Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Relevance of the two adjusting screws in data analytics: data quality and optimization of algorithms

Shreyas Bettadapura Raghavendra

Course of Study:	Computer Science
Examiner:	Prof. Dr. -Ing. habil. Bernhard Mitschang
Supervisor:	Alejandro Gabriel Villanueva Zacarias, M.Sc., Cornelia Kiefer, M.Sc.
Commenced:	January 26, 2017
Completed:	July 26, 2017
CR-Classification:	H.2.8, I.2.6

Abstract

In the context of learning from data, the impact on the performance of a learning algorithm has traditionally been studied through the perspective of data preprocessing and through that of empirical works. We attempt to provide a middle ground by employing an approach which enables a systematic analysis considering the interaction between the quality of the data provided for training, and the configurations applied to the learning algorithm. This is achieved through the concepts of a *Data Quality Profile*, which depicts quality indicators for the dataset and a *Classification Configuration Profile*, which depicts the configuration parameters applied to the learning algorithm. Both the profiles have the common characteristic of being able to distinctly view, and equally represent the variations in their properties, allowing for a systematic study. We demonstrate this through a prototypical implementation, considering the data quality indicators of missing values, label imbalance, and high cardinality, and evaluating it against the CART Decision Tree algorithm, configurable by its splitting criteria, early stopping criteria, and training data preprocessing operations. We were able to successfully observe a relationship between decreasing quality of the training data, and deterioration in the performance of the algorithm. The flexibility of the approach allows for easy progression to other algorithms, and implementations of more quality indicators.

Contents

List of Figures

List of Tables

1 Introduction

The context of this thesis is the analytics of large scale data. As noted by [SGM14], the advent of social media and the Internet of Things has brought about an explosion in the amount of data available, leading to a "data deluge". The primary reason for this is the availability of data through a large number of sources, leading to data of a heterogeneous nature. The challenges encountered due to this scale of data are primarily in the areas of data storage, and in analysis. Storage needs special emphasis due to the varied levels of quality associated with such heterogeneous data. Analysis of such data is not possible on conventional single core machines with regular statistical methods. It needs parallel computing, with technology and inferential techniques which can cope with this scale and distribution.

1.1 Data Analytics Pipeline

Modern data analytics pipelines are built to handle such challenges, as observed by [SO13]. Such a pipeline is shown in Figure 1.1. A brief description of the phases of this pipeline follows [SO13]. A graphical depiction of this pipeline has been attempted in Figure 1.1, based on the below descriptions.

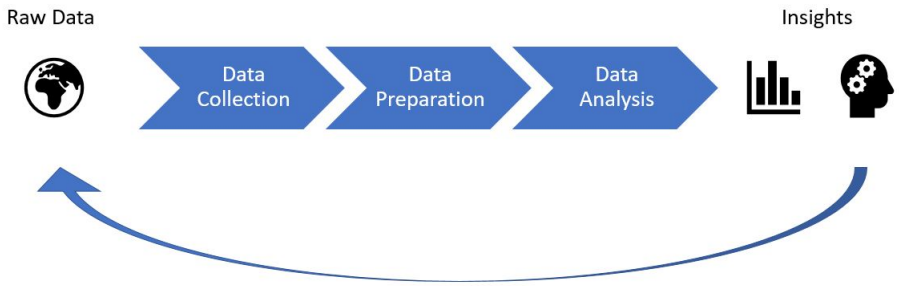

Figure 1.1: The Analytics Pipeline [SO13]

Data Collection is the process of surveying the data available in the real world, and using the subset relevant and necessary for the task at hand. In order to be able to identify such data, it is essential that the problem space and requirements are clearly defined [Pyl99]. Once these are defined, the subsequent challenge is to procure this data through an interface made available by the data source providers. The completion of this step indicates that the data required for the task at hand is now available. This data now needs to be explored to understand its essential characteristics, and its readiness for the subsequent analysis stage of the pipeline.

This is achieved through the *Data Preparation* phase, which deals with processing and transforming the data to a format best suited for the *Learning Algorithm*. This has been called as *Data Cleaning* [RD00], *Exploratory Data Analysis* [Pyl99], *Data Wrangling* [Kan+11], [Ter+15], and specifically in the area of learning algorithms, as *Data Preprocessing* [KKP06]. Irrespective of the terminology, the purpose served is the same. A common theme indicated in all these works is the necessity of performing this phase iteratively. That is, every transformation performed highlights some new characteristics of the data, which have to be learned from to take calculated decisions such as procuring new data, or obtaining a different perspective on the data.

Data Preparation, thus, can be seen through two distinct perspectives. One, with the aim of understanding characteristics of the data and taking corrective actions if necessary, and two, with the aim of transforming the data to a format better suited for the learning algorithm. The former deals with getting an understanding of the quality of the dataset at hand. The observations made here can be, for example, finding the amount of noise, missing values, invalid values and outliers in the data. Such problems with data quality are dealt with through *Data Cleaning*, which has the sole purpose of "improving the quality of data" [RD00]. On the other hand, *Data Preprocessing* suggests transformation operations such as missing value imputation, discretizing variables which have a continuous range and normalizing the scale of the variables. Though these operations might have an indirect effect on the quality of the dataset, their purpose is purely for the sake of the learning algorithm. The end result of the preparation phase, hence, is the dataset in a format which is ready for consumption by the learning algorithm.

The final phase of this pipeline deals with the application of a learning algorithm to analyse the data and derive insights. The most important decision which needs to be taken in this phase is the choice of learning algorithm to be used. This depends on the nature of the learning task, that is, whether the purpose of the learning task is mere

exploration to understand patterns in the data, or learning from the dataset to make real world predictions. In the latter case, an additional factor is the range of values this target variable can take, which could either be a continuous range, or a discrete set of values. Once an estimate on the most suitable algorithm has been made, their configurations must be tuned in the best way possible for the task at hand. These parameters which can be configured for an algorithm are referred to in this work as *hyperparameters*. Tuning using such hyperparameters constitute optimizing the learning algorithm. Using suitable testing techniques [KZP06], the quality of the learning model constructed is evaluated, and further decisions taken. This is an iterative process, and multiple combinations must be tried to come to a fruitful conclusion.

The appropriately configured learning algorithm is then applied to data in the real world, and the insights obtained are provided back so that calculated decisions can be taken. This forms a feedback loop, which in turn generates more data for the use case.

1.2 Research Questions

The task description for this work is based on [VZK16]. Before we get into the questions we attempt to answer through our work, we first define our area of focus. In the context of the analytics pipeline described above, we focus on the following two themes: the quality of the data collected, henceforth referred to as *Data Quality*, and the configurations applied to the *Learning Algorithm* used in the analysis phase. The place they occupy in the pipeline is shown in Figure 1.2. *Data Quality*, discussed in Section 2.1 is a measure of the goodness of fit of the data for the eventual consumer of this data, which in our context, is the *Learning Algorithm* used during the analysis phase. The configurations applied to the *Learning Algorithm* are used to configure the way in which the model is constructed, and the data transformation steps applied before being consumed for learning. These are further described in Section 2.5.3.

Given the above focus area, through our work, we wish to understand how the *interaction* between the quality of the dataset used to train a learning model, and the configurations applied to the learning algorithm which generates this model affect the performance of the learning algorithm. The motivation for undertaking this work comes originally from [VZK16], and existing work, which deal individually with the two aspects, and those which study the influence of quality on the performance of the learning algorithm through empirical evaluations.

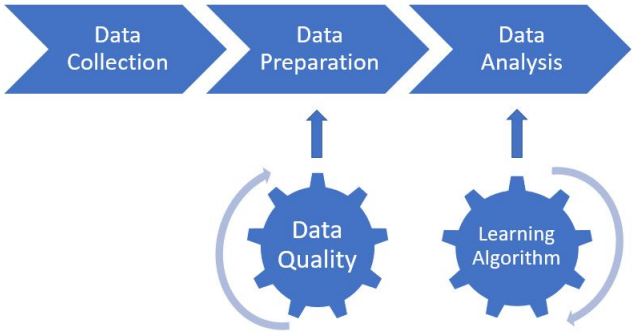

Figure 1.2: Scope of thesis in the analytics pipeline

As detailed in Chapter 3, we look at work in two categories. The first kind focus on the more obvious method of performing empirical evaluations by training a learning model using data with varying levels of quality, and observing and reasoning on the results. The most prominent example for this is the work by Quinlan, in which he formalized the C4.5 decision tree [Qui86], and explicitly argued on the effect of noise on the way the tree is constructed, and on its performance. The second kind focus on the preprocessing techniques which can be applied on the data before being supplied to a learning algorithm, with the aim of improving the latter's performance. Our work builds on the knowledge gained through such existing works and attempts to provide a cohesive perspective of the variations in both these aspects. We claim that obtaining such a perspective enables us to perform a systematic analysis of the *interaction*.

The approach we suggest in order to obtain such a perspective is the concept of a *profile*, as first described in [VZK16], which we initially define as a collection of information items which provide descriptive metadata for a concrete theme. We utilize this concept through the use of a *Data Quality Profile*, abbreviated as DQP, described in Section 4.2 and a *Classification Configuration Profile*, abbreviated as CCP, described in Section 4.3. A DQP for a dataset is a collection of indicators which describe the quality of the dataset. We define the following quality indicators for every field of the dataset: *Cardinality*, *Proportion of Missing Values*, *Proportion of Invalid Values*, and *Proportion of Noisy Tokens* for free text fields. A CCP for a particular classification task is a collection of parameters which detail the configurations applied to the learning algorithm. The configurations defined are particular to the learning algorithm in question, and for our focus of the CART Decision Tree algorithm, we define the following parameters: *Splitting Criteria*, *Early Stopping Criteria*, *Dataset Balancing*, and *Dataset Sampling*. Every variation in these two distinct aspects is represented through an instance of these profiles. This

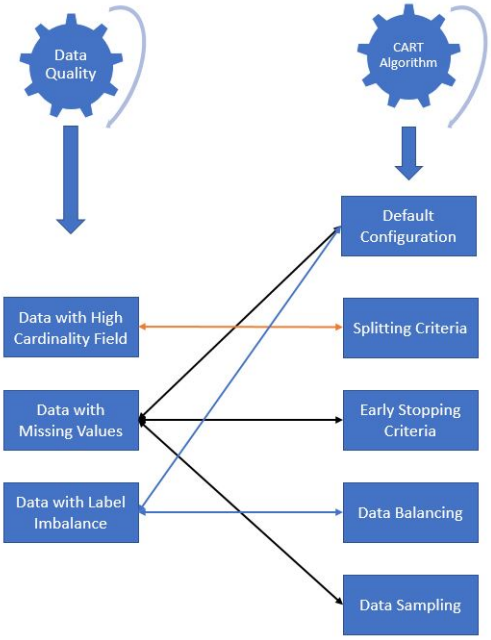

Figure 1.3: Interaction aspects considered for research goal

enables us to perform evaluations concerning the interaction of these two aspects by considering the interaction between the various instances of the DQPs and the CCPs.

The above is demonstrated by implementing a minimal solution by considering data with variations only in the *Proportion of Missing Values*, and in *Cardinality*. This is shown in Figure 1.3. These various aspects then interact with each other to produce performance metrics for the classification task, which are then analysed for evaluation. The arrows in the figure indicate the interactions which we consider for our evaluation. Thus, the research goals stated above would be justified if we are able to come to conclusions regarding the performance of the classifier based on the characteristics of the profiles involved. The specific questions which we answer for our implementation scope are shown in Table 1.1.

1.2.1 Document Structure

This thesis document is organized as follows. We first begin with descriptions and details of the concepts which form the basis for the rest of the document in Chapter 2. These include descriptions of data quality, the DQP, and the CTP, followed by information on machine learning algorithms and the steps required to run a typical machine learning pipeline. We then place emphasis on induction based learning using decision trees which

Data Quality Aspect	Learning Algorithm Aspects	Evaluation Questions
Proportion of Missing Values	Default Configuration	How do missing values in the training data affect the prediction performance and quality of tree generated?
	Splitting Criteria	How do the two splitting criterion behave with data of same quality?
	Early Stopping Criteria	How effective are tree growth early stopping criteria in mitigating the effect of missing values in the data?
	Data Sampling	How effective is training data sampling in mitigating the effect of missing values in the data, and how does it compare to the effect of early stopping criteria?
Data with High Cardinality Field	Splitting Criteria	How does the choice of splitting criteria help combat against the high cardinality bias?
Data with Label Imbalance	No Data Balancing	How does class imbalance affect the prediction performance?
	Data Balancing	How does data balancing affect prediction performance, and is it effective in improving prediction performance of low frequency samples?

Table 1.1: Evaluation questions for the minimal implementation

is the scope of our work. In Chapter 3 we look at existing work which look at the effects of data quality on learning algorithm performance, and optimization of decision tree algorithms. Chapter 4 deals with the approach taken by us in order to reach a solution. The implementation specifics for this approach are dealt with in Chapter 5. In Chapter **??**, we evaluate our research goals by analysing the results obtained for our sample analysis task, as defined in Table 1.1. We conclude our work in Chapter 7, and describe briefly the possibilities for further study.

2 Background

This chapter provides the foundations on which our work rests on. For this purpose, we begin this chapter with a discussion on Data Quality in Section 2.1. This consists of discussions on the various dimensions of data quality, and techniques to formalize this for a dataset through profiling. Techniques to better prepare the data for a learning algorithm, known as data preprocessing, is detailed net. We then move to the concepts of machine learning in Section 2.2. We discuss the basic concept of learning, followed by the procedure which needs to be followed to apply learning to a dataset. We conclude with a discussion on decision tree algorithms, which is the learning technique we use in our work for evaluation.

2.1 Data Quality

Information systems have always played a very crucial role in society and its operations. Early works describe the grave consequences of faultily functioning information systems in the area of law enforcement [Lau86], where the consequences of inaccurate data in criminal records can be false arrest and acquittals. Hence, there was a desire to ensure that these systems work as expected with minimal errors. For this to happen, it is essential that the data these systems process are free of errors, and are an accurate representation of the real world. This line of thought gave the motivation for further research in this area, which is termed as the study of data quality. This topic has had vast research attention since many decades. So much so, that a framework for categorizing and managing the vast research done in this area has been proposed [WSF95].

The findings of this work on the data quality research framework is that the emphasis is placed heavily on syntactic and semantic correctness of data. The former enforces the syntax and style in which the data must be present in order to be considered for processing, and the latter enforces restrictions on the values based on information obtained from the domain. The key takeaways here are that there is a need to look at data quality from a perspective that is beyond correctness, and that there is a need to

define formally methods to quantize data quality in the form of metrics. Both these issues are addressed in [WS96] through the following two measures:

1. Quality of data is given a broader perspective by looking at it from the perspective of suitability for the end consumer of the data
2. The many ways in which quality in the way above can be looked at is categorized under many dimensions, where each dimension then corresponds to a distinct aspect of the quality of the dataset

We adopt the consumer first perspective for looking at data quality in our work, and hence we define data quality as the "fitness for use" of the data by its consumer. We use this definition due to its versatility, as it has also been adopted for cases like ours, where the end consumer is a computer program [Kie]. One of the dimensions of data quality highlighted here is its *interpretability*, which is a measure of how close the data is to the ideal representation expected by the consumer. This consumer first viewpoint for data quality in case of learning algorithm as the end consumers has been studied, though indirectly, through the perspective of data preprocessing, by [KKP06], which we discuss later in Section 3.1. Using this view of data quality, we now explore its various dimensions in the next section.

2.1.1 Data Quality Dimensions

The quality of a dataset is a broad concept. In order to be able to refer to it using finely grained aspects, it is necessary to structure this concept into cleanly defined categories. In order to perform such a sub-division, [WS96] takes into account the end consumers of data itself, and performs a survey to learn which aspects of quality the consumers value the most. The responses of such an empirical approach clearly point to a conclusion that data quality is a multidimensional concept, which has to be looked at from perspectives of both the data custodians and the consumers of data[PLW02].

[Kie] adopts a similar approach, and discusses the quality dimensions of *accuracy*, *relevancy* and *interpretability* specifically for machine consumers, which is most appropriate for us, and hence we use them as the base for our discussion below. These dimensions are defined based on their closeness to an ideal dataset, which best represents the particular dimension. These are shown in Table 2.1. Every dimension in Table 2.1 can be defined as the distance of the dataset from the ideal dataset corresponding to that dimension. We also use some concepts from [WS96].

Dimension	Ideal Dataset
Accuracy	Represents the real world perfectly
Interpretability	Is perfectly suited for the consumer
Relevancy	Is optimal for the consumer's task

Table 2.1: Data Quality dimensions and their ideal datasets for comparison [Kie]

Accuracy of a dataset is a measure of how closely it represents the facts of the real world. In the context of building a learning model using this data, it is extremely essential that the data represents the behaviour of the real world in the best possible way, as the patterns of the real world as represented in the data, in terms of the relationships between the fields, are what is learnt by the learning algorithm. For example, if a real estate price prediction model is trained on inaccurate data, the prices predicted by the model will mostly be inaccurate. That is, the model learns the inaccuracies of the data, and hence in the context of learning algorithms, *Accuracy* is of prime importance. In addition, due to the heterogeneous nature of data collection, we have access to a large number of data sources. Hence, in addition to looking at the correctness of the data, it is also necessary to consider the trust factor associated with a particular data source in terms of its reputation and trustworthiness [WS96].

The idea of interpreting data quality as its applicability for the consumer is captured in the *Interpretability* dimension. It measures how close the data is to the one expected by the consumer. This requires the semantics of the data to be easy to understand, and have consistent representation. In case of learning algorithms, this could be a statistical requirement. For example, the Multinomial Naive Bayes algorithm expects the data to have a multinomial distribution [KZP06]. The interpretability of the data with respect to the expectations of the classifier, as can be represented through the training data must be high. For example, a stock risk category classifier trained on data from a few decades ago will not perform well with current data.

The *relevancy* of data indicates how appropriate the data is for the consumer's task. Highly accurate data, if unsuitable is not of much use. Extending the example of the stock risk classifier from above, despite access to current training data, we might still have to deal with poor performance if the data does not contain sufficient fields which can indicate the risk associated with the stock. In this case, even though the data is *accurate*, and highly *interpretable*, it is low on quality due to lack of *relevance*.

Given the multidimensional nature of data quality, we need a method to be able to depict this vast information in a clear and descriptive fashion. This is done by quantizing aspects of data quality. This is detailed in the next section.

2.1.2 Measuring Data Quality

To be able to derive value from information about the quality of a dataset, it is necessary to measure and quantize quality information of a dataset [HKK07]. This provides a tool for measuring improvement or deterioration of the quality of a dataset. For example, the quality of a dataset can be measured before and after a data cleansing operation to assess its effectiveness. Such quantized depictions of data quality are termed as data quality metrics. An important property of such metrics is that their value must be *normalized*. This has two implications [HKK07]:

1. Constraint 1: Their value must lie in the range 0 to 1
2. Constraint 2: The interpretation of 0 and 1 must be consistent. That is, 0 must mean either perfectly good or perfectly bad for all metrics

This immediately leads to the need to define metrics for the quality dimensions defined above in Section 2.1.1. Here, a distinction must be drawn between quality indicators for unstructured data such as free text fields, and structured data, which fit into the conventional relational model, with categorical or continuous values for the fields.

Quality indicators for free text fields have been discussed through a pure quality perspective as described above, through the perspective of the end consumer [Kie], and alternately, through a vectorized perspective, where the quality of the free text field is defined through characteristics of the tokens which comprise the text field. Consideration of individual tokens arises due to the need to represent the free text field in a vector format for compatibility with the learning algorithm (see Section 2.3.2.7). Since our use case deals with the latter, we consider the quality indicators set forth considering the characteristics of the tokens. These are described under.

Entropy is traditional used a measure of the distribution of data in a collection of instances, and is used as a measure of *purity* of the collection [Mit97]. [BRS12] extends this concept to measure the average amount of information contributed by the document corpus by associating with each token its entropy. A high entropy indicates that the distribution of tokens in the corpus is spread out across many tokens, and hence implies

Property	Significance
Max	Maximum value taken by the field
Min	Minimum value taken by the field
Distinct	Number of unique values taken by the field
Empty	Number of values which are empty
VarType	The type of data held by the field, whether numeric or string

Table 2.2: Quality indicators for structured data [Pyl99]

many tokens with small frequencies. A lower value of entropy implies fewer words with larger frequencies. It is calculated using Equation 2.1 [BRS12], where V is the vocabulary, $p(t_i)$ is the proportion of occurrence of token t_i. The interesting aspect to notice here is that the logarithm is taken to the base of the vocabulary length. This is necessary to meet the *normalized* [HKK07] requirement.

$$H = -\sum_{t_i \in V} p(t_i) log_{|V|} p(t_i) \tag{2.1}$$

[BRS12] also provides indicators which consist of taking ratios with respect with token frequencies and vocabulary size, etc. *Relative vocabulary size* is the ratio of the size of the vocabulary to the number of *meaningful* tokens. A token is said to be meaningful if it is not a function word. A smaller value is desired as it indicates simpler language with fewer tokenization errors. *Vocabulary dispersion* is the ratio of the number of tokens in the vocabulary which occur *infrequently* to the total size of the vocabulary. It is computed using Equation 2.2, where V_{low} is the set of tokens in the vocabulary V, which occur in less than 10 documents. This is a useful metric as higher values indicate spelling and tokenization errors, an give justification to applying vocabulary pruning methods, such as setting a threshold restricting tokens based on the number of documents they occur in [Seb02].

$$D_{Voc} = \frac{|V_{low}|}{|V|} \tag{2.2}$$

For structured data, we choose to focus on the concept of *data assay* introduced by [Pyl99], which is a collection of quality specific properties of the dataset, defined for every field of the dataset. These are described in Table 2.2. The interesting aspect to notice here is that these indicators do not satisfy the *normalized* restriction placed on quality indicators, according to [HKK07]. However, we state them here due to the multiple aspects focussed on in the assay.

2.1.3 Dataset Profiling

The data collection phase of the analytics pipeline (Section 1.1) relies on selecting appropriate datasets which provide the required information, and are most applicable to the nature of the task at hand [SO13]. This process, of discovering the right data source serves as the motivation for profiling data. This is because, in order to select the right dataset, we need a method by which we can compare data sources on certain criteria. The idea is to represent information about the dataset in a clear and understandable manner as part of a single artifact [Ell+]. Such an artifact which contains “descriptive metadata" about the dataset is termed as a “Dataset Profile", and the process of generating such a profile is termed as data profiling. The purpose of a dataset profile is to aid in the data discovery process. Hence, the information it provides for the dataset must be descriptive enough such that the nature of the dataset is captured, and the information it captures must representative of it, in the sense that it must allow for comparison against other datasets.

2.2 Machine Learning

Machine Learning is a collection of algorithmic techniques which aim at making computers intelligent by allowing them to learn from data, and from past experiences [Alp10]. This is achieved by feeding these algorithms with learning experience *E*, with respect to a class of tasks *T*, with the aim of optimizing a performance measure *P*. The knowledge to be learned using *E* is termed as the target function *V* [Mit97]. We now consider a sample example for the task of learning chess. Consider the problem of teaching a computer the task *T* of playing a game of chess. Its performance *P* is measured by the number of games won against a human opponent. In order to provide it with experience *E*, it could be provided with data of other games of chess, with every move tagged with a label indicating whether it led to a win or a loss. In this case, *V* becomes the problem of choosing the next best move.

The experience *E* provided to the learning algorithms consists of a collection of *instances*, where each *instance* is associated with a set of *features* which describe the properties of this *instance*. Every *instance* in such datasets consists of the same set of *features*. In the above example, each instance could be a move. The nature of the target function *V* depends on the style of the learning undertaken by the algorithm. If the training dataset maps every instance to a target label, then the problem is that of *supervised learning* in which *V* is the problem of mapping each instance in the unseen data to a target label. On the other hand, if no such target label exists, the problem

Type	Meaning	Example
Continuous	These can take a continuous range of values	Speed
Categorical	These can take one among a discrete set of values	City
Unstructured	These cannot be organized into a predefined model	Free text

Table 2.3: Types of Features [KZP06]

is that of *unsupervised learning* wherein the goal is to find patterns and irregularities in the dataset [KZP06]. The features which describe the instances can be continuous, categorical or unstructured [KZP06]. See Table 2.3.

2.3 Machine Learning Pipeline

We now make the analytics pipeline described in Section 1.1 concrete by describing it for the machine learning case. The process followed to apply machine learning is shown in Figure 2.1. Note that we have categorized the figure according the steps of the analytics pipeline, which was originally shown in Figure 1.1. Figure 2.1 is constructed through the concepts described in the below questions. Specifically, the sections of *Data Collection* and *Data Preparation* are derived from [Pyl99], and the *Data Analysis* section from [KZP06], and [SO13].

2.3.1 Data Collection

This step involves collecting *appropriate* data from various sources, and preparing it for analysis. The task of *Data Discovery* follows from the description given in Section 2.1.3. Suitable data sources are identified, and data is obtained from them using the interface exposed. Because of the heterogeneous nature of the sources, the data collected can be in different formats, and in different representations. For example, an e-commerce retailer could use geo-location data collected through user cell phones to find user travel patterns, and Twitter stream data to analyse user sentiment. Such varied data then has to be assembled into a common representation [SO13], and stored in a suitable *Raw Data* repository for further processing. Consider the sample example illustrated in Figure 2.2, which has been derived from experiences of everyday activities. The Twitter stream is made available in a JSON format. This has to be parsed appropriately, and the interesting fields have to be extracted. This can be combined with the latitude and longitude from the cellphone geographical location data to obtain a unified relational representation. Such raw data is then passed through stages of preparation for analysis.

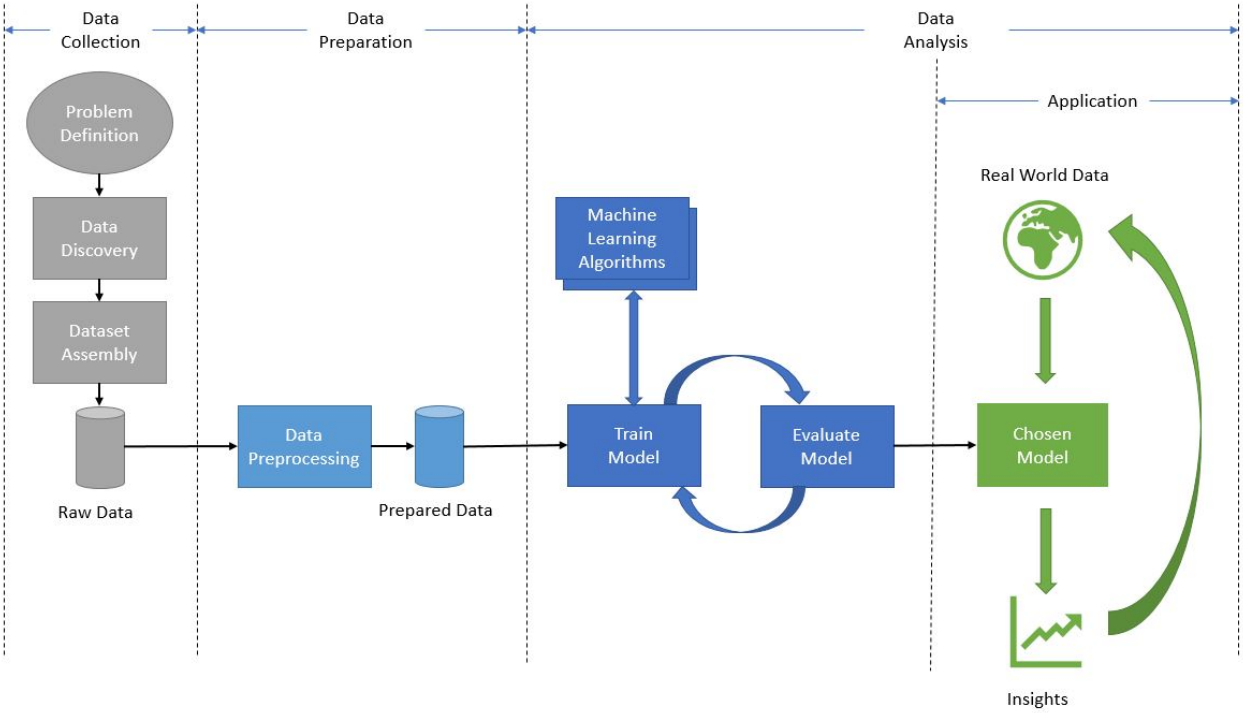

Figure 2.1: A Machine Learning Pipeline [Pyl99] [KZP06] [SO13]

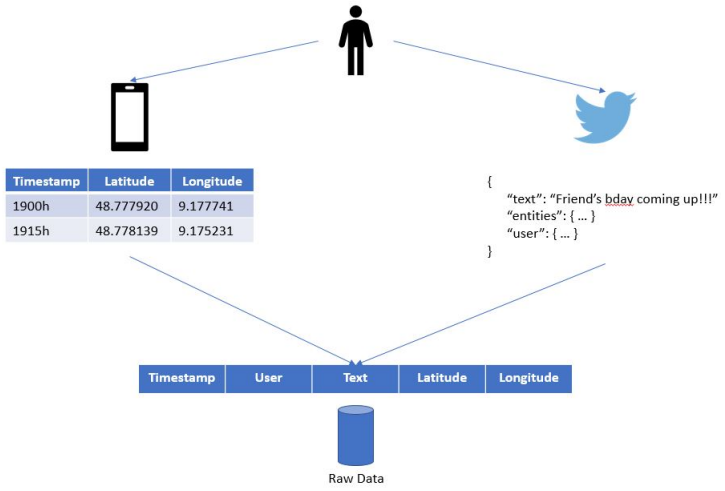

Figure 2.2: Example for Dataset Assembly

Objective	Achieved Using
Tackle quality issues	Instance Selection, Missing Value Imputation, Binning, Feature Construction, Balancing
Compliance with analysis environment	Data Projection, Text Vectorization

Table 2.4: Objectives of Data Preparation

2.3.2 Data Preparation

Data in its original form is not immediately suitable for analysis. It is passed through stages of processing, which transform the data to a suitable representation. The processing stages applied depend on the nature of the learning task, and the quality and semantics of the data. This transformation which is performed is termed as *Data Preparation* [Pyl99], and also referred to as *Data Preprocessing* in machine learning oriented literature [KKP06]. The quality of data supplied to the learning algorithm plays a prominent role in its performance. We detail literature related to this claim in Chapter 3, but here we summarize with the colloquial phrase, "garbage in, garbage out" [Pyl99], which best signifies this issue. We now detail the various preprocessing stages used in the context of application to learning algorithms. Note that the need for preprocessing arises due to two main criteria: 1) to improve the quality of the data which is supplied to the algorithm, and 2) to ensure the data is compatible with the environment in which the implementation of the learning algorithm is going to executed. The preprocessing steps which are associated with each of these motivations are given in Table 2.4. Sections 2.3.2.1 to 2.3.2.5 have been derived in their entirety from [KKP06], Section 2.3.2.6 from [CLS06], and Section 2.3.2.7 from [Seb02], unless indicated otherwise.

2.3.2.1 Instance Selection

One approach to improve the quality of the dataset is to target and remove instances in the dataset which contribute to the bad quality. This approach of selectively retaining only the acceptable instances is known as instance selection. This can be done by filtering out instances based on criteria set for feature values. If any of the feature values of an instance are qualified as invalid, then that instance is ignored for further processing. Table 2.5 shows possible rules for identification of invalid values according to feature type. This approach, however, adds the disadvantage of data loss. This could be disadvantageous because of the fact that the valid features in the instances eliminated could have useful information associated with them. Hence, an alternative approach is to replace, or *impute* the invalid values with a valid value. However, the trade off

Feature Type	Invalid Value Rule	Example
Categorical	Value outside valid set	Value of 'Y' for *Gender* with valid set {M, F}
Continuous	Value outside permissible range	Value of -1 for *Hour*, with permissible range 0 to 23
Free Text	Noisy tokens (abbreviations, unknown words, spelling mistakes, special symbols, etc) [Kie]	FPGA, autamble, !$#%%

Table 2.5: Rules for invalid feature values [KKP06]

here is between the value gained by not eliminating the instances, and the value lost by imputing values which are not obtained from the real world.

2.3.2.2 Missing Value Imputation

A missing value for a feature indicates an empty value, or a null value. There are many strategies available to deal with missing values. One is to deal with instances with missing values as invalid, and filter them out as described above. However, this approach suffers with the problem of data loss as in the above. Alternatives to this include to impute the missing value with an alternate value. One possibility here is to impute the missing values with the mean of all the values of the feature. However, this suffers with the problem of biasing the distribution of the feature values towards the mean. Alternative approaches are imputation with the median, or the mode of the feature values. These have been shown to be the simplest approaches with the least side effects [Zha16].

2.3.2.3 Binning

It can be advantageous to reduce the cardinality of features. Features with high cardinality have been known to be overestimated by inductive learning algorithms, adding a false bias. But this added bias affects the performance of the algorithm poorly due to the high variance associated with such features. Such high cardinality normally occurs with continuous features. One method to discretize such features is to bin them into a fixed number of categories, through a process called *binning*. It is a static partitioning method which does not consider the relationship between features, thereby performing individual partitioning of a feature into a fixed number of buckets. For binning with n buckets, the continuous range of values of the feature to be binned are

divided into n equal interval buckets, and the values within each bucket are thereafter referred to using only an identifier for the bucket. This reduces the cardinality of the feature to n.

2.3.2.4 Feature Construction

Sometimes, it can also be necessary to construct new features using existing features. This can be necessary if the information conveyed by a feature is of coarse granularity, and can be divided to obtain more concentrated information, giving rise to more accurate learning models. An example is shown in Figure 2.3. The coarse grained feature *Timestamp* is divided by constructing individual features for date, month and the year.

Figure 2.3: Feature Construction Example

2.3.2.5 Balancing

This preprocessing step becomes necessary for datasets supplied to supervised learning problems, where every instance of the dataset has an associated target label. A dataset is said to be imbalanced when the number of instances associated with each target label varies by large margins. This can lead to learning models which are biased towards the more frequently occurring target labels. This preprocessing step combats this by adjusting the number of instances associated per target label. This ensures the learning model is uniformly exposed to all target labels. To perform this, the following two approaches are suggested [JS02]:

1. Instances of the less frequently occurring labels can be oversampled by duplicating instances in the training dataset
2. Instances with higher frequencies are undersampled by eliminating instances of the higher frequency samples randomly thereby reducing the sample size

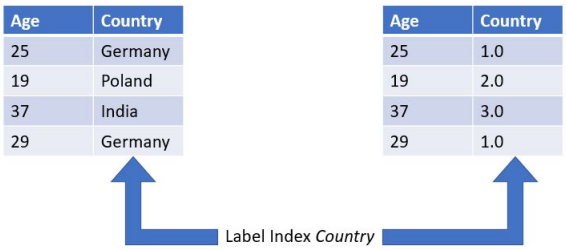

Figure 2.4: Example for Label Indexing

2.3.2.6 Data Projection

Data projection techniques are undertaken to change the representation to either better suit the learning algorithm, or ensure compatibility with the learning algorithm, and its technological implementation. We discuss two types.

Value Transformation becomes necessary when there is a need to map the values of features from their original representation to an alternate compatible representation. A technology specific variant of this is known as *Label Indexing* [Ped+11], which is necessary when textual content in feature values is not accepted by the analysis environment. It is applicable to categorical features with string content. The approach followed is to map every unique value of a categorical feature to a numeric index. An example is shown in Figure 2.4, where label indexing is applied to *Country*.

The example shown in Figure 2.4 indicates the inherent disadvantage associated with the label indexing approach. It gives a false sense of continuousness to discrete features. In the example, a induction learning algorithm can generate a rule *if Country >= 1.0*. In such cases, numerical mapping for the categorical attributes becomes necessary, in order to preserve the integrity of categorical attributes. A technology specific terminology for this approach is known as One-Hot Encoding [Ped+11]. Here, a n cardinality categorical feature is transformed into n binary features, one for each of the categories, with a *1* indicating the presence of that category for that particular instance. For the same example as above, a One-Hot Encoded feature would look like as shown in Figure 2.5, where *Country* is transformed into 3 further features. This has the disadvantage of increasing the dimensionality of the problem, and hence technological implementations which enforce the application of such a transformation, must also provide methods for effective memory management for a large amount of features. One approach for this is the use of a sparse matrix [Ped+11], which stores only the indices of the *1*s in memory, and not the entire data.

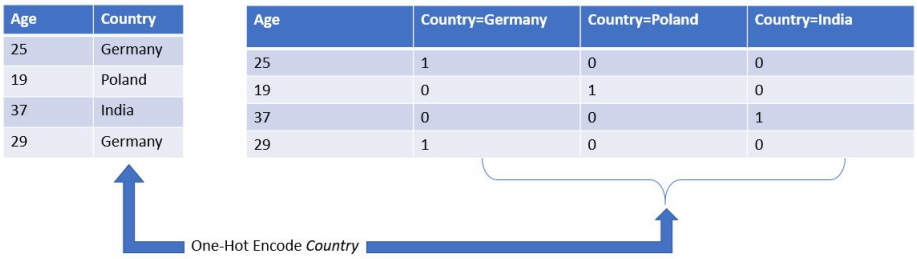

Figure 2.5: Example for One-Hot Encoding

2.3.2.7 Text Vectorization

Features which contain free text fields as their values cannot be directly supplied to a learning algorithm. Thus, we need an indexing mechanism for the free text fields which transforms them into a suitable format. This is done by constructing a vector representation consisting of *weights* assigned to every unique *term* which occurs in the document corpus [Seb02]. Such a vector can be represented as $\vec{d} =< w_1, ..., w_{|T|} >$, where w_i is the weight assigned term i in a document corpus with set of terms T. In this notation, a *term* corresponds to a word, and the weight can be assigned using the following three methods:

1. A binary weight indicating the presence of the term in the document for an instance
2. A weight indicating the number of times this term occurs in this document, known as the *term frequency* (*tf*)
3. *term frequency - inverse document frequency* (*tf-idf*) is an enhanced version of the above, which additionally offsets each *term frequency* weight by the number of documents the term occurs in, the *document frequency*. Equation 2.4 shows its computation, with *tf(t, d)* being the term frequency for term *t* in a document *d*, and *idf(t, D)* is the inverse document frequency for the same term in the document corpus *D*. The *inverse document frequency* is computed using Equation 2.3, where *df(t, D)* is the document frequency. This metric is used to quantize the following two intuitions: 1) The terms which occur across many documents are less representative of any correlation between the target label and the document, and 2) The terms which occur many times in a document are more representative of it

$$idf(t, D) = log \frac{|D|}{df(t, D)} \tag{2.3}$$

$$tfidf(t, d, D) = tf(t, d).idf(t, D) \tag{2.4}$$

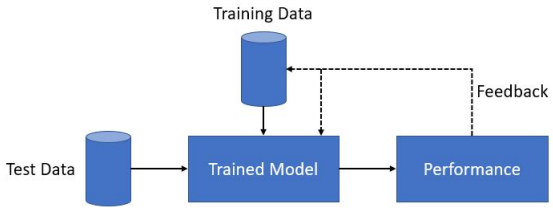

Figure 2.6: Model Evaluation using a test dataset [KKP06]

2.3.3 Data Analysis

In this stage of the pipeline, we apply a learning algorithm on the data prepared in the previous stage. This begins with selecting an appropriate learning algorithm for the given task, and thereafter evaluating the effectiveness of this model on the data available for the particular learning. Section 2.3.3.1 discusses possible evaluation methods, followed by a discussion on the metrics used to decide the effectiveness of the model, and choose the best one.

2.3.3.1 Model Evaluation

[SO13] describes the need for model evaluation as the following. Once a model has been trained using the available data, and before it is used over unseen data in the real world, it first needs to be evaluated on its effectiveness. This is done by maintaining separate evaluation datasets. The performance of the model is measured by applying these evaluation datasets on the trained model. The performance metrics obtained then help us decide if changes are required to obtain better results, if necessary. Such changes, known as *tuning*, can be performed across two levels [KKP06]:

1. On the training dataset: Either by changing the preprocessing steps applied or by obtaining data from a different source
2. On the learning algorithm: By tuning the configuration parameters (hyperparameters) of the learning algorithm

We now discuss the possible methods to obtain such evaluation datasets according to [KKP06]:

1. One approach is to maintain a separate test dataset obtained from the original dataset. This necessitates the partitioning of the initial dataset into two parts, one of which will be used to train the model, and other to test it. During the training phase, the test dataset is never brought in contact with the model, thus

ensuring that the performance of the model on the test data accurately represents its performance when applied on unseen data in the real world. Using this description, a depiction of this process is shown in Figure 2.6. The performance of the trained model is measured by applying the test data on the trained model. The partitioning of the dataset is done by first sampling a certain fraction of the data for the training dataset, and the remaining data then becomes the test dataset. Such a sampling can be done in two ways:

a) Random Sampling: The required fraction of the instances from the dataset are randomly sampled

b) Stratified Sampling: Sampling is done respecting the distribution of the instances across the target labels. That is, the required fraction is sampled per target label, thereby maintaining the nature of the distribution of instances in both the training and the test datasets. Maintaining similarity in the nature of training and test datasets is essential for good learning performance [Qui86]

A disadvantage with this approach is that the tuning performed is only aimed at improving the performance of the model against the generated test dataset. This can lead to the model to be optimized only with respect to this test dataset, leading to the model being *overfit* to the test dataset

2. An alternative to combat the above issue is *k-fold cross validation*. In this approach, the dataset is divided into *k* equal partitions. In every iteration, the model is constructed using *k -1* data partitions, and the left out partition acts as the validation dataset against which the performance is measured. This procedure is repeated *k* times, each time leaving out a different partition for validation. The final performance is the average of the metrics obtained in each iteration. After which, the tuning methods described above can be applied. The advantage with this approach is that the model considers a larger proportion of the dataset during construction, and the performance metrics depict the behaviour across the entire dataset

2.3.3.2 Performance Metrics

As seen above in Section 2.3.3.1, evaluation of the trained model depends on how good it performs on the evaluation dataset. This is done by measuring the performance of learning algorithms, and compare performances against different configurations of the learning algorithm, and different modifications to the training dataset. We restrict our scope to only metrics for the supervised family of learning algorithms in which every instance is tagged with a target label.

Metric	Formula
Accuracy	$\frac{tp+tn}{tp+fp+tn+fn}$
Precision	$\frac{tp}{tp+fp}$
Recall	$\frac{tp}{tp+fn}$

Table 2.6: Performance Metrics [SL09]

Accuracy signifies the fraction of instances which have been labelled correctly. However accuracy does not suffice in cases of label imbalance in the dataset. That is, when the target labels are unevenly distributed in the dataset [KKP06]. For example, consider a two class classification problem, where 90% of the instances are labelled with *ClassA*, and the other 10% with *ClassB*. A dummy classifier which only predicts *ClassA* for every instance, will still be able to achieve an accuracy of 0.9, which gives a wrong impression of the performance of the classifier. Hence, metrics are necessary which can take in label imbalance into account. These are *Precision* and *Recall* [SL09].

As a prerequisite to define the above metrics, we first define the following terms [SL09]. With respect to a target label *t* in the results of a classification problem, we have the following. *True Positives* (*tp*) are the number of instances correctly classified with *t*. *True Negatives* (*tn*) are the number of instances correctly classified as not belonging to *t*. *False Positives* (*fp*) are the number of instances wrongly classified with *t*. *False Negatives* (*fn*) are the number of instances which actually belong to *t*, but are wrongly classified with a label other than *t*.

With the above basic terminology, the performance metrics are shown in Table 2.6, and discussed here. For a classification problem with a target label *t*, *Precision* is the fraction of correct classifications out of all the instances labelled with *t*. *Recall* is the fraction of instances classified correctly with *t*, out of all the instances labelled with *t* [SL09]. Continuing the above example, we attempt to find the precision and recall values. Consider Figure 2.7, which visualizes the predictions made by the dummy classifier for our example, where the definitions of the terms introduced above hold. All 90 instances labelled with ClassA, are obviously correctly classified. However, all instances labelled with ClassB are classified incorrectly as ClassA. In order to better understand this, we compute the precision and recall metrics in Figure 2.8 using the formulae provided in Table 2.6. Note that the computations on the top are with respect to *ClassA*, and the one on the bottom with respect to *ClassB*. We then average the results to obtain the final value. Even though we obtain a high accuracy, the low precision and recall values hint in the direction of label imbalance and are hence necessary metrics.

n=100	Predicted: ClassA	Predicted: ClassB
Actual: ClassA	90	0
Actual: ClassB	10	0

Figure 2.7: Example predictions of a dummy classifier

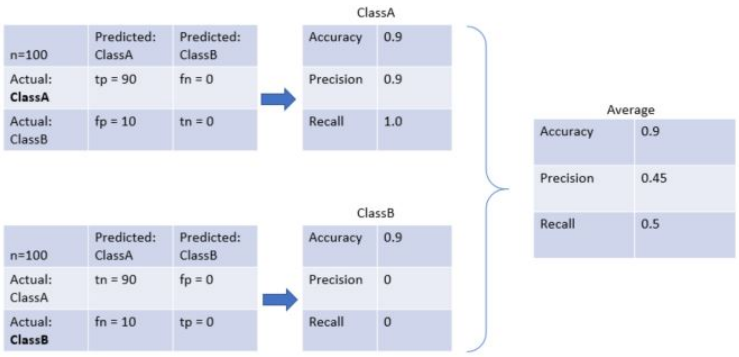

Figure 2.8: Example calculations of performance metrics

Note that the *Precision* and *Recall* metrics defined above are applicable for the target class for which they are computed. In case of multi-class classification problems, we need a method to be able to obtain global *Precision* and *Recall* values which take into account all labels which are part of the classification task [SL09]. Two averaging schemes have been suggested [SL01], *micro* and *macro* averaging. *Micro* averaging considers the global *tp*, *fp*, and *fn* values to compute the precision and recall. On the other hand, the *macro* average, simply takes the average of the precision and recall values of each individual token. The implication is that *micro* average gives equal importance to each instance in the dataset, whereas the *macro* average gives equal importance to each label in the dataset.

2.4 Classification based Learning

As described in Section 1.2, in our work, we focus exclusively on the classification problem, by using decision trees, which are an inductive learning technique [KZP06]. Wew first briefly describe the concept of a classification problem, and the relevant ingredients, followed by a more detailed look at decision trees.

Induction in the case of supervised learning to be the construction of a target function which maps each observation in the dataset to a target label. The hypothesis is that a target function which approximates well over a significantly large section of the training

data (Section 2.3.3) will also approximate well over unseen data. This has been referred to as the *trained model* in Section 2.3.3. It is said to have learnt a set of rules which are used to classify future data instances to one of the target labels. An individual case of applying classification in order to solve a problem is referred to as a *classification task* [Mit97].

The training dataset can be viewed as a pair consisting of the data instance and the associated target class label. Equation 2.5 [Alp10] represents the training dataset X consisting of N instances, where x^t is the t^{th} instance in the training dataset, and r^t is the target label for this instance.

$$X = \{x^t, r^t\}_{t=1}^{N} \tag{2.5}$$

Suppose there are k target classes, each denoted as $C_i, i = 1, ..., K$, then the target label r has K dimensions, and represented by Equation 2.6 [Alp10].

$$r_i^t = \begin{cases} 1 & \text{if } x^t \in C_i \\ 0 & \text{if } x^t \in C_j, j \neq i \end{cases} \tag{2.6}$$

The training dataset as described above, is said to abide by the target concept, which is the real world mapping between the feature vectors and the labels, and the target function maps the instances to classes which hold true for a "sufficiently large set of training examples". Such a real world outcome of the classification task is said to be the hypothesis, and a collection of possible hypotheses is known as the hypotheses space [Mit97]. In more formal terms, learning the target function involves finding the boundary separating the instances of the various target classes. Thus a K class classification problem can be viewed as K 2 class classification problems, where each class C_i represents the set of instances which hold true for the hypothesis h_i that the data instance belongs to class C_i, and does not belong to classes $C_j, j \neq i$ [Alp10]. Thus for a K class classification problem, we have K possible hypotheses, as shown by Equation 2.7 [Alp10]:

$$h_i(x^t) = \begin{cases} 1 & \text{if } x^t \in C_i \\ 0 & \text{if } x^t \in C_j, j \neq i \end{cases} \tag{2.7}$$

2.5 Decision Trees

Decision trees are non parametric learning methods. That is, they make no assumptions on the distribution of data. A prominent work formalizing the approach to

constructing decision trees was provided by [Qui86]. The principle followed is that of top down induction. The induction task here is the generation of the classification rule which accurately predicts the target label given the features of an instance. These rules are depicted as decision trees. The leaf nodes of the tree represent the target labels, and each intermediate node represents a decision taken based on a condition on the value of one of the features of the input dataset. It has to be ensured that the tree capture the essential characteristics of the data, but does not end up just learning the training data.

Many algorithms have been devised for construction of decision tree algorithms. Irrespective of the algorithm, most of them follow a top-down approach to induction [Mur98]. Such an approach consists of the following high level steps [Loh11]:

1. All instances of the training set are represented at the root node
2. The split which results in the most optimal value of the splitting criterion (Section 2.5.1) is computed
3. The split is performed and the instances are divided into the child nodes
4. Steps 2 and 3 are repeated for every child node until either:
    a) The node becomes *pure*, that is, all instances belong to the same class. In this case, the node is made into a leaf node, and is marked with the associated class
    b) Or, an early stopping criterion is reached. Then terminate tree growth at that node, and mark the node with the majority class (Section 2.5.2)

The goal is to split the instances present at each node into multiple groups of instances forming child nodes. The split which is chosen is the one which maximizes the homogeneity of the child nodes. The decision of which instances are part of which child nodes is taken based on the result of a test condition, which is evaluated against the values of the features which are represented at each node. The goal is to reach the situation where all instances belonging to a node are labelled with the same target class, and each target class is represented in the final tree. Such a node is the leaf node. This tree is then used to predict labels for instances of the test dataset by allowing the test dataset to flow through the constructed tree. Many algorithms have been proposed for construction of decision trees. They vary in the structure of the tree generated, and in the splitting criterion used. The ID3 algorithm [Qui86] generates multiway splits at each node, and uses the Information Gain splitting criterion. The CART (Classification And Regression

Tree) algorithm [Bre+84] generates binary splits, and originally uses the Gini Impurity splitting criterion.

2.5.1 Splitting Criteria

The key decision to be taken at each node is the feature on which to split the instances at that node. This is known as *split criterion*, and the aim of these split criterion is to choose the split which minimizes the *impurity* in the child nodes. That is, the split which results in more homogeneous child nodes is chosen. The two widely used split criterion are *Information Gain*, and *Gini Impurity*.

The information gain of a split is measured as the reduction in *entropy* achieved by performing this split. *Entropy* is a measure of impurity of a collection of instances. That is, it measures how homogeneously distributed the instances in this particular collection are, with respect to the target labels. A collection of instances *S*, is said to be pure, if they are all labelled with the same target class. In such a scenario, the entropy is at the lowest value of 0. Entropy is at its highest value of 1 when there are equal number of instances for each target class in the *S*. The entropy is given by Equation 2.8 [Mit97], where *S* is the collection of samples, *c* is the number of target classes, p_j is the proportion of instances in *S* belonging to target class *j*. The information gain then associated with a split is computed using Equation 2.9, where $IG(S, A)$ is the information gain obtained by splitting the collection of instances S on attribute A, $Values(A)$ is the set of all possible values which *A* can take, and S_v is the instances in *S*, for which the attribute *A* has value *v*. In effect, Equation 2.9, and hence Information Gain computes the reduction in entropy obtained by performing the split on attribute *A*.

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{2.8}$$

$$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.9}$$

The gini impurity is a criteria similar to entropy, and measures the reduction in impurity achieved by a particular split. It is computed using equation 2.10, with symbol conventions same as above. Then, the gain in purity obtained by splitting on an attribute *A* is known as *Gini_Split* and is computed using equation 2.11.

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2 \tag{2.10}$$

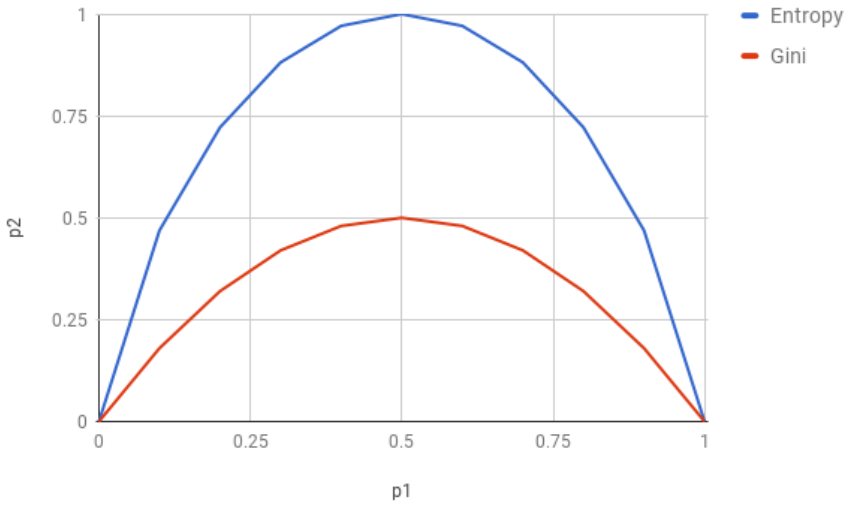

Figure 2.9: Comparison of Gini Index vs Entropy for a binary classification problem [Tan+06]

$$Gini\_Split(S, A) = \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Gini(S_v) \tag{2.11}$$

The crucial decision taken at every internal node during construction of a decision tree is the attribute and its value on which to split the samples at that node. The two most popularly used criterion, Gini Index and Information Gain have been discussed in Chapter 2. [RS04] performed a theoretical comparison, and also gives details of many empirical works performed in order to know the effects of using one among the two splitting criteria. It was shown that in less than 2% of the cases, the two splitting criteria disagreed with each other with respect to the attributes chosen for the splits. This finding has been backed by [Tan+06], by stating that the two splitting criteria are in fact consistent with each other. This has been illustrated through the example of a binary classification problem, where *p1* and *p2* are the proportion of samples which belong to the two class labels. Both the impurity measures are maximum in the case where the instances in the collection are uniformly distributed across the two class labels. This value reduces with increasing balance towards one target label. This behaviour can be seen in Figure 2.9.

2.5.2 Tree Construction Early Stopping Criteria

An issue encountered with tree construction is that of *overfitting* to the training dataset. This occurs when the training dataset does not accurately represent real world patterns, due to data quality issues [Mur98]. The consequence of *overfitting* is that the tree ends up learning the training dataset, but is unable to generalize well over the test data. The method to avoid this is restricting the depth to which the tree can grow to [Mur98]. Two approaches are possible:

1. Restrictions on tree growth applied using early stopping criteria [Mur98]. These can be:
    a) Maximum depth to which the tree can grow to
    b) Minimum number of nodes that must be present at a node for it to be considered for a split. If less than this number are present, it is marked as a leaf node, and the majority target class of the instances at that node is marked as the predicted class for that leaf node
2. Reducing the amount of data consumed by sampling beforehand [Seb+00]

An alternate approach to limiting the size of the tree is to reduce the amount of data used to train the model, through sampling [Seb+00]. It is shown that the relationship between the training data size and the depth of the tree can be proven empirically. And also, it is shown that sampling is a more effective way of limiting the effect of data quality issues.

2.5.3 Configurable aspects of a Decision Tree

Deriving from the discussion above, we now briefly focus on the properties of the tree which can be configured to alter the way in which the tree is constructed. Deriving from our discussion on the splitting criteria in Section 2.5.1, one aspect of modification can be the choice between multiple splitting criteria, even though the original algorithm was defined as using only one criterion. For example, a CART Decision Tree implementation [Ped+11] provides the choice between Gini Impurity and Information Gain, even though the original version of the CART algorithm [Bre+84] was defined only with the Gini Impurity. The early stopping criteria discussed above explicitly influence the structure of the tree [Mur98], and hence are candidates for configuration. And lastly, due to the prominence of sampling as discussed above, the data transformations applied before being used to train the model can also be a part of the configurations applied to the tree construction [Seb+00] [KKP06].

3 Related Work

The purpose of this chapter is to investigate existing literature which deal with the interaction between data quality, and the performance of the learning algorithm to which it is supplied. Individually exploring each of the two broad themes is beyond the scope of this work, and hence we stick to only those works which consider their interaction. Such literature can be broadly classified into:

1. those which deal with data preprocessing as a way of improving quality, with the eventual aim of improving the performance of the learning algorithm
2. those which deal with empirical evaluations on the effects of data quality on the performance of the learning algorithm

In the following sections, we discuss each of these individually.

3.1 Data Preparation

In Section 2.3.2, data preparation steps for the purpose of supplying to a learning algorithm were detailed. Here, we revisit the preprocessing operations, focussing on the specific quality issues which are tackled, and their expected influence on the learning algorithm. As seen in Section 2.3.2, works which deal with preprocessing either focus on quality improvement [KKP06], or compatibility [CLS06]. We now look at the specific issue in quality handled by the preprocessing steps. Note that phrases in bold indicate the quality aspect which the corresponding preprocessing operation, in italics, is intended to mitigate.

Instance Selection is used to filter out instances which contain **invalid values** in some of its features. The consequences of this are [KKP06]:

1. Elimination of irrelevant and noisy data
2. Reduction in the scale of data

Since the learning model is trained on the data supplied to it, elimination of noisy instances leads to the construction of better trained models [KKP06]. Specifically for decision trees, it has been experimentally proven that a reduction in noisy instances in the dataset lead to an improvement in the prediction performance [Qui86]. Elimination of such irrelevant instances also leads to a reduction in the scale of data. This is advantageous, as a smaller sample of the data allows the model to generalize better, and provide better results. In fact, the rate of improvement in prediction performance reaches a plateau with increase in data size [KKP06], thus showing that smaller datasets can provide a performance similar to that of larger datasets. An additional advantage is the increase in computational efficiency, that is, the time taken to train the model [CLS06]. Thus, due to both the above consequences of instance selection, an improvement in algorithm performance can occur.

A **missing value** indicates an absence of a feature value for an instance. This implies a lack of information for the learning algorithm. Decision tree algorithms have the capability to deal with missing values, by not considering such instances at the computation of each node. [KKP06] discusses various approaches to deal with the problem of missing values. However, no specific study is done on the specific impact of missing values on learning model construction. Another point of investigation is a comparison between instance elimination method and imputation methods.

Binning is the process of transforming continuous features into discrete features by reducing their **cardinality**. The approach of binning reduces the effect of noisy data, and that of outliers [CLS06]. This is a direct consequence of constructing representative bins to represent the continuous values. The number of bins, and the strategy used to find the split points to assign data to the bins play a direct role in how much advantage binning can provide. This is because, the amount of details of the original data captured depend on the granularity of the bins chosen. That is, the more the number of bins, and the more sensitive to the original data distribution the split selection strategy is, the more representative the discrete feature is of the original continuous feature. Here, a trade off needs to be made between the improvement achieved due to the reduced effect of noise, and the reduction in the representation of the patterns in the original data.

Balancing ensures that every instance of the dataset is seen equally during the model construction process, thus avoiding bias for the high frequency target labels. Such a bias occurs due to the nature of inductive learners to minimize errors over the training dataset [KKP06]. This allows the model to ignore low frequency labels because the penalty for such an action are low. Another cause for this is the model being

overfit to the **imbalanced dataset**. This will lead to the model generalizing well over only the instances with the high frequency labels, and ignoring the under-represented instances.

3.1.1 Data Wrangling

As already described, preprocessing is an integral part of an analysis pipeline. However, it also needs to be emphasized that it can also be performed iteratively, wherein, the results of application of a certain number of preprocessing steps are studied, and depending on the results, further preprocessing, or changes to the existing pipeline may be done. Such a style of data preparation has been referred to as *Data Wrangling* [Kan+11]. Wrangling is catch-all term used for the collective processes performed above to get the dataset ready for analysis in the best possible representation. We mention it here because wrangling is a way of looking at improving data quality with the aim of obtaining better algorithm performance.

[Kan+11] notes that upto 80% of the time in analysis are spent on tasks related to wrangling. Because of this significant portion of time taken in this process, wrangling activities now occupy a first class activity status, and hence, are now viewed as an integral part of the activities leading up to analysis. The way wrangling occupies first class status is by using the feedback obtained from the performance of the analysis algorithm to redo some parts of the wrangling. Thus, wrangling here becomes an iterative task with constant feedback. The emphasis is placed on making the data by the analysis algorithm which agrees with the view of data quality taken by us. Visual analytics is suggested to be the ideal tool to combine iterative wrangling and the eventual analysis. This is so because deriving insights from visual means is a lot faster and intuitive than other methods. In a similar fashion, [Ter+15] recommends maintaining a data lake on which curated data is placed. By curated, they mean data which has gone through the wrangling process, and is hence immediately usable by the following analysis stage. This is a difference to traditional thought processes in which the persistent store is the source of truth. An encouragement to maintain a store for wrangled data is a true indication that it is being considered as a first class activity.

3.1.2 Discussion

There is a clear agreement that preprocessing is essential to obtain a good quality dataset, and that it has an impact on the performance of the learning algorithm to which it is supplied to. However, the works in this area are heavily focussed on the techniques

and the procedures of applying these preprocessing techniques, but do not explicitly study the effects of these preprocessing steps on the construction of learning models.

3.2 Empirical Evaluation

The theme of interaction between data quality and learning algorithm performance has also been viewed in research through empirical evaluations. That is, evaluations of varying data quality against learning algorithms. Across all these works, a common theme is that of generation of datasets of varying quality. This is then supplied to various learning algorithms, and the results compared. The analysis of results forms the study of the how quality affected the learning algorithm. The central theme of having datasets of varying quality to supply to algorithms is adapted by us in our approach. However, as will we see below, we bring more structure to such an analysis.

Performing such an empirical analysis is a classic way of analysis, as it has been used in the very early work which formalized decision trees itself [Qui86]. Here, it is used as a way of validating the resiliency of the decision tree approach to issues in data quality. The following data quality issues are dealt with: noise, fields with unknown values, and fields with a large number of unique values. *Noise* is seen here as an irregularity in the data, which does not describe data as in the real world, and is a spurious pattern in the dataset. That is, they do not represent the reality of the world, and hence lead to incorrect information being supplied to the model during construction. Noise is said to interact with the tree construction directly by increasing the complexity of the tree generated. *Complex* means a tree with greater depth and size. This is due to the fields of the dataset becoming *inadequate* to be able to perform classifications. This work then goes on to suggest approaches which could be used to deal with noise in the data. These approaches suggested are specific to the way in which the algorithm for tree construction can be modified, and thresholds which can be applied to the values of the impurity measures used to calculate the splits (Section 2.5). The empirical evaluation is then performed to evaluate the resilience of this approach against noise in the datasets. For this purpose, the approach followed is to generate datasets with varying levels of noise, by artificially injecting increasing proportions of the data with noisy data. A noisy data value for a field is one among the other value of the field. Thus, different datasets are generated each with a noise level in between 5% and 100%, in increments of 5% in each case. Each such dataset is then used as the training set to train a decision tree model. The same set of objects, in their original form, were then corrupted again to the same extent as the training set, and used as the test dataset, and the error rate of the classifier was determined. It is observed that with increasing amounts of noise in the data, the

classification error increases linearly and reaches a peak, and beyond 80% corruption, a slight decrease in the error rate, by about 1% to 2% is observed. Explanations for both these observations are given as under:

1. The increase in error rate with increasing noise is attributed to the decision tree model essentially being constructed using increasingly random choices. That is, due to the fields becoming inadequate, the split choices during tree construction now reflect an attempt to learn the noise in the data. This leads to the decision rules to become complex, and hence leads to more complex trees
2. The slight decrease in error rate at very high noise levels is attributed to the fact that predictions will now be biased towards objects of the majority class due to very high levels of random split choices, and hence this behaviour will be more pronounced in case of classification problems with high label imbalance

The other data quality issue dealt with is unknown values present in the data. A study similar to the one above has been performed, where the proportion of missing values in the dataset is gradually increased, and the prediction performance assessed. Objects in the dataset are injected with missing values by replacing each value of an attribute with an 'unknown' token with m% probability. This then forms a dataset with m% *ignorance*. It is observed that classification error increases gradually with increasing *ignorance*.

Another observation made by [Qui86], is the effect of fields with a high number of unique values on split selection. The information gain criterion described above is biased towards selecting fields with a large number of unique values. A consequence of this is described through the example of an arbitrary random field with an extremely high cardinality, such that no two objects in the dataset have the same value for this field. It is shown that such an attribute would have maximum information gain, leading it to be selected as the root of the tree. Since this field contributes no information with respect to patterns for the target label, this is a poor choice. Two remedies have been proposed to combat this bias [Qui86]. The first, is the concept of *Gain Ratio*, which penalizes high cardinality features by dividing the Information Gain obtained for this feature by the *Entropy* of that feature. An additional method suggested to combat this bias, is that of constructing trees purely through performing binary splits. Since, for performing binary splits, individual values of features are tested for impurity rather than entire features, this effectively compensates the high cardinality bias. Our approach heavily rests on corrupting datasets incrementally, and observing variations in the performance associated with each varying level of quality issues. Also, we test the effect of high cardinality fields by introducing fields with high cardinality for classification. Our approach is however, different on the following grounds:

1. Our approach in considering the quality of the dataset extends beyond considering a single metric, which in this case is Noise. That is, we provide an approach which allows to consider further such metrics, and perform comparisons of this nature
2. We also consider further metrics in assessing the performance of the algorithm beyond the error rate
3. Most importantly, the perspective taken here is limited to considering the effects of varying one aspect of the data quality. We wish to provide a more complete approach, which allows for comparisons not only through the perspectives of data quality, but also through the perspective of the configurations of the learning algorithm

[PK04] looks at such an empirical study by explicitly defining a cost model which accounts for the costs associated with a data mining initiative in terms of the quality of the dataset, and also in terms of the effort spent in preprocessing the dataset before the mining operation (Section 2.3.2). They consider the following aspects in accounting for the cost of a data mining initiative:

1. Proportion of missing data, m
2. Proportion of noisy data, n
3. Proportion of inconsistent data, i
4. Amount of data preprocessing necessary, dp

Given the above terminology, the cost associated is then given by Equation 3.1, where $cost$ is the cost computed for the data mining initiative, and functions f_1 to f_4 are functions which can quantize the aspects they are associated with, and normalize the obtained scores to a uniform scale.

$$cost = f_1(m) + f_2(n) + f_3(i) + f_4(dp) \tag{3.1}$$

The authors acknowledge that vast liberties have been taken in order to realize the cost function consisting of the entities above. These liberties include:

1. The assumption that the noisy and inconsistent data are disjoint. Though in reality, strong overlaps can be present between the two
2. The existence of functions f_1 to f_4

For the purpose of evaluations, the scope of the work is limited to considering a cost function using only the quality aspect of missing data. Also, no explicit $cost$ is calcuated. Instead, the percentage of correctly classified instances by the classifier is considered. The empirical analysis proceeds in a way similar to [Qui86], wherein data with varying levels of corruptness are supplied to different classifier families, and the percentage of correctly classified instances is studied. For the purpose of evaluation, there datasets are considered. One is used to construct the classifier model, and other two are used to test the performance of the classifiers. The following four classifiers are evaluated against: Neural Networks, Logistic Regression, C5.0 ad the Apriori Algorithm. Datasets with proportions of missing data ranging from 0% to 40%, in increments of 5% are prepared, and are used for the classifier construction and testing purpose. Such datasets are generated by randomly selecting the required proportion of instances in the dataset, and for each such instance, randomly omitting the value of one of its fields. The correct classification percentages obtained using the two test datasets is then analysed. The results obtained are inconclusive as the classification percentages do not indicate any pattern of deterioration with increasing missingness in the data. Though for the C5.0 algorithm, an enhanced decision tree algorithm, an actual improvement in the classification accuracy is observed with increasing missingness in the data. A thorough justification for this is not provided, but the following are given as possible reasons:

1. The supplied levels of missing data are insufficient to observe a deterioration in performance
2. Omitting values could very well be removing the noisy instances, leading to the improved classification performance

The analysis performed in this work is closer to what we wish to achieve. Specifically the following:

1. The interaction between data quality and algorithm performance is seen in a more systematic way than in [Qui86] by explicitly defining the cost function. The implication of this approach is that variations in the quality of the dataset can be viewed through different instances of this cost function and the computed cost can be used as a tool to perform a comparison and analyse the relative effects of the variations in the quality of the dataset
2. The flexibility of the cost function approach allows for extension with other metrics of data quality
3. Similarly, the flexibility in the definition of the associated $cost$ again allows us to define it in terms of different metrics of algorithm performance, or perhaps a combination of them

Our approach adapts the advantages described above. However, we provide a simpler, and more intuitive approach to systematically view this interaction. Functions which can normalize and scale quantities for abstract concepts like the preprocessing effort are hard to achieve, and out of scope of our work. And again, in this work, the ability to be able to specify configuration parameters for the learning algorithm is lacking, which our approach provides.

[SV06] use the setup described above to study the effects of data of varying levels of quality on the PC algorithm, which is an algorithm named after its founders, which allows the learning of Bayesian networks from data. Their primary motivation is to show that data quality is an important consideration for learning algorithms, and that learning algorithms should be designed keeping in mind such quality issues in mind. The issue the authors choose to deal with is *accuracy*. Inaccurate data is defined as the data which is not reflective of the real world situation it represents. Accuracy is the chosen metric due to its ease in being manipulated in the data, and the its high relevance in research.Modifying the accuracy in the data results in datasets of varying quality. The PC algorithm is then evaluated against these datasets. The performance of the algorithm is studied based on the network learnt by the algorithm when supplied with corrupted data, as compared to the network constructed when supplied with completely accurate data. That is, the edges of the network constructed using accurate data are said to be the correct edges. The observation is that initially, with rising inaccuracy in the data, the number of incorrect edges in the constructed network increases. However, with worse quality data, the number of incorrect edges surprisingly drop. This is said to occur because of the fact that with increasing inaccuracy in the data, the network learns not just the correct edges, but also incorrect edges which are present due to the inaccuracy in the data. This leads to double the number of edges, and hence higher count of inaccurate edges. At the lowest and highest levels of inaccuracy however, the network learns, either, only from the correct set, or the incorrect set, leading to a drop in the number of incorrect edges at the extremes. Their work goes one step further by trying to understand why the PC algorithm is so sensitive to inaccurate data. This is achieved by performing an analysis on the worst case time complexity of the PC algorithm, which increases by a magnitude when inaccurate data is introduced into the system. This occurs due to the graph being complete, as the noisy instances form additional connections. The work done by [SV06] closely reflects what we wish to achieve with our work. An evaluation was performed with datasets of varying quality, and the results analyzed. However, the concepts considered for evaluation are limited to the effects of inaccurate data, and the variations in performance of the algorithms are not studied by giving specific emphasis to the aspects of the data, and the learning algorithm which caused the changes.

3.3 Discussion

Both categories of the works described above give a clear description of the importance of data quality to learning algorithms with the aim of improving the performance of the latter. The unquestionable opinion is that lesser the quality of data, lesser the performance of the learning algorithm, stated more colloquially as “garbage in garbage out” [Pyl99]. However, what is missing is an evaluation of

1. The specific characteristics of the data quality which led to the improvement, or lack thereof, in the performance of the learning algorithm
2. How changes to the configurations of the learning algorithm can impact performance
3. And, how an interaction of the above impacts the performance

In order to enable the above, we need to devise a procedure to systematically track the various changes to quality done with the aim of improvement in algorithm performance, and the changes done to the algorithm configurations. Thus, our approach retains the spirit of the works detailed above by systematically applying quality varying transformations and maintaining information regarding their sequence. We go one step ahead and associate with each of these variations a concrete classification task along with its classification performance. The variations in these two distinct aspects lets us study the interaction amongst the two.

4 Approach

In this chapter we discuss the approach taken by us in order to achieve the goals discussed in Section 1.2. Our approach to a solution is based on the concept of profiles first introduced in [VZK16]. The major ingredients in our work are the DQP and the CCP. We begin this chapter with a description of each of these, followed by a description of their usage. The line of arguments which this approach allows us to take is detailed thereafter. How we view the information obtained from the profiles in order to accommodate the discriminators for decision trees closes off this chapter.

4.1 Concept of a Profile

The requirement for our work is to be able to systematically view the interaction between the two disparate concepts of data quality, and learning algorithm configurations, and the impact of this interaction on the algorithm's performance. When viewed individually, they are distinct, and belong to distinct areas of study. However, when viewed as part of the data analytics pipeline described in Section 1.1, understanding the impact of their interaction becomes essential since one follows the other. In order to understand this interaction, we need to be able to systematically look at the variations in these two aspects, and for each combination, be able to reason on the performance obtained.

In order to achieve this, we use the concept of a *Profile* [VZK16]. Every *Profile* deals with a single theme. For example, in the case of data analytics, possible themes can be data source description, data quality, learning algorithms, and data preprocessing. In its most basic form, a *Profile* is a collection of *descriptors*, identified by a name. Every *descriptor* for a *Profile* indicates a distinct aspect associated with that theme. That is, these *descriptors* indicate the aspects of the themes which we wish to study, and whose variations in values we are interested to track. Every such *Profile* can have any number

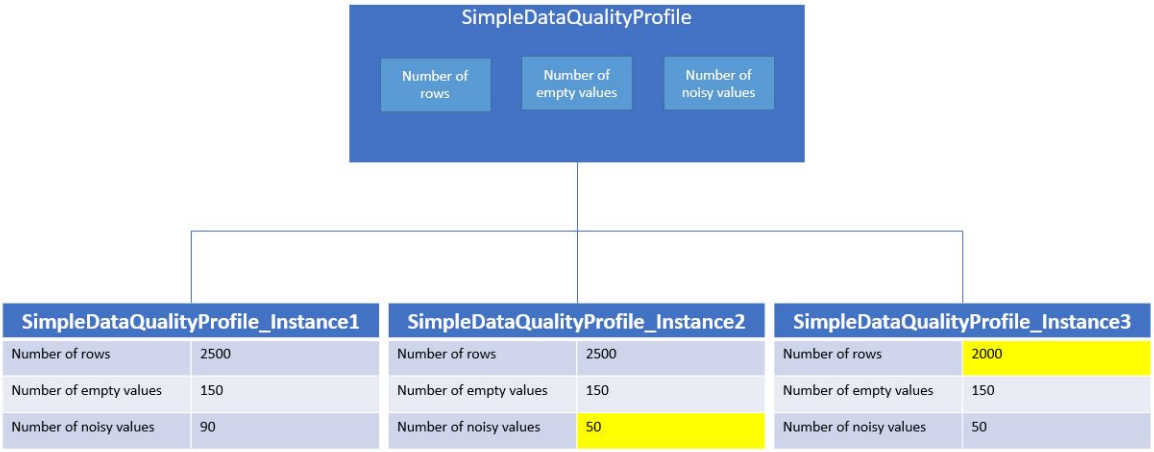

Figure 4.1: Example profile for the data quality theme

of instances, where each instance has a value associated with each descriptor. The descriptors remain fixed, but the values associated with them can change in accordance to the variations. An *instance* of a *Profile* has all the values for each descriptor filled. The granularity of these descriptors decide the specificity to which we can carry out our analysis. An example for this approach using the data quality theme is given in Figure 4.1. The name of the profile shown is the *SimpleDataQualityProfile*, which has the descriptors *Number of rows*, *Number of empty values* and *Number of noisy values*. In each of its 3 instances, the value of only a single descriptor changes.

For our work, we define profiles for the quality of the dataset, and the configurations applied to the learning algorithm used for classification. We will refer to them as the Data Quality Profile (DQP), and the Classification Configuration Profile, (CCP), respectively. The DQP indicates the quality characteristics of the dataset. The CCP indicates the parameters and their values used to configure the learning algorithm for a particular classification task. These are detailed in the following sections.

4.2 Data Quality Profile

As detailed in Section 2.1.3, a dataset profile captures descriptive metadata about the dataset. We extend this idea here for data quality through the use of a *Data Quality Profile*, which uses such descriptive metadata for the quality of a dataset as its *descriptors*. These *descriptors* are defined by using the data quality indicators described in Section 2.1.2. We define data quality indicators considering the *Interpretability* dimension

Descriptor	Significance	Value Space
Missing Values	Proportion of values having a missing or empty value (Section 2.1.1)	Real numbers between 0 and 1, 0 indicating no missing values
Invalid Values	Proportion of values which are considered invalid according to the field syntax or the domain semantics (Section 2.1.1)	Real numbers between 0 and 1, 0 indicating no invalid values
Entropy	Indicator of the balance of values in the field (Section 2.3.2.5)	Real numbers between 0 and 1, 0 indicating equal distribution of values of the field
Vocabulary Dispersion	For free text fields only. Proportion of tokens with low frequency (occurrence <= 10) (Section 2.1.2)	Real numbers between 0 and 1, 0 indicating no tokens with low frequency
Cardinality	Number of unique values	Set of all Whole Numbers

Table 4.1: Descriptors of the Data Quality Profile

described in Section 2.1.1. We cannot consider the *Accuracy* dimension, due to the fact that we have no way of verifying if the data represented in the dataset is indeed a true, accurate representation of the real world. To consider the *Relevancy* dimension, we would have to define quality indicators as optimized for our classification task, which is out of scope of this work.

As described in Section 2.1.1, *interpretability* deals with the view of data quality from the perspective of what is ideally suited for the end consumer. In our work, we deal with a learning algorithm as the end consumer. This specific case is discussed in Section 3.1 where the following quality issues were identified: missing values in data, invalid values in data, fields with high cardinality, and data with label imbalance. Looking at the *interpretability* dimension, the metrics we use to quantize this dimension must hence depict how well suited the data is for the learning algorithm. Hence, we define quality metrics keeping these quality issues in mind, with the constraint that they can be quantized into numeric values. These are summarized in Table 4.1, and described below.

For a field, the *proportion of missing values*, M is computed as shown in Equation 4.1, where N is the total number of samples available for the field, m is the total number of values, out of the N values, which are deemed missing. We use the information provided

by the dataset providers to classifies values as *missing*. In addition conventional values such as 'NA' and *blank* are also considered as missing in our work.

$$M = \frac{m}{N} \tag{4.1}$$

The *proportion of invalid values* is calculated using the ratio similar to the above. We consider the following two aspects to deem a value for a field as invalid:

1. Knowledge about the type of data which will be stored in the field (Table 2.2) to place syntactic constraints on
2. Domain specific information about the nature of the value held by the field to place semantic constraints

Given the above constraints, for a field, if i values are deemed invalid, with N having the same meaning as before, then the proportion of invalid values for the field will be given by I in Equation 4.2.

$$I = \frac{i}{N} \tag{4.2}$$

In Section 2.1.2, we saw that Shannon's Entropy was used described as a possible metric for free text fields, which measured the distribution of tokens amongst the documents. To measure *label imbalance*, we use the same metric here, normalized the cardinality of the field. We rewrite Equation 2.1 to calculate the Entropy, E of a field, given by Equation 4.3, using convention we used to discuss Entropy as a splitting criteria for decision trees in Section 2.5.1. A is the field for which we are calculating the entropy E, and $Values(A)$ provides all unique values of A, and $p(v)$ is the proportion of occurrence of the value v. Note that it is essential to use the cardinality of the field as the base of the logarithm to meet the normalization requirement set forth for quality metrics, stated in Section 2.1.2.

$$E = -\sum_{v \in Values(A)} p(v) log_{|Values(A)|} p(v) \tag{4.3}$$

In order to represent a field's *cardinality*, we have sacrificed the normalized constraint described in Section 2.1.2. Normalizing the *Cardinality* descriptor would lead to very low values for some fields, and would not accurately depict its meaning. Thus, we represent *Cardinality* using its actual value.

Parameter	Significance	Choices
Splitting Criteria	The measure used to calculate the purity of a split	Information Gain, Gini Index
Early Stopping Criteria	The measures used to control the size of the tree growth	Maximum Tree Depth, Minimum samples at each node, Threshold on impurity values
Label Balancing	Measures taken to offset the imbalanced labels problems	Training data balancing
Training Sample size	The amount of training data considered	A percentage of the training data sampled through stratification

Table 4.2: Classification Task Profile

Metric	Significance
Tree Depth	Shallower trees which generalize the training set well are preferred
Training Time	Shorter training times are computationally efficient and hence preferred
Accuracy	Accuracy obtained on the test dataset
Precision	Macro average of the precision obtained for all labels in the prediction
Recall	Macro average of the recall obtained for all labels in the prediction

Table 4.3: Performance Metrics

4.3 Classification Configuration Profile

The *descriptors* of the CCP consist of the parameters used to configure the learning algorithm used for the classification task. This is centred on the learning algorithm which is used for the classification. The configurable properties for the decision tree family were described in Section 2.5.3. These are the measure used to compute the purity of a split (Section 2.5.1), thresholds for early stopping of tree growth, (Section 2.5.2) and data preprocessing tasks of sampling (Section 2.3.3.1), and offsetting the imbalanced labels problem through balancing (Section 2.3.2). These parameters and associated values form our CCP, and these are indicated in Table 4.2

4.4 Performance Metrics

We use the metrics for measuring the prediction performance of the algorithm described in Section 2.3.3.2. These are shown in Table 4.3. We use the macro average values of precision and recall as described in Section 2.3.3.2 to be able to capture the variations caused due to label imbalance. Along with the prediction performance metrics of accuracy, precision, and recall, we also use the depth of the decision tree generated, and the time taken to train the model as tree quality metrics. The motivation for this comes from the descriptions given in Section 2.5.2, about *overfitting* of the tree to the data quality issues in the training data. Hence, we use the depth of the tree as a metric to understand its response to quality issues in the training data. Increasing levels of noise imply more complex rules which translate to higher training times.

4.5 Usage

We now describe how *profiles* [VZK16] help us in accomplishing our goal, stated in Section 1.2. Our purpose of defining the profiles above is to be able to study the *interaction* between the various aspect that encompass the two themes of data quality and learning algorithm configurations. The descriptors of the profiles described above define the scope of the interactions we wish to observe. The purpose of using profiles is to be able to capture the variations in these two themes, and give individual representation to each such variation. This is possible due to the very nature of profiles, described in Section 4.1, as every variation is represented by its concrete *instance*. This, in effect, transforms the study of interaction between two disparate themes to a systematic study of the interaction between various combinations of well defined profiles. Thus, to demonstrate the required interaction, we define multiple instances of each profile, and then perform evaluations using various combinations of the profiles. The performance of each such evaluation is defined using the performance metrics described in Section 4.4, which are used to compare multiple executions against one another.

This gives us a broad range of observations, and an instant tool for comparison. This is depicted in Figure 4.2. It shows how the variations in data quality lead to multiple instances of DQPs, and variations in the algorithm configurations lead to multiple instances of the CCPs. Every execution using a combination of the profiles leads to certain values for the performance metrics. Analysis is performed by observing the metrics obtained, and the information which can be *derived* by considering the combination of the two profiles.

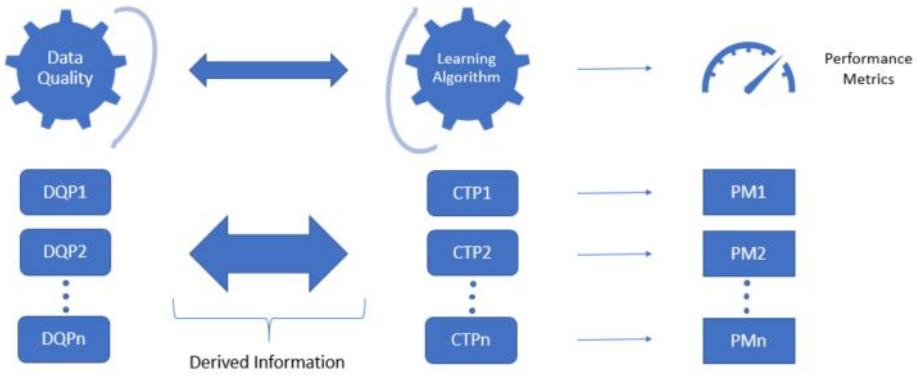

Figure 4.2: Usage of Profiles

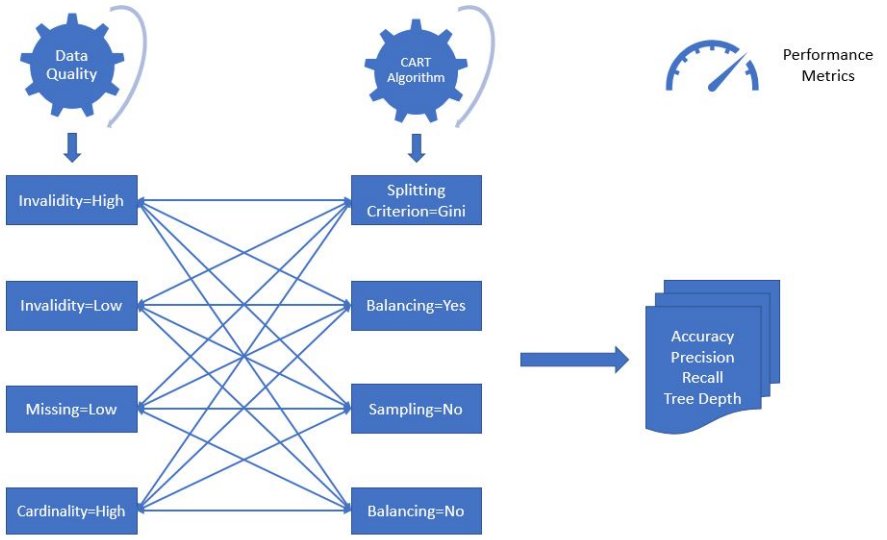

Figure 4.3: Example for the approach using the profiles

Figure 4.3 shows an example of our usage of *profiles*. Various instances of profiles are defined The names given to the profile instances are self explanatory. *Every* interaction between the profiles results in values for the performance metrics. We don't define any specific values for the *High* and *Low* values. They are provided for indicative purposes. One possible analysis using the *derived* information described above is, for example, the interaction between data containing low proportion of missing values and with sampling, and that without sampling.

As mentioned in Section 1.2, our work is restricted to considering the quality aspects of missing values, and cardinality. In order to realize the goals set forth, we define instances of the DQP which represent variations in the proportion of missing values, and

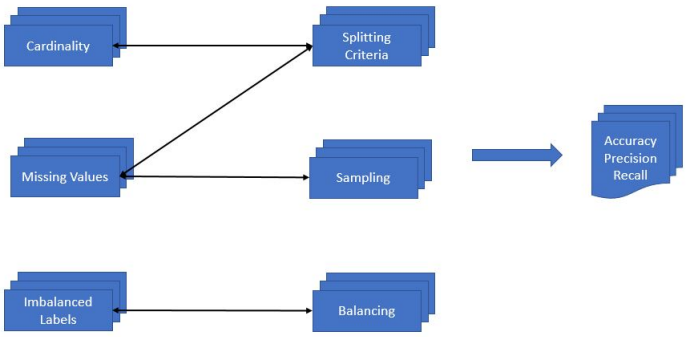

Figure 4.4: Approach using profiles for our scope

in the cardinality. This is shown in Figure 4.4, and directly follows the goals set forth in Figure 1.3. We now define concepts which we will evaluate through the interaction of the profiles shown in Figure 4.4.

4.6 Concepts for evaluation

To evaluate our research goals, we employ multiple scenarios with a range of DQPs and CCPs. To be able to justify the observed variations, or lack thereof, we select the following concepts based on the theory of decision trees covered in Section 2.5, which affect their construction. The concepts evaluated through the interaction of the profiles shown in Figure 4.4 are described below.

4.6.1 Interaction between Missing Values and Splitting Criteria

Here, we study two scenarios. First, we study the differences in behaviour of both the metrics Information Gain and Gini Impurity. As described in Section 2.5, we expect no significant differences in the trees constructed using the two splitting criteria. However, we wish to study this with a large dataset and varying levels of quality. Specifically, we want to analyse if this claim holds in the face of datasets with a large amount of missing values, and if one splitting criterion is more or less sensitive to the quality of supplied data. Also, due to the additional logarithmic computation required for Entropy, we suspect runs using the Information Gain as split criterion to have slightly longer training times.

Next, we choose the splitting criterion which gave the better performance in the evaluations above, and study the effect of increasing amounts of missing values in the data on the prediction performance. As seen in Section 2.5, missing values lead to providing erroneous information during the construction of the tree. This is because information regarding the correlation between the values of the feature and the target value are lost, making the information provided by the feature *inadequate*. Hence, it has been observed in several empirical studies seen in Section 3.2, that the prediction accuracy of the trained model drops with increasing levels of noise in the data. The concept of overfitting described in Section 2.5 is responsible for deterioration in the accuracy of the model over the test dataset.

4.6.2 Interaction between Missing Values and Data Sampling

As discussed in Section 2.5, one of the techniques to combat performance deterioration due to overfitting is to sample the training data beforehand. One of the consequences of sampling the training data is a decrease in the depth of the tree generated. The purpose of evaluating the impact of a smaller training sample size is to see whether the, presumably, simpler tree generated provides the any benefit to combat against increasing amounts of missing values in the data.

4.6.3 Interaction between Cardinality and Splitting Criteria

As described in Section 2.5, the Information Gain splitting criterion is biased towards features with high cardinality. However, in case of trees with a binary splitting nature, this tendency becomes reduced. In order to study this, we introduce a feature with high cardinality, and observe the features selected to perform the splits at the nodes of the constructed tree. Through this we analyse if the high cardinality feature had any influence on the split selections made by Information Gain criterion.

4.6.4 Interaction between dataset with Imbalanced Labels and Data Balancing

The imbalanced labels problem is a classical problem in classification. One way to combat this is by balancing the dataset so that there are equal number of samples for all target labels, as discussed in Section 2.3.2.5. This will need undersampling of the rows with the majority labels and oversampling the rows with the minority labels. In a non balanced case, we expect the less frequently occurring labels to get under-represented.

With balancing, however, we expect both accuracy, precision and recall to improve, as the minority labels get more emphasis, and hence the model generalizes unseen data better.

5 Implementation

In this chapter, we describe the steps taken by us in order to realize a minimal implementation of the concepts described so far. We first describe the dataset we use, and the classification task which we use for demonstration. We then describe the architecture of the system used to build this pipeline, and follow this up with specifics of the technology used. Our evaluation approach rests on the usage of profiles. These are implemented by artificially modifying data, and by configuring the parameters associated with the algorithm. This is described next. We close the chapter with the approach used to execute the code for the test runs, and the challenges involved.

5.1 The Dataset

We use the National Highway Transport Safety Administration (NHTSA) Complaints dataset [DI] for our evaluation. This dataset consists of complaints filed by vehicle owners regarding their cars after the occurrence of an incident, in an online form. The dataset is made available as an open dataset in a tab delimited format (.tsv) file. This dataset consists of 515209 rows, and 50 fields per row. The 50 fields present in the dataset provide information regarding the vehicle involved in the incident, the owner of the vehicle, and some specifics about the actual incident itself.

For our work, we restrict ourselves to a subset of the fields available. We filter the fields based on the following criteria:

1. Fields with an excess of 5% missing values. We carry forward the definition of missing values used in Section 4.2
2. Fields which act as primary keys

After filtering fields based on the above criteria, we remain with 17 fields out of the original 50, These are described in Table 5.1.

Field Name	Meaning	Type	Example
MFR_NAME	The name of the car manufacturer	String	'BMW'
MAKE_MODEL_TXT	The make and model of the car manufacturer	String	'FORD ESCAPE'
CRASH	Indicates occurrence of a crash	Boolean	Y
FAILDATE	The date of occurrence of the incident	Date	'20120321'
FIRE	Indicates occurrence of a fire	Boolean	Y
COMPDESC	Description of the component involved in the incident	String	'AIR BAGS: FRONT'
CITY	The car owner's city	String	'BEAUMONT'
STATE	The car owner's state represented using the official abbreviation	String	'TX'
DATEA	The date the incident was added to the file	Date	'20130107'
LDATE	The date the complaint was received by NHTSA	Date	'20120214'
CMPL_TYPE	Code of the complaint	String	'EVOQ', 'IVOQ'
POLICE_RPT_YN	Indicates whether a police complaint was filed	Boolean	Y
ORIG_OWNER_YN	Indicates whether the owner is the original owner	Boolean	Y
ANTI_BRAKES_YN	Indicates whether the anti-break locking system was on	Boolean	Y
CRUISE_CONT_YN	Indicates whether the cruise control system was on	Boolean	Y
PROD_TYPE	Code of the product type	String	'C', 'E'
YEARTXT	The car model year	Numeric	1975
CDESCR	Description of the complaint as provided by the car owner	Free Text	'The brakes do not work!!!'

Table 5.1: Fields of the NHTSA Complaints Dataset used for our work

Some of the fields of the dataset were available for user entry only in the recent past. Hence, a large number of the fields contain a significant proportion of missing values. For the purpose of this work, we have not considered fields which contain in excess of 5% missing values. The reason for such a strict threshold is twofold:

1. We use the data in its original form as the ground truth dataset. Thus, it is ideal to have a dataset relatively free of noise
2. The technology in use does not support blank values for features. Hence, this constrains us to impute missing values for numeric features. Excessive imputation can bias the dataset, which is undesirable

5.2 The Classification Task

The classification task which we will use for evaluation purposes is to predict the maker and model of the car involved in the incident. This is represented through the field 'MAKE_MODEL_TXT' in Table 5.1. We use every other field as a feature for prediction. Also note that the free text field is vectorized, and is appended to the feature vector generated by the other structured fields.

5.3 Technology Considerations

The following technologies were considered for our work:

1. Apache Spark v 2.1.0, along with its Machine Learning Library (MLLib)
2. Scikit Learn v 0.18.1

The primary reason to consider Apache Spark was the out of the box support provided for machine learning algorithms, and the efficient in memory handling of large data sets. However, the issue occurred with usage of the sparse matrices. Our task generates a feature vector of large scale, due to amalgamation with the textual feature vector, and hence, using sparse matrices is absolutely essential to be able to perform in memory computation. This is even more important due to the fact that Spark performs pure in-memory computations. But during executions of distributed learning algorithms, the sparse matrices are made dense, and this does not fit into memory, which caused out of memory exceptions. Scikit-Learn effectively combats this by ensuring sparse matrices remain sparse during usage for training learning models. Hence, our prototype has been implemented using Scikit-Learn.

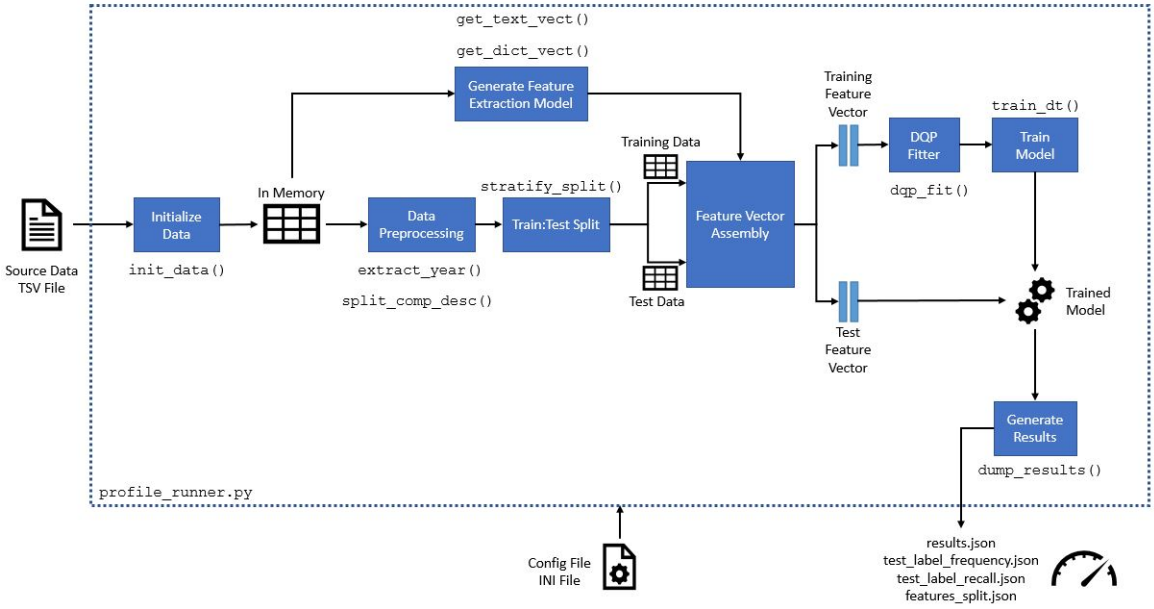

Figure 5.1: System Architecture

Python specific terminology *list* and *dictionary* have been used in the descriptions of these chapter. A Python *list* is equivalent to a classic array, and is a contiguous list of objects. A Python *dictionary* is a map which holds key-value pairs. According to usual convention, the keys are unique.

5.4 System Architecture

Our solution is built using the Scikit-Learn library, version 0.18.1. The Scikit-Learn API has been exposed in Python, and hence, we use Python for all operations. We use Python version 2.7.6. In addition, as prerequisites, Scikit-Learn v0.18.1 requires Python libraries NumPy version 1.6.1 and SciPy version 0.9 [JOP+01]. The solution was executed on a machine running Ubuntu 14.04.5 LTS, having 49 GB RAM, 109 GB hard disk and 16 cores, each with a clock speed of 2.5 GHz.

The system implemented is a Python script named *profile_runner.py*. And its architecture is shown in Figure 5.1. Method names close to component objects indicate the methods which are used in the code to implement corresponding functionality. Note that the paramter names are omitted due to space constraint. They are elaborated below. Descriptions of individual components follow.

5.4.1 Configuration File

Various parts of the script are configurable. Hence, the diagram shows the configuration file as being applicable to the entire script. This configuration file is maintained as an INI file, and is parsed in Python using the `ConfigParser` library. The configurable properties for each component are detailed in corresponding sections.

5.4.2 Initialize Data

Here, we assemble the data into an in-memory representation from the flat file format in which it originally is in. To be able to this, we need the following information:

1. Format of data storage.This means the way in which the entities of the dataset, and the fields os each entity are separated
2. After obtaining the separate values as above, we need to know the meaning of the each value based on what position it is in. This is schema information
3. Finally, we need to know how much of the data must be considered

We obtain both the above information through a file provided with the dataset. The file is in the *Tab Separated Values*, or *tsv* format. This implies every line in the file indicates a different entity. Here, each entity corresponds to a complaint of an incident. Each field per entity is separated by a tab delimiter ('\t'). Hence, we split the file on two levels. The first on the new line delimiter ('\n'), and then, for each line generated, we split using the tab delimiter, to get access to every field of the row. The fields per row are available as a list. This is advantageous, because this lets us access each field individually, and consider only the fields we find appropriate, as described in Section 5.1. Programmatically, this results in a two dimensional array with dimensions [515209][50]. We use this representation to generate a Python dictionary for each entity, with keys being the names of the field, and the value having the content of that field. Thus, our final in-memory representation of the data is a list of dictionaries, where the length of the list is 515209, and the number of keys per dictionary is 18, which correspond to the 17 fields described in Table 5.1, plus the target label field *MAKE_MODEL_TXT*. Table 5.2 shows the properties which can be configured for this component.

Configuration	Significance
Source Path	Path of the flat file
Field Separator	Delimiter separating fields in each line of the flat file
Schema	Comma separated values indicating headers for the data represented in the flat file
Categorical Indices	Comma separated values indicating the positions of the categorical fields in the flat file
Numerical Indices	Comma separated values indicating the positions of the numerical fields in the flat file
Text indices	Comma separated values indicating the positions of the free text fields in the flat file
Label Index	Position of the target label

Table 5.2: Configurable elements of the Initialize Data component

5.4.3 Data Preprocessing

We first preprocess the data to transform it in a way which will be beneficial to the construction of the learning algorithm. Here, we apply the Feature Construction operation (Section 2.3.2.4) in the ways described as under.

Fields *FAILDATE*, *DATEA* and *LDATE* consist of a timestamp as a String field in the format YYYYMMDD. We are interested here only in the year component for two reasons:

1. Date and month are too fine grained units to have an impact in this case on the target label
2. Transforming to only the year acts as binning into years which also helps reduce the dimensionality

Thus we construct the numeric year field from the String timestamp field for these features.

Additionally, we observe that the field *COMPDESC* contains of hierarchical items. Consider the following values for the field: AIR BAG: ALLEGED COUNTERFEIT AIR BAG, AIR BAG,AIR BAGS: ROLL PROTECTION, AIR BAGS:FRONTAL. As observed, all of these values correspond to the AIR BAG component. Hence, we split these values amd retrieve only the parent component.

5.4.4 Training Test Split

Here, the task is to perform a stratified split of the dataset to training and test data. We use 70% of the data for training and 30% for the test data. The purpose of *sampling* is to make the scale of the data more manageable to the infrastructure. Here, we use it as a method to evaluate the effect of dataset size on the construction of the decision tree. While sampling, it has to be ensured that the nature of the dataset is not altered with. Hence, a stratified split, as described in Section 2.3 has been performed to preserve the label distribution characteristics of the dataset. The SciKit Learn library for performing a stratified split, `sklearn.model_selection.train_test_split` is unusable for us due to the fact that many of the target labels have a frequency of 1, and single frequency labels are supported by this library. Hence, stratified split has been performed manually.

We first construct a dictionary mapping the labels in the dataset to the row indices of the rows they is labelled to. If the desired sampling quantity is x%, we then randomly sample x% of these rows per label, using the Python `random.sample` method. In case there is only a single sample for a target class, it is selected unconditionally. Using this we obtain a x% stratified sample from the data, where x is the target sampling ratio. is taken to first generate the 70% split for the training data. The rows not selected though this approach form the remaining 30% samples for the test data. This results in a stratified split without repetition. Note that, however, this results in the single frequency samples being present in both the training and the test datasets. Currently, no approach has been tried to combat this, nor have the impacts of this been studied.

5.4.5 Feature Extraction

The purpose of feature extraction is to transform the data into a representation acceptable by the learning algorithm. The constraints put forth by the technology here dictate the extent to which feature extraction is necessary. Constraints put forth by SciKit-Learn are shown in Table 5.3

The *Numerical Feature Vectors* constraint clearly only affects features which contain a textual value. One way to deal with this is to performing mapping of these textual values to numerical indices using `sklearn.preprocessing.LabelEncoder`. This performs the mapping of string to numerical indices preprocessing described in Section 2.3.2.6. The issues already described there can affect a decision tree construction, because the CART decision tree implementation made available in SciKit-Learn assumes pure numeric variables, and hence splits are based on inequality comparisons. Such comparisons do not make sense

Constraint Name	Constraint Description
Numerical Feature Vectors	Feature vectors (Section 2.3.2) may consist purely of numerical values only. No other data type is allowed
Categorical Features	Categorical features are not explicitly supported [jbl]. That is, there is no specific data type for dealing with categorical features

Table 5.3: SciKit-Learn constraints

on nominal categorical features. Hence, we do not use this approach. The approach used by us to perform one-of-K encoding, specifically the OneHot encoding, available through the `sklearn.preprocessing.OneHotEncoder` function. This offsets the continuousness bias, and effectively represents categorical features. Hence, this also solves the *Categorical Features* constraint.

Both the above described constraints are dealt with SciKit-Learn through a cleanly packaged function `sklearn.feature_extraction.DictVectorizer`, which generates the feature vector which can be immediately used by the decision tree function. Numerical features are not modified, and are added as is to the feature vector. OneHot Encoding is performed for the categorical features, and then assembled in the feature vector. Differentiation between numerical and categorical features is done using the data type of the feature values. The former can have `int` or `double` type, and the latter can have `string` type. The dictionary representation of the dataset assembled by us, as described in Section 5.4.2, is immediately compatible with the `DictVectorizer` function.

The text field, *CDESCR*, which contains the complaint description has to be dealt with differently. We use the Tf-Idf vectorizer, available through the `sklearn.feature_extraction.text.TfidfVectorizer` function us to generate the feature vector for the text field. This has to be then assembled with the feature vector generated as described above using the `DictVectorizer`. This is done by horizontally *stacking* the sparse matrices generated using both these vectorization functions. Horizontal stacking implies the order of both these matrices must be of the form nXm, and nXk, where n is the total number of samples, m is the length of the feature vector generated using the `DictVectorizer`, and k is the length of the feature vector generated using the `TfidfVectorizer`. This results in a sparse matrix of order $nX(m+k)$, which represents the final feature vector supplied to the decision tree algorithm.

Note from diagram 5.1 that the feature extraction model is generated using the *whole* dataset first and then individually applied to the training and test datasets. This is necessary to ensure consistency in the feature vectors generated for both the datasets, and also to ensure that unseen values are not encountered in either dataset.

5.4.6 DQP Fitter

We implement a minimal DQP considering only the *Mising Values* indicator, shown in Section 4.2.

The dataset obtained in memory is now in its original source format. For the purposes of our work, we need to artificially generate datasets with varying levels of quality. Thus, the purpose of this component is, hence, to alter the dataset so that it conforms to the requirements of an instance of a DQP. As detailed in Section 4.5, we focus the implementation of DQP to the missing values, and the cardinality indicators.

In order to generate data with varying levels of missing values in it, the approach followed is to randomly inject values which indicate a missing value into the features. For the categorical features, we inject the value 'NA', and for the numerical features, we inject the value 9999. We choose these values as these are the values defined by the dataset providers as representing unknown information. Hence, we stick to the convention, and use these values to inject missing values.

First, we describe how a single row is corrupted with missing values. A row is corrupted by randomly selecting m% of the features in the row, and replacing the value at that row with either a 'NA', or 9999, depending on the type of the feature. These m% features are randomly selected from among the features which were selected the most for splits at the intermediate nodes, when the tree was constructed using the original *uncorrupted* data. In order to restrict this further, we consider only the top 10% most used features for the split. In addition, we do not corrupt all 10% of these features per row. Instead, we further randomly select 50% among these top 10% features and corrupt these selected features. We select features only among the 10% most split features in order to ensure that the corruption actually makes an impact during split selection. This ensures that while calculating a split, a loss of information is injected, hence successfully simulating the effect of missing values. This additional step of random selection is done to ensure that no bias is injected into the data by corrupting the same features for every row. If such random selection is not performed, it is possible that due to the corruption

Configuration	Significance
Corruption Percentage	Percentage of the rows in the feature vector to corrupt
Corrupt Values	Comma separated values indicating the values to corrupt with
Feature Range	Range of features to corrupt in each row
Features to select	Random percentage of features to corrupt with in the above range

Table 5.4: Configurable elements of the DQP Fitter component

being performed for the same features in every row, a pattern can be recognized thereby adding information, which is against what we are trying to achieve.

The number of rows to corrupt using the method described above is decided based on the requirements of the DQP instance. If n% of the rows have to be corrupted, then n% rows are randomly selected from the dataset, and the row corruption operation described above is performed for each of these n selected rows. We use this approach to corrupt the dataset with 5%, 10% and 15% missing values.

The number of rows to corrupt, and the range of features to corrupt per row are all configurable through the configuration file. Table 5.4 shows these components. Hence, using such a configurable approach, it is easily possible to alter the features being corrupted per row. For example, setting the range as just 1 feature, and selecting 100% of those values leads to corrupting just one feature per row. Also note that the values to corrupt with are also configurable, hence allowing operations such as corrupting with invalid values, and different values for missing values.

It is important to note that we perform all quality modification operations only on the training dataset. This is done to ensure that the test data remains a depiction of the patterns of the real world, and all quality altering operations affect only the training dataset which affects the construction of the model.

5.4.7 Train Model

This component consumes the final training dataset created, and constructs the decision tree model. In accordance to the profiles we implement, Table 5.5 shows the configurable properties.

Configuration	Significance
Depth	Depth to which to restrict the tree growth
Split Criteria	Splitting criterion to use
Balance Data	Whether to balance the training data
Sample Data	Whether to stratify sample the training data

Table 5.5: Configurable elements of the Train Model component

Parameter Name	Value Space	Description
`criterion`	`gini`, `entropy`	The splitting criterion to be used at each node
`splitter`	`best`, `random`	The technique used to find the best split at each node
`max_depth`	Set of natural numbers	The maximum depth to grow the tree to

Table 5.6: `DecisionTreeClassifier` hyperparameters used

We use the CART decision tree algorithm exposed through the `sklearn.tree.DecisionTreeClassifier` function. The feature vector generated for the training dataset is supplied to the function to construct the learning model. The parameters which can be passed to the function are the hyperparameters which configure the construction of the tree. Our CQP requires us to use the parameters defined in Table 5.6, and detailed under:

1. The `criterion` parameter defines whether the Gini impurity (`gini`), or Information Gain (`entropy`) is used as the splitting criterion at each node to find the best split
2. The `splitter` parameter defines the way the split point is computed for each selected feature. `best` selects the best possible feature split which gives the maximum reduction in impurity in the child nodes, and `random` computes a random threshold within the bounds of the selected feature, and uses this as the split point for the feature

We also provide the following two data modification operations configurable as part of this component. Balancing the training data, and sampling it. The reason these operations are placed here is because operations directly affect the nature of construction of the tree as described in Section 4.6. These are described here.

With *balancing*, we want to achieve an equal distribution of samples for each target label. The approach to implement this is to first find the ideal balance of samples per target label. For example, the label *JEEP GRAND CHEROKEE*, has 3430 samples

associated with it. But many labels like *GRAND DESIGN SOLITUDE*, and *PORSCHE PANAMERA TURBO* have only 2 and 1 samples respectively. In order to find the balanced number of samples per target label, we use Equation 5.1, where b is the number of samples per target label in the balanced case, N is the total number of samples, and y is the total number of distinct target labels. We then iterate through our data and store for each label, the row index of the samples labelled with it, as a Python dictionary. We then iterate through each label in this dictionary, and depending on the number of samples associated with it, do one of two things:

1. If the number of samples associated with a label is more than the balance target b, we randomly sample b number of samples for that label. This constitutes under-sampling
2. If the number of samples associated with a label is lesser than b, then we randomly sample x rows for this label, where $x = b - N$. This constitutes over-sampling. Note the caveat that for labels with very low frequency, less than 10 for example, the over-sampled rows might create a bias in favour of these target labels. As was the case before, this has not been dealt with here

$$b = \frac{N}{y} \tag{5.1}$$

In order to apply sampling, the same procedure as described in Section 5.4.4 is applied.

5.4.8 Results Generation

Results are made available by generating the following result artefacts, as JSON files:

1. results.json: Contains the performance metrics described in Section 4.4
2. test_label_frequency.json: For every target label, their frequency of occurrence in the test dataset
3. test_label_precision.json: For every target label, their precision in the test dataset
4. test_label_recall.json: For every target label, their recall in the test dataset
5. features_split.json: The features used for splits at nodes, along with the number of times splits have been made for that feature

The `sklearn.metrics` package provides `accuracy_score`, `recall_score`, and `precision_score`, which we use to obtain the corresponding prediction metrics. Note that in order to compute average precision and recall, we the `average='macro'` setting, which indicates that the precision and recall for all target labels are individually computed, and then averaged to obtain the final result. The generated tree model provides the `max_depth` attribute which gives the depth of the tree generated. The path to store the result files is configurable.

5.4.9 Helper Methods

The following methods do not directly contribute to the architecture depicted above, however, we mention them here for completeness:

1. `get_column`: For the representation of the data generated by the *Initialize Data* component, this method returns all values of the field. It accepts the index of the field as the input, and a boolean flag which toggle returning of only the unique values
2. `get_label_report`: Combines the test_label_frequency.json, test_label_precision.json and test_label_recall.json result artefacts mentioned above into a helpful CSV file for instant import into Spreadsheet applications for generation of visualizations
3. `compute_entropy`: Given a column index, computes the Entropy (Section 4.2)
4. `compute_dispersion`; Given the index of the free text field, computes the vocabulary dispersion (Section 4.2)

5.5 Execution Procedure

Our script `profile_runner.py`, needs the following artefacts before actual evaluation can begin:

1. The same training and test dataset have to be used for every run to maintain consistency. Hence, initially, training and test datasets are generated, and they are dumped onto storage using the Python `pickle.dump` method, which is a serialization helper. Subsequently, for every evaluation, the same datasets are loaded using the `pickle.load` method

2. The random feature selection of the DQP fitter relies on information of the top features used for splits in the non corrupted case. Hence, initially, the non-corrupted classification case is run, and using the features_split.json artefact, the features are ranked in **descending** order of their frequency, and stored in a simple text file, with new line as delimiter

5.6 DQP Computation

We give a quick description of computation of the *Entropy* and *Vocabulary Dispersion* descriptors.

To compute *Entropy*, we make use of the SciPy method `scipy.stats.entropy`, which gives us the flexibility to specify a custom base for our computation. To compute the distribution of values we use the `Counter` method. The ratio of the frequency of occurrence to the total sample size gives us the required value p. We compute this proportion for every value in the dataset, and the required Entropy is `e = scipy.stats.entropy(p, base = len(p))`, where `p` is a list containing the proportions of occurrence of every field in the dataset.

To compute the dispsersion, we first need the list of all tokens in the free text field. We use the regex pattern `r"(?u)\b\w\w+\b"` [Ped+11] to split every document into distinct tokens. We then use the `Counter` method to obtain the frequencies for each token. We then filter the tokens based on the criteria that their frequency be strictly greater than 10. The length obtained is then divided by the total number of tokens to obtain the required value.

5.7 Profile Examples

Here, we give example of a DQP for reference, in Table 5.7 To understand the low Entropy of CRASH, consider here the distribution of its values: 'Y': 0.06, 'N': 0.94. The Low Entropy occurs due to skew in the favour of a single value 'N'.

Name	Cardinality	Missing Values	Invalid Values	Entropy	Dispersion
MFR_NAME	808	0.007	0	0.5	
CRASH	3	0	0	0.2	
FAILDATE	5936	0	0	0.9	
MILES	54177	0.18	0	0.59	
POLICE_RPT_YN	3	0	0.008	0.17	
MAKE_MODEL_TXT	4936	0	0	0.67	
CDESCR					0.79

Table 5.7: DQP Example

6 Evaluation

In this chapter, we first define the profiles we will use for our evaluation in Section 6.1. We then analyse the results obtained by executing combinations of the profiles. And we evaluate our results against the claims made in Section 4.6. Detailed results are available in Appendix A.

6.1 Profiles

In the above sections, we described how we construct the various profiles necessary for our evaluation. Here, we detail the profiles we use, along with their contents. These are based on our original research goals discussed in Section 1.2.

6.1.1 Data Quality Profile

We use the indicators of the DQP described in Table 4.1 to describe the profiles we construct for our work. Our minimal implementation considers only the factor of missing values. This gives the quality profiles as described in Table 6.1. Note that the corruption takes place in accordance to the description in Section 5.4.6

Profile Name	Contents
GroundTruth	The data in its original form as obtained from source
Missing5	5% of the rows randomly corrupted
Missing15	15% of the rows randomly corrupted
Missing25	25% of the rows randomly corrupted
HighCardinality	Feature MILES introduced

Table 6.1: Data Quality Profiles

Profile ID	Contents
Gini	Gini index as the splitting criteria
Entopy	Information Gain as the splitting criteria
MaxDepth75	Depth of the tree restricted to 75% of the depth of the tree grown to full depth with the same configurations
Balanced	Dataset balanced in order to obtain equal samples for all target classes
Sample50	50% of the training dataset sampled with stratification before commencing the learning

Table 6.2: Classification Task Profiles

Profile Name	Splitting Criterion	Maximum Depth	Balanced	Sampled
Gini	`gini`	Till leaf purity	No	No
Entropy	`entropy`	Till leaf purity	No	No
Max Depth 75	`gini`	75% of depth obtained for *Gini*	No	No
Balanced	`gini`	Till leaf purity	Yes	No
Sample 50	`gini`	Till leaf purity	No	50% stratified sampling

Table 6.3: Descriptor values for the CTPs

6.1.2 Classification Configuration Profile

The configuration of the learning algorithm is set in the CCPs. These are described in Table 6.2. The descriptor values for each profile, according to the descriptors defined in Table 4.2, are detailed in Table 6.3.

6.2 Baseline Evaluation

In order to be able to compare the performance of the decision tree classifier, which performs predictions based on the patterns learnt from the training set, we use another dummy classifier which performs predictions purely based on the frequencies of the labels in the test dataset. This is done as follows. For every instance in the test dataset,

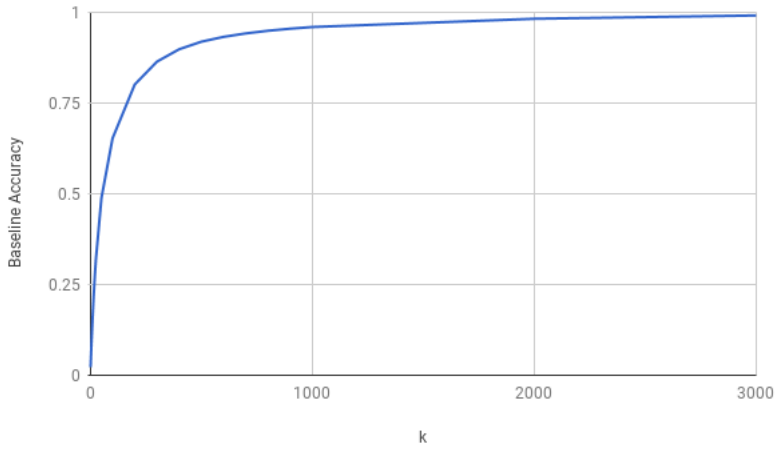

Figure 6.1: Accuracy results obtained for *k* most frequent labels in the test dataset

and for a positive integer *k*, the baseline classifier returns the top *k* labels in the test dataset when sorted from highest to lowest according to their frequencies in the test dataset [KM16]. This baseline classifier is said to have made an accurate prediction for an isntance in the test dataset, if the label associated with that instance is one among the *k* labels returned by the baseline classifier. The accuracies obtained using this baseline classifier is shown in Figure 6.1. We evaluate the baseline for values $k = 1$ to $k = 3000$. Note that the instance of $k = 1$, corresponds to predicting the most frequently occurring label in the test dataset. Accurate predictions of 80% are obtained for the case when $k = 200$, indicating that 80% of the test dataset instances are covered by the 200 most frequent labels in the test dataset.

6.3 Observed Results

The profiles introduced in Chapter 5 are implemented, and executed in various combinations for our analysis. The procedure described in Section 5.4 is followed in each case. That is, the training dataset is first fit according to the DQP necessary, and then used to train the model. Our discussion rests on the concepts introduced in Section 4.6, and we attempt to answer the questions set forth in Table 1.1. Using the profiles defined below, we now redraw Figure 1.3 in Figure 6.2. We first provide the results obtained by considering each concept, and in a later section we summarize all the results.

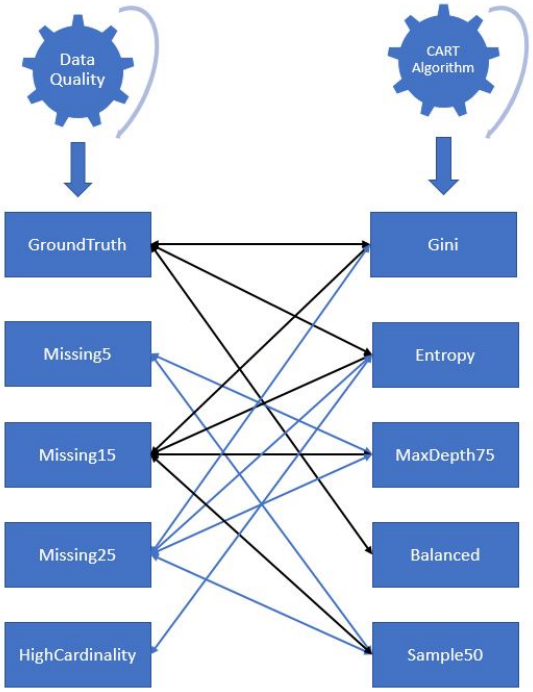

Figure 6.2: Combination of Profiles for Evaluation

Missing Values	Split Criterion	Accuracy	Precision	Recall	Training Time (s)
0%	Gini Impurity	0.52	0.48	0.49	6,228
0%	Information Gain	0.49	0.42	0.43	17,019

Table 6.4: Performance metrics with different splitting criterion, and varying missing values proportion

6.3.1 Interaction between Missing Values and Splitting Criteria

We first evaluate the most basic case with no missing values, that is using the original data through the *GroundTruth* profile. Here, we consider the criterion used at each node to select the best possible split. The Gini Imurity and Information Gain split selection criterion, introduced in Section 2.5 are used here, made available through the CCPs *Gini* and *Entropy*, introduced in Section 6.1.2. The data as obtained from source, through the *GroundTruth* DQP is supplied as the dataset for classifications performed through both the above CCPs. The performance metrics obtained are shown in Table 6.4.

Our expectations are twofold (Section 4.6.1). One, that the features selected for splits will be relatively same when using both criterion. And two, that the time taken to train using the Information Gain criterion will be significantly larger due to the additional logarithmic complexity. In order to analyse this, we first analyse the features selected during the splits, and their frequency of selection. We take the top 500 features selected by both the criterion, by frequency of selection, and find this similarity. It is seen that 93% of the features are the same. Which gives justification for the similar behaviour. Note that the top 500 features also consist of the vectorized text tokens. Thus, we are selecting the top 500 among approximately 300,000 features, and hence we justify that a similarity in the top 0.2% of the features is indeed an indicator of the similarity in the behaviour of the two criterion.

Our other expectation that the difference in the prediction performance should be minimal holds true. See Table 6.4. Though a uniformly slight drop in overall prediction performance is seen when Information Gain is used. Note the following observations to be made:

1. Computing information gain increases training time by over 200%. This can be attributed to the use of logarithms in the computation of the split. Though it might seem that a mathematical computation would account for such a significant rise in training time, we believe that the scale of the feature set makes this possible. Proving this analytically is out of scope of this work
2. Invalidity seems to have similar effects to the behaviour of both criterion, with uniform drops in prediction performance with rise in invalidity

6.3.2 Interaction between Missing Values and a single Splitting Criterion

Now, we evaluate the effect of increasing missing values on a single splitting criterion. For this evaluation, we consider data with varying levels of missing values, as made available through the DQPs described in Section 6.1.1. This data with increasing levels of missing values is passed through the analysis pipeline, with the *Gini* CCP (Section 6.1.2) in each case. The reason for using the *Gini* CCP is due to the fact that the best performance for the data in its original form was obtained with the Gini impurity as the split criterion, as shown above in Table 6.4. Table 6.5 shows the results observed by applying missing data. Note that the data made available through the *GroundTruth* data quality profile is shown as having 0% missing values in Table 6.5 for convenience.

Missing Values	Accuracy	Precision	Recall	Depth	Training Time (s)
0%	0.51	0.48	0.49	604	6228
5%	0.5	0.46	0.48	607	11,073
15%	0.48	0.42	0.44	617	19,625
25%	0.47	0.37	0.4 0	565	27,987

Table 6.5: Performance metrics for data with increasing levels of missing values using Gini Impurity as the splitting criterion

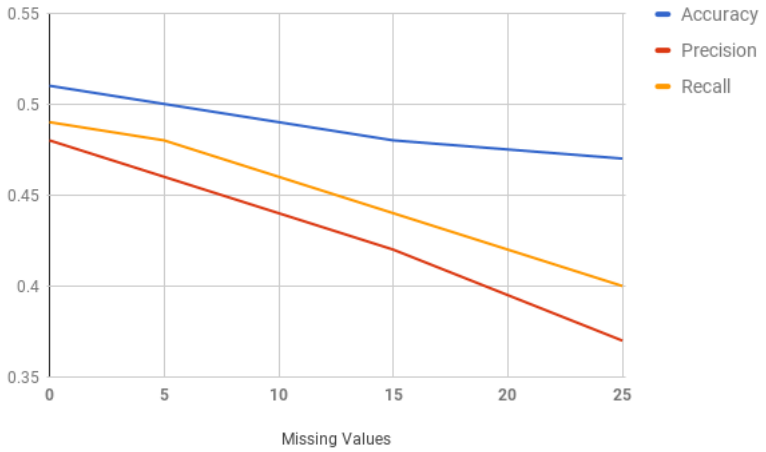

Figure 6.3: Effect of missing values on prediction performance

Consider Figure 6.3. A consistent drop in the prediction performance metrics is observed with increasing missing values proportion. Drops in accuracy are relatively minimal, with only a 1 - 2% drop for every 10% increase in missing values. However, drops in precision and recall are more pronounced. A 4 - 5% drop for every 10% increase in missing values. The drop in prediction performance in consistent with the assumptions made in Section 4.6.1. However, we wish to further investigate the reasons for the sharper drops in precision and recall. For this, we use the per label precision and recall metrics artefact generated through our script. In Section 6.2, we observed that, for the test data, 80% of the samples are covered by the top 200 most frequently occurring labels. Hence, we find the average precision and recall values for these top 80% labels, and for the bottom 20% labels. This is shown in Table 6.6. As it can be seen, for the top 80%, the average precision and recall values fall by 4%. However, for the bottom 20%, they fall by approximately 10%. Thus, we suspect that this is the cause for the steeper falls in the precision and recall values.

Missing Values	Precision top 80%	Recall top 80%	Precision bottom 20%	Recall bottom 20%
0	0.48	0.48	0.47	0.49
5	0.47	0.47	0.46	0.48
15	0.45	0.45	0.42	0.44
25	0.44	0.44	0.37	0.4

Table 6.6: Average Precision and Recall values for the top 80% and the bottom 20% labels

Figure 6.4: Effect of missing values on depth of constructed tree

Figure 6.5: Effect of missing values on training time

Missing Values	Accuracy	Precision	Recall
0%	0.52	0.48	0.49
5%	0.5	0.39	0.39
15%	0.48	0.36	0.37
25%	0.47	0.32	0.33

Table 6.7: Prediction performance with a 75% depth restriction

As seen in Figure 6.4, the depth of the constructed tree increases initially with increasing proportions of missing values in the data, which indicates an attempt to overfit to the noise in the data, as was assumed by us in Section 4.6.1. However, strangely, the depth drops drastically for the case with the highest proportion of missing values. But consequently, observing Figure 6.5, the training times increase linearly with no exceptions. Thus, we suspect that with increasing levels of missing values in the data, the time taken to train is a better estimate of the tree being overfit to the noise in the data, rather than purely the depth alone.

6.3.3 Interaction between Missing Values and Early Stopping Criteria

Here, our purpose is to evaluate the effects of early stopping in the tree growth against increase in missing values in the data. As seen in Section 6.3.1, performance deteriorates with increase in missing values. This is attributed to the tree being overfit to the noise in the training dataset. As described in Section 2.5.2, one approach to deal with overfitting is to explicitly restrict the depth to which the tree can grow to. We evaluate this strategy here by using CCP *MaxDepth75* (Section 6.1.2). We want to see whether early stopping the growth of the tree provided us any benefit. This is shown in Table 6.7, and depicted in Figure 6.6.

With increasing proportion of missing values, no improvements in any of the performance metrics is observed through use of the early stopping criteria. This indicates that the drop in accuracy due to increase in invalidity might not be due to the tree being overfit to the noise in the dataset. The drops is precision and recall indicate that problem of label imbalance might be adding sufficient bias in the dataset, so that improving prediction performance is not possible any more. A possible future work is to study the effect of early stopping criteria on balanced datasets.

A prominent observation from Figure 6.6 is the sharp drop in the precision and recall values , but a far less drastic drop in the Accuracy. To investigate this, we find the

Figure 6.6: Prediction performance with depth restriction at 75%

Test Label Frequency →	1	200	4628
Precision	0.47	0.38	5% Missing Values
Recall	0.47	0.39	
Precision	0.46	0.36	15% Missing Values
Recall	0.46	0.36	
Precision	0.44	0.31	25% Missing Values
Recall	0.44	0.32	

Figure 6.7: Average Precision and Recall for the most frequent and infrequent test data labels

precision and recall of the first 200 labels, and the rest, due to the motivation described in Section 6.2, and this is shown in Figure 6.7. As can be seen clearly, the precision and recall values of the bottom 80% test labels are consistently around 10% lower than that of the top 20%. This explains the drastic drop in these metrics.

6.3.4 Interaction between Cardinality and Splitting Criteria

Our assumption is that the theoretical bias of information gain for high cardinality features is offset in our case, because the CART decision tree performs only binary splits, and as described in Section **??**, this eliminates the bias for high cardinality features. We evaluate this by using the *HighCardinality* DQP, described in Section 6.1.2, by introducing the feature *MILES* with a cardinality of 54077 in the pipeline, and analysing

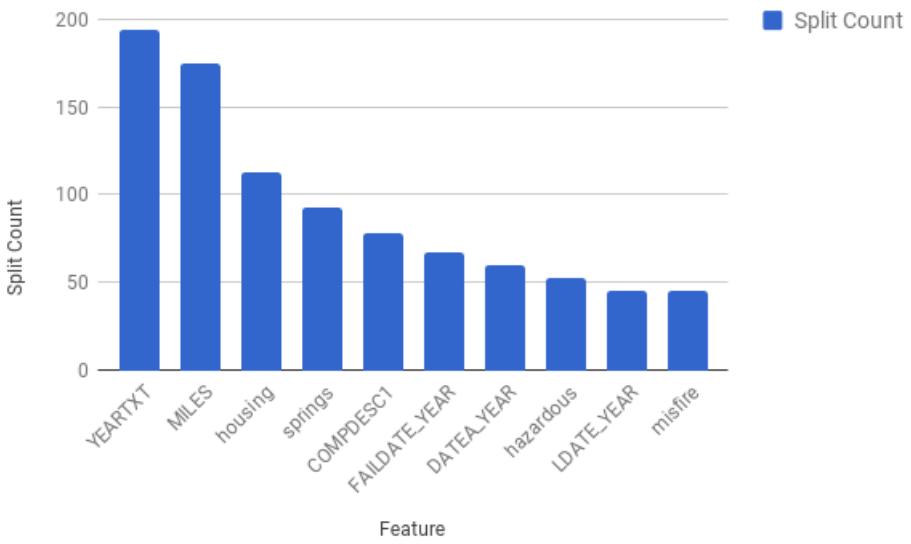

Figure 6.8: Top 10 most selected features for split with high cardinality feature and Information Gain split criterion

Balanced	Accuracy	Precision	Recall
No	0.51	0.48	0.49
Yes	0.21	0.43	0.49

Table 6.8: Performance metrics obtained with balancing applied to the dataset

whether this causes a difference to the features selected during the splits. The top 10 features selected for splits is shown in Figure 6.8. As can be seen *MILES* is indeed chosen may times for the split, at position 2. Thus, we observe that binary splits is not a robust enough tactic to deal with high cardinality features.

6.3.5 Interaction between dataset with Imbalanced Labels and Data Balancing

In order to combat target class imbalance, the approach we take is to balance the dataset so that there are equal number of samples for every target class. This was achieved by oversampling the under-represented labels, and undersampling the others. The results for this are in Table 6.8. For convenience, the results with the unbalanced case are also provided.

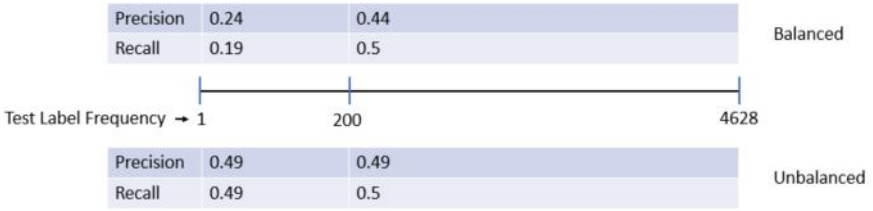

Figure 6.9: Average Precision and Recall for the most frequent and infrequent labels for the balanced and unbalanced case

The major observation is of the sharp drop in accuracy, however, a not so sharp drop in the precision and recall values. Comparing with what we saw in Section 6.3.3, accuracy seems to be the metric which is robust compared to the other metrics. Our understanding in Section 4.6.4 is that this occurs due to the majority labels over-compensating for the minority labels. However, here we see a contrast. A possible hint is that due to the undersampling of the majority labels performed, the prediction performance of the majority labels has reduced, thereby leading to a lower accuracy. We try to make this concrete using the similar top 80% figure, which is shown in Figure **??**. Our assumptions are indeed true as can be seen. The precision and recall values of the most frequent labels drops drastically. Thus, undersampling the majority labels is not an effective solution for the problem of label imbalance.

6.3.6 Interaction between Missing Values and Data Sampling

As stated in Section 4.6.2, the purpose of testing with a sampled amount of the dataset is to evaluate whether a smaller data sample does indeed result in simpler trees, and if the simpler trees compensate for the invalidity in data.

As can be seen in Table 6.9, sampling the training data failed to provide any improvement with increasing missing values in the data. Comparing with Table 6.7, sampling leads to an even further deterioration in performance, with almost similar depths as of the 75% depth restriction. Hence, in our case, we have been unable to achieve any benefit with sampling in face of missing values. However, depths of the tree have seen reduction by approximately 65%, as was our assumption in Section 4.6.2.

Missing Value	Accuracy	Precision	Recall	Depth	Training Time (s)
0%	0.42	0.38	0.41	482	3,014
5%	0.42	0.36	0.39	473	5,159
15%	0.41	0.33	0.36	480	9,538
25%	0.39	0.28	0.32	426	12,947

Table 6.9: Performance metrics with 50% sampled data

Evaluation Question	Finding
How do missing values in the training data affect the prediction performance and quality of tree generated?	Deterioration in both the prediction performance and the tree quality is observed. With sharper deterioration for precision and recall
How do the two splitting criterion behave with data of same quality?	Behaviour is similar as can be seen in the top features picked for splits
How effective are tree growth early stopping criteria in mitigating the effect of missing values in the data?	We were unable to observe any improvement due to early stopping of tree growth. A consistent drop in prediction performance was observed
How effective is training data sampling in mitigating the effect of missing values in the data, and how does it compare to the effect of early stopping criteria?	No improvement was observed with sampled data. A drop in tree depth was observed
How does data balancing affect prediction performance, and is it effective in improving prediction performance of low frequency samples?	Undersampling of the majority sampled led to a drop in overall prediction performance

Table 6.10: Evaluation Questions from Table 1.1 and our Findings

6.4 Summary

We summarize the findings of the evaluations by answering our original evaluation questions from Table 1.1.

6.5 Conclusion

To reiterate, our research goals were as under:

1. A holistic evaluation of the impact of the interaction between the quality of the dataset, and the choice and configuration of the learning algorithm used on the performance of the learning algorithm
2. An approach to systematically perform such evaluation through the use of DQPs and CTPs
3. Demonstration of the applicability of such an approach though the use of a sample analysis task, with the CART Decision Tree algorithm

The profiles defined in Section 6.1 serve as our tool to perform a holistic evaluation. The various concepts discussed above form the grounds on which we claim to show the impact the two screws have on the performance of the learning algorithm, which in our case is the CART Decision Tree. Hence, we assert that we have studied the relevance of the various aspects of data quality and the classification algorithm on the performance of the latter.

7 Conclusion and Future Work

The purpose of this work is to holistically study the influence of interaction between data quality, and the configurations of the learning algorithm, on the performance of the learning algorithm. This work is motivated by the need to perform a systematic analysis, and to establish a procedure for the same. Though existing work exists which extensively study these two themes in depth, a lack of a systematic approach in studying their interaction was felt. Through our work, we attempt to provide a middle ground which bridges the gap between the above two approaches, and allows to perform a holistic analysis.

It was observed that in order to be able to perform such an analysis, we need a formal notation for representing both the themes, and importantly, this approach must allow us to perform comparisons against multiple versions of similar things. This was achieved through the use of descriptive metadata known as *profiles*, for both the data quality, and the classification algorithm theme. The data quality profile consists of quality indicators for the data, and the classification task profile consists of the configuration of the classification task applied, and the preprocessing stages applied. Variations in each of the two themes are represented through a distinct profile which gives us the tool necessary to be able to perform a systematic analysis.

The viability of such an approach was demonstrated through a sample classification task on an open dataset. The technological scope was limited to the CART decision tree algorithm to construct the classification model. For evaluation purposes, the following aspects of decision trees were considered: the effect of splitting criterion, the effect of missing values in the training data, the effect of high cardinality features, effect of restricting the depth of the tree, and the effects of balancing and sampling the dataset prior to constructing the learning model.

It was observed that the claims made by us based on theoretical backings were supported in some cases. A consistent deterioration was observed in the prediction performance and in the quality of the tree generated. The information gain bias to high

cardinality features was also demonstrated. However, we were unable to mitigate the effects of missing values in data either through early stopping the tree construction, or through sampling the training data.

Thus, we were able to make evaluations considering the aspects of both data quality, and the classification algorithm. Such an evaluation was possible due to flexibility provided by using a profile based approach. Thus we claim that such an approach can be a valuable tool in order to perform evaluations concerning two disparate themes. The evaluations performed by us also enable us to claim that certain aspects of data quality can have an influence on the construction of learning models, and in the results of these models.

7.1 Future Work

The scope of the work was heavily restricted. This naturally leads to the motivation to further this work by expanding on the following accounts:

1. By considering additional learning algorithms. The most immediate possibility can be consideration of other decision tree families like C4.5 and ID3
2. Normalizing quality indicators is a non trivial task for some quality issues like label imbalance. This hints that the data quality profile can in itself warrant significant attention
3. Balancing for multi-class datasets is also a non-trivial concern. Even though balancing has been studied and mitigated extensively for binary classification problems, it is still a challenging task to accomplish for multi-class problems

Specifically with respect to the implementation, the associated script *profile_runner.py* allows the flexibility to select, to a certain degree, the features to corrupt per row. This can be exploited to achieve multiple combinations like single feature, or all feature corruption etc. Another feature provided by the script is the ability to provide values to use for corruption. This allows for study along the lines of the effect of single value corruption againt multi value corruption.

A Appendix

A.1 Execution Results

The results presented in Chapter 6 only focus on the results which convey a useful concept. Here, we provide the entire results of all executions performed. The profile descriptions can be found in Section 6.1.

DQP	CTP	Accuracy	Precision	Recall	Depth	Training Time (s)
Ground Truth	Gini	0.51	0.48	0.49	604	6,228
Ground Truth	Entropy	0.49	0.42	0.43	64	17,019
Missing5	Gini	0.5	0.46	0.48	607	11,073
Missing15	Gini	0.48	0.42	0.44	617	19,625
Missing25	Gini	0.47	0.37	0.4	565	27,987
Ground Truth	Sample50	0.42	0.38	0.41	482	3,014
Ground Truth	Balanced	0.21	0.43	0.49	1206	11,665
Missing5	Sample50	0.42	0.36	0.39	473	5,159
Missing15	Sample50	0.41	0.33	0.36	480	9,538
Missing25	Sample50	0.39	0.28	0.32	426	12,947
Missing5	MaxDepth75	0.5	0.39	0.39	455	10,834
Missing15	MaxDepth75	0.48	0.36	0.37	463	19,041
Missing25	MaxDepth75	0.47	0.32	0.33	424	29,760

Table A.1: All Execution Results

Bibliography

[Alp10] E. Alpaydin. *Introduction to Machine Learning*. 2nd. The MIT Press, 2010. ISBN: 026201243X, 9780262012430 (cit. on pp. 22, 34).

[BRS12] M. Bank, R. Remus, M. Schierle. "Textual Characteristics for Language Engineering." In: *LREC*. 2012, pp. 515–519 (cit. on pp. 20, 21).

[Bre+84] L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen. *Classification and regression trees*. CRC press, 1984 (cit. on pp. 36, 38).

[CLS06] S. F. Crone, S. Lessmann, R. Stahlbock. "The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing." In: *European Journal of Operational Research* 173.3 (2006), pp. 781–800 (cit. on pp. 25, 39, 40).

[DI] N. Office of Defects Investigation. *Office of Defects Investigation Flat File Downloads*. URL: https://www-odi.nhtsa.dot.gov/downloads/ (cit. on p. 59).

[Ell+] M. B. Ellefi, Z. Bellahsene, J. G. Breslin, E. Demidova, S. Dietze, J. Szymanski, K. Todorov. *Dataset profiling–a guide to features, methods, applications and vocabularies* (cit. on p. 22).

[HKK07] B. Heinrich, M. Kaiser, M. Klier. "How to measure data quality? A metric-based approach." In: (2007) (cit. on pp. 20, 21).

[JOP+01] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed <today>]. 2001–. URL: http://www.scipy.org/ (cit. on p. 62).

[JS02] N. Japkowicz, S. Stephen. "The class imbalance problem: A systematic study." In: *Intelligent data analysis* 6.5 (2002), pp. 429–449 (cit. on p. 27).

[KKP06] S. Kotsiantis, D Kanellopoulos, P. Pintelas. "Data preprocessing for supervised leaning." In: *International Journal of Computer Science* 1.2 (2006), pp. 111–117 (cit. on pp. 12, 18, 25, 26, 30, 32, 38–40).

[KM16] L. Kassner, B. Mitschang. "Exploring Text Classification for Messy Data: An Industry Use Case for Domain-Specific Analytics." In: *EDBT*. 2016, pp. 491–502 (cit. on p. 77).

[KZP06] S. B. Kotsiantis, I. D. Zaharakis, P. E. Pintelas. “Machine learning: a review of classification and combining techniques.” In: *Artificial Intelligence Review* 26.3 (2006), pp. 159–190 (cit. on pp. 13, 19, 23, 24, 33).

[Kan+11] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, P. Buono. “Research directions in data wrangling: Visualizations and transformations for usable and credible data.” In: *Information Visualization* 10.4 (2011), pp. 271–288 (cit. on pp. 12, 41).

[Kie] C. Kiefer. “Assessing the Quality of Unstructured Data: An Initial Overview.” In: () (cit. on pp. 18–20, 26).

[Lau86] K. C. Laudon. “Data quality and due process in large interorganizational record systems.” In: *Communications of the ACM* 29.1 (1986), pp. 4–11 (cit. on p. 17).

[Loh11] W.-Y. Loh. “Classification and regression trees.” In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1.1 (2011), pp. 14–23 (cit. on p. 35).

[Mit97] T. M. Mitchell. *Machine Learning*. 1997 (cit. on pp. 20, 22, 34, 36).

[Mur98] S. K. Murthy. “Automatic construction of decision trees from data: A multi-disciplinary survey.” In: *Data mining and knowledge discovery* 2.4 (1998), pp. 345–389 (cit. on pp. 35, 38).

[PK04] L. Pipino, D. P. Kopcso. “Data Mining, Dirty Data, and Costs.” In: *IQ*. 2004, pp. 164–169 (cit. on p. 44).

[PLW02] L. L. Pipino, Y. W. Lee, R. Y. Wang. “Data quality assessment.” In: *Communications of the ACM* 45.4 (2002), pp. 211–218 (cit. on p. 18).

[Ped+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. “Scikit-learn: Machine Learning in Python.” In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 28, 38, 72).

[Pyl99] D. Pyle. *Data preparation for data mining*. Vol. 1. morgan kaufmann, 1999 (cit. on pp. 12, 21, 23–25, 47).

[Qui86] J. R. Quinlan. “Induction of decision trees.” In: *Machine learning* 1.1 (1986), pp. 81–106 (cit. on pp. 14, 31, 35, 40, 42, 43, 45).

[RD00] E. Rahm, H. H. Do. “Data cleaning: Problems and current approaches.” In: *IEEE Data Eng. Bull.* 23.4 (2000), pp. 3–13 (cit. on p. 12).

[RS04] L. E. Raileanu, K. Stoffel. “Theoretical comparison between the gini index and information gain criteria.” In: *Annals of Mathematics and Artificial Intelligence* 41.1 (2004), pp. 77–93 (cit. on p. 37).

[SGM14] K. Slavakis, G. B. Giannakis, G. Mateos. “Modeling and optimization for big data analytics:(statistical) learning tools for our era of data deluge.” In: *IEEE Signal Processing Magazine* 31.5 (2014), pp. 18–31 (cit. on p. 11).

[SL01] A. Sun, E.-P. Lim. “Hierarchical text classification and evaluation.” In: *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE. 2001, pp. 521–528 (cit. on p. 33).

[SL09] M. Sokolova, G. Lapalme. “A systematic analysis of performance measures for classification tasks.” In: *Information Processing & Management* 45.4 (2009), pp. 427–437 (cit. on pp. 32, 33).

[SO13] R. Schutt, C. O’Neil. *Doing data science: Straight talk from the frontline*. " O’Reilly Media, Inc.", 2013, pp. 41–43 (cit. on pp. 11, 22–24, 30).

[SV06] V. Sessions, M. Valtorta. “The Effects of Data Quality on Machine Learning Algorithms.” In: *ICIQ* 6 (2006), pp. 485–498 (cit. on p. 46).

[Seb+00] M Sebbanü, R NockO, J Chauchat, R Rakotomalala. “Impact of learning set quality and size on decision tree performances.” In: *IJCSS* 1.1 (2000), p. 85 (cit. on p. 38).

[Seb02] F. Sebastiani. “Machine learning in automated text categorization.” In: *ACM computing surveys (CSUR)* 34.1 (2002), pp. 1–47 (cit. on pp. 21, 25, 29).

[Tan+06] P.-N. Tan et al. *Introduction to data mining*. Pearson Education India, 2006, pp. 158–172 (cit. on p. 37).

[Ter+15] I. Terrizzano, P. M. Schwarz, M. Roth, J. E. Colino. “Data Wrangling: The Challenging Yourney from the Wild to the Lake.” In: *CIDR*. 2015 (cit. on pp. 12, 41).

[VZK16] A. G. Villanueva Zacarías, C. Kiefer. “Relevance of the two adjusting screws in data analytics: data quality and optimization of algorithms.” Institute for Parallel and Distributed Systems, University of Stuttgart, Stuttgart. 2016 (cit. on pp. 13, 14, 49, 54).

[WS96] R. Y. Wang, D. M. Strong. “Beyond accuracy: What data quality means to data consumers.” In: *Journal of management information systems* 12.4 (1996), pp. 5–33 (cit. on pp. 18, 19).

[WSF95] R. Y. Wang, V. C. Storey, C. P. Firth. “A framework for analysis of data quality research.” In: *IEEE transactions on knowledge and data engineering* 7.4 (1995), pp. 623–640 (cit. on p. 17).

[Zha16] Z. Zhang. “Missing data imputation: focusing on single imputation.” In: *Annals of translational medicine* 4.1 (2016) (cit. on p. 26).

[jbl] jblackburne. *Categorical splits for tree-based learners*. URL: https://github.com/scikit-learn/scikit-learn/pull/4899/ (cit. on p. 66).

All links were last followed on July 26, 2017.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature