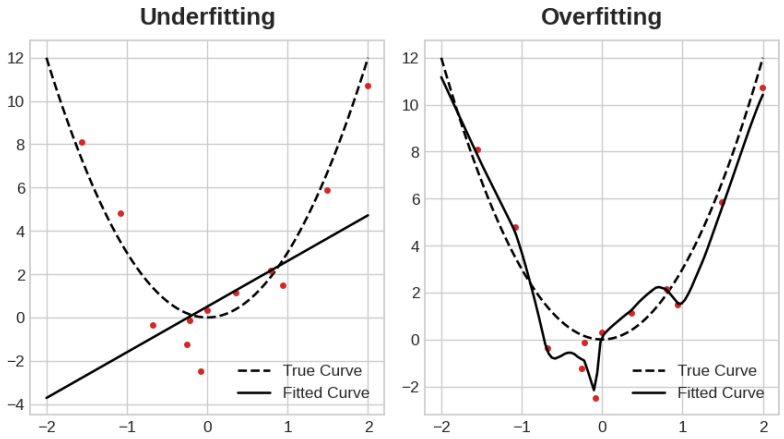


Figure 3.4: Underfitting and Overfitting Holbrook R. (2020)

3.11.1 Overfitting and Underfitting

Overfitting might appear when a spurious pattern in the original dataset is captured and replicated, increasing its frequency. This expands the amount of unnecessary random noise in the dataset. ML models trained with overfitted datasets are negatively impacted performance-wise. The random fluctuations of data are picked and learned as recurring moments.

Underfitting is the opposite of overfitting. It happens when the synthetic dataset does not capture enough relevant information from the original. Original metrics are not replicated, leading to a dataset with different distributions and metrics. ML models that used underfit datasets displayed poor performance ratios.

A graphical representation of overfitted and Underfitting can be found in Fig. 3.4

3.11.2 Missing Data

Missing data is a problem that might also affect the quality of a dataset. It happens when no data value is stored for the variable in an observation. Missing data can significantly affect the conclusions that can be drawn from the data.

Of course, when looking at data generation, two possible origins of missing data arise. It could be a bug of the generation software that can ultimately be corrected with due IT development, or it could be a problem existing in the original dataset used for the generation.

In actual datasets, missing data can occur due to a series of factors, with the most common being human error Laberge (2011).

3.11.3 Data Imbalances

Imbalanced data refers to those types of datasets where the target class has an uneven distribution of observations, which is a problem in cases where small observations are relevant despite their

minimal existence. For example, this can be seen in finance datasets for fraud detection, where the amount of fraud observations is very diminutive concerning the general data.

Mechanisms to prevent overfitting might increase the amount of imbalance in the dataset as relevant data with minor occurrences could be considered noise.

3.12 Conclusion

In this section, we took a look at a series of methodologies and algorithms used in dataset generation. This analysis provides a series of metrics that can be helpfull for the development phase.

However, bibliographic analysis of several dataset generation articles used different methods to generate datasets for different ends. We can see that each study develops its generation process per the final utilitarian purpose.

However, when looking at our generation, it should be noticed that application on the existing generator may not be optimal while these methodologies are helpful. The generator currently uses a configuration file that defines a series of restrictions and rules used in the synthesis. We can then see that the generation process in our case consists of either an optimization problem to obey all restrictions while creating a dataset in the most optimal way possible or a simple generation with statistical distribution in mind; everything depends on the configuration imputed by the user.

Looking at classification, one problem also arises. Comparisons of applied improvements can be analyzed by looking at datasets currently generated and datasets generated at the end of this study using the same parameters. However, as the outcome of the generation depends on the user inputs, the system should allow for the generation of datasets of worse qualities.

Chapter 4

Problem and proposed solution

The problem at hand is the extraction and inclusion of meta features on the generation process of the Synthetic Dataset Generator SNOOKER. This generator was initially produced by this study's second supervisor Leonardo Silva Ferreira. Simple generators face the problem of not generating realistic enough data or disclosing private information [Goncalves et al. (2020b)] when using a real dataset as a basis for the generation process.

The generator interface should allow users to select a variety of meta features (explored in Chapter 2) that would be used in the generation process and be present in the final synthetic dataset, allowing for the creation of realistic datasets.

When picking a generation algorithm, the first step will be to analyse the current generator and check its' methodology. If the current algorithm is not satisfiable enough, a new implementation will have to be made.

Some evaluation metrics should also be applied, and their results should be made available for the user at the end of the generation process.

4.1 Snooker - An Overview

Before analysing the problem at hand, we should first take a look at the original generator, trying to understand it's internal workings. SNOOKER was developed to bridge the availability gap for Helpdesk's datasets.

In this section we will present a general overview of the generator, talking about it's main thecnologies, features and the characteristics of it's synthetically generated datasets.

4.1.1 Technologies

SNOOKER was developed using Python and several of its libraries. Python[1] is an interpreted, high-level programming language considered the standard for exploratory interactive and computation-driven scientific research [Millman and Aivazis (2011)]. The user-generation interaction was handled by creating a graphical user interface (GUI).

[1] python.org

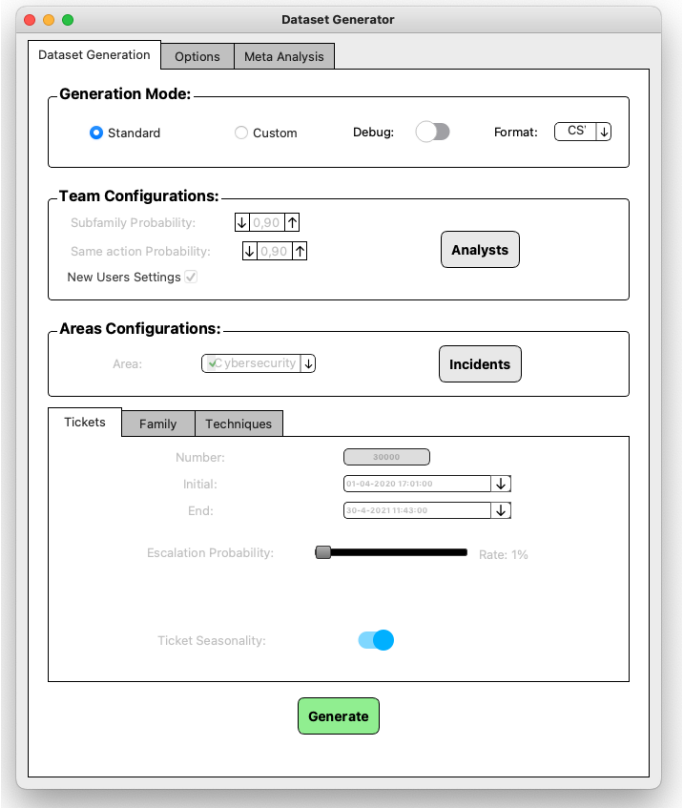


Figure 4.1: SNOOKER's User Interface

The user interface of the generator was developed using PyQt. PyQt[2] is a combination of Python bindings for cross-platform applications that combine all the strengths of Qt and Python [Siahaan and Sianipar (2019)]. This enables the development of GUI elements using Python Code. A print of the generator's GUI can be found in Figure 4.1.

The user can use the standard parameters in the presented interface or personalize the generation inputs as he wishes, fumbling with several interface options. Upon clicking the Generate button, after a short delay, a synthetical tabular dataset is generated.

SNOOKER: A DataSet GeNeratOr fOr HelpdesK SERvices was developed with both these technologies. A change in technologies on a study aiming to enhance the original generator would start the process from scratch, which would not be a viable option.

4.1.2 Features

The primary objective of SNOOKER was to enable synthetic helpdesk dataset generation. This dataset would take into account a series of features. The main features included in the generator are ticket management, team management and area management

Tickets should be able to be customized (quantity, creation time, type, etc.), scheduled, treated (generation of appropriate treatment methods) and replicated if needed.

[2] pypi.org/project/PyQt5/

The team management module handles the team's statistics (shift management, ticket percentages, etc.) and the singular team member information (general data connected to each member).

Lastly, the area management feature takes care of customizable incidents that might happen in each domain previously described.

It should also be noted that the generator uses as input not only the information provided by the user in the generator's UI but also data from a YAML configuration file.

4.1.3 Synthetic Datasets

In this section, we will look at the final output from SNOOKER's generation process, explaining the information in each column of the dataset. The information regarding SNOOKER's outuputed dataset can be found on Appendix A.

4.2 Looking at our objectives

Looking back at Chapter 1 we should take a new look at the questions asked.

How is the current generator working? As we previously analysed in this chapter, the generator extracts some information from an existing dataset. It combines that information with a configuration file with a series of parameters. Generation uses simple probabilistic distribution models to generate and populate the synthetic dataset.

What selection of meta features should be calculated from the data in the synthetic dataset? In Chapter 2, we extracted several meta features that could be used. The final selection of meta features extracted from our datasets can be found in Chapter 5.

What generation methods can we use? This was answered in 3. Several options exist, but the generation used in SNOOKER was not touched after closer inspection.

Can we include specific values of meta features inside the generation process? This is touched on in the final development chapter of this study (Chapter 6), where we study and discuss the possibility of including specific values of meta features inside the generation process.

What optimisations can be applied to the generator? Allied with the meta feature extraction module, we included a full fledged meta feature extraction feature into the generator. More information regarding this feature can be seen in Chapter 5.

4.3 Schedule

Development of the final product will go through several steps that should involve a variety of tasks. To help with the planning a Gantt diagram was created. It can be found in figure 4.2.

Task 1 is the analysis of the current code. Taks 2 is the implementation of improvements to the dataset generation process. Task 3 will explore the implementation of some meta-feature selection. Task 4 regards quality validation of generated datasets. Taks 5 will look at general improvements to the generator, namely in UI/UX. Finally Taks 6 is the writting of the dissertation and scientific paper.

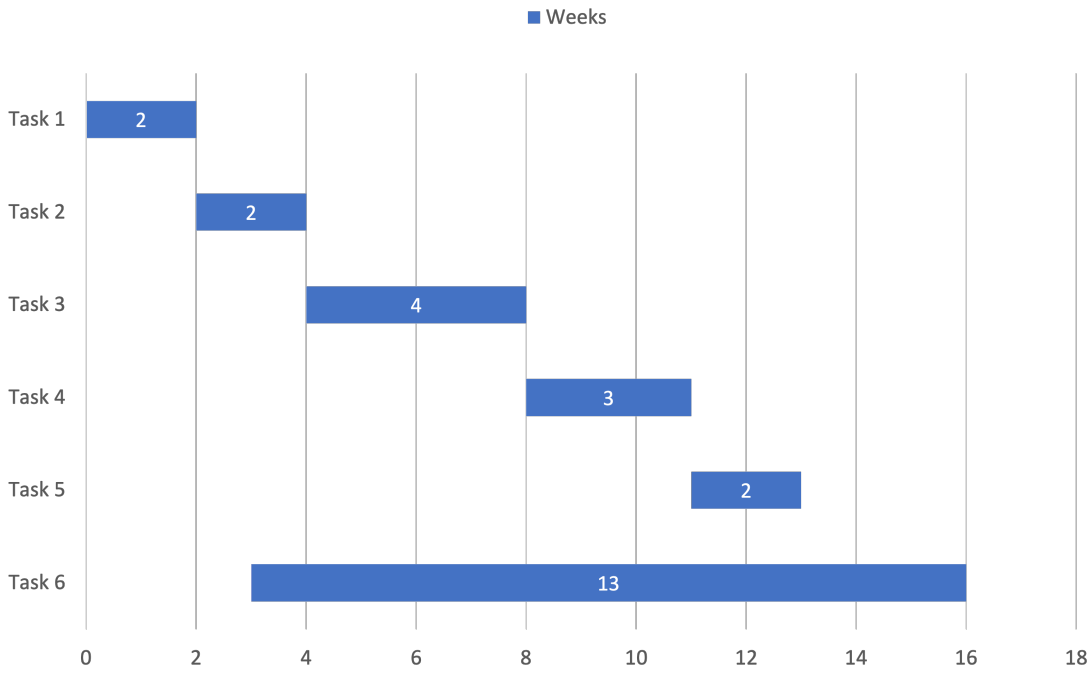


Figure 4.2: Gantt diagram for the developed process

4.4 Risk Analysis

The development of this project may face a series of risks. These risks could hinder the development process of the tasks previously depicted. We will now analyse a series of potential risks.

To start, we have to look at time constraints. The task at hand is challenging, and the short amount of time available to complete it may make it impossible to deliver every single proposal task at the end.

Hardware limitations may also impact development. As some generation processes and meta-features have high computational costs, an inefficient appliance may delay the development, especially during trial-and-error periods.

One other aspect that might impact development is the know-how. Insufficient know-how from the student may delay the development of the project.

4.5 Conclusions

During this chapter we introduced the problem at hand during this study. That was achieved by looking at the requirements of the project. We also created groups of tasks and dividem them by weeks.

We also performed a risk analysis, depicting subjects that might delay the development phase.

The chapters that follow depict the development journey. We hope to arrive at successful results by the end.

Chapter 5

Classification

The first addition to SNOOKER: A DataSet GeNeratOr fOr HelpdesK SERvices was the inclusion of a classification model. The objective was to allow users to extract meta-features from the generated datasets, making it possible to extrapolate the quality of the synthetic data. It was also essential to enable classifications of other datasets, making the product usable for classification tasks for datasets external to the generation.

Past development of the generator was made using Python. As the classification feature was idealized as an increment to the current product, it used the same language and foundation.

The evolution of this advancement proved quite troublesome; the development steps and the results are described in this chapter.

5.1 Meta feature extraction

Development started with manually coding the extraction of meta features from small (about 100 rows) synthetic datasets. A list of meta features to be included in the analysis was made, considering the information acquired in Chapter 2.

Testing started with the simple meta feature analysis of a dataset. As the generator creates synthetic datasets in a .csv (Comma-separated values) format, a way to analyse such data was necessary. One major issue for would-be statistical Python programmers in the past has been the lack of libraries implementing standard models and a cohesive framework for specifying models. Pandas is a Python library of rich data structures and tools for working with structured data sets. It is commonly used in statistics, finance, social sciences, and many other fields McKinney et al. (2011). It intends to close the gap in the richness of available data analysis tools between Python, general-purpose systems and scientific computing language, and the numerous domain-specific statistical computing platforms and database languages. Pandas was therefore used.

General meta feature analysis was relatively simple to implement due to being directly observable from the dataset, representing basic information and being the most explicit set in terms of computation cost. Problems arrived when looking at the analysis of statistical meta features.

The development of models to analyse every statistical meta feature was slow. Continuing down that path would limit the scope of the work given the existing time frame and delivery dates. The solutions available were to limit the number of analysed meta features (extracting only a hand full per family) or to rely on external tools to perform that same task. The latter option was chosen. After consideration, the Python library Python Meta Feature Extractor (pyfme[1]) was selected and used to perform meta feature extraction tasks.

The pymfe library provides a comprehensive set of meta-features implemented in python. According to the library developers, the package brings cutting-edge meta features, following recent literature proposals. The pymfe architecture was idealised to systematically make the extraction, which can produce a robust set of meta features. The review performed by Alcobaça et al. (2020) agreed with our decision and proved a suitable validator. As such, development moved forth using this library.

Pymfe can extract several families of meta features from a dataset. Those families are depicted in the following list, with tables for every family describing every meta feature presented in Appendix B.

- **General**: General information related to the dataset, also known as simple measures, such as the number of instances, attributes and classes (Table B.1).
- **Statistical**: Standard statistical measures to describe the numerical properties of data distribution (Table B.2).
- **Information-theoretic**: Particularly appropriate to describe discrete (categorical) attributes and their relationship with the classes (Table B.3).
- **Model-based**: Measures designed to extract characteristics from simple machine learning models (Teable B.4).
- **Landmarking**: Performance of simple and efficient learning algorithms (Table B.5).
- **Relative Landmarking**: Relative performance of simple and efficient learning algorithms.
- **Subsampling Landmarking**: Performance of simple and efficient learning algorithms from a subsample of the dataset.
- **Clustering**: Clustering measures extract information about dataset based on external validation indexes (Table B.6).
- **Concept**: Estimate the variability of class labels among examples and the examples density (Table B.7).
- **Itemset**: Compute the correlation between binary attributes (Table B.8).
- **Complexity**: Estimate the difficulty in separating the data points into their expected classes (Table B.9).

[1] pymfe.readthedocs.io

The vast amount of families and collections of meta features inside those families proved more than enough to use in our classification model. Some limitations were applied non the less.

Right in the start, the library requires the definition and extraction of the first column from the dataset to analyse CSV data. The definition of the column name can be surpassed using pandas' ability to read information from CSV data. The program can automatically grab the first column from a dataset. The generator already creates an expendable 'ID' column in its first position, which can be consumed without affecting the meta feature analysis result. However, this does not apply to other possible datasets, forcing users to edit datasets not generated by SNOOKER to get valid results. In the snippet bellow an example of code used to read the CSV file while consuming the first column can be found.

Listing 5.1: Importing the csv dataset with ID column

```
df = pd.read_csv(path, sep=";", index_col=False)
X, y = df.drop(df.columns[0], axis=1), df[df.columns[0]]
```

Another simple limitation was defining a sep for the CSV data. In the case of SNOOKER, CSV is generated using semi-colons, which was the used value for steps.

The classification feature was intended as the main addition to SNOOKER. It needed to be integrated into the generator's interface, making it possible to be accessible to users in a simple way. The development of the interface will be tackled in the next section.

Each meta feature family selected outputs a file at the end of the analysis. This file contains the path to the analysed dataset, the results of each meta feature inside that family and the time it took to perform that meta feature extraction. A transcription of an actual output file from a meta feature extraction analysis can be found in Appendix C.

5.2 Interface

As the meta feature extraction feature is an inclusion to the already existing generator, its interface was appended to the generators' frontend. This interface was, therefore, developed using the Pyqt framework.

"Python is probably the easiest to learn and nicest scripting language in widespread use, and Qt is probably the best library for developing GUI applications. The combination of Python and Qt, "PyQt", makes it possible to develop applications on any supported platform and run them unchanged on all supported platforms" (quoted from Summerfield (2007)).

The interface went through an idealization period, and the concept and objectives changed. Originally the interface was supposed to have a button that would trigger a file selection. This selection would generate a series of rows containing that file's column names. Next to each column name, we would have a toggle feature to select columns to use in the calculations. It would also be

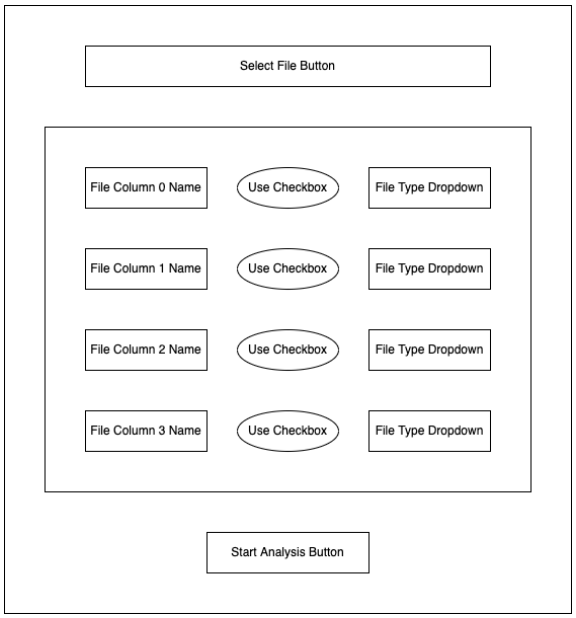


Figure 5.1: Original idealization of classification feature interface

accompanied by a dropdown containing options to select the data type in that column (Numeral, Text, Date, etc.). The concept for the original interface can be found in Figure 5.1.

The meta feature calculations were coded by hand, using a series of measures requiring user input on each column's type of data. The system could not distinguish between categorical and numeric features, creating error breaks in the analysis. Although implementation started, it was quickly dropped, and no visual record was kept of its development.

When moving to the pymfe library, the interface changed to focus on more relevant features, the selection of meta-features. The current implementation keeps the file selection button. Still, it forgoes the column toggle and data type selection, using a simple toggle of meta feature familes. Midway through development, SKOOKER's interface theme was changed, and the classification module was adapted to fit the change. The original version can be found in Figure 5.2. In contrast, Figure 5.3 depicts the final interface.

5.3 Library Limitations

Opting to use the pymfe library proved to be the right choice, as it propelled the development of a large number of steps. It was possible to classify datasets with a giant galery of meta features. However, some limitations were accompanied by the library.

One such limitation was quickly found. The time needed to perform meta feature extraction analysis. While small datasets took only a few minutes to analyse, as the size of the dataset increased, so did the time the analysis took to finish. Several tests were made and data extracted to see what families of meta-features were causing the output delay. That data can be found in Table 5.1. It should be noted that way more tests were made during the development of this study.

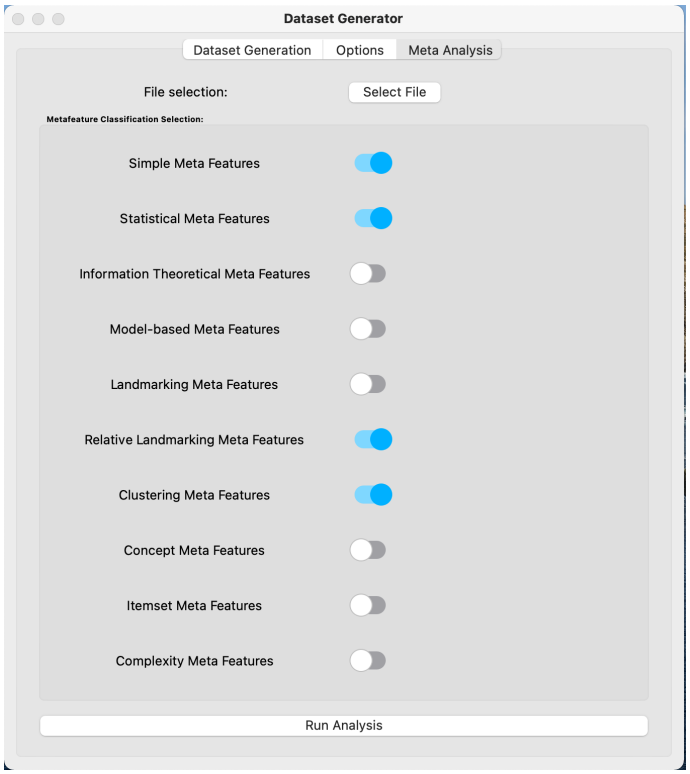


Figure 5.2: Classification feature interface, initial theming

Nonetheless, results were sometimes incomplete and/or used datasets with different amounts of columns. The datasets chosen to perform this first analysis were all generated by SNOOKER.

Table 5.1: Testing made on the classification feature, using several datasets.

Meta Feature family	DS1	DS2	DS3	DS4
Simple Meta Features	<1s	<1s	2.21s	32.70s
Statistical Meta Features	1.5s	2:55 min	28:16 min	51:31 min
Information Theoretical Meta Features	2.18s	6s	24.82s	1:26 min
Model-based Meta Features	<1s	<1s	4.47s	>2h
Landmarking Meta Features	<1s	<1s	<1s	38.73s
Relative Landmarking Meta Features	<1s	<1s	6.25s	>2h
Clustering Meta Features	<1s	9.12s	3:13 min	>2h
Concept Meta Features	<1s	<1s	8.52s	57.46s
Itemset Meta Features	1.6s	26.04s	5.14 min	>2h
Complexity Meta Features	53.62s	1:24:53 h	>12h	>12h

DS1, DS2, DS3 and DS4 were SNOOKER-generated datasets with 100, 500, 1000 and 5009 entries each, respectively. From the get-go, it can be seen that Statistical and Complexity Meta-Analysis were outliers. Their analysis took more time and resources than the rest of the meta features. Initially, the sentiment was that hardware limitations were accountable for these analysis

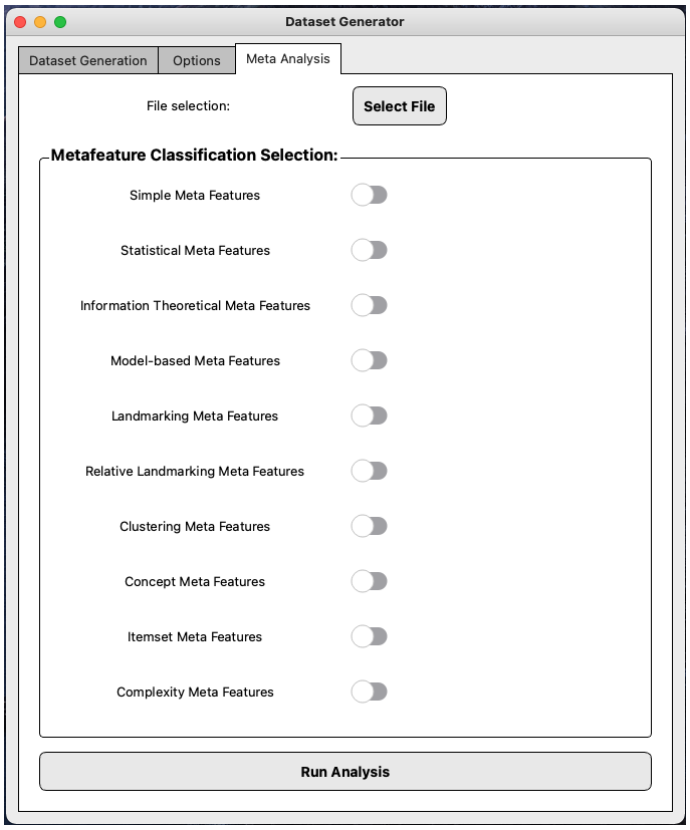


Figure 5.3: Classification feature interface after a theme change

times. Yet, when testing the classification feature on a much more powerful machine, the end result times were similar to those achieved locally. It could then be concluded that the problem was not the computational power. Still, the library took some time to make the required calculations to perform this analysis.

Regular analysis was kept running for a maximum amount of 2 hours per meta family. However, complexity analysis took way more time from the beginning. We tried to see how far we could go on their analysis, making 2 tests with 2 different datasets with a time limit of 12 hours. Both these tests were unable to lead to results. Complexity analysis calculation times were so egregiously large that further analysis of these meta features was dropped. The option to analyse them is still present in the current product. Still, time constraints made their study impossible to pursue.

While Complexity Meta-Analysis was dropped from further development, the same could not apply to Statistical Meta Features. The idea was to test each statistical meta feature extraction. We would save each analysis's time and remove more time-consuming meta features, making the analysis faster. We used a series of simple publicly available datasets (from source: https://perso.telecom-paristech.fr/eagan/class/igr204/datasets). Several other datasets were also used in the analysis, but as dataset dimensionality grew, so did the time required to analyse all 19 meta features plus the full analysis. Runtime of each analysis is presented in Table 5.2.

Table 5.2: Statistical meta analysis duration

Meta Feature	cars	cereal	CausesOfDeath_France_2001-2008
can_cor	50.97s	0.48s	13:07min
cor	52.99s	0.46s	12:31min
cov	52.40s	0.46s	12:25min
eigenvalues	52.54s	0.47s	11:25min
g_mean	52.55s	0.46s	11:37min
gravity	52.16s	0.45s	11:38min
h_mean	52.11s	0.45s	11:38min
iq_range	1:44min	0.46s	11:31min
kurtosis	1:07min	0.51s	11:36min
lh_trace	54.09s	0.45s	11:34min
mad	52.51s	0.45s	11:34min
max	52.51s	0.44s	11:34min
mean	51.85s	0.44s	11:36min
median	51.88s	0.44s	11:33min
min	51.57s	0.45s	11:31min
nr_cor_attr	51.03s	0.44s	11:29min
nr_disc	50.70s	0.44s	11:30min
nr_norm	51.09s	0.45s	11:29min
nr_outliers	50.99s	0.44s	11:33min
p_trace	50.77s	0.44a	11:35min
range	51.02s	0.44s	11:35min
roy_root	52.16s	0.44s	11:36min
sd	51.04s	0.45s	11:40min
sd_ratio	50.71s	0.45s	11:37min
skewness	50.95s	0.49s	12:05min
sparsity	50.67s	0.42s	11:24min
t_mean	50.95s	0.43s	11:13min
var	51.76s	0.42s	11:10min
w_lambda	50.65s	0.43s	11:08min
FULL ANALYSIS	51.44s	62.65	11:25min

The outcome of this analysis was unexpected. Even with some calculations taking more time than others, the differences are predominantly negligent. In fact, the last row shows that the library takes roughly the same time to perform a complete analysis as it does on every individual. With this result, proceeding with analysing more enormous datasets proved unnecessary. This left us with no choice but to accept the high wait times when researching some meta feature families.

The classification module was considered finished with the decision that no further improvements could be made.

5.4 Conclusion

In this chapter, we took a look at the development process for the meta feature Classification module, created as an addition to SNOOKER. We started the chapter by displaying our path until we arrived at an agreeable solution for the meta feature extraction problem. After trying to make all the calculations ourselves, we used the pymfe library to accelerate the process, adding an enormous amount of meta features to the analysis pool.

In the second section, we explore the development of the module's interface, developed using PyQt5. The interface went through a series of alterations and experimentations until we arrived at the current presentable stage.

Lastly, we did some experiments on the final product, identifying problems and finding a way to correct them. Unfortunately, no way to solve the detected problems was found. Later in this document, you can find information regarding this aspect of the project.

Ultimately the Classification feature ended up going further than what was initially planned. The original concept was to develop this meta-analysis from scratch, getting only some examples from each family. Ultimately, pyfme allowed us not to get some meta feature results but tens of results per analysis. On the matter of optimization, the feature does, however, underperform.

Chapter 6

Meta Feature Integration

In the following chapter, we will explore the conclusive part of this dissertation: the study and methodology for including meta features in the generation process.

Synthetic datasets with specific meta features can help generate better datasets. For starters, randomly generated datasets cannot ensure a good distribution of meta-features, leaving sparsely populated areas inside the "meta feature space". Secondly, generating datasets with specific values of meta features allows more controlled experiments that might lead to conclusions about the usefulness of particular meta features Reif et al. (2012).

However, using the current generator and applying meta feature forcing proved too tacky and time-consuming. As seen in Chapter 4.1, SNOOKER generates datasets with helpdesk ticket solving; the final dataset contains a lot of information (regarding ticket generation, team management, and scheduling). Every entry was too large and had too many variables to correctly be able to be influenced on a global scale. Dimensionality was, therefore, a problem. As the dataset's final dataset could be understood as a kind of DATALAKE dataset, what was decided was to study the insertion of a meta feature on one of those modules.

We developed some experiments using a subset of a dataset output from SNOOKER, using columns related to ticket generation, reducing the dimensionality of the dataset. Experimentation was simple and purely exploratory. A more prominent focus was taken on exploring possible meta features to be included and methods that would facilitate that inclusion.

What was left was to choose a set of meta features, study their characteristics and see how to better include them in the generation process. While a collection of meta-features is presented below, it was unknown if their inclusion would be considered helpful at the moment of selection. Each meta feature was analysed on its individuality, taking into no account the result from a combination of forcing these meta features in the same generation process.

6.1 Dimentionality

Dimensionality in statistics refers to how many attributes a dataset contains. In the case of tabular datasets, this data could be represented in a spreadsheet, with one column representing each di-

mension. In this study, we analysed dimensionality as the ratio between the number of instances and attributes in the dataset. Therefore the attr_to_inst meta feature from the general family was considered a possible meta feature to represent dimensionality.

High-dimensional data relates to a dataset where the number of Attributes (or columns) is more significant than the number of Instances (or lines). For example, we can regard a dataset with twenty attributes and merely seven instances as a high-dimensional dataset. As we can witness, the number of features is larger than the number of observations. High dimensionality datasets are challenging to address. It is also problematic to categorise such high dimension data. High dimensionality also carries some redundant features, ushering to data losses Ray et al. (2021).

It should be mentioned that even if a dataset has many attributes, if the number of instances surpasses that point, it cannot be regarded as a high dimensionality dataset Ziegel (2003).

Regarding the inclusion of a specific dimensionality value to a synthetic dataset, how can we force a certian value of attr_to_inst during generation? To start it off, let's take a look at the needed calculations. Let's use α to represent the number of attributes and n to represent the number of instances.

$$attr\_to\_inst = \frac{\alpha}{n} \tag{6.1}$$

Now to proceed, some basic rules were applied. Both α and n need to be natural integer numbers more significant than zero. We also decided that adding or removing entries was fair game. However, doing the same for the number of columns could remove information essential to the dataset. Some rules were added to guarantee the preservation of the original dataset's attributes.

$$\alpha = O + \delta \tag{6.2}$$

Where O is the original dataset's column count, and $/delta$ represents the number of added columns to the synthetic dataset.

Two more conditions should be added to certify that no error occurs during the calculations.

$$\delta \geq 0 \tag{6.3}$$

$$\alpha, O, n \geq 1 \tag{6.4}$$

And this last one guarantees that all values are positive integers.

$$\alpha, n, O, \delta \in \mathbb{I} \tag{6.5}$$

Having all the needed formulas, we can concur that we need to solve an optimization problem to achieve a desirable dimensionality value. Using the following equations that come from simplifying the previously shown ones, we should be able to solve the optimization problem that leads to the desired result.

$$\begin{aligned} D &= \frac{\alpha}{n} \\ \alpha &\geq O \\ \alpha, n, O &\geq 1 \\ \alpha, n, O &\in \mathbb{I} \end{aligned} \tag{6.6}$$

Knowing our attr_to_inst value (represented in the previous equation by D), we can solve the optimization problem by getting the values of α and n that are a solution to the problem. According to the results, the generator can tweak its parameters to grow vertically and/or horizontally.

6.2 Skewness

In probability theory and statistics, skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean Dean and Illowsky (2018). The skewness value can be positive, zero, negative, or undefined. It describes the distribution shape of the measured values in terms of symmetry Rivolli et al. (2019).

A visual representation of skewness can be found in Figure 6.1. In this figure, the first graphic depicts a left skewness, informing us that in this case the mean of the values is smaller tham the median, that in turn is smaller tham the mode. The central graph depicts a symetrical normal, where the mean, mediam and mode all share the same value. Finaly the right figure depicts a skewed right graph. In this types of skewes, the mean is greater tham the medium that is by turn greater tham the mode.

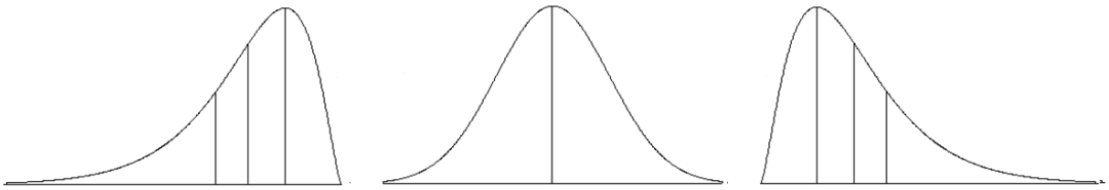


Figure 6.1: Visual representation of skewness. Removed from Doane and Seward (2011)

The skewness of attributes is calculated throug the expression:

$$skewness_x = \frac{m_3}{sd_x^3} \tag{6.7}$$

Where sd_x^3 and m_3 are extracted from the following formulas:

$$sd_x^3 = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \tag{6.8}$$

$$m_j = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^j \tag{6.9}$$

Applying skewness to the generation process can actually require only a small tweeking in dataset generators that already use normal distribution to create syntetic datasets. In order to generate skewness to either side we just need to ensure that the mean value stays the same as it would on normal generation, while shifting the mode and median to the other side. This is donne by generating a large but focused amount of values on the opsite side of the skew, while creating a spread out number of entries on the side of the skew, generating a skewness tail on the desired side of the aisle.

6.3 Kurtosis

Kurtosis is a "measure of the "tailedness" of the probability distribution of a real-valued random variable" (adapted from Ramya and Nanda (2017)). Like skewness, kurtosis describes the shape of a probability distribution. There are different ways of quantifying it for a theoretical distribution and corresponding ways of estimating it from a sample from a population. Other measures of kurtosis may have varied interpretations. As Balanda and Macgillivray (1988) said, and I quote: "It is best to define kurtosis vaguely as the location- and scale-free movement of probability mass from the shoulders of a distribution into its center and tails and to recognize that it can be formalized in many ways."

Kurtosis is sometimes confused with a measure of the peakedness of a distribution Darlington (1970). However, kurtosis is a measure that describes the shape of a distribution's tails concerning its overall format. All kurtosis measures are analogised against a typical normal distribution or bell curve.

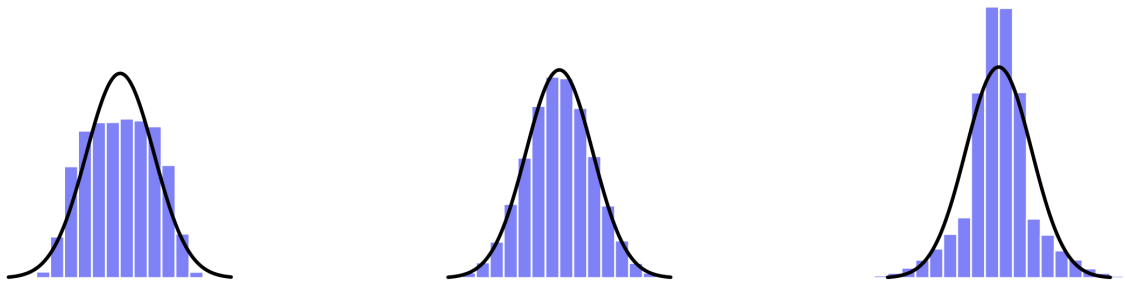


Figure 6.2: Visual representation of kurtosis in histogram form. Removed from Navarro and Foxcroft (2019)

As depicted in Figure 6.2 kurtosis can be represented in 3 types. The left graph is called a platykurtic distribution. These distributions have short tails with flattened curves. The middle graph represents a mesokurtic kurtosis distribution where kurtosis is almost exactly 0. Finally, on the right, we have a leptokurtic kurtosis, where the values of the normal distribution peak. The modal curve represented by a black curve in the figure has a kurtosis value equal to 0.

Kurtosis can be calculated using the folowing formula:

$$kurt_x = \frac{m_4}{sd_x^4} - 3 \tag{6.10}$$

Where m_4 is calculated using the formula presented in Equation 6.9

As with Skewness integration, adding desired skewness values could be done by changing the parameters of the output entries. For negative skewness, we need to reduce the number of entries closer to the mean of the distribution while assuring that output continues to be a normal distribution (only flattened in comparison to the original). Positive skewness would do the inverse, increasing the production of entries closer to the mean, creating a peak of values.

6.4 Analysis

After looking at several options for including meta features in synthetic dataset generation, let's now see how valuable these techniques would actually be.

We started by looking at dimensionality, using the general meta feature attr_to_inst to represent the dimensionality metric. However, how useful is this method?

The dataset could expand both vertically and horizontally. Vertical expansion is already used in synthetic dataset generation. It would simply change the number of entries to the dataset. Definition of the number of entries is an essential feature required in dataset generation, typically using user input to get this value. On the other hand, horizontal growth is a more particular feature that isn't generally available on generators using already existing datasets as a basis. However, a critique should be made of the final synthetic dataset. How valuable would the extra randomly generated data actually be? Especially when looking at machine learning, adding useless information would actually be prejudicial.

We concur that dimensionality should be analysed, and datasets with high values of this metric should be discarded or improved upon to add new data entries. However, forcing a set value for dimensionality is not a helpful feature in dataset generation.

Moving on to kurtosis and skewness. Assembling datasets with set values for these meta-features would facilitate the creation of more diverse and valid datasets with different characteristics, all with the same original dataset as a basis.

It should, however, be noted that forcing these meta-features would require an internal change to the original input parameters in the generator. This means that a lot of user tweaking could be internally changed and/or ignored.

6.5 A case on landmarking integration

We arrived at strange conclusions regarding landmarking meta features when analysing possible meta features to validate their inclusion in synthetic dataset generation processes. These meta features use algorithms to characterise datasets. To include specific values of these meta features,

we would need to create entries while also knowing how well the entire dataset would behave when fronting said algorithm.

A funny analogy to this problem would be to drive a train while building the train tracks, taking into account the performance of every vehicle on the railroad system. The complexity of this problem vastly overshadows the scope of this study.

Landmarking inclusion would, nonetheless, be quite valuable, especially in cases involving machine learning. Users could generate datasets with good and bad results, teaching the MLM to follow datasets with good results. However, their inclusion could also prove troublesome as data could serve the dataset way too perfectly.

Generating datasets with set algorithmic results would lead to synthetic datasets riddled with overfitting problems.

6.6 Conclusion

In this chapter, we looked at the original approach taken to test some of the methodologies later described in the chapter.

We then analysed some possible meta-feature inclusions to the generation process. Dimensionality, skewness and kurtosis were studied in this phase. The validity of inclusion of these meta features was analysed in the following section, where we discarded Dimensionality forcing and favoured kurtosis and skewness integration.

Lastly, we took a look at the problem of forcing landmarking into synthetic dataset generation processes.

When looking at what was achieved in this part of the study and what was initially idealised, it is impossible not to feel underwhelmed. We analysed ways to force certain meta features into the generation process and proved that this implementation can be done. However, the same does not apply to other families when looking at simple and statistical meta features. We conclude that a complete study on the subject should be made to get to any plausible conclusion on including meta features with more tricky calculations.

Chapter 7

Conclusion

During the development of this study, we tried to tackle the theme of 'Meta Feature classification and insertion on a Synthetic dataset generation process.' Having SNOOKER as the original basis for this study (after a small introduction in Chapter 1), we started Chapter 2 by analysing the concept of meta feature, exploring the subject and presenting the most commonly known families of meta features. While groups of meta features are not uniformly described, the list presentive is extensive and varied enough to correctly characterise a dataset.

The study advanced to dataset generation methodologies in Chapter 3. While explored, this subject was not developed further during the development phase. This and the previous part described earlier made our state of the art.

Chapter 4 was dedicated to the project's planning phase. Development was divided into tasks, and those tasks were grouped in weeks. Reality ended up deviating from the planning during the study, as the focus changed from a core product indentation to an exploratory subject.

The begging of the development phase was spearheaded by the deep dive into SNOOKER. A concise overview can be found in Chapter 4.1.

The Classification feature was appended to SNOOKER. Its development is described in Chapter 5. We used the pyfme library to help us extract as many meta features from generated datasets as possible, allowing for vast classification options. The development of this feature's UI is also referenced in this chapter.

Lastly, in Chapter 6, we explored ways to force specific meta feature results to synthetic datasets. While achieving only favourable results, we conclude that a much more focused study is necessary to dive deeper into the matter.

Ultimately the whole development part ended up not going as far as initially expected. We did not expect so many problems during the planning at the beginning of this study and presented in Chapter 4. Problems that ultimately left us still with much open to exploring. The author idealised this study as an engineering problem, with a core product that would be built upon, but ended up diving more into exploratory subjects, with no concrete answer achieved regarding the inclusion of meta features in generation processes. While we know it is possible to do so, only small results were achieved and not enough to lead to valid proof of concept.

On the other hand, the interface serves as suitable proof of concept for the problem at hand. While the classification of high dimensional datasets is still resource-intensive, the tool can perform each classification task, given enough time.

7.1 Future Works

As said before, the study of meta feature inclusion on generation could benefit from further research, being a possible project to be.

We can now classificate datasets using the meta feature classification tool. We could test performance on machine learning models trained using similar datasets with different meta feature values. This is also an interesting topic that requires further research.

Lastly, we present possible advances to SNOOKER as a product. After exploring SNOOKER, it is easy to see that computing power and runtime are essential factors needed when dealing with the generator. We conclude that SNOOKER would beneficiate from a major overhaul, becoming decentralized. With all the necessary calculations, donne remotely on a server. As meta analysis of some families could take a long time, making them run in a server, sending them to the user on a later date, would prevent users form needing to keep processes running for over 12 hours (in cases of high dimentionality).

References

Hervé Abdi, Dominique Valentin, and Betty Edelman. *Neural networks*. Sage Publications, Inc, 1999.

Arnisha Akhter, Uzzal K Acharjee, and Md Masbaul A Polash. Cyber bullying detection and classification using multinomial naïve bayes and fuzzy logic. 2019.

Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís Paulo F Garcia, Jefferson Tales Oliva, André CPLF de Carvalho, et al. Mfe: Towards reproducible meta-feature extraction. *J. Mach. Learn. Res.*, 21(111):1–5, 2020.

Nantheera Anantrasirichai, Juliet Biggs, Fabien Albino, and David Bull. A deep learning approach to detecting volcano deformation from satellite imagery using synthetic datasets. *Remote Sensing of Environment*, 230:111179, 2019.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. arxiv 2017. *arXiv preprint arXiv:1701.07875*, 30:4, 2017.

Theodoros N. Arvanitis, Sean White, Stuart Harrison, Rupert Chaplin, and George Despotou. A method for machine learning generation of realistic synthetic datasets for validating healthcare applications. *medRxiv*, 2021. doi: 10.1101/2021.02.11.21250741. URL https://www.medrxiv.org/content/early/2021/02/12/2021.02.11.21250741.

Kevin P. Balanda and H. L. Macgillivray. Kurtosis: A critical review. *The American Statistician*, 42(2):111–119, 1988. doi: 10.1080/00031305.1988.10475539. URL https://www.tandfonline.com/doi/abs/10.1080/00031305.1988.10475539.

Besim Bilalli, Alberto Abelló Gamazo, and Tomàs Aluja Banet. On the predictive power of meta-features in openml. *International Journal of Applied Mathematics and Computer Science*, 27 (4):697–712, 2017.

Hans Binder and Henry Wirth. Analysis of large-scale omic data using self organizing maps. In *Encyclopedia of Information Science and Technology, Third Edition*, pages 1642–1653. IGI global, 2015.

T Birsan and Dan Tiba. One hundred years since the introduction of the set distance by dimitrie pompeiu. In *IFIP Conference on System Modeling and Optimization*, pages 35–39. Springer, 2005.

Nathaniel Boggs, Hang Zhao, Senyao Du, and Salvatore J Stolfo. Synthetic data generation and defense in depth measurement of web applications. In *International Workshop on Recent Advances in Intrusion Detection*, pages 234–254. Springer, 2014.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

Gregory Caiola and Jerome P Reiter. Random forests for generating partially synthetic, categorical data. *Trans. Data Priv.*, 3(1):27–42, 2010.

Giovanni Campagna, Silei Xu, Mehrad Moradshahi, Richard Socher, and Monica S Lam. Genie: A generator of natural language semantic parsers for virtual assistant commands. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 394–410, 2019.

Ciro Castiello, Giovanna Castellano, and Anna Maria Fanelli. Meta-data: Characterization of input features for meta-learning. In Vicenç Torra, Yasuo Narukawa, and Sadaaki Miyamoto, editors, *Modeling Decisions for Artificial Intelligence*, pages 457–468, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31883-5.

Stanley Cohen. The evolution of machine learning: past, present, and future. In *Artificial Intelligence and Deep Learning in Pathology*, pages 1–12. Elsevier, 2021.

Jessamyn Dahmen and Diane Cook. Synsys: A synthetic data generation system for healthcare applications. *Sensors*, 19(5), 2019. ISSN 1424-8220. doi: 10.3390/s19051181. URL https://www.mdpi.com/1424-8220/19/5/1181.

Ashish Dandekar, Remmy AM Zen, and Stéphane Bressan. A comparative study of synthetic dataset generation techniques. In *International Conference on Database and Expert Systems Applications*, pages 387–395. Springer, 2018.

Richard B. Darlington. Is kurtosis really "peakedness?". *The American Statistician*, 24(2):19–22, 1970. doi: 10.1080/00031305.1970.10478885. URL https://doi.org/10.1080/00031305.1970.10478885.

Manoranjan Dash and Huan Liu. Feature selection for classification. *Intelligent data analysis*, 1 (1-4):131–156, 1997.

Susan Dean and Barbara Illowsky. Descriptive statistics: skewness and the mean, median, and mode. *Connexions website*, 2018.

David P. Doane and Lori E. Seward. Measuring skewness: A forgotten statistic? *Journal of Statistics Education*, 19(2):null, 2011. doi: 10.1080/10691898.2011.11889611. URL https://doi.org/10.1080/10691898.2011.11889611.

Jörg Drechsler. Using support vector machines for generating synthetic datasets. In *International Conference on Privacy in Statistical Databases*, pages 148–161. Springer, 2010.

David B Dunson and Chuanhua Xing. Nonparametric bayes modeling of multivariate categorical data. *Journal of the American Statistical Association*, 104(487):1042–1051, 2009.

Andrey Filchenkov and Arseniy Pendryak. Datasets meta-feature description for recommending feature selection algorithm. In *2015 Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT)*, pages 11–18, 2015. doi: 10.1109/AINL-ISMW-FRUCT.2015.7382962.

Sachin S. Gavankar and Sudhirkumar D. Sawarkar. Eager decision tree. In *2017 2nd International Conference for Convergence in Technology (I2CT)*, pages 837–840, 2017. doi: 10.1109/I2CT.2017.8226246.

Andre Goncalves, Priyadip Ray, Braden Soper, Jennifer Stevens, Linda Coyle, and Ana Paula Sales. Generation and evaluation of synthetic patient data. *BMC Medical Research Methodology*, 20(1):108, 2020a. doi: 10.1186/s12874-020-00977-1. URL https://doi.org/10.1186/s12874-020-00977-1.

Andre Goncalves, Priyadip Ray, Braden Soper, Jennifer Stevens, Linda Coyle, and Ana Paula Sales. Generation and evaluation of synthetic patient data. *BMC Medical Research Methodology*, 20(1):108, 2020b. doi: 10.1186/s12874-020-00977-1. URL https://doi.org/10.1186/s12874-020-00977-1.

T Hamsapriya, D Karthika Renuka, and M Raja Chakkaravarthi. Spam classification based on supervised learning using machine learning techniques. *DIGITAL WORLD*, 2(04), 2011.

Donald Olding Hebb. *The organisation of behaviour: a neuropsychological theory*. Science Editions New York, 1949.

Tin Kam Ho and Mitra Basu. Complexity measures of supervised classification problems. *IEEE transactions on pattern analysis and machine intelligence*, 24(3):289–300, 2002.

Cook A Holbrook R. Intro to deep learning. Technical report, Kaggle, 2020.

Jorge Kanda, Andre de Carvalho, Eduardo Hruschka, Carlos Soares, and Pavel Brazdil. Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features. *Neurocomputing*, 205:393–406, 2016. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2016.04.027. URL https://www.sciencedirect.com/science/article/pii/S0925231216302867.

Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4551–4560, October 2019.

KATRIN Kirchner, K-H Tölle, and J Krieter. Decision tree technique applied to pig farming datasets. *Livestock Production Science*, 90(2-3):191–200, 2004.

Thomas Kollar, Danielle Berry, Lauren Stuart, Karolina Owczarzak, Tagyoung Chung, Lambert Mathias, Michael Kayser, Bradford Snow, and Spyros Matsoukas. The alexa meaning representation language. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3*, pages 177–184, 2018.

Christian Köpf and Ioannis Iglezakis. Combination of task description strategies and case base properties for meta-learning. In *Proceedings of the 2nd international workshop on integration and collaboration aspects of data mining, decision support and meta-learning*, pages 65–76, 2002.

Oliver Kramer. K-nearest neighbors. In *Dimensionality reduction with unsupervised nearest neighbors*, pages 13–23. Springer, 2013.

Indu Kumar, Kiran Dogra, Chetna Utreja, and Premlata Yadav. A comparative study of supervised machine learning algorithms for stock market trend prediction. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 1003–1007. IEEE, 2018.

Yves Laberge. Advising on research methods: A consultant's companion. *Journal of Applied Statistics*, 38(12):2991–2991, 2011. doi: 10.1080/02664763.2011.559375. URL https://doi.org/10.1080/02664763.2011.559375.

Masitah Abdul Lateh, Azah Kamilah Muda, Zeratul Izzah Mohd Yusof, Noor Azilah Muda, and Mohd Sanusi Azmi. Handling a small dataset problem in prediction model by employ artificial data generation approach: A review. In *Journal of Physics: Conference Series*, volume 892, page 012016. IOP Publishing, 2017.

Daren Ler, Hongyu Teng, Yu He, and Rahul Gidijala. Algorithm selection for classification problems via cluster-based meta-features. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4952–4960. IEEE, 2018.

Guido Lindner and Rudi Studer. Ast: Support for algorithm selection with a cbr approach. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 418–423. Springer, 1999.

Ana C. Lorena, Luís P. F. Garcia, Jens Lehmann, Marcilio C. P. Souto, and Tin Kam Ho. How complex is your classification problem? A survey on measuring classification complexity. *ACM Comput. Surv.*, 52(5), sep 2019. ISSN 0360-0300. doi: 10.1145/3347711. URL https://doi.org/10.1145/3347711.

Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

Geoffrey J McLachlan. *Discriminant analysis and statistical pattern recognition*. John Wiley & Sons, 2005.

Donald Michie, David J Spiegelhalter, and Charles C Taylor. Machine learning, neural and statistical classification. *Ellis Horwood Series in Artificial Intelligence*, 1994.

K. Jarrod Millman and Michael Aivazis. Python for scientists and engineers. *Computing in Science & Engineering*, 13(2):9–12, 2011. doi: 10.1109/MCSE.2011.36.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 1(7), 2014.

Mario A Muñoz, Laura Villanova, Davaatseren Baatar, and Kate Smith-Miles. Instance spaces for machine learning classification. *Machine Learning*, 107(1):109–147, 2018.

Danielle Navarro and David Foxcroft. Learning statistics with jamovi: A tutorial for psychology students and other beginners (version 0.70). *Tillgänglig online: http://learnstatswithjamovi.com [Hämtad 14 december]*, 2019.

Sejong Oh. A new dataset evaluation method based on category overlap. *Computers in Biology and Medicine*, 41(2):115–122, 2011.

Neil Mac Parthalain, Qiang Shen, and Richard Jensen. Distance measure assisted rough set feature selection. In *2007 IEEE International Fuzzy Systems Conference*, pages 1–6, 2007. doi: 10.1109/FUZZY.2007.4295518.

Eduardo Perez and Larry A Rendell. Learning despite concept variation by finding structure in attribute-based data. In *In Proceedings of the Thirteenth International Conference on Machine Learning*. Citeseer, 1996.

S Selva Priyanka, Sudeep Galgali, S Selva Priya, BR Shashank, and KG Srinivasa. Analysis of suicide victim data for the prediction of number of suicides in india. In *2016 International Conference on Circuits, Controls, Communications and Computing (I4C)*, pages 1–5. IEEE, 2016.

S Ramya and S Nanda. Breast cancer detection and classification using ultrasound and ultrasound elastography images. *IRJET*, 4:596–601, 2017.

Papia Ray, S Surender Reddy, and Tuhina Banerjee. Various dimension reduction techniques for high dimensional data analysis: a review. *Artificial Intelligence Review*, 54(5):3473–3515, 2021.

Matthias Reif, Faisal Shafait, and Andreas Dengel. Prediction of classifier training time including parameter optimization. In *Annual Conference on Artificial Intelligence*, pages 260–271. Springer, 2011.

Matthias Reif, Faisal Shafait, and Andreas Dengel. Dataset generation for meta-learning. *KI-2012: Poster and Demo Track*, pages 69–73, 2012.

Matthias Reif, Faisal Shafait, Markus Goldstein, Thomas Breuel, and Andreas Dengel. Automatic classifier selection for non-experts. *Pattern Analysis and Applications*, 17(1):83–96, 2014. doi: 10.1007/s10044-012-0280-z. URL https://doi.org/10.1007/s10044-012-0280-z.

Jerome P Reiter. Using cart to generate partially synthetic public use microdata. *Journal of official statistics*, 21(3):441, 2005.

Allen H Renear, Simone Sacchi, and Karen M Wickett. Definitions of dataset in the scientific and technical literature. *Proceedings of the American Society for Information Science and Technology*, 47(1):1–4, 2010.

Jason DM Rennie. Mixtures of multinomials, 2005.

Adriano Rivolli, Luıs PF Garcia, Carlos Soares, Joaquin Vanschoren, and André CPLF de Carvalho. Towards reproducible empirical research in meta-learning. *arXiv preprint arXiv:1808.10406*, pages 32–52, 2018.

Adriano Rivolli, Luís P. F. Garcia, Carlos Soares, Joaquin Vanschoren, and André C. P. L. F. de Carvalho. Characterizing classification datasets: a study of meta-features for meta-learning, 2019.

Adriano Rivolli, Luís PF Garcia, Carlos Soares, Joaquin Vanschoren, and André CPLF de Carvalho. Meta-features for meta-learning. *Knowledge-Based Systems*, page 108101, 2022.

Donald B Rubin. Basic ideas of multiple imputation for nonresponse. *Survey Methodology*, 12 (1):37–47, 1986.

Donald B Rubin. Statistical disclosure limitation. *Journal of official Statistics*, 9(2):461–468, 1993.

Ricardo Dos Santos, Jose Aguilar, and Eduard Puerto. A meta-learning architecture based on linked data. In *2021 XLVII Latin American Computing Conference (CLEI)*, pages 1–10, 2021. doi: 10.1109/CLEI53233.2021.9640223.

Saddys Segrera, Joel Pinho, and María N Moreno. Information-theoretic measures for meta-learning. In *International Workshop on Hybrid Artificial Intelligence Systems*, pages 458–465. Springer, 2008.

Paras Sethi, Vaibhav Bhandari, and Bhavna Kohli. Sms spam detection and comparison of various machine learning algorithms. In *2017 international conference on computing and communication technologies for smart nation (IC3TSN)*, pages 28–31. IEEE, 2017.

Tegjyot Singh Sethi and Mehmed Kantardzic. When good machine learning leads to bad security: Big data (ubiquity symposium). *Ubiquity*, 2018(5):1–14, may 2018. doi: 10.1145/3158346. URL https://doi.org/10.1145/3158346.

Vivian Siahaan and Rismon Hasiholan Sianipar. *POSTGRESQL FOR PYTHON GUI: A Progressive Tutorial to Develop Database Project*. SPARTA PUBLISHING, 2019.

Chris Snijders, Uwe Matzat, and Ulf-Dietrich Reips. " big data": big gaps of knowledge in the field of internet science. *International journal of internet science*, 7(1):1–5, 2012.

Yan-Yan Song and LU Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015.

Mark Summerfield. *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming (paperback)*. Pearson Education, 2007.

Kristina Toutanova, Chris Brockett, Ke M. Tran, and Saleema Amershi. A dataset and evaluation metrics for abstractive compression of sentences and short paragraphs. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 340–350, November 2016. URL https://www.microsoft.com/en-us/research/publication/dataset-evaluation-metrics-abstractive-sentence-paragraph-compression/.

Robert Julius Trumpler and Harold F Weaver. *Statistical astronomy*. Dover Books on Astronomy and Space Topics, 1953.

Joaquin Vanschoren. *Understanding Machine Learning Performance with Experiment Databases (Het verwerven van inzichten in leerperformantie met experiment databanken) ; Understanding Machine Learning Performance with Experiment Databases*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 2010. URL https://lirias.kuleuven.be/handle/123456789/266060.

Guangtao Wang, Qinbao Song, Heli Sun, Xueying Zhang, Baowen Xu, and Yuming Zhou. A feature subset selection algorithm automatic recommendation method. *CoRR*, abs/1402.0570, 2014. URL http://arxiv.org/abs/1402.0570.

Guangtao Wang, Qinbao Song, and Xiaoyan Zhu. An improved data characterization method and its application in classification algorithm recommendation. *Applied Intelligence*, 43(4): 892–912, 2015.

Herbert Weisberg and Herbert F Weisberg. *Central tendency and variability*. Sage, 1992.

Yew Kee Wong. The use of big data in machine learning algorithm. In *CS & IT Conference Proceedings*, volume 11. CS & IT Conference Proceedings, 2021.

Xiao Yu, Ang Pan, Lu-An Tang, Zhenhui Li, and Jiawei Han. Geo-friends recommendation in gps-based cyber-physical social network. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 361–368. IEEE, 2011.

Eric R Ziegel. The elements of statistical learning, 2003.

Appendix A

SNOOKER Synthetic Dataset

In this appendix, we will look at the final output from SNOOKER's generation process, explaining the information in each column of the dataset. When looking at the 1st line of the CSV file (the header), we can extract all the columns that will be analysed:

```
ID; Location; Raised (UTC); Allocated; Stages; Fixed; Client;
    Family; Family Action; Subfamily; Subfamily Action;
    Subfamily Action Duration; Team; Users in the Shift; Users
    Next Shift; Users Competent; User actions; User Chosen;
    Action Chosen; Action Chosen Status; Action Chosen Duration
    ; Action Chosen (With Outlier); Ticket Duration; Escalate;
    Status; Outlier
```

The first column displays the ticket's ID, a unique numerical identifier; in this case, the ID is also the ticket's number. The second column contains the location of the ticket (it's country of origin). Raised (UTC) displays the DateTime the ticket was submitted in UTC format. At the same time, the next column (Allocated) does the same for the time the ticket was allocated to a helpdesk team member.

Stages registers many timestamps concerning the ticket treatment. Fixed, like Allocated did before, keeps the DateTime tickets solved. The next metric is simple: the Client column gives information regarding the client the ticket belongs to.

Family, Family Action, Subfamily and Subfamily Action characteristics keep information regarding the nature of the incident that caused that ticket. The Actions metrics save the techniques usually used to solve those problems. Lastly, the Subfamily Action Duration illustrates the time needed to perform the actions presented in the Subfamily Action's column.

Team, Users in the Shift and Users Next Shift give information regarding the team management, the current team, its active members and the members that will become available during the next shift, respectively. Users competent present a list of the most suitable member to perform the task. The User actions column contains data regarding the action that each available user would perform. User Chosen presents us the team member that will take care of the ticket. Action Chosen reflects the action chosen to fix the incident, followed by the Action Chosen Status that

represents the status of the action chosen. Action Chosen Duration and Action Chosen Duration (With Outlier) display the action's duration and the action's duration if an outlier exists.

The final four columns are Ticket Duration, Escalate, Status and Outlier. The first one depicts the time the ticket has endured without being solved, followed by Escalate, which is responsible for informing the user that the ticket should be passed to the next team. The Status column provides the status of the ticket (Closed or Transferred). Finally, the Outlier notifies us if the ticket should be considered an outlier.

Appendix B

Pymfe Meta feature tables

Table B.1: Pymfe general meta features

Family	Meta Feature	Description
general	attr_to_inst	Compute the ratio between the number of attributes.
general	cat_to_num	Compute the ratio between the number of categoric and numeric features.
general	freq_class	Compute the relative frequency of each distinct class.
general	inst_to_attr	Compute the ratio between the number of instances and attributes.
general	nr_attr	Compute the total number of attributes.
general	nr_bin	Compute the number of binary attributes.
general	nr_cat	Compute the number of categorical attributes.
general	nr_class	Compute the number of distinct classes.
general	nr_inst	Compute the number of instances (rows) in the dataset.
general	nr_num	Compute the number of numeric features.
general	num_to_cat	Compute the number of numerical and categorical features.

Table B.2: Pymfe statistical meta features

Family	Meta Feature	Description
statistical	can_cor	Compute canonical correlations of data.
statistical	cor	Compute the absolute value of the correlation of distinct dataset column pairs.
statistical	cov	Compute the absolute value of the covariance of distinct dataset attribute pairs.
statistical	eigenvalues	Compute the eigenvalues of covariance matrix from dataset.
statistical	g_mean	Compute the geometric mean of each attribute.
statistical	gravity	Compute the distance between minority and majority classes center of mass.
statistical	h_mean	Compute the harmonic mean of each attribute.
statistical	iq_range	Compute the interquartile range (IQR) of each attribute.
statistical	kurtosis	Compute the kurtosis of each attribute.
statistical	lh_trace	Compute the Lawley-Hotelling trace.
statistical	mad	Compute the Median Absolute Deviation (MAD) adjusted by a factor.
statistical	max	Compute the maximum value from each attribute.
statistical	mean	Compute the mean value of each attribute.
statistical	median	Compute the median value from each attribute.
statistical	min	Compute the minimum value from each attribute.
statistical	nr_cor_attr	Compute the number of distinct highly correlated pair of attributes.
statistical	nr_disc	Compute the number of canonical correlation between each attribute and class.
statistical	nr_norm	Compute the number of attributes normally distributed based in a given method.
statistical	nr_outliers	Compute the number of attributes with at least one outlier value.
statistical	p_trace	Compute the Pillai's trace.
statistical	range	Compute the range (max - min) of each attribute.
statistical	roy_root	Compute the Roy's largest root.
statistical	sd	Compute the standard deviation of each attribute.
statistical	sd_ratio	Compute a statistical test for homogeneity of covariances.
statistical	skewness	Compute the skewness for each attribute.
statistical	sparsity	Compute (possibly normalized) sparsity metric for each attribute.
statistical	t_mean	Compute the trimmed mean of each attribute.
statistical	var	Compute the variance of each attribute.
statistical	w_lambda	Compute the Wilks' Lambda value.

Table B.3: Pymfe Information-theoretic meta features

Family	Meta Feature	Description
info-theory	attr_conc	Compute concentration coef. of each pair of distinct attributes.
info-theory	attr_ent	Compute Shannon's entropy for each predictive attribute.
info-theory	class_conc	Compute concentration coefficient between each attribute and class.
info-theory	class_ent	Compute target attribute Shannon's entropy.
info-theory	eq_num_attr	Compute the number of attributes equivalent for a predictive task.
info-theory	joint_ent	Compute the joint entropy between each attribute and class.
info-theory	mut_inf	Compute the mutual information between each attribute and target.
info-theory	ns_ratio	Compute the noisiness of attributes.

Table B.4: Pymfe Model-based meta features

Family	Meta Feature	Description
model-based	leaves	Compute the number of leaf nodes in the DT model.
model-based	leaves_branch	Compute the size of branches in the DT model.
model-based	leaves_corrob	Compute the leaves corroboration of the DT model.
model-based	leaves_homo	Compute the DT model Homogeneity for every leaf node.
model-based	leaves_per_class	Compute the proportion of leaves per class in DT model.
model-based	nodes	Compute the number of non-leaf nodes in DT model.
model-based	nodes_per_attr	Compute the ratio of nodes per number of attributes in DT model.
model-based	nodes_per_inst	Compute the ratio of non-leaf nodes per number of instances in DT model.
model-based	nodes_per_level	Compute the ratio of number of nodes per tree level in DT model.
model-based	nodes_repeated	Compute the number of repeated nodes in DT model.
model-based	tree_depth	Compute the depth of every node in the DT model.
model-based	tree_imbalance	Compute the tree imbalance for each leaf node.
model-based	tree_shape	Compute the tree shape for every leaf node.
model-based	var_importance	Compute the features importance of the DT model for each attribute.

Table B.5: Pymfe Landmarking meta features

Family	Meta Feature	Description
landmarking	best_node	Performance of a the best single decision tree node.
landmarking	elite_nn	Performance of Elite Nearest Neighbor.
landmarking	linear_discr	Performance of the Linear Discriminant classifier.
landmarking	naive_bayes	Performance of the Naive Bayes classifier.
landmarking	one_nn	Performance of the 1-Nearest Neighbor classifier.
landmarking	random_node	Performance of the single decision tree node model induced by a random attribute.
landmarking	worst_node	Performance of the single decision tree node model induced by the worst informative attribute.

Table B.6: Pymfe meta features

Family	Meta Feature	Description
clustering	ch	Compute the Calinski and Harabasz index.
clustering	int	Compute the INT index.
clustering	nre	Compute the normalized relative entropy.
clustering	pb	Compute the pearson correlation between class matching and instance distances.
clustering	sc	Compute the number of clusters with size smaller than a given size.
clustering	sil	Compute the mean silhouette value.
clustering	vdb	Compute the Davies and Bouldin Index.
clustering	vdu	Compute the Dunn Index.

Table B.7: Pymfe concept meta features

Family	Meta Feature	Description
concept	cohesiveness	Compute the improved version of the weighted distance, that captures how dense or sparse is the example distribution.
concept	conceptvar	Compute the concept variation that estimates the variability of class labels among examples.
concept	impconceptvar	Compute the improved concept variation that estimates the variability of class labels among examples.
concept	wg_dist	Compute the weighted distance, that captures how dense or sparse is the example distribution.

Table B.8: Pymfe itemset meta features

Family	Meta Feature	Description
itemset	one_itemset	Compute the one itemset meta feature.
itemset	two_itemset	Compute the two itemset meta feature.

Table B.9: Pymfe complexity meta features

Family	Meta Feature	Description
complexity	c	Compute the entropy of class proportions.
complexity	c2	Compute the imbalance ratio.
complexity	cls_coef	Clustering coefficient.
complexity	density	Average density of the network.
complexity	f1	Maximum Fisher's discriminant ratio.
complexity	f1v	Directional-vector maximum Fisher's discriminant ratio.
complexity	f2	Volume of the overlapping region.
complexity	f3	Compute feature maximum individual efficiency.
complexity	f4	Compute the collective feature efficiency.
complexity	hubs	Hub score.
complexity	l1	Sum of error distance by linear programming.
complexity	l2	Compute the OVO subsets error rate of linear classifier.
complexity	l3	Non-Linearity of a linear classifier.
complexity	lsc	Local set average cardinality.
complexity	n1	Compute the fraction of borderline points.
complexity	n2	Ratio of intra and extra class nearest neighbor distance.
complexity	n3	Error rate of the nearest neighbor classifier.
complexity	n4	Compute the non-linearity of the k-NN Classifier.
complexity	t1	Fraction of hyperspheres covering data.
complexity	t2	Compute the average number of features per dimension.
complexity	t3	Compute the average number of PCA dimensions per points.
complexity	t4	Compute the ratio of the PCA dimension to the original dimension.

Appendix C

Meta Analysis Output

Statistical meta feature analysis output file:

```
Statistical Meta Feature Analysis of file in /Users/joaolemos/DatasetGen/
    Output/Generation/test.csv at 2022-06-02 00:15:09.428844.
can_cor.mean                    1.0
can_cor.sd                      2.863353569008059e-16
cor.mean                        0.0031818656040018055
cor.sd                          0.029273671606693415
cov.mean                        0.00018169921467679913
cov.sd                          0.06544132326412351
eigenvalues.mean                8.517833528021107
eigenvalues.sd                  584.2881540682121
g_mean.mean                     0.02200758566217588
g_mean.sd                       0.8030517936688358
gravity                         14.491376746189438
h_mean.mean                     0.017814411761191074
h_mean.sd                       0.6219661342834908
iq_range.mean                   0.018441748588161472
iq_range.sd                     0.6913274616349807
kurtosis.mean                   878.1186853398123
kurtosis.sd                     283.23712818907694
lh_trace                        inf
mad.mean                        0.009141821376281114
mad.sd                          0.35732979367673556
max.mean                        1.3132189918427106
max.sd                          19.345451307983584
mean.mean                       0.0399734463403152
mean.sd                         1.6495148276889133
median.mean                     0.01930767621836436
median.sd                       0.6700330673798763
min.mean                        0.006065676636686885
min.sd                          0.21001299789299555
nr_cor_attr                     0.0007765221295242062
nr_disc                         1000
nr_norm                         0.0
```

nr_outliers	4774
p_trace	999.9999999999998
range.mean	1.3071533152060237
range.sd	19.23267679677396
roy_root	inf
sd.mean	0.08891679211579023
sd.sd	2.9174831022006753
sd_ratio	nan
skewness.mean	28.866039090472135
skewness.sd	6.782417085249947
sparsity.mean	0.13476022492442502
sparsity.sd	0.19501828184302594
t_mean.mean	0.02068489003024659
t_mean.sd	0.7202330927954221
var.mean	8.517833528021098
var.sd	584.2706323926961
w_lambda	0.0

Analysis took 0:28:16.250728.