FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Meta Feature Classification on a Synthetic Dataset Generator

João Carlos Fonseca Pina de Lemos

Master in Informatics and Computing Engineering

Supervisor: Daniel Augusto Gama de Castro Silva

Second Supervisor: Leonardo Silva Ferreira

August 1, 2022

Meta Feature Classification on a Synthetic Dataset Generator

João Carlos Fonseca Pina de Lemos

Master in Informatics and Computing Engineering

Approved by . . . :

August 1, 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, its 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 focus on the study of meta feature extraction methods and meta feature insertion analysis. Using an already existing Ticket-based Synthetic DataSet Generator as a basis, we also developed a meta extractor feature.

Each dataset has its group of meta features - its fingerprint. Our meta feature extractor feature can dissect any kind of CSV tabular dataset, even if those datasets were not synthetically generated by our generator. 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 are explored. Besides extracting parameters from existing datasets' meta features, it is also essential to analyse the parameter-parameter relationship.

We also explored the concept of meta feature inclusion in the generation process, where we try to force values of certain meta features into the final synthetic dataset.

Keywords: Synthetic Dataset, Meta Features, Machine Learning

Agradecimentos [1]

Um trabalho de mestrado é uma longa caminhada, marcada por diversos desafios, incertezas, alegrias e tristezas. Apesar do processo solitário a que qualquer investigador está destinado, o seu percurso é também pautado pelo apoio de várias pessoas, às quais dedico este trabalho.

Em primeiro lugar, gostaria de exprimir os mais sinceros agradecimentos ao meu orientador, professor doutor Daniel Castro Silva e ao meu co-orientador, Leonardo Silva Ferreira, pela orientação exemplar, rigor científico, disponibilidade e conselhos facultados ao longo dos últimos meses.

Agradeço aos meus grandes amigos e colegas de curso, Ricardo Figueiredo e Luís Guilherme Neves, por terem sempre puxado por mim e me terem ajudado quando mais precisei – é também graças a vocês que este trabalho é possível.

Obrigado aos meus amigos Sérgio Pinto, Rahul Pires, João Brandão, João Pedro Morais, Mafalda Neves, Inês Faria e Sara Aguiar por todos os momentos passados e amizade transmitida.

Um obrigado à Tuna de Engenharia da Universidade do Porto (TEUP) - lugar onde encontrei a minha casa longe de casa, e por ter sido a escola que fez de mim grande parte da pessoa que sou hoje.

Agradeço à minha terapeuta Dra. Sara Malheiro, pela atenção, disponibilidade, prontidão e amizade que me permitiram trilhar o caminho certo para aqui chegar.

Ao professor doutor António Augusto Sousa, pelas constantes palavras de encorajamento e por uma conversa especial que fez com que fosse capaz de olhar para este percurso com outros olhos.

Por fim, e como não podia deixar de ser, dirijo os meus maiores agradecimentos à minha família, que revelou ser o maior porto de abrigo e refúgio nos momentos mais difíceis.

Aos meus pais, que sempre foram um exemplo de dedicação, trabalho e esforço.

Ao meu irmão Pedro, meu melhor amigo, com quem partilho memórias e momentos muito felizes.

Às minhas tias Helena e Elsa, que ao longo da vida (em especial dos anos no Porto) foram sempre a principal rede de apoio e ajuda para tudo, e a quem serei eternamente grato.

Às minhas tias São, Zeza, e à minha prima Xana pelo carinho e amizade desde sempre.

À Mariana, um especial e muito grande agradecimento por, mesmo nos momentos mais difíceis, nunca me ter deixado desistir e ter acreditado que tudo isto seria possível. Obrigado pelo carinho, dedicação e amor ao longo dos últimos nove anos.

João Carlos Lemos

[1] Unlike the rest of this study, the acknowledgements are written in Portuguese to be read by every person mentioned, some of which are native only speakers.

“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
PET	Privacy-enhancing technologies
SD	Synthetic Dataset

Chapter 1

Introduction

Dataset is the name given to a collection of related, discrete items of associated data. The Oxford Dictionary defines the term as "a collection of related sets of information that is composed of separate elements but can be manipulated as a unit by a computer.." This data may be accessed apiece, in combination or handled as a complete entity (Akhter et al., 2019). The information 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, making data with similar or better results than 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 that can be used to train intelligent systems. However, several improvements can be made to the current program. 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 data analysis technique 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 the 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 dataset description 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 further detailed in Chapter 2.

Scrapping and collection tools are commonly used to circumvent the scarceness of data. Still, their quality is questionable and highly dependent on the data compilation techniques and annotation procedures (Jose et al., 2020). Challenges still arise when more large-scale datasets are collected. Small datasets are not as effective in producing accurate prediction models, needing further treatment until their use is valid (Lateh et al., 2017). Problems do exist even in cases where a large amount of data is available. Its 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, 1988). 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 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 an MLM with SDs to group personally and geographically close users, increasing the potential for real-world relationships to flourish. Sethi et al. (2017) and Renuka 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.

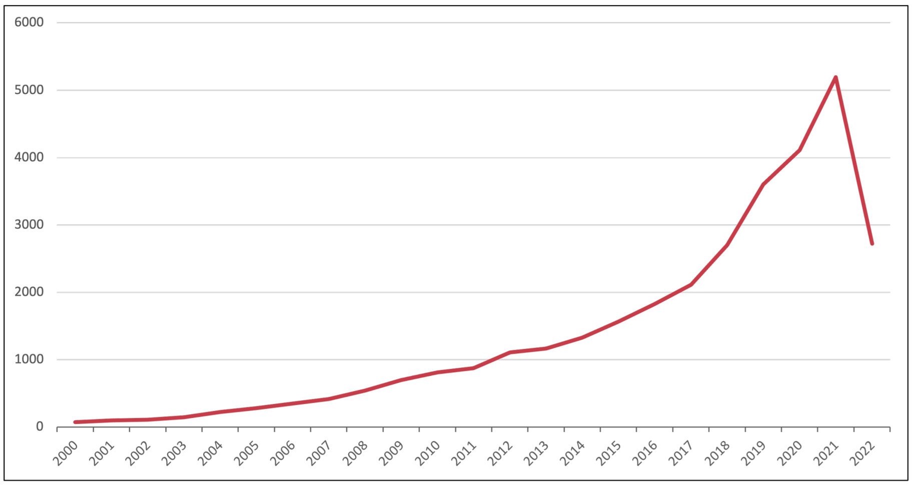


Figure 1.1: The number of SD papers published on Scopus from 1990 to June 2022. Data was collected by searching papers using "Syntetic Datasets" as keywords and applying a yearly filter.

Meta-learning supports the recommendation of machine learning algorithms and their configurations. These recommendations are based on meta-data, consisting of performance evaluations of algorithms and characterizations of prior datasets. These characterizations, also called meta features, are important metrics to study and will be the focal point of this study.

SDs that use real data, replicating its features in the generation process, have better results. The dynamics and intricacy of real-world systems are practically impossible to be modelled in generators. Skopik et al. (2014) called the process of generating such datasets "Semi-synthetic Data Generation".

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.

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 examine 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. 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.

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

3. 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.

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

At the end of this study, we should have a complete set of meta features collected through bibliographic analysis. We will then use this meta feature list as metrics to be extracted from a dataset, letting us fully characterise it through meta-extraction. We should also be able to understand ways to insert some meta feature values into the generation process.

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 look at the characteristics of datasets and 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. This will be done by investigating some commonly used algorithms and reviewing how other SDs generation projects performed their generation undertakings. We will also 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 study 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., 2018). 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 of 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 use mathematical distributions to describe data distribution, and information-theoretic uses 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., 2018; Filchenkov and Pendryak, 2015). In this chapter, we took a look at various sets of commonly used meta features, creating a collected list of features that can be useful in analyzing datasets.

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 regarding 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 *instToAttri* 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., 1999). They are the most extensive and 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 a high 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), and 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 (Rayens, 1993). 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 describe 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). It represents the amount of data needed to specify one class.

The joint entropy meta feature (*jointEnt*) displays the relative significance of the attributes in the mather of representation. On the other hand, mutual information meta features (*mutInf*) represent the common information in the attributes, correlating to the degree of interdependence.

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 percentage 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 on 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. 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 on each level. 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 inequality in the tree. The shape of the tree (*treeShape*) symbolises the entropy of the probabilities of randomly reaching 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 used for 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* uses the least informative attribute to initialise the model.

The elite-Nearest Neighbor (*eliteNN*) landmarker uses the kNN algorithm. In this algorithm, 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 to maximise class separability. The second technique is based on Bayes' theorem. It calculates, for each feature, the probability of an instance to belong to each class.

Besides the ones described before, relative and subsampling landmarking 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 the distance between the instances, their density or 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 are defined dynamically, this simple informative measure becomes more valuable to calculate. This doesn't happen in cases where the number of clusters becomes an input parameter specified 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 on the value returned. Despite 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 measures: feature-based, linearity, neighbourhood, network, and dimensionality. 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 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 compared 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.

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.

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 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 exploring similarities 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.

With this family, we conclude our collection of meta features. Later, in Chapter 5 we provide a new list depicting the selected meta features in the extraction process. A subset of the General, Statistical, Information-theoretic, Model-based, Landmarking, Clustering, Concept, Itemset and Complexity meta feature families were used.

Chapter 3

Generation

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 synthetic text data to bootstrap Alexa's language options (Kollar et al., 2018)). An example of its usage can be seen in Fig. 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 artificial video, image, or sound. Media is rendered with properties close-enough to real-life data. This similarity allows synthetic media to replace the original data as a drop-in replacement.

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 sensitive attribute value (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. An imputer alters the values of a set of attributes for a subset of data points to protect sensitive information. 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. We chose a close record in the synthetic data for each random record of actual data. 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 a synthetic data generation method that is 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 generator integration can be seen in Fig 3.1. This figure depicts an overview of the Genie Semantic Parser Generator, a toolkit for creating a seman-

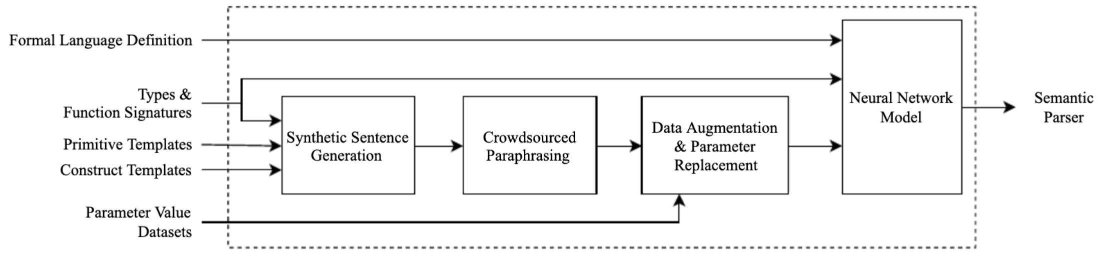

Formal Language Definition
Types & Function Signatures
Primitive Templates
Construct Templates
Parameter Value Datasets
Synthetic Sentence Generation
Crowdsourced Paraphrasing
Data Augmentation & Parameter Replacement
Neural Network Model
Semantic Parser

Figure 3.1: Overview of the Genie Semantic Parser Generator (Campagna et al., 2019).

tic parser for new virtual assistant capabilities. In this generator sentences are synthetically created (Campagna et al., 2019).

A statistical distribution is the method typically used to generate synthetic data. 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".

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

3.1 Conditionial Distribution

A conditional distribution is a distribution of values for one variable that exists when the values of other variables are specified. This distribution allows one to estimate the distribution of one's variable of interest under specific conditions.

The marginal probability (presented below) represents the probability of a single event occurring, independent of other events. On the other hand, a conditional probability represents the likelihood that an event occurs, given that another specific event has already occurred.

According to Zhou et al. (2022), the aim of conditional distribution applied to dataset generation is that a random sample from the target conditional distribution can be obtained by the action of the conditional generator on a sample drawn from a reference distribution.

3.2 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 (Perrakis et al., 2013).

The marginal distribution of a group of arbitrary variables is the probability distribution of the variables included in the group. It gives the likelihood 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 considering the values of the other variables (Trumpler and Weaver, 1953).

The independent marginals method approach has its advantages: it is computationally efficient. It possibly estimates 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 (Goncalves et al., 2020).

3.3 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 (Chen and Pollino, 2012).

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 variables' pairwise conditional dependencies (or lack thereof). The second step is estimating the conditional probability tables (Goncalves et al., 2020).

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.

3.4 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 trial has a set number of outcomes and a constant probability for each outcome. These are independent, meaning that a previous trial does not influence future trials.

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

The multinomial model over term frequencies is defined in the Equation (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.

3.5 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 synthetic data from the first dataset. Due to the competition, each network pushes the other, leading to better results (Creswell et al., 2018).

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. GANs cannot compute "gradients latent categorical variables for training vis backpropagation" - (in Goncalves et al. (2020)). 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 challenging to train. Solving the associated min-max optimization problem can be very unstable. This can, however, be circumvented if the variation Wasserstein GAN is applied (Gulrajani et al., 2017).

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 regarded as 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).

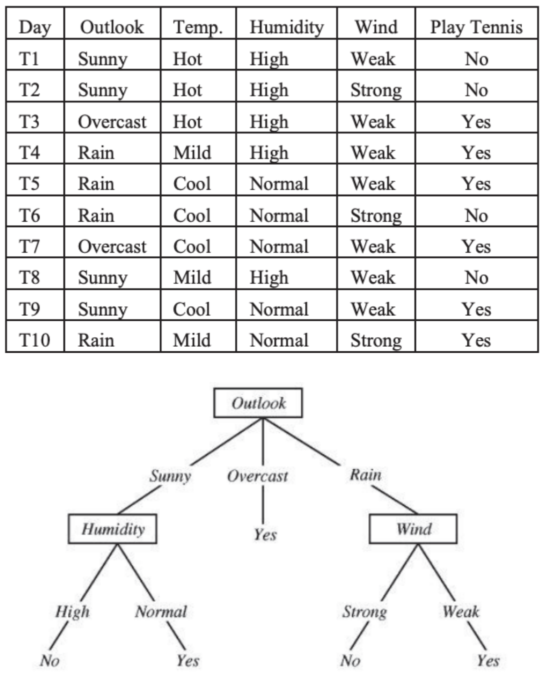
To synthesise every attribute Y_i in a dataset, we learn the parameters of the regression model using the dataset with attributes in Y_{-i}. We generate values for Y_i by sampling from a Gaussian distribution with constant variance and the mean as determined regression parameters.

Zocchi and Manly (2006) used linear regression to generate different datasets with the same estimates. This was developed for teaching purposes to demonstrate that different sets of regression data can give precisely the same estimated regression functions.

3.7 Decision Tree

Reiter (2005) proposed a generation technique that uses classification and regression tree. That author applied it to the generation of a partially synthetic 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).

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



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 using a dataset [Gavankar and Sawarkar (2017)].

the tree using the known attributes of the datapoint until reaching the leaf node. Kirchner et al. (2004) applied this method to swine production and concluded that "the decision tree algorithm can detect relationships and patterns in real swine breeding datasets". Figure 3.2 depicts a decision tree model created from a tabular dataset. An ML model was used to decide on a playing tennis binary problem.

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, and voting occurs for each class. The class with the most votes becomes our model's prediction.

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

3.9 Neural Network

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

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. This allows us to categorise and cluster data at a high rate.

3.10 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.

3.10.1 Overfitting and Underfitting

Overfitting might appear when a spurious pattern in the original dataset is captured and replicated with increasing 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 (Roelofs et al., 2019).

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. 3.3

3.10.2 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 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. It could also be a problem in the original dataset used for the generation.

In actual datasets, missing data can occur due to a series of factors: data may be missing due to test design, failure in the observations or failure in recording observations.

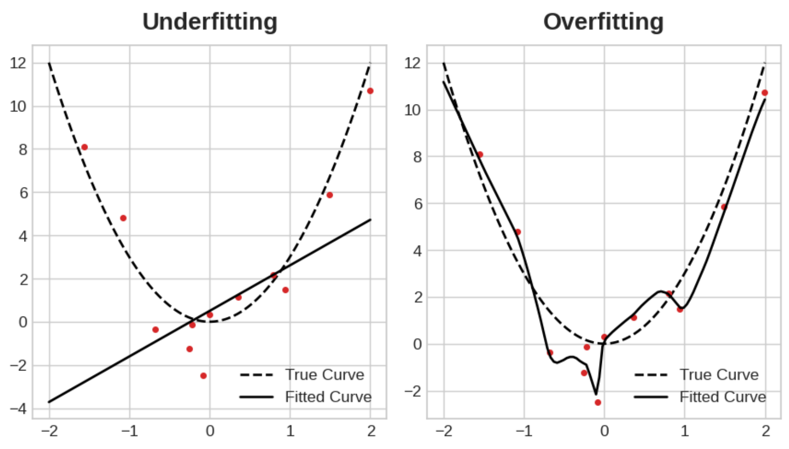


Figure 3.3: Underfitting and Overfitting [Holbrook R. (2020)]

3.10.3 Data Imbalances

Imbalanced data refers to datasets where the target class has an uneven distribution of observations. This 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 imbalance in the dataset as relevant data with minor occurrences could be considered noise.

3.11 Data Evaluation

For synthetic data to be considered valid, there needs to be evidence demonstrating the utility of that data. El Emam (2020) collected seven methods to evaluate the validity of synthetic data. Those methods are present in the following paragraphs.

The first method presented is structural similarity. Structural similarity indicates that the synthetic data should pass edit checks and have the same variable types and formats, variable names, metadata, file formats, table names and structure as real data does. The same methods used to analyse real data can be used on synthetic ones.

General utility metrics assess general statistical parameters and model evaluation results. Using these metrics, it is also possible to assess the distinguishability between real and synthetic data. The synthetic data can be considered valid if an automated model cannot distinguish between real and synthetic data.

The utility of synthetic data can also be accessed through the replication of studies. Using this approach, we use an already-published study using the same dataset. We then replicate the results of that published study using the synthetic data. The data's validity depends on the success of the replication of results.

Letting domain experts examine the synthetic dataset and compare it to real data can also be a way to evaluate its quality. This evaluation is, however, subjective.

Another evaluation metric is the Bias and Stability Assessment. This assessment consists of generating large quantities of synthetic datasets and calculating the general utility metrics on average. This evaluation is also valuable for evaluating the stability of the generation process.

Public available data could also be used in order to compare datasets. Synthetic datasets that reflect the same reality as publicly available real ones would be considered valid.

Comparison with Privacy-enhancing technologies is an option that can also be available. This assessment can help decision makers decide the relative strengths and weaknesses of particular PETs for providing data access.

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 helpful for the development phase.

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

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. In our case, we can see that the generation process consists of an optimization problem to obey all limits while creating a dataset in the most optimal way possible. It could also be a simple generation with statistical distribution in mind. Everything depends on the configuration imputed by the user.

Comparisons of applied improvements can be analyzed by looking at currently generated 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.

We also presented ways to evaluate the quality of generated data. However, it should be noted that some of the presented evaluation methods will not be available in a purely synthetic generation.

Chapter 4

Problem and proposed solution

The problem at hand is the extraction and inclusion of meta features in the Synthetic Dataset Generator SNOOKER generation process. 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., 2020) when using a real dataset as a basis for the generation process.

The generator interface should allow users to select various meta features (explored in Chapter 2). Those meta features 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 must 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

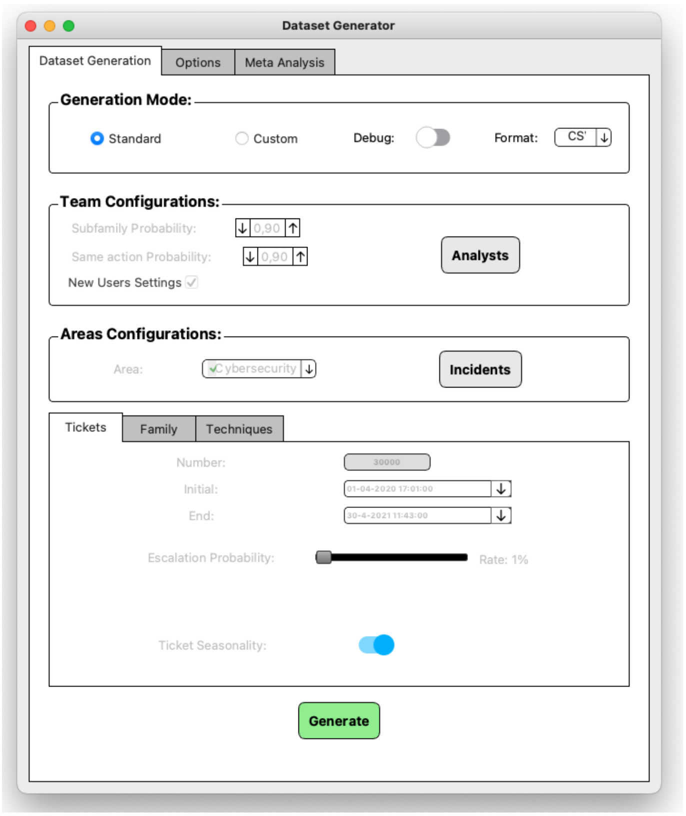
Before analysing the problem, we should first look at the original generator, trying to understand its internal workings. SNOOKER was developed to bridge the availability gap for Helpdesk's datasets.

This section will present a general overview of the generator, talking about its leading technologies, features and the characteristics of its synthetically generated datasets.

4.1.1 Technologies

SNOOKER was developed using Python[1] and several of its libraries. Python 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



Dataset Generator

Dataset Generation | Options | Meta Analysis

Generation Mode:

[✓ Standard] Custom Debug: Format: CS'

Team Configurations:

Subfamily Probability: 0,90

Same action Probability: 0,90

New Users Settings ✓

Analysts

Areas Configurations:

Area: ✓Cybersecurity

Incidents

Tickets | Family | Techniques

Number: 30000

Initial: 01-04-2020 17:01:00

End: 30-4-2021 11:43:00

Escalation Probability: Rate: 1%

Ticket Seasonality:

Generate

Figure 4.1: SNOOKER's User Interface

The user interface of the generator was developed using PyQt[2]. PyQt 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. After a short delay, a synthetical tabular dataset is generated after clicking the Generate button.

SNOOKER: A DataSet GeNeratOr fOr HelpdesK SERvices was developed using both technologies. A change in technologies in 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

SNOOKER's generated datasets contain much information commonly present in helpdesk datasets. While a complete representation of the synthetic dataset is not provided in this section, it can be examined in greater detail in Appendix A.

4.1.4 SNOOKER's generation process

After receiving input from the previously mentioned configuration file, SNOOKER performs a series of tasks to fully generate each entry on the dataset.

It starts the process by doing a simple Ticket Creation Task, where all the tickets are generated with simple data, and more complex data is added iteratively. Finally, the ticket family information is inserted in the final phase, where the family distribution over time is considered.

The following module in the generation is the Team Assignment. As its name suggests, this process is responsible for allocating teams and users available for each ticket. Shifts are considered for scheduling ticket-solving tasks by considering the team, work shift, and days not working for each team member.

The User Actions Creation module follows, providing the generator with valid actions (subtechniques) to solve each ticket problem based on its subfamily.

The following module is the Users Assignment. Each ticket is assigned a user and action by using the previously described modules in this process.

The fifth module takes care of Ticket replication tasks. This happens when a ticket is passed to more advanced teams. This event can be triggered when a ticket is created, when a set maximum number of tickets of that subfamily is detected, or when a set distance between the subfamily action and the user action exceeds the maximum distance defined in the interface.

Finally, the last module takes care of generating the Ticket Status. This status can either be 'closed' or 'transferred'. Transferred tickets also need to be replicated and passed to higher teams.

4.2 Study Objectives

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

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? In Chapter 3 we analysed a large group of algorithms that can be used to generate synthetic datasets. Of the available options, the one chosen and used later in Chapter 6's experiments was Conditional distribution. It is also the method used in SNOOKER's generation process.

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 new additions can be added 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. In Chapter 7 we also reference some future features that can be added to the application. Besides the inclusion of the new feature no other additions or changes were made to the original generator.

The development of this study can be considered successful after fully answering the questions and furfilling the objectives presented in this section.

4.3 Schedule

Development of the final product went through several steps involving various tasks. A Gantt diagram was created to help with the planning. It can be found in figure 4.2.

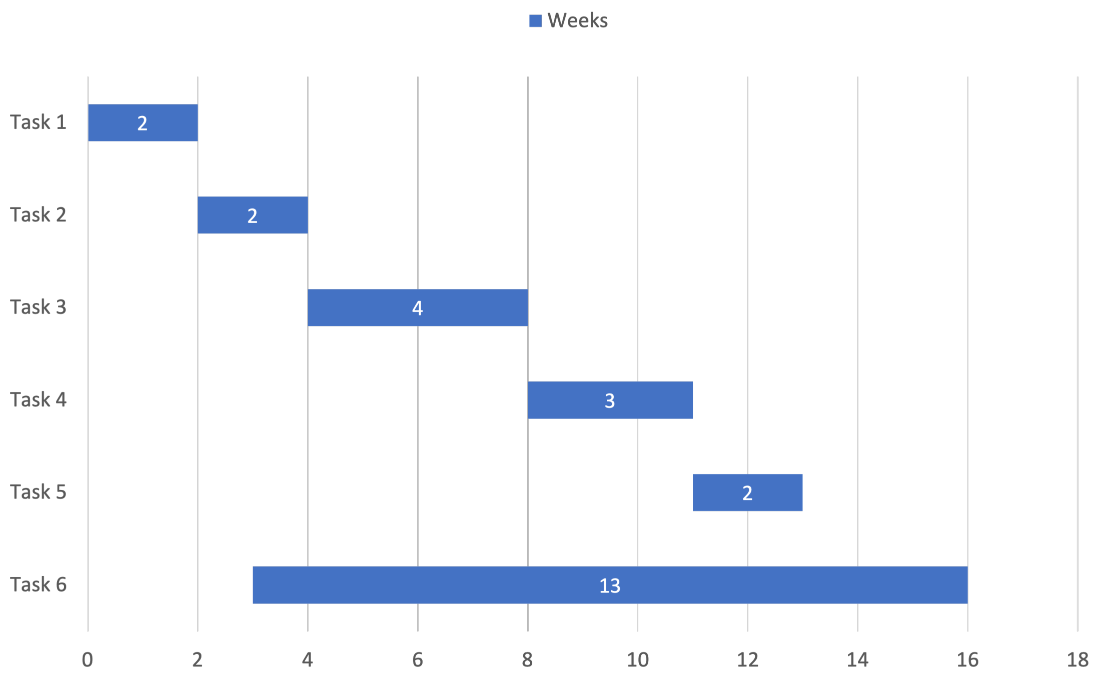


Figure 4.2: Gantt diagram for the developed process

Task 1 is the analysis of the current code. Task 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. Task 5 will look at general improvements to the generator, namely in UI/UX. Finally, Task 6 is the writing of the dissertation and scientific paper.

4.4 Risk Analysis

The development of this project faced a series of risks. These risks were hindrances to the development process of the tasks previously depicted. We will now present a series of such potential threats.

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

Hardware limitations also impacted development. As some generation processes and meta feature calculations have high computational costs, an inefficient appliance delays the development, especially during trial-and-error periods.

Another aspect that impacted development was the lack of know-how when dealing with data analysis and manipulation and the technologies used in the product.

4.5 Projected vs Real work

Ultimately the whole development part ended up not going as far as initially expected. We did not expect so many problems during the development of this project. Problems that ultimately left us still with much open to explore. 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 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.

The real work ended up not following the scheduled plan. Analysis tasks took way more time than initially outlined. The quality assurance task became the meta feature extraction feature that took half of the available time to complete. The meta feature implementation (or integration as we now call it) was not fulfilled, being only theoretically explored.

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

Left behind were the practical meta feature inclusion module and the inclusion of enhancements to SNOOKER.

Chapter 5

Meta Feature Extraction

The first addition to SNOOKER: A DataSet GeNeratOr fOr HelpdesK SERvices was the inclusion of a meta feature extraction 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 meta-extraction of other datasets, making the product usable for meta-extraction tasks for datasets external to the generation.

Past development of the generator was made using Python. As the meta-extraction 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[1] 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.

[1] pandas.pydata.org

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[2]) 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 describing every family's 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).

[2]pymfe.readthedocs.io

The vast amount of families and collections of meta features inside those families proved more than enough to use in our meta-extraction model. Some limitations were, however, applied. Pandas is unable to analyse datasets with missing data. Meta features that use missing data as a parameter had to be left out.

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. The snippet 5.1 shows an example of code used to read the CSV file while consuming the first column.

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 meta-extraction 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

The Meta Feature Extraction Module was always idealized as an increment made to SNOOKER during this study. As such, it needed a user interface to allow user-module interaction. An interface was created and appended to the generator's frontend. As seen in Chapter 4, SNOOKER's interface was developed using PyQt, which meant that this interface had to be developed using the same technology.

"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)).

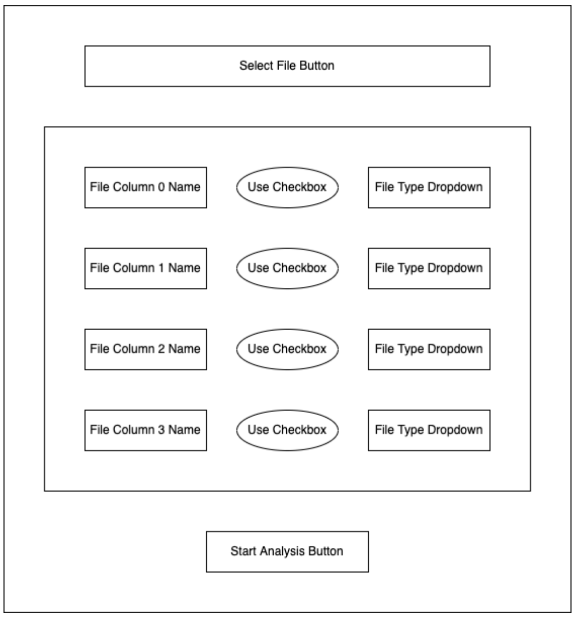


Figure 5.1: Original idealization of meta-extraction feature interface

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 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 data type. 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 families. Figure 5.2 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 gallery 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

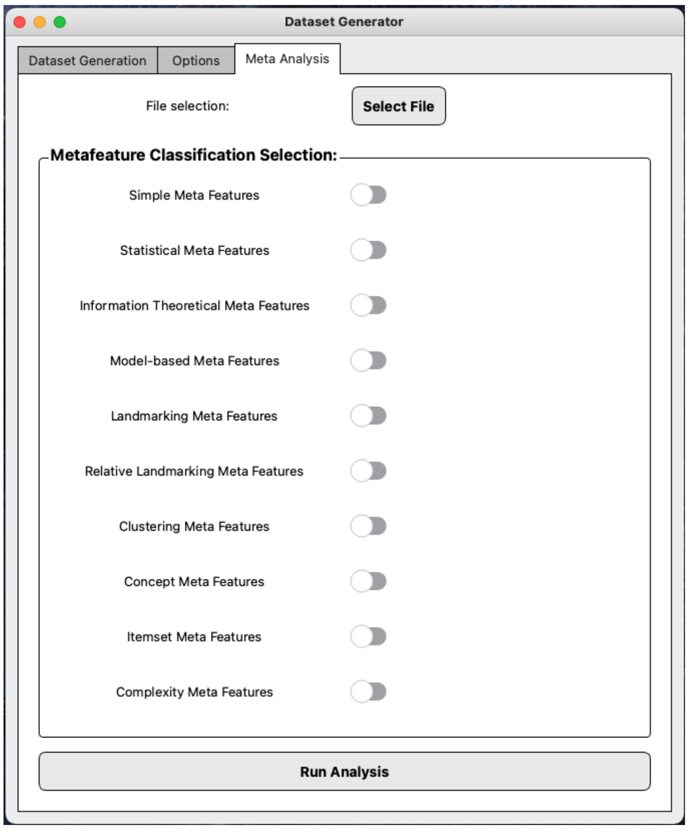


Figure 5.2: Meta Feature Extraction feature interface

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. 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 meta-extraction feature, using several datasets.

Meta Feature family	DS1	DS2	DS3	DS4	DS5
Simple	2.21s	3.25s	6.27s	11.90s	32.70s
Statistical	28:16 min	17:06 min	12.22 min	>2h	51:31 min
Information Theoretical	24.82s	21.9s	45.72s	43.55s	1:26 min
Model-based	4.47s	1:15 min	25:42 min	>2h	>2h
Landmarking	<1s	2.69s	5.34s	23.07 s	38.73s
Relative Landmarking	6.25s	2.19s	5.23s	17:06 min	>2h
Clustering	3:13 min	4:29 min	13:19 min	45:23min	>2h
Concept	8.52s	29.02s	58.32s	52.15s	57.46s
Itemset	5.14 min	10:38 min	23:57 min	>2h	>2h
Complexity	>4h	>4h	>4h	>4h	>4h

DS1, DS2, DS3 and DS4 were SNOOKER-generated datasets with 1000, 2000, 3000, 4000 and 5000 entries, respectively. Some outliers can be found on datasets with more entries where certain calculations took way more time than the rest. Many outliers can be attributed to hardware limitations. Overheating and memory problems were faced when leaving the meta-extractor module running for hours. However, we can see that statistical analysis took more time than any other analysis. Complexity analysis was also incapable of reaching any result, even having double the available time limit to perform said analysis. Initially, the sentiment was that hardware limitations were accountable for these analysis times. Yet, when testing the meta-extraction feature on a much more powerful machine, the end result times were similar to those achieved locally. It could then be concluded that the statistical and complexity times problem was not the computational power.

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 research, making 2 tests with 2 different datasets with a time limit of 12 hours. Both these tests were unable to lead to results. We then tried using datasets with a low number of entries. This analysis's result is displayed in Table 5.2. We used 5 SNOOKER-generated datasets with 100, 200, 300, 400 and 500 entries. We compared the time it took to perform a statistical analysis (the second most time-consuming analysis) to the time taken by complexity calculations. On that same table, we also have a Ratio row. This ratio indicates the $ComplexityTime/StatisticalTime$ relation.

From the analysis, we could conclude that Complexity analysis took way too much time to complete compared to other meta-calculations. Further investigation of these meta features was dropped as time constraints made their study impossible to pursue. The option to analyse them is still present in the current product. However, due to the limitations mentioned earlier, its use is not recommended on local machines.

While Complexity Meta-Analysis was dropped from further development. The same could not apply to Statistical Meta Features as their values are some of the most important we could extract from a dataset. So what if we could remove some meta features from the module? 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 calculation faster.

We used the previously used SNOOKER-generated datasets to get the statistical analysis times. However, it soon became obvious that it would take too much time. Complete statistical analysis of SD1 took more than 15 hours to complete (the output of this analysis can be seen in Appendix

Table 5.2: Complexity and Statistical runtimes on small datasets

Entries	100	200	300	400	500
Statistical	1.5s	10.60s	38.52s	1:25 min	2:55 min
Complexity	53.62s	5:53 min	17:46 min	40:17 min	1:24:53 h
Ratio	35.75	33.30	27.67	28.44	29.01

Table 5.3: Statistical meta analysis duration

Meta Feature	SD1	cars	cereal	CausesOfDeath_France_2001-2008
can_cor	26:32 min	50.97s	0.48s	13:07 min
cor	27:16 min	52.99s	0.46s	12:31 min
cov	25:29 min	52.40s	0.46s	12:25 min
eigenvalues	31:03 min	52.54s	0.47s	11:25 min
g_mean	25:42 min	52.55s	0.46s	11:37 min
gravity	26:54 min	52.16s	0.45s	11:38 min
h_mean	25:32 min	52.11s	0.45s	11:38 min
iq_range	27:45 min	1:44 min	0.46s	11:31 min
kurtosis	26:54 min	1:07 min	0.51s	11:36 min
lh_trace	26:04 min	54.09s	0.45s	11:34 min
mad	28:52 min	52.51s	0.45s	11:34 min
max	26:29 min	52.51s	0.44s	11:34 min
mean	26:56 min	51.85s	0.44s	11:36 min
median	27:36 min	51.88s	0.44s	11:33 min
min	26:31 min	51.57s	0.45s	11:31 min
nr_cor_attr	47:58 min	51.03s	0.44s	11:29 min
nr_disc	27:54 min	50.70s	0.44s	11:30 min
nr_norm	27:40 min	51.09s	0.45s	11:29 min
nr_outliers	24:35 min	50.99s	0.44s	11:33 min
p_trace	24:35 min	50.77s	0.44a	11:35 min
range	35:49 min	51.02s	0.44s	11:35 min
roy_root	26:51 min	52.16s	0.44s	11:36 min
sd	26:14 min	51.04s	0.45s	11:40 min
sd_ratio	27:26 min	50.71s	0.45s	11:37 min
skewness	24:42 min	50.95s	0.49s	12:05 min
sparsity	27:49 min	50.67s	0.42s	11:24 min
t_mean	25:17 min	50.95s	0.43s	11:13 min
var	24:23 min	51.76s	0.42s	11:10 min
w_lambda	23:56 min	50.65s	0.43s	11:08 min
FULL ANALYSIS	28:16 min	51.44s	62.65s	11:25 min

D). We decided to use a series of smaller, publicly available datasets[3]. Several other datasets were also used in the analysis, but as dataset dimensionality grew, so did time required to analyse all 19 meta features plus the full analysis. Runtime of each analysis is presented in Table 5.3.

The outcome of this analysis was unexpected. Even with some calculations taking more time, the differences are predominantly negligible. In fact, the last row shows that the library

[3]perso.telecom-paristech.fr

takes roughly the same time to perform a complete analysis as it does on every individual meta-extraction computation. With this result, proceeding with analysing more enormous datasets proved unnecessary.

Further exploration of the pymfe library proved that improving the calculations it made would be a daunting task. Pymfe was developed by Alcobaça, Siqueira, Rivolli, Garcia, Oliva, and de Carvalho (2020) the leading experts in the field regarding meta analysis. A full review of the library found on kandii[4] also showed that the library is of the highest quality. Its most significant flaws were its high code complexity (9614 lines of code, 461 functions and 56 files) and lack of current releases (it had no major release in the last 12 months). As code complexity directly impacts the maintainability of the code, trying to influence its results would require more time and resources than what was available. We were left with no choice but to accept the high wait times when researching some meta feature families.

While abandoning problematic families could alleviate the problem, it would not solve it. Even general meta feature extraction could take a long time to complete, given a large enough dataset. In fact, even problematic familes can achieve results in smaller datasets. Ultimately, looking at the tool as a product, we find it more beneficial to leave the choice up to the end user than to remove it outright.

The meta-extraction 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 extraction 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.

Ultimately the Meta-extraction 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.

[4]https://kandi.openweaver.com/ - kandi is a code and library review website

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 Section 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 influence globally. 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 some of the previously referred meta families. Then, select 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 General

General meta features are the simplest ones to force onto a dataset, as they are closely connected to the main parameters and characteristics of the generated synthetic dataset. One aspect associated with the general meta feature family is dataset dimensionality.

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 dimension.

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 in 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).

We will now look at one meta feature connected with dimensionality - *attr_to_inst*. This meta feature represents the ratio between the number of instances and attributes in the dataset. Let's now see if it is possible to force a value of this meta feature onto the generation process and if this inclusion is valuable.

Regarding the inclusion of a specific general meta feature value to a synthetic dataset, how can we force a particular 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. The formula to calculate *attr_to_inst* meta feature can be seen in Equation 6.1.

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

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. 6.2 is one of such rules.

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

Where O is the original dataset's column count, and δ 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. Those conditions are presented in Equations 6.3 and 6.4.

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

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

And Equation 6.5 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 equations presented in 6.6 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.

Now that we solved the forced inclusion of a set *attr_to_inst* value, let's now see how sound is this inclusion.

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 the dimensionality of a dataset should be extracted. However, forcing a set value for a dimensionality-related meta feature (like *attr_to_inst*) is not helpful in dataset generation.

6.2 Statistical

Assembling datasets with set values for statistical meta features would facilitate the creation of more diverse and valid datasets. These would have different characteristics, all using the same original dataset. Indicating that there is value in diving into forcing some of these meta features into a synthetic dataset. In this section, we will look at skewness and kurtosis, seeing how their inclusion would change the dataset and displaying a way to force them into the generation process.

6.2.1 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., 2018).

Figure 6.1 shows a visual representation of skewness. In this figure, the first graphic depicts a left skewness, informing us that the mean of the values is smaller than the median, which is smaller than the mode. The central graph illustrates a symmetrical normal, where the mean, median and mode share the same value. Finally, the right figure depicts a skewed right graph. In this type of skewness, the mean is greater than the medium, that is, greater than the mode.

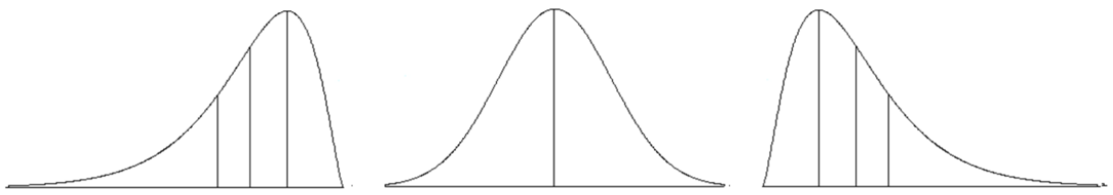


Figure 6.1: Visual representation of skewness. Removed from Doane and Seward (2011)

The skewness of attributes is calculated through the Equation 6.7:

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

Where sd_x^3 and m_3 are extracted from the formulas 6.8 and 6.9.

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

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

Applying skewness to the generation process can require only a tiny tweaking in dataset generators that already use a normal distribution to create synthetic datasets. Luckily, python's library script Skewnorm's function[1] had the required functions that allowed us to perform our experimentation. This function takes a number as a skewness parameter. When that number equals 0, the distribution is identical to a normal distribution.

In Listing 6.1 we display the used code to perform the insertion of a skewness of value -2 into a normal distribution.

[1]docs.scipy.org/

Listing 6.1: Integrating skewness

```
NumValues = 10000
maxValue = 6
skewness = −2

random = skewnorm.rvs(a=skewness, loc=maxValue, size=numValues)

random = random − min(random)
random = random / max(random)
random = random * maxValue
```

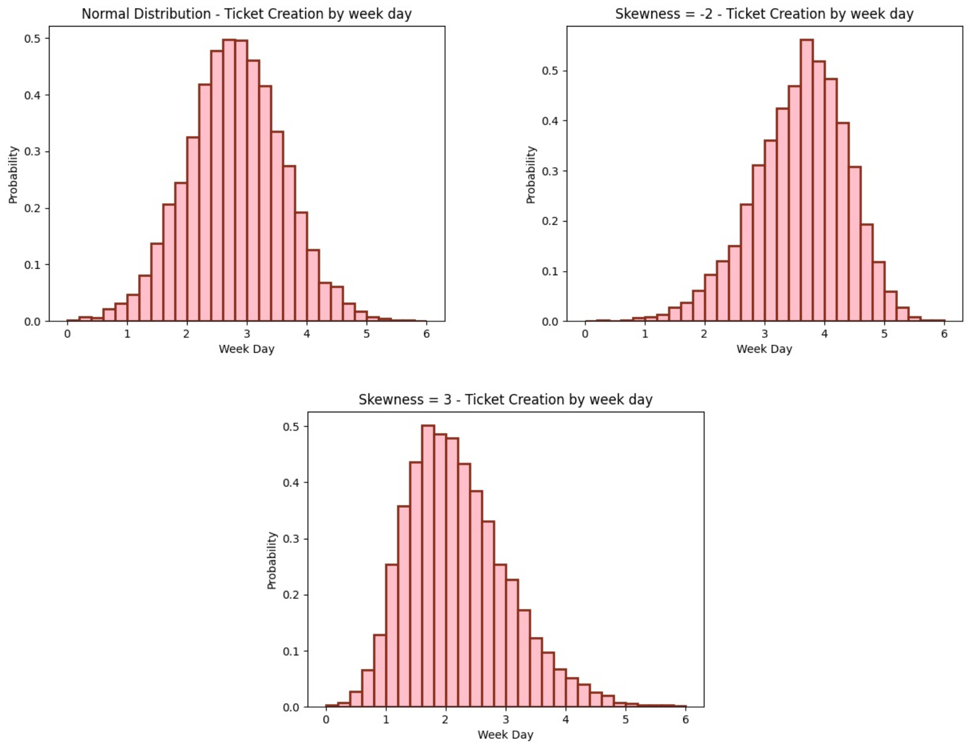


Figure 6.2: Skewness integration

We could then move on to experimentation. Using a normal distribution, we studied the distribution of 10000 tickets generated by day of the week (0 being Sunday and 6 Saturday). We tried to influence the distribution probability by inputting a skewness value.

The results of this testing can be seen in Figure 6.2. We started by creating a normal distribution that had a skewness of 0. We changed that value to -2 and 3 and exported the end results in graphical form.

Having these outcomes, we can conclude that we can indeed generate a synthetic dataset with a chosen value of kurtosis by influencing the probability distribution of values in the generator.

6.2.2 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 analogized against a typical normal distribution or bell curve.

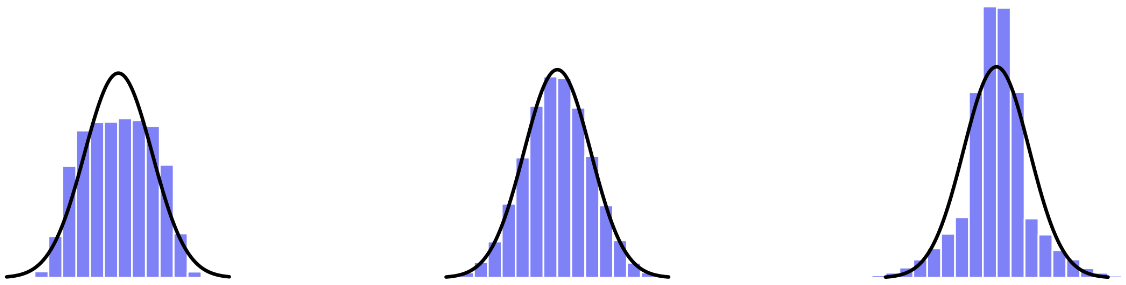


Figure 6.3: Visual representation of kurtosis in histogram form. Removed from Navarro and Foxcroft (2019)

As depicted in Fig. 6.3 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 formula 6.10.

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

Where m_4 is calculated using the formula presented in Equation 6.9

As with Skewness integration, adding desired kurtosis values could be done by changing the parameters of the output entries. However, influencing this value is not as easy as initially thought. Using a simple value of Kurtosis was not possible. However, we could try using different distributions and see how their Kurtosis values would influence the shape of the output dataset.

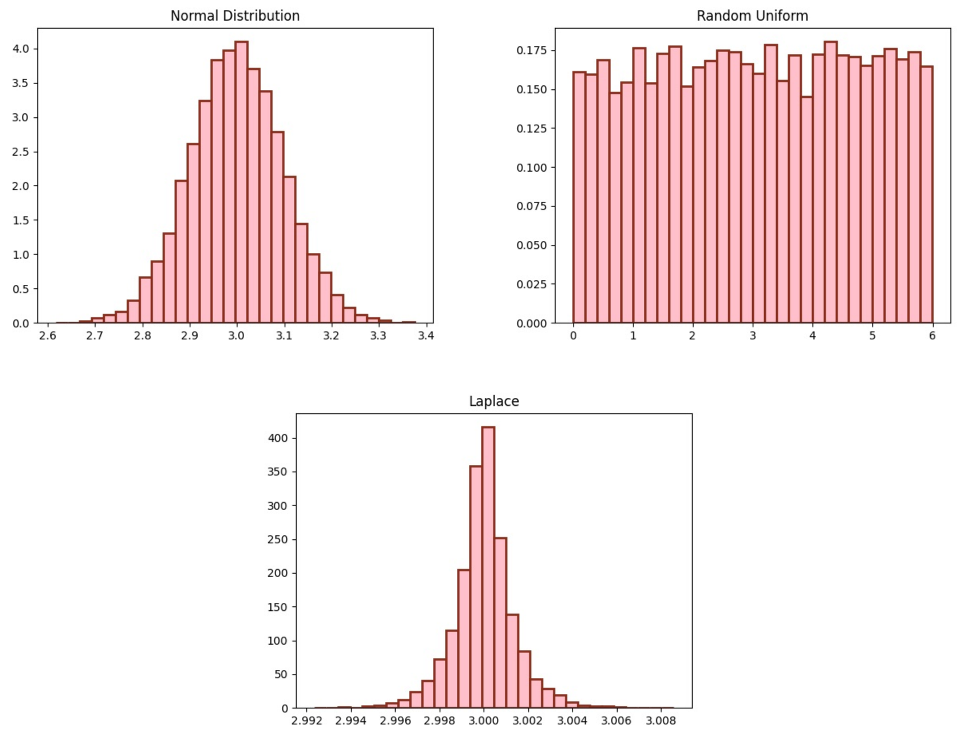


Figure 6.4: Normal, Random Uniform and Laplace distributions

Results of this testing can be seen in Figure 6.4. We used different distributions to generate data with distinguishable characteristics. We then calculated kurtosis on these distributions and see if its values changed according to the distribution. The uniform distribution had a kurtosis value of 1.8, Normal distribution had a value of 3. Finally, with Laplace, we arrived at a kurtosis value of 6.59.

While we found that it is possible to influence the value of kurtosis we found to way of correctly influencing that same value with a simple user input.

6.3 Information-theoretic

Information-theoretic meta features are based on entropy, capturing the data's amount of information and complexity. The entropy (H_x) can be given by the formula 6.11.

$$H_x = -\sum_{x=1}^{N} P_x log_2(P_x) \tag{6.11}$$

Where P_x represents the probability of a category (x) to occur.

We can use this formula to calculate certain meta features in this family, such is the case of the classEnt meta feature.

Solving this problem to force a specific entropy value is a mathematical one that exceeds our capabilities. Nevertheless, from an engineering standpoint, we propose the analysis of entropy at set intervals during the generation process. For each gap, we would freeze the generation, calculate entropy and see how far we are from our target. In cases where values are superior to the desired ones, we should try to generate more noise, reducing the more significant percentages and lowering the entropy. In cases where we were already lower, we should focus the generation on existing, more probable values. This would increase their final probability and levelling our entropy values.

Would setting a particular value of entropy be helpful to the final dataset? In a simple answer, yes. Entropy measures disorder or impurities in the information processed in machine learning. The ability to choose that purity level has obvious benefits from an ML study standpoint.

6.4 Landmarking

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

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

6.5 Dataset Morphing

One option we came across to use to create datasets with specific values of meta features was dataset morphing. With dataset morphing, we could change it after having a complete dataset and altering its values to achieve the desired meta feature value results.

This, however, comes with a set of problems. Let's use the same case we already used with kurtosis and skewness. So we want to alter the ticket generation time by day of the week. To do this, we need to finish the generation process and then change the data on columns related to that information. This change also needs to impact the rest of the dataset, not just that one column. The generation process would need to be halted in columns whose data depends on the morphed columns. Only after the data connected to the morphing finishes being generated and is morphed can the generation process continue.

This process is more time and resource expensive than normal run-time generation.

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. We explored the possibility of including general, statistical, information-theoretic and landmarking meta features. We also looked at the validity of incorporating these meta features in each analysis.

We also looked at the possibility of dataset morphing to insert meta feature values into a synthetic dataset.

While we found it possible to insert meta features, we did it in a singular factor. By inserting a meta feature, we cannot guarantee that the values of other meta features in the dataset will be maintained.

Datasets generated with dataset integration, be it by runtime integration or morphing should them be analysed to see if they continue to be valid and realistic. A debugging analysis should be made, in order to guarantee the stability of buisiness rules inside the dataset to guarantee the validity of the data. Some of the data evaluation methods described in Chapter 3 should also be used to check the validity of the final output.

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. We proved that this implementation can be done on 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 complicated 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.' With SNOOKER as the original basis, we analyzed the concept of meta features, exploring the subject and presenting the most commonly known family of meta features. While groups of meta features are not uniformly described, the list presentive is extensive and varied enough to characterize a dataset correctly.

The study advanced to dataset generation methodologies. While explored, this subject was not developed further during the development phase.

The Classification feature was appended to SNOOKER. We used the pyfme library to help us extract as many meta features from generated datasets as possible, allowing for vast classification options.

We also explored ways to force specific meta feature results on synthetic datasets. Due to a lack of practical results, our observations were purely theoretical, having no sound proof to correctly evaluate our hypotheses. While we believed that inserting meta features into synthetic datasets is a possible and (in some cases) beneficial endeavour, no testing was finalized.

The development of this study went through more problems than expected. The meta-extraction tool ended up occupying most of the time available for development. There is also an evident lack of knowledge and know-how regarding dataset analysis and generation processes that hindered the development and limited the scope of the study.

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 profit from a significant overhaul, becoming decentralized. Having all the necessary calculations done remotely on a server. As meta-analysis of some families could take a long time, making them run in a server would prevent users from needing to keep processes running for long times. The results could be sent to the user at a later date. We conclude that SNOOKER would benefit from evolving into a web service.

References

Hervé Abdi, Dominique Valentin, and Betty Edelman. *Neural networks*. Quantitative Applications in the Social Sciences. Sage Publications, Inc, 1999. ISBN 9781412985277. doi: 10.4135/9781412985277.

Arnisha Akhter, Uzzal K Acharjee, and Md Masbaul A Polash. Cyber Bullying Detection and Classification using Multinomial Naïve Bayes and Fuzzy Logic. *Modern Education and Computer Science*, 2019. doi: 10.5815/ijmsc.2019.04.01.

Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís Paulo F. Garcia, Jefferson Tales Oliva, and André C. P. L. F. de Carvalho. MFE: Towards Reproducible Meta-Feature Extraction. *Journal of Machine Learning Research*, 21:111:1–111:5, 2020. URL http://jmlr.org/papers/v21/19-348.html.

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. ISSN 0034-4257. doi: https://doi.org/10.1016/j.rse.2019.04.032. URL https://www.sciencedirect.com/science/article/pii/S003442571930183X.

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, and Tomís Aluja-Banet. On the Predictive Power of Meta-Features in OpenML. *International Journal of Applied Mathematics and Computer Science*, 27(4):697–712, dec 2017. ISSN 1641-876X. doi: 10.1515/amcs-2017-0048. URL https://doi.org/10.1515/amcs-2017-0048.

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.

Nathaniel Boggs, Hang Zhao, Senyao Du, and Salvatore J. Stolfo. Synthetic Data Generation and Defense in Depth Measurement of Web Applications. In Angelos Stavrou, Herbert Bos, and Georgios Portokalidis, editors, *Research in Attacks, Intrusions and Defenses*, pages 234–254, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11379-1. doi: 10.1007/978-3-319-11379-1_12.

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. *Transactions on Data Privacy*, 3(1):27–42, apr 2010. ISSN 1888-5063. doi: 10.5555/1747335.1747337.

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*, Programming Language Design and Implementation 2019, page 394–410, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367127. doi: 10.1145/3314221.3314594. URL https://doi.org/10.1145/3314221.3314594.

Ciro Castiello, Giovanna Castellano, and Anna Maria Fanelli. Meta-Data: Characterization of Input Features for Meta-Learning. In *Proceedings of the Second International Conference on Modeling Decisions for Artificial Intelligence*, International Conference on Modeling Decisions for Artificial Intelligence'05, page 457–468, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3540278710. doi: 10.1007/11526018_45. URL https://doi.org/10.1007/11526018_45.

Serena H. Chen and Carmel A. Pollino. Good Practice in Bayesian Network Modelling. *Environmental Modelling & Software*, 37:134–145, 2012. ISSN 1364-8152. doi: https://doi.org/10.1016/j.envsoft.2012.03.012. URL https://www.sciencedirect.com/science/article/pii/S1364815212001041.

Stanley Cohen. Chapter 1 - The Evolution of Machine Learning: Past, Present, and Future. In Stanley Cohen, editor, *Artificial Intelligence and Deep Learning in Pathology*, pages 1–12. Elsevier, 2021. ISBN 978-0-323-67538-3. doi: https://doi.org/10.1016/B978-0-323-67538-3.00001-4. URL https://www.sciencedirect.com/science/article/pii/B9780323675383000014.

Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative Adversarial Networks: An Overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018. doi: 10.1109/MSP.2017.2765202.

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 A. M. Zen, and Stéphane Bressan. A Comparative Study of Synthetic Dataset Generation Techniques. In Sven Hartmann, Hui Ma, Abdelkader Hameurlain, Günther Pernul, and Roland R. Wagner, editors, *Database and Expert Systems Applications*, pages 387–395, Cham, 2018. Springer International Publishing. ISBN 978-3-319-98812-2. doi: 10.1007/978-3-319-98812-2_35.

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.

Susan Dean and Barbara Illowsky. Descriptive Statistics: Skewness and the Mean, Median, and Mode. *Connexions website*, 2018. URL https://resources.saylor.org/wwwresources/archived/site/wp-content/uploads/2011/06/MA121-1.3.3.pdf.

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 Josep Domingo-Ferrer and Emmanouil Magkos, editors, *Privacy in Statistical Databases*, pages 148–161, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15838-4. doi: 10.1007/978-3-642-15838-4_14.

David B. Dunson and Chuanhua Xing. Nonparametric Bayes Modeling of Multivariate Categorical Data. *Journal of the American Statistical Association*, 104(487):1042–1051, 2009. doi: 10.1198/jasa.2009.tm08439. URL https://doi.org/10.1198/jasa.2009.tm08439.

Khaled El Emam. Seven ways to evaluate the utility of synthetic data. *IEEE Security & Privacy*, 18(4):56–59, 2020. doi: 10.1109/MSEC.2020.2992821.

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. ISBN 978-9-5268-3970-7. 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. ISBN 978-1-5090-4307-1. 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, 2020. doi: 10.1186/s12874-020-00977-1. URL https://doi.org/10.1186/s12874-020-00977-1.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5769–5779, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964. doi: 0.5555/3295222.3295327.

Donald Olding Hebb. *The Organisation of Behaviour: a Neuropsychological Theory*. Science Editions New York, 1949. ISBN 978-0805843002. doi: 10.4324/9781410612403.

Tin Kam Ho and M. Basu. Complexity Measures of Supervised Classification Problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):289–300, 2002. doi: 10.1109/34.990132.

Cook A Holbrook R. Intro to Deep Learning. Technical report, Kaggle, 2020.

Navya Jose, Bharathi Raja Chakravarthi, Shardul Suryawanshi, Elizabeth Sherly, and John P. McCrae. A Survey of Current Datasets for Code-Switching Research. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 136–141, 2020. ISBN 978-1-7281-5197-7. doi: 10.1109/ICACCS48705.2020.9074205.

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.or g/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. arXiv, 2019. ISBN 978-1-7281-4803-8. doi: 10.48550/ARXIV.1904.11621. URL https://arxiv.org/abs/1904.11621.

K. Kirchner, K.-H. Tölle, and J. Krieter. Decision Tree Technique Applied to Pig Farming Datasets. *Livestock Production Science*, 90(2):191–200, 2004. ISSN 0301-6226. doi: https://doi.org/10.1016/j.livprodsci.2004.04.003. URL https://www.sciencedirect.com/science/article/pii/S0301622604001009.

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 (Industry Papers)*, pages 177–184, New Orleans - Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-3022. URL https://aclanthology.org/N18-3022.

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 Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38652-7. doi: 10.1007/978-3-642-38652-7_2. URL https://doi.org/10.1007/978-3-642-38652-7_2.

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, 2018. doi: 10.1109/ICICCT.2018.8473214.

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 a Artificial Data Generation Approach: A Review. In *Journal of Physics: Conference Series*, volume 892, page 012016. IOP Publishing, 2017. doi: 10.1088/1742-6596/892/1/012016.

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, 2018. ISBN 978-1-5386-5035-6. doi: 10.1109/BigData.2018.8621982.

Guido Lindner and Rudi Studer. Ast: Support for algorithm selection with a cbr approach. In Jan M. Zytkow and Jan Rauch, editors, *Principles of Data Mining and Knowledge Discovery, Third European Conference, PKDD 99, Prague, Czech Republic, September 15-18, 1999, Proceedings*, volume 1704 of *Lecture Notes in Computer Science*, pages 418–423. Springer, 1999. ISBN 3-540-66490-4. doi: 10.1007/978-3-540-48247-5_52.

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.

D. Michie, D. Spiegelhalter, and Charles Taylor. *Machine Learning, Neural and Statistical Classification*, volume 37. 01 1999. ISBN 978-0-13-106360-0. doi: 10.2307/1269742.

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, 2014. URL https://arxiv.org/abs/1411.1784.

Mario A. Muñoz, Laura Villanova, Davaatseren Baatar, and Kate Smith-Miles. Instance Spaces for Machine Learning Classification. *Machine Learning*, 107(1):109–147, 2018. doi: 10.1007/s10994-017-5629-5. URL https://doi.org/10.1007/s10994-017-5629-5.

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. doi: 10.24384/hgc3-7p15.

Eduardo Perez and Larry A. Rendell. Learning despite Concept Variation by Finding Structure in Attribute-Based Data. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, page 391–399, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. ISBN 1558604197. doi: 10.5555/3091696.3091743.

Konstantinos Perrakis, Ioannis Ntzoufras, and Efthymios Tsionas. On the Use of Marginal Posteriors in Marginal Likelihood Estimation Via Importance-Sampling. *Computational Statistics & Data Analysis*, 77, 11 2013. doi: 10.1016/j.csda.2014.03.004.

S Selva Priyanka, Sudeep Galgali, S Selva Priya, B R Shashank, and K G 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, 2016. ISBN 978-1-5090-5369-8. doi: 10.1109/CIMCA.2016.8053293.

S Ramya and S Nanda. Breast Cancer Detection and Classification Using Ultrasound and Ultrasound Elastography Images. *International Research Journal of Engineering and Technology*, 4:596–601, 2017. ISSN 2395-0056.

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. doi: 10.1007/s10462-020-09928-0. URL https://doi.org/10.1007/s10462-020-09928-0.

William S. Rayens. Discriminant Analysis and Statistical Pattern Recognition. *Technometrics*, 35(3):324–326, 1993. doi: 10.1080/00401706.1993.10485331. URL https://www.tandfonline.com/doi/abs/10.1080/00401706.1993.10485331.

Matthias Reif, Faisal Shafait, and Andreas Dengel. Prediction of Classifier Training Time Including Parameter Optimization. In Joscha Bach and Stefan Edelkamp, editors, *KI 2011: Advances in Artificial Intelligence*, pages 260–271, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-24455-1. doi: https://doi.org/10.1007/978-3-642-24455-1_25.

Matthias Reif, Faisal Shafait, and Andreas Dengel. Dataset Generation for Meta-Learning. *KI-2012: Poster and Demo Track*, pages 69–73, 2012. doi: 10.1007/978-3-030-33607-3_43.

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, 09 2005. URL https://www.proquest.com/scholarly-journals/using-cart-generate-partially-synthetic-public/docview/1266792149/se-2. Copyright - Copyright Statistics Sweden (SCB) Sep 2005; Last updated - 2013-01-07.

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. doi: https://doi.org/10.1002/meet.14504701240. URL https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/meet.14504701240.

Jason DM Rennie. Mixtures of Multinomials, 2005. URL http://people.csail.mit.edu/jrennie/writing/mixtureMultinomials.pdf.

D. Karthika Renuka, T. Hamsapriya, M. Raja Chakkaravarthi, and P. Lakshmi Surya. Spam Classification Based on Supervised Learning Using Machine Learning Techniques. In *2011 International Conference on Process Automation, Control and Computing*, pages 1–7, 2011. doi: 10.1109/PACC.2011.5979035.

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. *arXiv: Learning*, 2018. doi: 10.48550/ARXIV.1808.10406. URL https://arxiv.org/abs/1808.10406.

Adriano Rivolli, Luís P.F. Garcia, Carlos Soares, Joaquin Vanschoren, and André C.P.L.F. de Carvalho. Meta-Features for Meta-Learning. *Knowledge-Based Systems*, 240:108101, 2022. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2021.108101. URL https://www.sciencedirect.com/science/article/pii/S0950705121011631.

Rebecca Roelofs, Sara Fridovich-Keil, John Miller, Vaishaal Shankar, Moritz Hardt, Benjamin Recht, and Ludwig Schmidt. A Meta-Analysis of Overfitting in Machine Learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Donald B Rubin. An Overview of Multiple Imputation. In *Proceedings of the survey research methods section of the American statistical association*, pages 79–84. Citeseer, 1988.

Donald B. Rubin. Discussion Statistical Disclosure Limitation. *Journal of Official Statistics*, 9 (2):461, 06 1993. URL https://www.proquest.com/scholarly-journals/discussion-statistical-disclosure-limitation/docview/1266818482/se-2. Copyright - Copyright Statistics Sweden (SCB) Jun 1993; Last updated - 2013-01-07.

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. ISBN 978-1-6654-9503-5. doi: 10.1109/CLEI53233.2021.9640223.

Saddys Segrera, Joel Pinho, and María N. Moreno. Information-Theoretic Measures for Meta-learning. In Emilio Corchado, Ajith Abraham, and Witold Pedrycz, editors, *Hybrid Artificial*

Intelligence Systems, pages 458–465, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-87656-4. doi: https://doi.org/10.1007/978-3-540-87656-4_57.

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, 2017. ISBN 978-1-5386-0627-8. doi: 10.1109/IC3TSN.2017.8284445.

Tegjyot Singh Sethi and Mehmed Kantardzic. When good machine learning leads to bad security: Big data (ubiquity symposium). *Ubiquity*, 2018(May), 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. ISBN 978-1686474712.

Florian Skopik, Giuseppe Settanni, Roman Fiedler, and Ivo Friedberg. Semi-Synthetic Data Set Generation for Security Software Evaluation. In *2014 Twelfth Annual International Conference on Privacy, Security and Trust*, pages 156–163, 2014. ISBN 978-1-4799-3503-1. doi: 10.1109/PST.2014.6890935.

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. ISSN 1662-5544.

Yan-Yan Song and LU Ying. Decision Tree Methods: Applications for Classification and Prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015. doi: 10.11919/j.issn.1002-0829.215044.

Mark Summerfield. *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming (paperback)*. Pearson Education, 2007. ISBN 978-0134393339.

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, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1033. URL https://aclanthology.org/D16-1033.

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*. 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. *Journal of Artificial Intelligence Research*, abs/1402.0570, 2014. doi: https://doi.org/10.1613/jair.3831. 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. doi: 10.1007/s10489-015-0689-3. URL https://doi.org/10.1007/s10489-015-0689-3.

Herbert Weisberg and Herbert F Weisberg. *Central Tendency and Variability*. Number 83. 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. doi: 10.5121/csit.2021.1119 11. URL https://csitcp.org/paper/11/1119csit11.pdf.

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, 2011. ISBN 978-1-61284-758-0. doi: 10.1109/ASONAM.2011.118.

Xingyu Zhou, Yuling Jiao, Jin Liu, and Jian Huang. A deep generative approach to conditional sampling. *Journal of the American Statistical Association*, 0(0):1–12, 2022. doi: 10.1080/0162 1459.2021.2016424. URL https://doi.org/10.1080/01621459.2021.2016424.

Eric R Ziegel. The Elements of Statistical Learning. *Technometrics*, 45(3):267–268, 2003. doi: 10.1198/tech.2003.s770. URL https://doi.org/10.1198/tech.2003.s770.

SilvioSandoval Zocchi and Bryan FJ Manly. *Generating Different Data Sets for Linear Regression Models With the Same Estimates*. 2006.

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 dataset column. Columns can be classified as primary and secondary. While primary columns display information essential for the dataset, secondary columns facilitate debugging actions. Other columns exist when generating datasets for different ends. Still, the study and analysed are based only on the helpdesk dataset type output. 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 (its 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 register 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 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 with 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 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, responsible for informing the user that the ticket should be passed to the next team. The Status column provides the ticket status (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 Clustering 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                     0.022007585662175888
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.
```

Appendix D

Statistical Analysis Output

```
Analysis of file ../../Output/generatedDatasets/SD1.csv.
can_cor.mean                1.0
can_cor.sd                  2.863353569008059e-16
Analysis took 0:26:32.801149.
cor.mean                    0.0031818656040018055
cor.sd                      0.029273671606693415
Analysis took 0:27:16.633176.
cov.mean                    0.00018169921467679913
cov.sd                      0.06544132326412351
Analysis took 0:25:29.343186.
eigenvalues.mean            8.517833528021107
eigenvalues.sd              584.2881540682121
Analysis took 0:31:03.024574.
g_mean.mean                 0.02200758566217588
g_mean.sd                   0.8030517936688358
Analysis took 0:25:42.758934.
gravity                     14.491376746189438
Analysis took 0:26:54.196286.
h_mean.mean                 0.017814411761191074
h_mean.sd                   0.6219661342834908
Analysis took 0:25:32.138877.
iq_range.mean               0.018441748588161472
iq_range.sd                 0.6913274616349807
Analysis took 0:27:45.718662.
kurtosis.mean               878.1186853398123
kurtosis.sd                 283.23712818907694
Analysis took 0:26:54.988853.
lh_trace                    inf
Analysis took 0:26:04.834030.
mad.mean                    0.009141821376281114
mad.sd                      0.35732979367673556
Analysis took 0:28:52.903699.
max.mean                    1.3132189918427106
max.sd                      19.345451307983584
Analysis took 0:26:29.736283.
```

```
mean.mean                         0.0399734463403152
mean.sd                           1.6495148276889133
Analysis took 0:26:56.504050.
median.mean                       0.01930767621836436
median.sd                         0.6700330673798763
Analysis took 0:27:36.376224.
min.mean                          0.006065676636686885
min.sd                            0.21001299789299555
Analysis took 0:26:31.345163.
nr_cor_attr                       0.0007765221295242062
Analysis took 0:47:58.595005.
nr_disc                           1000
Analysis took 0:27:54.741807.
nr_norm                           0.0
Analysis took 0:27:40.943515.
nr_outliers                       4774
Analysis took 0:24:35.350784.
p_trace                           999.9999999999998
Analysis took 0:24:35.025303.
range.mean                        1.3071533152060237
range.sd                          19.23267679677396
Analysis took 0:35:49.239344.
roy_root                          inf
Analysis took 0:26:51.358588.
sd.mean                           0.08891679211579023
sd.sd                             2.9174831022006753
Analysis took 0:26:14.329449.
sd_ratio                          nan
Analysis took 0:27:26.405710.
skewness.mean                     28.866039090472135
skewness.sd                       6.782417085249947
Analysis took 0:24:42.426164.
sparsity.mean                     0.13476022492442502
sparsity.sd                       0.19501828184302594
Analysis took 0:27:49.408391.
var.mean                          8.517833528021098
var.sd                            584.2706323926961
Analysis took 0:24:23.294747.
w_lambda                          0.0
Analysis took 0:23:56.400821.
```