12-2013

Knowledge Extraction from Work Instructions through Text Processing and Analysis

Abhiram Koneru
Clemson University, abhirak@g.clemson.edu

KNOWLEDGE EXTRACTION FROM WORK INSTRUCTIONS THROUGH TEXT PROCESSING AND ANALYSIS

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Mechanical Engineering

---

by
Abhiram Koneru
December 2013

Accepted by:
Dr. Gregory M. Mocko, Committee Chair
Dr. Lonny L. Thompson
Dr. Mary E. Kurz

ABSTRACT

The objective of this thesis is to design, develop and implement an automated approach to support processing of historical assembly data to extract useful knowledge about assembly instructions and time studies to facilitate the development of decision support systems, for a large automotive original equipment manufacturer (OEM). At a conceptual level, this research establishes a framework for sustainable and scalable approach to extract knowledge from big data using techniques from Natural Language Processing (NLP) and Machine Learning (ML).

Process sheets are text documents that contain detailed instructions to assemble a portion of the vehicle, specification of parts and tools to be used, and time study. To maintain consistency in the authorship process, assembly process sheets are required to be written in a standardized structure using controlled language. To realize this goal, 567 work instructions from 236 process sheets are parsed using Stanford parser using Natural Language Toolkit (NLTK) as a platform and a standard vocabulary consisting of 31 verbs is formed.

Time study is the process of estimating assembly times from a predetermined motion time system, known as MTM, based on factors such as the activity performed by the associate, difficulty in assembling, parts and tools used, distance covered. The MTM compromises of a set of tables, constructed through statistical analysis and best-suited for batch production. These MTM tables are suggested based on the activity described in the work instruction text. The process of performing time studies for the process sheets is

time consuming, labor intensive and error-prone. A set of (IF <Verb> AND <object type> THEN <MTM table>) rules are developed, by analyzing 1019 time study steps from 236 process sheets, that guide the user to an appropriate MTM table. These rules are computationally generated by a decision tree algorithm, J48, in WEKA, a machine learning software package.

A decision support tool is developed to enable testing of the MTM mapping rules. The tool demonstrates how NLP techniques can be used to read work instructions authored in free-form text and provides MTM table suggestions to the planner. The accuracy of the MTM mapping rules is found to be 84.6%.

DEDICATION

To my parents, Sarada and Gandhi. For their unconditional love towards me, I will be forever grateful.

ACKNOWLEDGMENTS

First and foremost I would like to express my deepest gratitude to my advisor, Dr. Gregory Mocko, for his continual support and guidance throughout my time at Clemson. I wish to thank BMW for funding this research and Dr. Killian Funk for his valuable suggestions on the project. I would also like to thank Dr. Lonny Thompson and Dr. Mary Beth Kurz for serving on my committee. A special thanks to the faculty and members of the CEDAR group for their feedback and encouragement.

I would like to take this opportunity to acknowledge all those who have contributed in any way, shape or form to the completion of this thesis. A special mention goes out to the people who contribute to Stack Overflow.

Last but not least, I would like to thank all my friends without whom this thesis would have definitely finished earlier.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Page

CHAPTER ONE: MOTIVATION AND RESEARCH OBJECTIVES

The objective of this thesis is to design and implement an automated approach to support processing of historical assembly data. Specifically, this thesis aims to extract useful knowledge about assembly instructions and time studies to facilitate the development of decision support systems, for a large automotive original equipment manufacturer (OEM). This will reduce the cognitive load on the planner by providing decision support during the generation of assembly time estimates. This is achieved by employing the tools and techniques from Natural Language Processing (NLP), Data Mining (DM) and Machine Learning (ML).

Assembly process sheets or process sheets are documents that contain detailed steps, known as work instructions, to assemble a portion of the vehicle, specification of parts and tools to be used, and time study. The consistency in the process sheets can be maintained by standardizing the authorship process through the use of a standardized structure and controlled language. To develop a controlled vocabulary, an automated approach to extract information is required.

Time estimates for each activity described in the process sheets is carried out to perform line balancing. The time estimates are obtained from a pre-determined motion time system containing tables describing various activities. Assigning assembly time estimates is an arduous task dealing with ambiguity. By providing decision support (directing the planner to an appropriate table in the time standards) and automating the process of assigning assembly times estimates, the user effort can be reduced.

1.1 Motivation and Research Objectives

This section provides a brief overview of the research objectives and the issues that are being addressed. A detailed description and the outcomes of each objective will be presented in further chapters.

1.1.1 Research Objective One

The first research objective is to establish an automated approach to extract information from assembly process sheets written using unrestricted grammar and vocabulary [1–4]. The framework processes the unstructured assembly instructions and captures knowledge to develop a controlled vocabulary of verbs to aid in the standardization of process sheet authorship.

An assembly process sheet includes a complete set of instructions describing the sequence of operations to be performed. Authoring assembly process sheets is a labor intensive process and prone to possible human errors and ambiguity. Currently the process sheets are authored without any restriction on grammar, structure and controlled language. Also the level of detail in assembly instructions greatly varies based on the planner authoring the process sheet. This non-uniformity in authorship between planners leads to inconsistency in process sheets. To address this problem, Peterson[4] has proposed a system to author process sheets using standardized structure and controlled language. The standard vocabulary for the controlled language was developed from data acquired from a sample set of existing process sheets. The individual process sheets have been analyzed and the required information was extracted. This process was performed manually and therefore is a time consuming and error-prone process. Also, manual

extraction of information is not suitable when dealing with many process sheets. The purpose of first research objective stems directly from trying to automate the process of knowledge extraction. The system must be quick, capable of analyzing large amounts of data, and flexible to accommodate different formats in authorship while requiring less effort from a user.

1.1.2 Research Objective Two

The second research objective is to develop decision support system using machine learning to aid the planner in estimating assembly times for the work instructions authored in the process sheets.

The process of standardizing work instruction authoring brings about a viable opportunity to estimate assembly times. Renu [5] has explored this area and developed decision support tools to reduce the effort expended by planner during assembly line planning. The assembly time for work instructions is estimated based on the activity to be performed by the associate, from a predetermined motion time system called MTM [6]. Large automotive manufacturers use adapted versions of the MTM for time estimation. The MTM compromises a set of tables, constructed through statistical analysis of historical data. The planner is provided suggestions regarding the MTM table based on rules developed by manually analyzing time studies from existing process sheets. The data analyzed to generate these rules can be overwhelming and be continuously expanding. Manual generation of the rules could lead to loss of information that is not explicit. Also only a small sample set of process sheets were analyzed for generation of rules. The second research objective addresses this issue by developing a decision

support system to automatically and computationally form rules, to assist planners in assembly time estimation, with the support of machine learning algorithms and data analysis. The method must be able to process new information added on a regular basis and generate knowledge for decision support that is reliable.

1.1.3 Research Objective Three

The third goal of this research is to develop and implement a decision support tool to enable testing of the MTM mapping rules that are generated from research objective two. The tool is provided with a GUI to demonstrate how NLP techniques can be used to read work instructions, written in free-form text, and provide MTM table suggestions to the planner.

The tools developed to address the first two research objectives are integrated to develop a decision support tool. To standardize the authorship of process sheets Peterson [4] used text element structures in the controlled language. This system restricts the planner's input and the planners cannot freely author process description. A system is proposed that allows the planners to author work instructions in free form text.

1.2 Research Objectives Overview

The first research objective aims to develop a system capable of extracting information from thousands of process sheets.. The second research objective aims to develop a decision support system to automatically form rules that aid the planner in assembly time estimation. The outcome of the first two research objectives is to transform unstructured data into useful knowledge that can be utilized to develop tools to

better the processes in domains handling large amounts of data. The third research objective presents the development and implementation of a tool to test and validate the knowledge generated from research objective two. Figure 1.1 illustrates the framework to extract knowledge from unstructured data. The three research objectives are outlined.

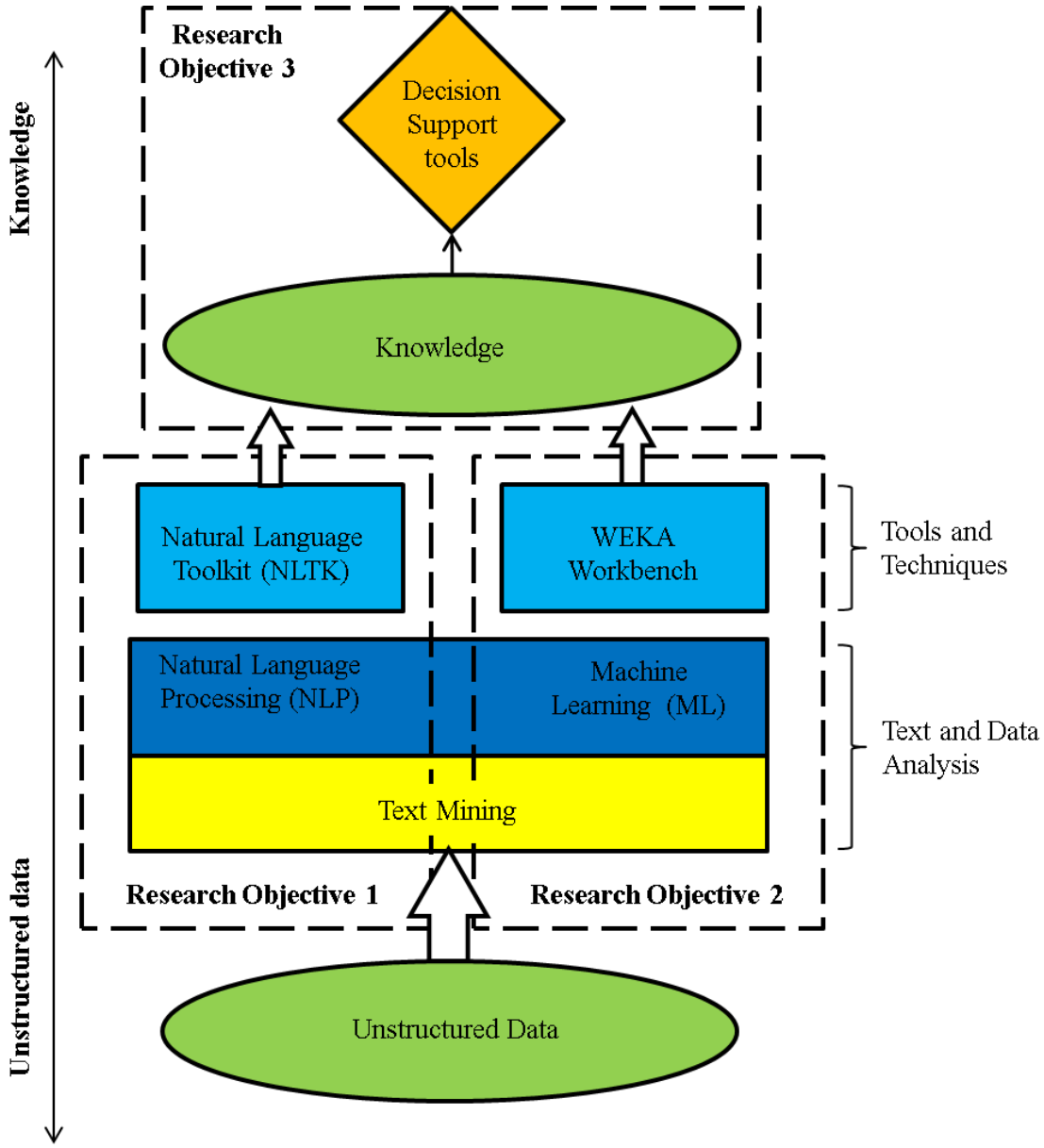

Figure 1.1: Framework to extract knowledge from unstructured data

1.3 Thesis Outline

Error! Reference source not found.A summary of the thesis is included in **Error! Reference source not found.**.

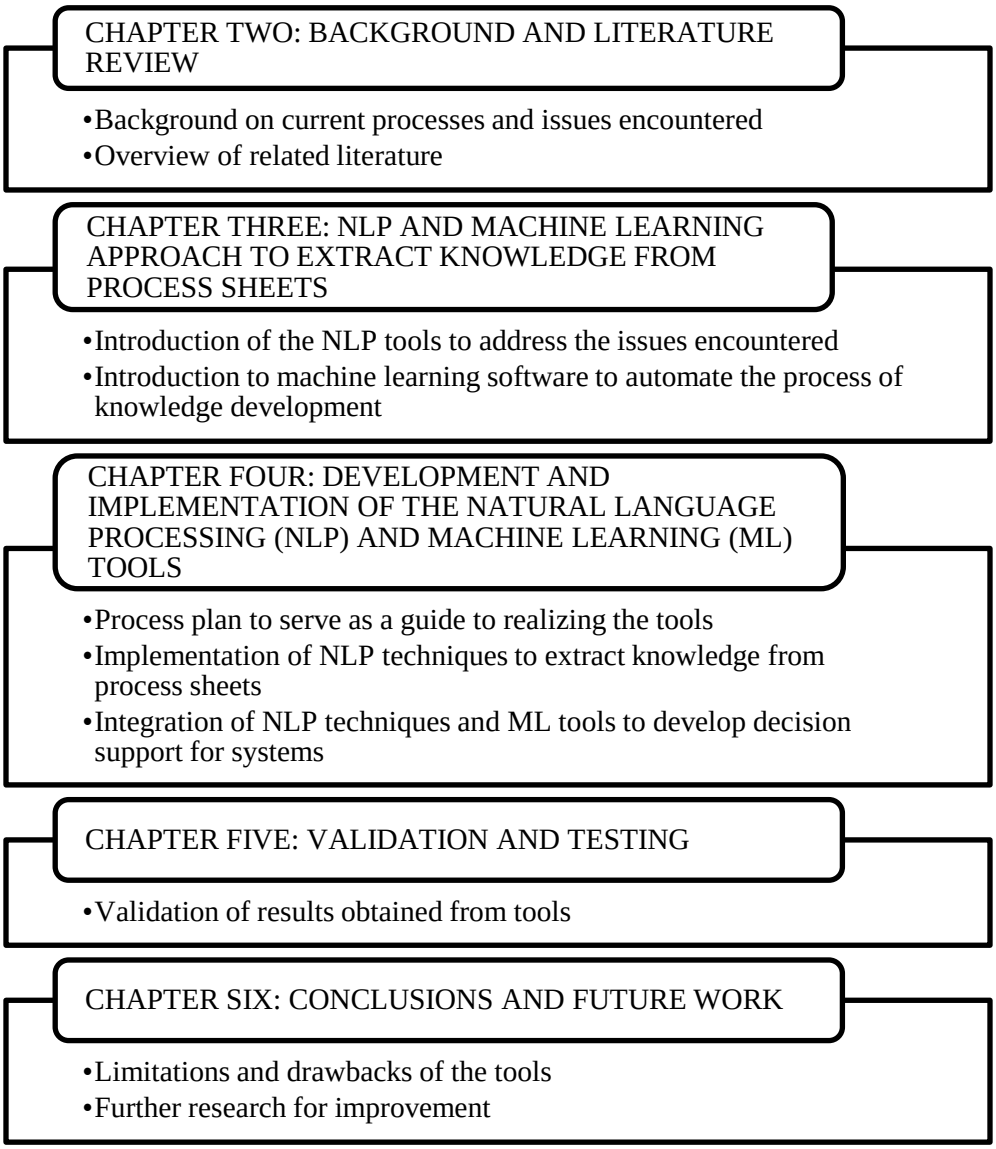

Figure 1.2: Thesis outline

The necessary background and introduction to the current work and processes is included in Chapter Two. Current literature in related research fields is reviewed to better

understand the problems at hand. Best possible methods to address these issues are highlighted.

Chapter Three introduces the NLP tools that are used to perform the necessary operations to extract knowledge from the process sheets. A brief description of WEKA, a machine learning software, is provided. WEKA is used to automate the process of developing knowledge for the decision support tools. A deatiled discussion on each of the tools and the tasks involved is provided and how these tools are inegrated to achieve the desired result.

The development of the tools to realize the two research objectives is presented in Chapter Four. A process plan detailing how each phase serves to solve the problems identfied is presented. The chapter concludes with the implementation of the tools that are developed. A GUI is developed that integrates the tools to direct the planner to an appropriate MTM table based on the work instruction entered.

Chapter Five deals with testing and validation the tools developed in chapter four. The results obtained from the tools are checked against existing data and decision support tools to validate the accuracy of the tool.

The closure for the thesis is presented in Chapter Six with a summary of the tools developed to address the research objectives and the broader reach of the work. This section identifies certain limitations and drawbacks of the developed tools and provides a brief discussion on future work.

CHAPTER TWO: BACKGROUND AND LITERATURE REVIEW

This chapter provides the necessary background regarding the current assembly process in automotive industry. The literature of relevant topics is reviewed to determine the preferred approach to support the development of tools to address the issues outlined.

2.1 Current trends in automotive industry

The current automotive market is highly competitive, characterized by intense competition and increasing demands for innovative and customer-oriented products. Recent automotive manufacturing trend has seen a shift from mass production to a JIT (Just-In-Time) production to meet the demands of a more wide and diverse customer base [7]. The customer requirement for product variety needs flexible and intelligent manufacturing systems to be integrated to the current manufacturing processes to achieve low-cost of production, high product variety, high productivity and short delivery times [8].

The application of Artificial Intelligence (AI) in automotive industry is seen in a wide variety of domains ranging from design, manufacturing, and vehicle functionalities [3]. Recent advances in CAD and Artificial Intelligence (AI) have further augmented the manufacturing process by presenting opportunities to perform assembly planning by functional precedence and connectivity relationships [9].

The integration of AI systems in manufacturing processes in automobile industries has seen development of applications in areas such as machine translation of

process sheets, robotic alternative to manual operation, and ergonomic analysis of assembly process [3,10,11].

Abdullah et al. [12] point out that almost half of all production work comprises of assembly process and assembly costs amount to 50% of the entire production cost. There is immense scope of cost cutting, workforce reduction and effective management in the assembly process of automotive industry. Therefore there has been much research in development of tools to improve the assembly planning process.

Rychtyckyj [1–3,11,13] discussed the development of a knowledge based system, known as Direct Labor Management System (DLMS), that supports and manages data pertaining to all stages of the assembly planning process [1–3,11,13]. Process sheets are formal documents that contain detailed instructions, called work instruction, to build a portion of a vehicle. The DLMS allows planners to create process sheets using a restricted vocabulary that are machine readable. The system makes use of AI to check for any conflict among the instructions or ergonomic issues that can occur. The work instructions are mapped onto MODAPTS, a predetermined time standard, to estimate the time required to complete the activities. Furthermore, the system also provides the capability of translating the process sheet to other languages to support activities in other assembly plants that do not use the same language used for writing process sheets as their main language [1,3,13]. Further discussion on assembly planning is provided in the following section.

2.1.1 Assembly Process Planning

In an automotive manufacturing industry, the product is carried through a succession of workstations on moving flow line called an assembly line [12]. The complete assembly of the vehicle is performed sequentially on this assembly line by associates allocated to each workstation. This sequence of steps to complete the assembly of a product based on the connectivity relationship of the parts or subassemblies is known as assembly planning [12]. The process of assembly planning is a critical activity in the final production of a vehicle. The cost of assembling a product can be minimized by optimal process planning [14]. A schematic representation of an assembly line is shown in Figure 2.1. The base part moves from work station 1 to work station 4 along the work flow. At each work station, a value adding task is performed and the final product is obtained at the end of the assembly line.

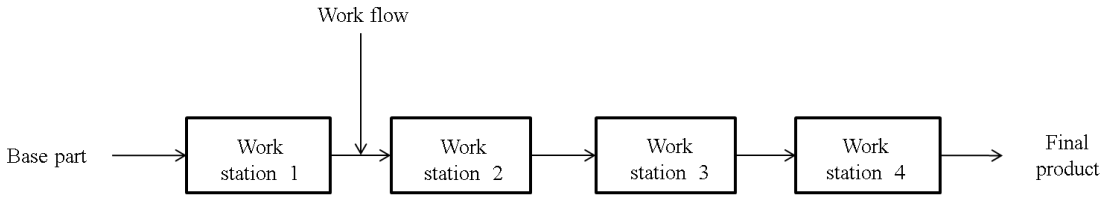

Figure 2.1: Schematic representation of an assembly line

The result of the assembly process planning is the assembly process sheet. Process sheets convey the vehicle assembly information from the process planning department to the shop floor [1–3]. Process sheets, or TVGs, are text documents that contain detailed instructions required to assemble a portion of the vehicle [1]. In addition, process sheets also include information regarding the vehicle model, specifications of parts and tools to be used, quality checks, and assembly time estimates. The complete

assembly of a single vehicle requires about three thousand to five thousand process sheets [15]. These process sheets are allocated to the respective workstations as a reference document for the associates during assembly. Figure 2.2 shows a sample assembly process sheet.

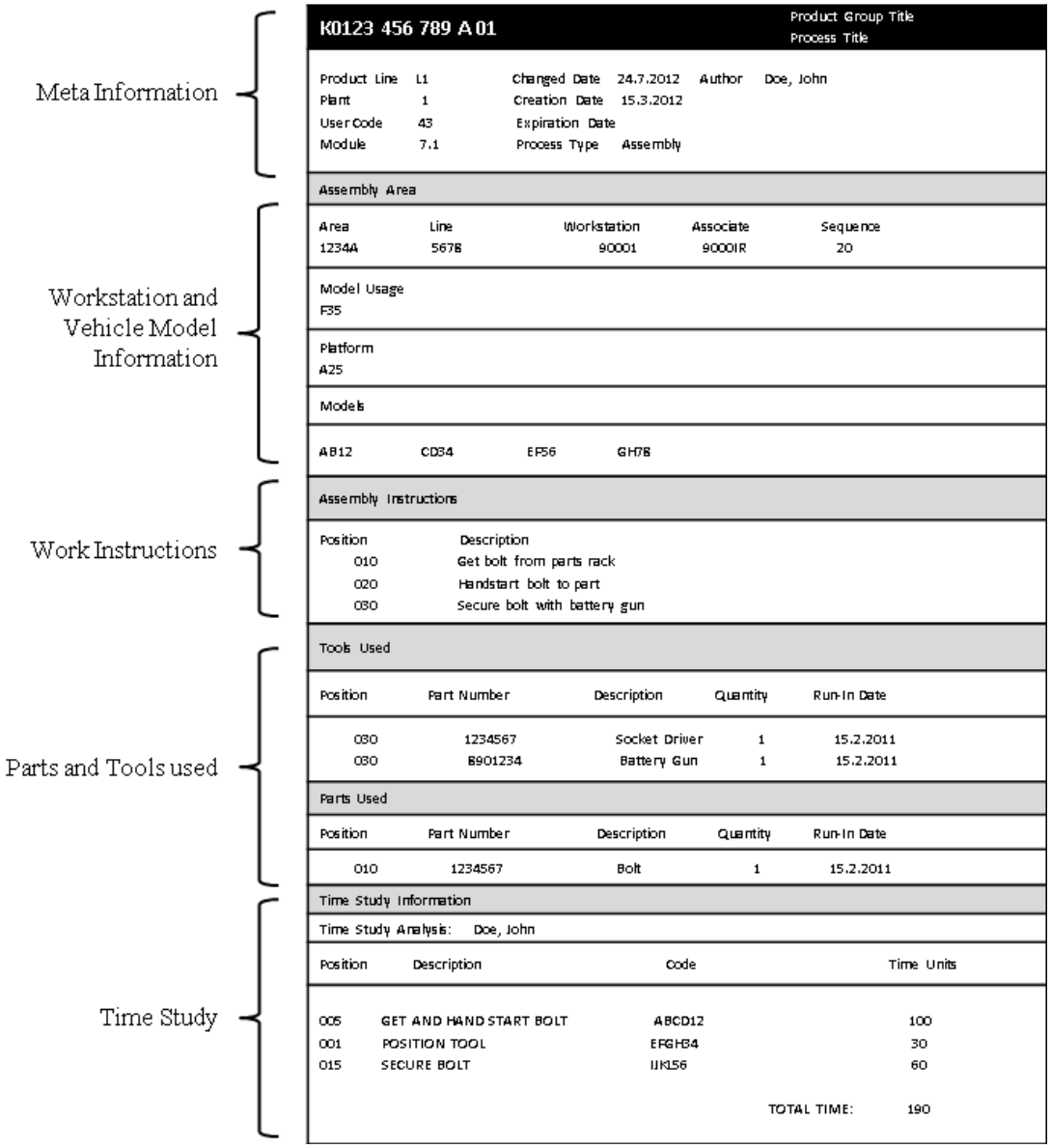

Figure 2.2: Sample process sheet

To realize the automation of process planning, to support machine translation, assembly time estimation, and perform ergonomic analysis standardization of information contained within the process sheet is required. Peterson [4] developed a knowledge database system to support standardization the process of creating process sheets, for a global automobile manufacturer. Similar to Ford's DLMS, Peterson [4] standardized the process of authoring process sheets through controlled language and vocabularies. This restricted use of language and syntax helps maintain consistency of the structure and also the level of detail in the work instructions. The use of controlled language and its benefits has been well documented in Ford's Direct Labor Management System (DLMS) [1–3]. To develop the standard vocabulary, Peterson and colleagues analyzed a large number of existing process sheets to extract the most frequently used action verbs to generate a reduced standard list of verbs. This process can be automated to reduce the effort and time consumed.

2.1.2 Assembly time estimation

The assembly planning activity is performed prior to the start of the vehicle production [15]. This is crucial to optimize the layout of the assembly line, work allocation, and efficient management of personnel. In the conventional method, a process engineer records the time taken by personnel to complete the task. But this procedure is time consuming and burdensome. Also when the personnel is aware that he is being observed and evaluated, his performance can suffer and lead to miscalculation of the time estimates [1,6]. To eliminate these issues, predetermined motion time systems have been developed. Boothroyd and Dewhurst, MTM, and MODAPTS are few frequently used

time standards [1,6,16]. The assembly time estimates obtained from the process sheets allows the users to predict the total time taken to build the vehicle. This time estimation for each assembly process allows planners to perform optimal line balancing, effectively procure material, reduce costs incurred by carrying large inventories, and schedule for shipment to customers, minimize storage costs.

Time study is the process of estimating the time required to carry out a certain task. Maynard et al. [6] have developed a predetermined motion time system, known as Methods-Time Measurement (MTM). The MTM comprise of a set of tables, constructed through statistical analysis, that contain specific codes and time units for all value and non-value adding manual activities that are performed during assembly process. Large automotive manufacturers used adapted versions of the MTM as per their requirement. Based on factors such as the task performed by the associate, difficulty in assembling, parts and tools used, and distance covered the planners assigns each work instruction a time estimate by traversing through the tables. Each MTM table consists of various options that the planner has to narrow down in order to select one code and corresponding time units that relates to the activity described in the work instruction. A sample table from the MTM is shown in Table 2.1.

Table 2.1: Sample MTM Table

GET AND PLACE			**Distance range in cm**	**<20**	**>20 < 50**	**>50**
Weight	Conditions	Place accuracy	Code	1	2	3
< 1 kg	Easy	Approximate	AA	20	35	50
		Loose	AB	30	45	60
		Tight	AC	40	55	70
	Difficult	Approximate	AD	20	45	60
		Loose	AE	30	55	70
		Tight	AF	40	65	80
> 1 kg < 8 kg		Approximate	AH	25	45	55
		Loose	AJ	35	55	65
		Tight	AK	45	65	75
> 8 kg < 22 kg		Approximate	AL	80	105	115
		Loose	AM	95	120	130
		Tight	AN	120	145	160

Manually performing time studies for all the process sheets involved in the complete assembly of an automobile is a tedious process. There is a need to automate the process of estimating the assembly time. This need has been addressed by Ford's Direct Labor Management System (DLMS). Ford's DLMS uses standard language, known as SLANG to construct all work instructions [1,2]. By standardizing the work instructions through standard vocabulary, the system is capable of reading and interpreting each work instruction and assign time estimates. As mentioned earlier, similar work has been carried out by Peterson [4] and Renu [5].

Peterson's [4] model to author process sheets using controlled language is leveraged by Renu [5] to assign time estimates for each work instruction. Peterson's model, to standardize the process of writing work instructions and Renu's tool, for assembly time estimation are discussed in detail in the following chapter. The basic elements that constitute a work instruction are Verb and Object. Renu's decision support tool gathers the verb and objects information from a work instruction and directs the user to the MTM table based on a rule set developed from historical data. But these rules have been developed manually from a small set of data. For effective utilization of the decision support tool, the generation of rules must be automated and large amounts of data need to be analyzed. The proposed research aims to bridge the gaps that are encountered. By automating the process of extracting information from unstructured data and generating rules through machine learning, this research further augments the work by Peterson and Renu.

2.2 Text Mining

Exploitation of existing knowledge and knowledge acquisition are a key to compete at a global level, in any industry. Text mining or text data mining is the process of extracting useful knowledge from unstructured data. Text mining is a multidisciplinary field, involving information retrieval, knowledge extraction, machine learning and data mining [17].

Recent studies indicate that 80% of the data in an industry is stored in textual format [17]. Though freely available, this data is not availed at the right time and in the right manner and hence it is not utilized to its full potential. The reason is due the

overwhelming nature of the data collected. This problem has been aptly coined as “rich data, poor information” [18]. Large amount of data is accumulated from various sources but no means to filter it into knowledge that can aid in decision making and enhance productivity. It is evident that the availability of information and the ability to exchange and process it is the key to success in global market [19]. Knowledge provides the means to solve problems and predicting future market. Efficient knowledge acquisition necessitates intelligent systems that are capable of gathering large amounts of information and deduce patterns that are implicit.

2.2.1 Natural Language Processing (NLP)

As the interaction with computational machines is ever increasing, the need to reduce the gap between man and machine is predominant. Researchers have observed very early on that, a machine that can analyze and respond using natural language rather than a machine language is much more effective and easier to interact with, from a user perspective. This ideology has culminated into the research and development of systems capable of processing natural language. Natural language processing (NLP) is the ability of a system to understand, manipulate and communicate using natural language. The field of NLP brings together tools and techniques from a number of disciplines, namely, Artificial Intelligence (AI), linguistics, and computer science [20–22].

Research efforts into NLP have been ongoing for several decades and the roots trace back to the early 1950’s [20,22–24]. Early application of NLP was seen in automatic machine translation of phrases from one language to the other. This automation

is a consequence of Turing's model of algorithmic computation, considered to be the foundation of modern computer science [20]. NLP involves design and implementation of computer systems that can effectively read, understand and communicate in human languages. The applications of NLP extend from speech recognition to cross-language information retrieval.

The ability of Natural Language Processing (NLP) is exploited within the scope of this research to extract information from assembly process sheets. Process sheets are a classic example of a technical document written using natural language. NLP tools and techniques are best suited to extract information from large unstructured process sheets and transformed into knowledge to provide decision support within the manufacturing domain. Further discussion on NLP tools and techniques used to extract knowledge from process sheets are presented in Chapter Three.

2.3 Chapter Summary

This chapter provides an overview of the assembly process planning in the automotive industry. The chapter presents how standardization of process sheets will aid in automation of assembly time estimation process. The chapter concludes by describing how Information Retrieval (IR) through NLP and Machine Learning (ML) can be used to automatically extract information and develop knowledge for decision support tools.

CHAPTER THREE: NLP AND MACHINE LEARNING APPROACH TO EXTRACT KNOWLEDGE FROM PROCESS SHEETS

This chapter lays the framework to realize the research objectives and forms the core of this thesis. To put this research in context, standardization of process sheets and decision support tools to estimate assembly time are discussed in detail. This discussion serves as an introduction to address the gaps identified.

To standardize the process of authoring process sheets, Peterson analyzed the process sheets for the vocabulary used and the sentence structure used within work instruction text. A process sheets contains work instruction, tools and parts used, time studies, and other meta-information that range from details on allotment of the process sheet to a certain assembly line and location to the diagrammatic representation of the approximate location of the part in the vehicle. The information required to develop a standard vocabulary and controlled language is contained in the section consisting assembly work instructions.

A total of 236 process sheets have been analyzed for this purpose. Each process sheet contains multiple assembly instructions, averaging about three to four instructions. A total of 697 assembly instructions are gathered from the 236 process sheets. A list of frequently used verbs that describe a unique action are gathered. Thus by identifying the verbs used to describe the work instruction actions, a preliminary list of verbs to be used in controlled language is developed. The standard vocabulary for verbs consists of 31 unique actions, each describing a certain action performed by the associate during the assembly process. The sentence structure is developed, based on the existing work

instructions, that is minimalistic while sufficient to clearly write a work instruction. The standard sentence structure of the work instruction is shown in Figure 3.1.

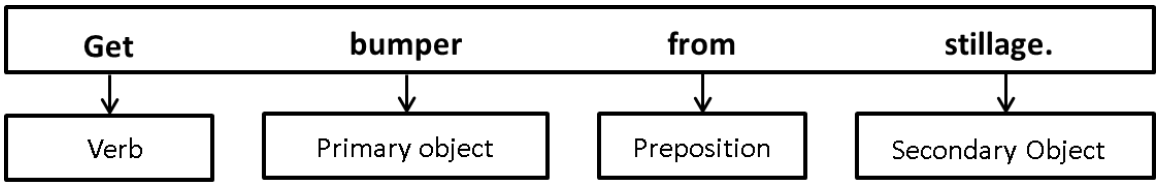

Figure 3.1: Sentence structure of work instructions

Peterson [4] developed a process sheet authorship tool based on the controlled language and standard vocabulary. The tool allows the planner to create work instructions for a process sheet using a standard structure format with the help of drop down menus and free form entry fields.

The process of developing a standard vocabulary required manual extraction of verb from each work instruction. This time consuming activity can be simplified by the NLP approach. NLP tools can be used to automatically read each work instruction from the process sheets and find the verb that describes the primary action of the assembly step. This method will also have the advantage of processing a large number of process sheets in a significantly shorter time since it is a computational method.

In addition to extracting the verb, the primary object from each work instruction is also extracted. This data is required to generate a tool (Object type classifier) which will assist in the development of the decision support to estimate assembly time. Further discussion on the object type classifier will be provided in further chapters.

Each process sheet has several work instructions that are carried out by an associate on the shop floor for that particular assembly activity. The process sheet also

contains a set of time study steps that list out all the time defining actions that will occur during the assembly activity. Each time study step provides information on the time required to carry out a certain action. The combined time of these time study steps provides an estimate of the time required to complete all the assembly instructions described within the process sheet. These time estimates are essential in any manufacturing industry to perform optimal line balancing.

Currently the time studies for each process sheet are written by a planner by observing an associate performing the assembly activity and recording the action steps. The corresponding MTM tables are referred and each activity is denoted a code and time units based on certain parameters. This process is labor intensive and time taking [1].

For example consider the work instruction – 'Get and place bumper on car body'. Presume the bumper weighs 3 kg and is place tightly onto the car body. The associate moves 25 cm in order to pick up the part and place. The steps to determine a suitable MTM code and time units for the work instruction is as follows.

Step 1: Select the appropriate MTM table based on the activity described. In this case, the work instruction statement describes picking up a part and placing it on a sub-assembly. Therefore, the MTM table 'Get and Place' is selected.

Step 2: The first column in the MTM table describes the weight of the part. Since the bumper weighs 3 kg, the rows corresponding to the 'Weight' parameter (> 1 kg < 8kg) are chosen.

Step 3: The type of fit is described as tight. Therefore in the 'Place accuracy' column, the option 'Tight' is chosen. The selection of 'Weight' and 'Place accuracy' parameters points to the MTM code 'AK'.

Step 4: The user is provided with three choices for the time units. Since the distance moved by the associate is 25 cm, the second column in the 'Distance range' is selected and the corresponding time unit of 65TMU is obtained. MTM tables contain time units in TMU (1 TMU = 0.036s).

The illustration to the example is shown in **Error! Reference source not found.**. he parameters and the time unit are highlighted.

Table 3.1: Obtaining time estimates for sample work instruction statement

GET AND PLACE			**Distance range in cm**	**<20**	**>20 < 50**	**>50**
Weight	**Conditions**	**Place accuracy**	**Code**	**1**	**2**	**3**
< 1 kg	Easy	Approximate	AA	20	35	50
		Loose	AB	30	45	60
		Tight	AC	40	55	70
	Difficult	Approximate	AD	20	45	60
		Loose	AE	30	55	70
		Tight	AF	40	65	80
> 1 kg < 8 kg		Approximate	AH	25	45	55
		Loose	AJ	35	55	65
		Tight	AK	45	65	75
> 8 kg < 22 kg		Approximate	AL	80	105	115
		Loose	AM	95	120	130
		Tight	AN	120	145	160

The process of selecting a MTM table based on the activity described in the work instruction and determining a single MTM code based on the parameters is a tedious process. To automate the process of performing time studies, Renu [5] has developed a set of rules that direct the planner to the appropriate MTM table based on the information gathered from the assembly instruction. This automation reduces the cognitive load and repetitive work load on the planner. It is to be noted that these rules only direct the planner to an appropriated MTM table. Work instructions do not contain all the information required to narrow down to a single MTM code and time unit but sufficient information to select a MTM table. Each MTM table has specific set of parameters that drive the planner to single code. Information regarding the parameters is obtained from other information sources such as CAD data, which is not within the scope of this research. The MTM rules are a set of simple IF THEN rules, that utilize the verb and object to determine the table. The MTM rules are in the format shown below.

IF <verb> AND <object> THEN <MTM table>

During the assembly of the vehicle, the associates interact with thousands of objects. This would result in a huge list of rules, which is impractical. To reduce the number and simplify the rules, five object types were created and all the objects belonged to one and only one type. The five object types being – Part, Tool, Consumable, Fixture, and Plant item. Each object is assigned to one object type and this resulted in a simplified rules list as shown below.

IF <verb> AND <object type> THEN <MTM table>

Therefore if the planner chooses the verb ‘Scan’ and an object with type ‘Part’, the tool will direct the planner to the appropriate table, in this case ‘Marking and Documenting’. These rules have been manually developed by analyzing 1019 time study steps from 236 process sheets. The verb, object and the MTM table have been extracted and the instances with highest frequency, derived through statistical analysis, are used to form rules. These rules have a mapping accuracy of 75 %. A sample set of the rules are presented in tabular form in Table 3.2.

Table 3.2: Sample MTM mapping rules

MTM mapping rules	
Action verb	
& Object Type	**MTM Table Name**
Align	
& Fixture	Place
& Plant Item	Place
& Tool	Motion Cycles
Attach	
& Consumable	Working with Adhesives
& Fixture	Get and Place
Clean	
& Consumable	Cleaning

Since manually extracting verb and object from each time study and then assigning an object type for each of the extracted object is a burdensome task, NLP tools can be utilized to process a large set of data in a very short time. Also, the rules when developed manually are subject to human error. This can be avoided by using machine learning algorithms to generate rules. Machine learning algorithms are capable of processing large amounts of data and also bring out the implicit relationships between the data that may not be noticed by a human.

To better understand the underlying process of knowledge extraction, a brief description of the NLP techniques and machine learning tools, that are used to develop the decision support tools, is provided. These NLP techniques and machine learning tools work in tandem to extract selected information from a large dataset and transform the data into resourceful knowledge. The following section talks about Natural Language Processing and few of the techniques within NLP.

3.1 Natural Language Processing (NLP)

The primary intent of NLP is to extract the meaning of text. Text can be a word, statement, paragraph or an entire document depending on the analysis [25]. In process sheets, this text is in the form of sentences. NLP provides tools to perform syntactic and semantic analysis involving text using computational methods. Syntactic analysis is performed to understand structure of the sentence. It involves the part of speech of the words and parse trees [25]. Semantic analysis provides the meaning, which involves the context of the sentence. It provides the relationship between the syntactic elements.

3.1.1 Parsing

Parsing is the process of breaking down text into its components, identifying the part of speech (PoS), outlining the function and syntactic relationship between each component based on the rules of formal grammar and generating a parse tree structure of the text. Essentially parsing pertains only to the process of creating tree structures, but in most cases the entire process is considered parsing. Parsing is preceded by two processes – Tokenizing and Tagging. The first step involves splitting a sentence into single entities

called tokens, by means of user-specified separator. In the second step, tagging, the tokens are assigned a part of speech (PoS) depending on the nature of the token in the sentence. A tree structure is then created based on the grammatical structure of the sentence. The three analyses of the parsing process are shown in Table 3.3.

Table 3.3: Analyses in parsing process

Analysis	**Process**	**Definition**
Lexical	Tokenizing	Breaking down a sentence into single entities, known as tokens.
Syntactic	Tagging	Assigning a part of speech (PoS) tag to each of the tokens.
Syntactic	Parsing	Creating tree structures of the sentence.

The Stanford parser, developed by the Natural Language Processing Group (NLPG) at Stanford University, is a computational implementation of a statistical parser. The Stanford parser analyses the input sentence and constructs a constituent structure that adheres to the syntax [26,27]. The Stanford parser provides Java implementations of probabilistic natural language parsers. In this research, an unlexicalized PCGF (Probabilistic Context Free Grammar) parser is used. The PCGF parser is provides in three different languages apart from English – German, Chinese, and Arabic [26]. The PCFG parser is trained on a large corpus consisting annotated text. Recent studies have showed that unlexicalized parsers have higher accuracy than previously thought [26,28]. Klein and Manning's [26] research has showed that unlexicalized parsers have a high accuracy of 86.31%, almost as high as state-of-the art parsers.

The parser uses the Penn Treebank schema to denote phrasal categories and annotate the text with Part of Speech (PoS) tags. The Penn Treebank is a huge corpus

consisting of syntactically bracketed and PoS tagged texts. A list of twelve syntactic tags and thirty-six Part of Speech (PoS) tags are used within the Penn Treebank and the Stanford parser to syntactically bracket and annotate the text [29]. Table 3.4 and Table 3.5 show a sample of the PoS and syntactic tag set with their description.

Table 3.4: Sample PoS tag set

Tag	Description
NN	Noun
VB	Verb
JJ	Adjective
RB	Adverb
CC	Conjunction
IN	Preposition
CD	Cardinal number

Table 3.5: Sample syntactic tag set

Tag	Description
S	Simple declarative clause
NP	Noun phrase
VP	Verb phrase
PP	Prepositional phrase

The Stanford parser analyzes the input text and provides the user with various outputs – phrase structure trees, typed dependencies, and plain PoS tagged tokens. The parse tree of a sample sentence "*The quick brown fox jumps over the lazy dog.*" is shown below in Figure 3.2

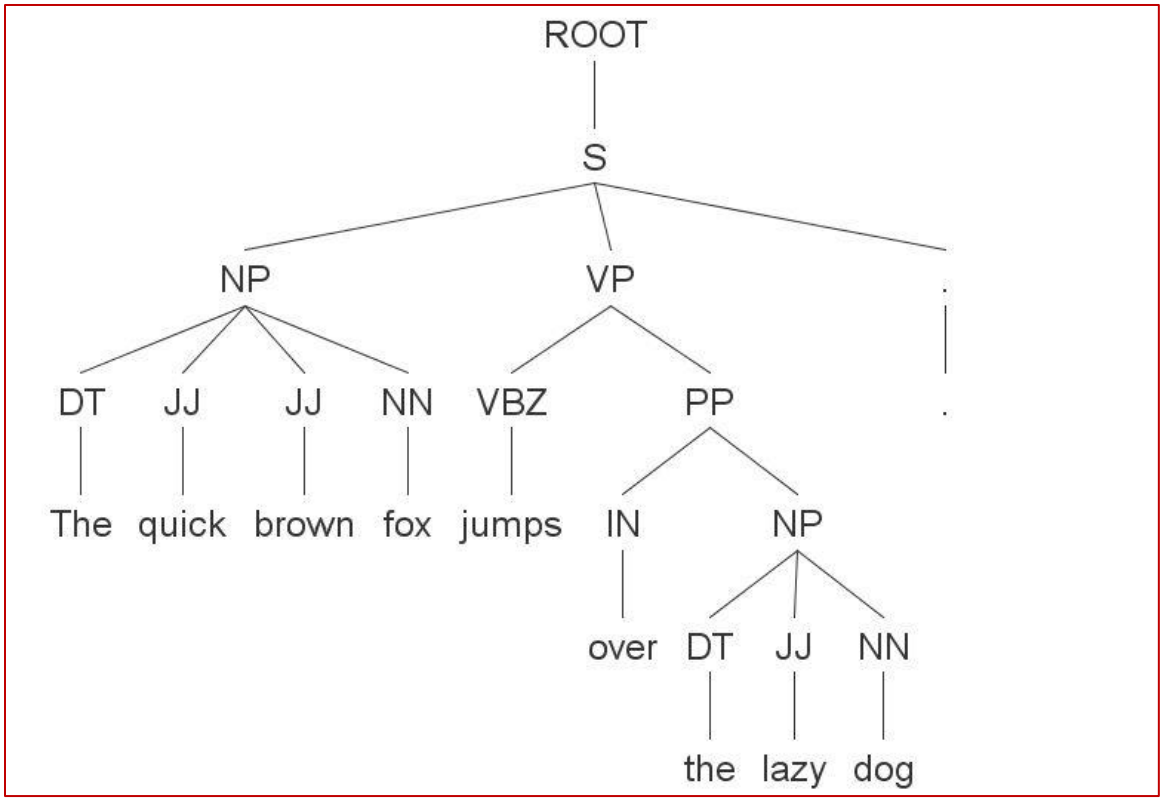

Figure 3.2: Parse tree of sample sentence

To develop the standard vocabulary and decision support for the tools, it is required to establish a method to automatically extract the verb and object list from the existing 236 TVGs. It is observed that in each of the assembly instruction and time study step, the primary action is a verb (VB) and the object, the verb acts on, is a noun (NN). The Stanford parser is leveraged to accurately tag the action verb and object from each assembly instruction and time study step. A program is written in a suitable scripting language to search for the VB and NN tags from each parsed sentence and extract the corresponding tokens into a text file. A detailed discussion provided in the following sections.

3.2 Natural Language Toolkit (NLTK)

The NLTK is a software package for building Python modules to perform linguistic research in Natural Language Processing (NLP). NLTK consists of a set of NLP tools and that provides access to corpora and data manipulation. It provides a suite of text processing libraries for tokenizing, tagging, parsing and classification. Large number of text files can be imported through NLTK, analyzed and presented in a suitable format.

The verb (VB) and noun (NN) tokens relate to the corresponding action verb and objects in an assembly instruction or time study step. Therefore by parsing 236 process sheets, the verb and object from each work instruction and time study can be extracted thus avoiding manual work. The required information can be extracted from the parsed work instructions and time study steps by a python code written using the library of functions available in NLTK. But to process the text, certain amount of pre-processing and editing is required to structure the statements. Further discussion on the required pre-processing is provided below.

Due to the absence of a standard format, many work instructions have been compounded into a single sentence, describing more than one activity to be performed. For the purposes of ease and simplicity, these compound work instructions have been separated into single action steps as shown in Table 3.6. Therefore each step represents only one action to be performed by the associate. This is the first step in simplifying the data for effective information extraction.

Table 3.6: Compound work instructions split into single action work instructions

S.No	Compound work instruction text	S.No	Single action work instruction text
1	Take EMS hanger hook and attach hanger hook to spring damper	1.a	Take EMS hanger hook and
		1.b	attach hanger hook to damper
2	Get kim-wipe from line side and apply isoproponal to wipe	2.a	Get kim-wipe from line side and
		2.b	apply isoproponal to wipe

The Stanford parser requires the text input in a certain format to accurately tag words. The work instructions from the TVGs do not follow a standard structure or grammar; therefore it is crucial to perform certain text-preprocessing for effective parsing. Figure 3.3 and Figure 3.4 illustrate the parsed tree structure for work instruction in upper case and sentence case respectively. It is evident that the parser performs poorly when the sentence is entered in upper case and tags each token as a noun, the default tag. The parser performs better while the text is inputted in sentence case and accurately tags each token with the appropriate tag. Therefore the work instruction text from the TVGs is converted to a standard format with punctuation rules for better text analysis.

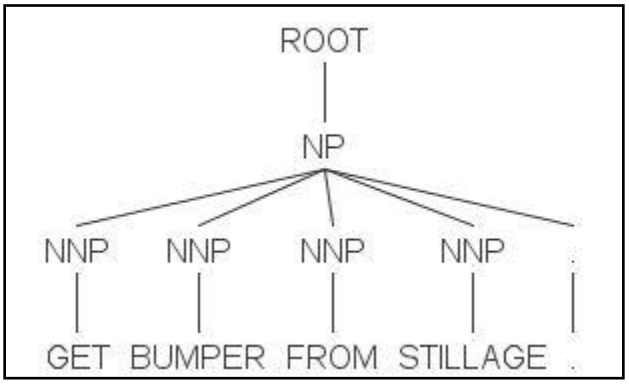

Figure 3.3: Parse tree of sample sentence in upper case

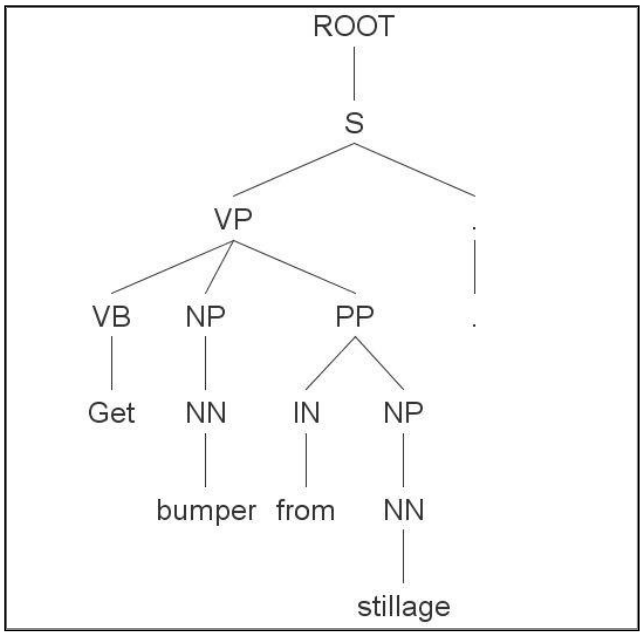

Figure 3.4: Parse tree of sample sentence in sentence case

Parsers are trained on corpuses of hand-parsed and complete sentences and therefore are able to almost accurately tag each token with a part of speech (PoS) [30]. The work instructions in the existing TVGs and written in bullet point grammar. The parser cannot adequately identify all the PoS tags of the tokens in a sentence unless additional information is provided. Figure 3.5 shows the token ‘Align’ is tagged as a noun (NNP), highlighted in red. But in fact the token describes an action to be performed by the associate and hence it is a verb (VB).

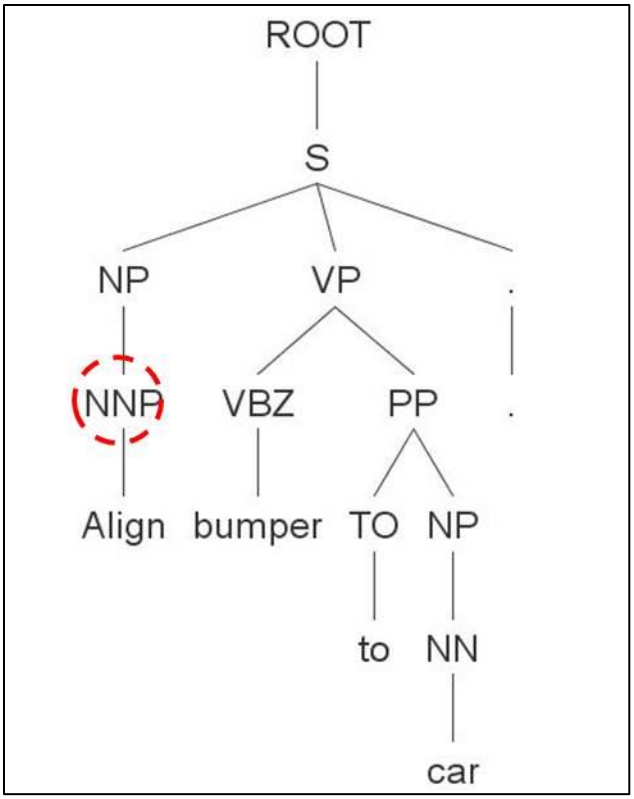

Figure 3.5: Incorrectly tagged WI text

To address this issue, each work instruction is concatenated with the term "The associate must" at the start of the sentence to provide contextual meaning. The edited work instruction is parsed and the token 'Align' is accurately tagged as a verb (VB) by the parser, highlighted in green in Figure 3.6.

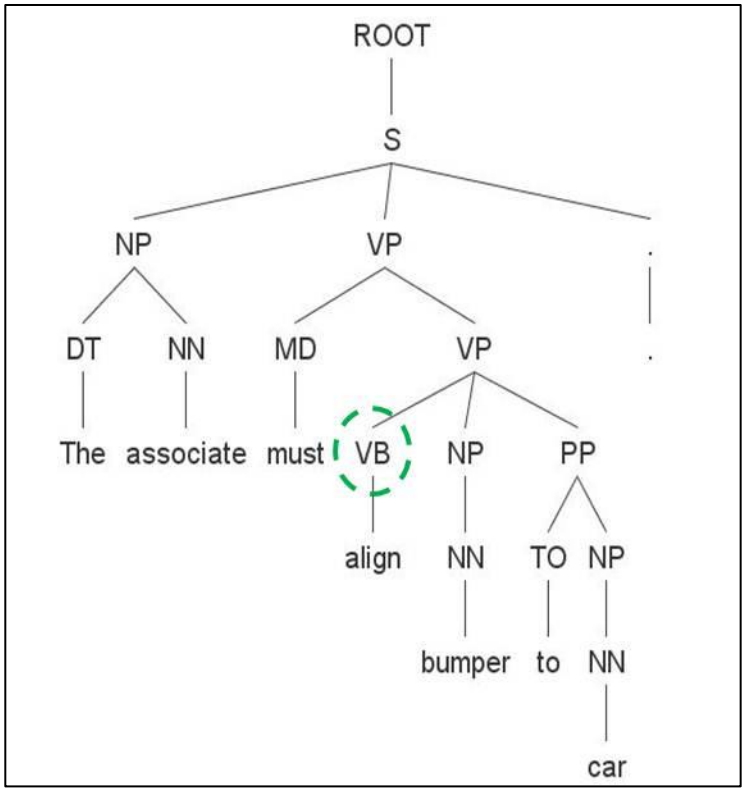

Figure 3.6: Accurately tagged WI text

All the work instructions are edited to a suitable format by simple text pre-processing so as to be accurately parsed. The required data from the process sheets is imported into an excel worksheet, edited and exported to a text file to be parsed. Table 3.7 shows a sample of raw work instruction text that has been edited to obtain accurate PoS tags.

Table 3.7: Sample edited work instruction text

Raw work instruction text	Edited work instruction text
MOVE TO BUMPER STILLAGE.	The associate must move to bumper stillage.
ENSURE BUMPER IS FLUSH WITH FEN DER	The associate must ensure bumper is flush with fender.
Get the correct roof rail from line side.	The associate must get the correct roof rail from line side.

3.3 Text classification

Text classification is the process of categorizing text documents or text files among a set of pre-defined groups [31]. Text classification of data is natural language plays a pivotal role in information retrieval [32]. Text classification goes beyond regular text categorization and document retrieval and finds its application in many real world challenges such as sorting emails, sentiment detection, and search engines [32,33]. In basic text classification, a text document or input is analyzed and then assigned a label that is most appropriate [27]. The classification tasks are generally carried out by machine learning algorithms that can identify certain attributes or features extracted from the input and label the document based on the data the algorithm has been trained on.

A classification process which involves training a classifier model on pre-labeled data is known as supervised learning. Therefore supervised learning requires a training set to learn data properties [34]. The training data consists of text documents that are manually pre-annotated with one or more labels. The feature extractor generates the features and associates them with the relevant labels. The feature-label pair forms the basis for the algorithm to generate the classifier model. The features from the test documents are also extracted and the classifier model and are checked against the feature-label pair and then assigned one or more labels. Figure 3.7 shows the schematic representation of the classification process with supervised learning.

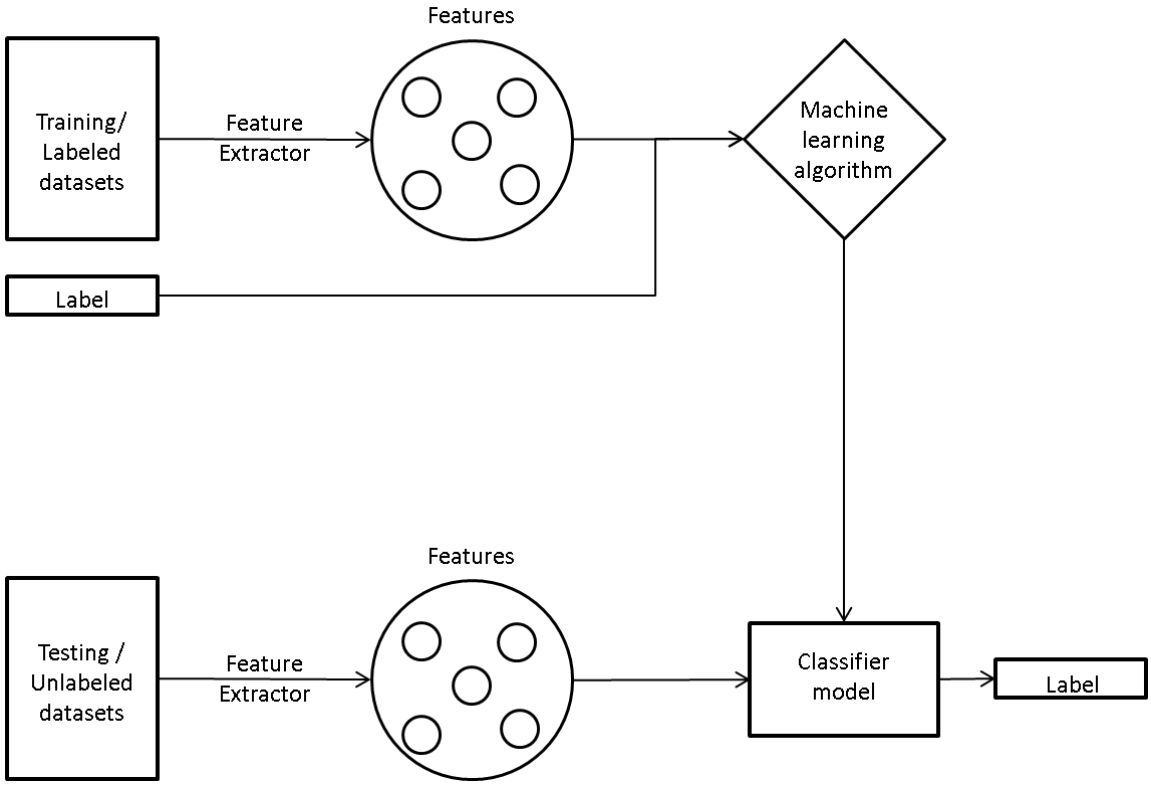

Figure 3.7: Schematic representation of text classification

There are three types of text classification – binary classification, multi-class classification, and multi-label classification. The binary classification involves classifying a given input into either of the two available classes. Classifying a document into only one of many labels is known as a multi-class classification. Essentially a multi-class classification is an extension of binary classification and the same techniques can also be applied to create a multi-class classifier. The third type of classification is the multi-label classification, which involves classify the input into one or more labels. A multi-label classifier can be developed by combining a binary classifier for each label [31].

As discussed previously, each object is assigned one of the five object types that are proposed. The rules are developed in the following format and consist of the elements – verb and object type.

$$\{Verb, Object\ type\} \Rightarrow \{MTM\ table\}$$

Each unique set of verb and object type narrows down the choice of MTM tables. Therefore classification of objects into categories is required to provide a mapping between the standard verb and the MTM table. Text classification is most suitable for this purpose since it is memory efficient, versatile and a large training set is available. An object type classifier is developed using the scikit-learn machine learning library to categorize the objects into their respective classes [34]. The object type classifier is a multi-class classifier that categorizes the objects into one of the five pre-defined object types. The main process blocks of text classification are discussed below in detail.

3.3.1 Training dataset

In supervised learning, a training dataset is initially provided as an external data source to the algorithm. Based on this dataset, the algorithm generates a model which predicts the label for the test input based on the data properties of the training set [35]. The training set for the object classifier is a manually labeled set of 794 objects with their respective object types. The training data is inputted as two arrays: an array of objects of and an array of their corresponding object types [34].

3.3.2 Feature extractor

Feature extraction from text document is a major component of the classification process. A feature extractor analyzes the text data, identifies the data properties and transforms them into numerical features. The training data for the object type classifier is a list of objects with labels, and each object sample represents a document and the features are extracted from the object.

Machine learning algorithms support only certain formats of the features extracted from the datasets. And the format of the features is dependent on the type of algorithm being used to build the model. For a multi-class classification, Support Vector Machines (SVM) are preferred since they are more robust and its ability to process large data when compared to conventional text classification methods [33,36]. Support Vector Machines requires the features to be in the form of vectors [35]. Vectorization involves the process of transforming text documents into a set of numerical feature vectors [34]. The training data is vectorized using the modules provided by the scikit-learn library. Support Vector Machines and machine learning will be discussed in detail in the following sections.

To convert the raw text into feature vectors the text documents are tokenized using whitespaces as separators. The occurrence of the tokens in each document is computed and finally the token are normalized and assigned weights based on their occurrence and importance in the training documents. Each text document in the training set is transformed into an array of numerical feature vectors as shown in Figure 3.8. The training set is arranged into a matrix where each row denotes a text document and each column denotes a feature. Each text document is represented as a binary vector with a

value 1 if the document contains the feature and 0 if the feature does not appear [33]. The vector matrix is provided as input to the algorithm.

```
>>> import numpy as np
>>> import sklearn
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> vectorizer= CountVectorizer(min_df=1)
>>> training_data = ['bumper','engine','clips','screwdriver','screws','dunnage']
>>> vectorized_data = vectorizer.fit_transform(training_data)
>>> vectorized_data
<6x6 sparse matrix of type '<type 'numpy.int64'>'
        with 6 stored elements in COOrdinate format>
>>> vectorized_data.toarray()
array([[1, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0],
       [0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 0, 0]], dtype=int64)
>>> vectorizer.get_feature_names()
[u'bumper', u'clips', u'dunnage', u'engine', u'screwdriver', u'screws']
```

Figure 3.8: Input vectorization

This process of feature set extraction is called the “Bag of Words” representation [31,34]. The bag of words representation is a collection of individual tokens, also called as unigrams, which disregard word dependencies. Misspelling, phrases, multi-word expressions, and word derivations are also not taken into consideration in bag of words representation. To counter this drawback, a consecutive set of unigrams are considered, known as n-grams representation, to include word dependencies. The feature extraction module provides parameters that can be modified to extract meaningful features from the data. The maximum and minimum number of characters for the n-grams, analyzer, and cut-off parameters are set for the object type classifier.

3.3.3 Machine learning

Machine learning deals with developing systems that are designed to learn and act without being explicitly programmed. The systems adapt to solve a given problem by utilizing sample data or past experience [37]. Text classification in machine learning employs algorithms to generate a decision function that is learned automatically from the data.

Support Vector Machines (SVM) was introduced by Vapnik [38] as a new machine learning algorithm that maps the input data vectors onto a high dimensional feature space and determining a separating hyper-plane between the classes [35,36,38]. They are based on the structural risk minimization principle, which involves finding a hypothesis that guarantees the least true error [36].

SVMs provide functions to classify data that is not linearly separable, by mapping the data on a higher dimensional space without the losing relatedness between the data points. These functions are known as kernel functions [32]. The commonly used kernel functions are linear, radial based function (RBF), polynomial, and sigmoid [35]. Kernel functions are specified for decision functions and are capable of multi-class classification. SVMs are designed to handle high dimensional feature spaces, as is the case of text classification [34,36]. This is possible since SVMs use overfitting protection, which is independent of the number of features. Each document contains only few 1s and mostly 0s, where 1 represents an occurrence of a feature and 0 represents that the feature does not exist in the document. SVMs are capable of handling both dense and sparse vectors as inputs [36].

For a multi-class classification, two approaches are most common – one-vs-one (OVO) and One-vs-all (OVA). In one-vs-one (OVO) approach, one classifier per each pair of classes is constructed and the class which receives the most prediction score is chosen. One-vs-all (OVA), one classifier per each class is constructed. Each class is fitted against all the other classes and the class which classifies the test data with greatest margin is chosen. OVA is the preferred approach for its simplicity, faster processing time, and computational efficiency [32,34]. LinearSVC class with a linear kernel is used to generate the object type classifier since it implements a One-vs-all (OVA) approach [34].

3.3.4 Issues with text classification

Though text classification has been greatly advanced over the last decade, certain issues are still open to research efforts. Most machine learning algorithms work well with balanced datasets. But in the case of imbalanced datasets, the overall performance of text classifier deprecates [39]. Imbalanced datasets refers to situations wherein there are far fewer instances of one class when compared to the other class. This results in a skewed classifier that leans towards the majority class. Though the overall accuracy is very high due to the presence of a large dataset of the majority class, the minority class is misclassified, which is usually a major concern. Imbalanced datasets are very common in real world situations like gene profiling and fraudulent credit card detection [40]. The imbalanced dataset problem has also been encountered while developing the object type classifier, and hence requires addressing. Techniques to counter the effects of an

imbalanced dataset, both at data level and algorithmic level, have been proposed. The most commonly used and effective methods are discussed in the following section.

Some of the basic adjustments, done at the data level, to balance the datasets are sampling techniques - under-sampling and over-sampling. In under-sampling approach, the majority class is diminished by extracting a smaller set from the large set of data while maintain the initial dataset of the minority class intact. Under-sampling greatly reduces the training time but at the same time, a risk of information loss exists due to a diminished dataset. Over-sampling is the exact opposite of the under-sampling process. The size of minority class is expanded by replicating the initial instances to reduce the imbalance ratio between the majority class and the minority class. Although this technique avoids information loss, it does not address the issue that the minority class lacks data. New data is not created; rather existing data is duplicated. Also, over-sampling increases computational cost and the effect of labeling errors are greatly multiplied [39–41].

Over-sampling is the preferred approach towards balancing the dataset for the following reasons:

1. The object classifier deals with simple classifying tasks and therefore does not require extensive, complex and computational costly algorithms.
2. Under-sampling of the training data causes further information loss.
3. As the sample data contains only one object for each instance, labeling errors are almost nonexistent.

3.4 WEKA, machine learning workbench

The Waikato Environment for Knowledge Analysis (WEKA) is a suite of Java class libraries that aid in the application of machine leaning and data mining algorithms to real world problems [42,43]. The principal algorithms in WEKA are the classifiers that generate decisions trees and rule sets that structure the dataset. WEKA also provides tools for data manipulation; visualization of results, cross-validation and comparison of rule set [43]. The WEKA workbench brings together several established algorithms that include decision trees, data clustering methods, feature selection and data filtering to a common graphical user interface to extract useful information while providing flexibility to add new algorithms as desired by the user. It allows the user to perform research pertaining to data mining and knowledge extraction without burdening the user with machine learning algorithms. The flexibility and user friendly interface of WEKA workbench is utilized in this research to generate MTM mapping rules.

The primary graphical interface in WEKA is the "Explorer", which provides easy access to the various algorithms and functionalities [44]. The Explorer window has six different panels that can be accessed from the tabs present at the top as shown in Figure 3.9: WEKA Explorer user interfaceFigure 3.9. The six panels are – Preprocess, Classify, Cluster, Associate, Select attributes, and Visualize. A brief description of each panel and the corresponding data mining tasks supported is presented below.

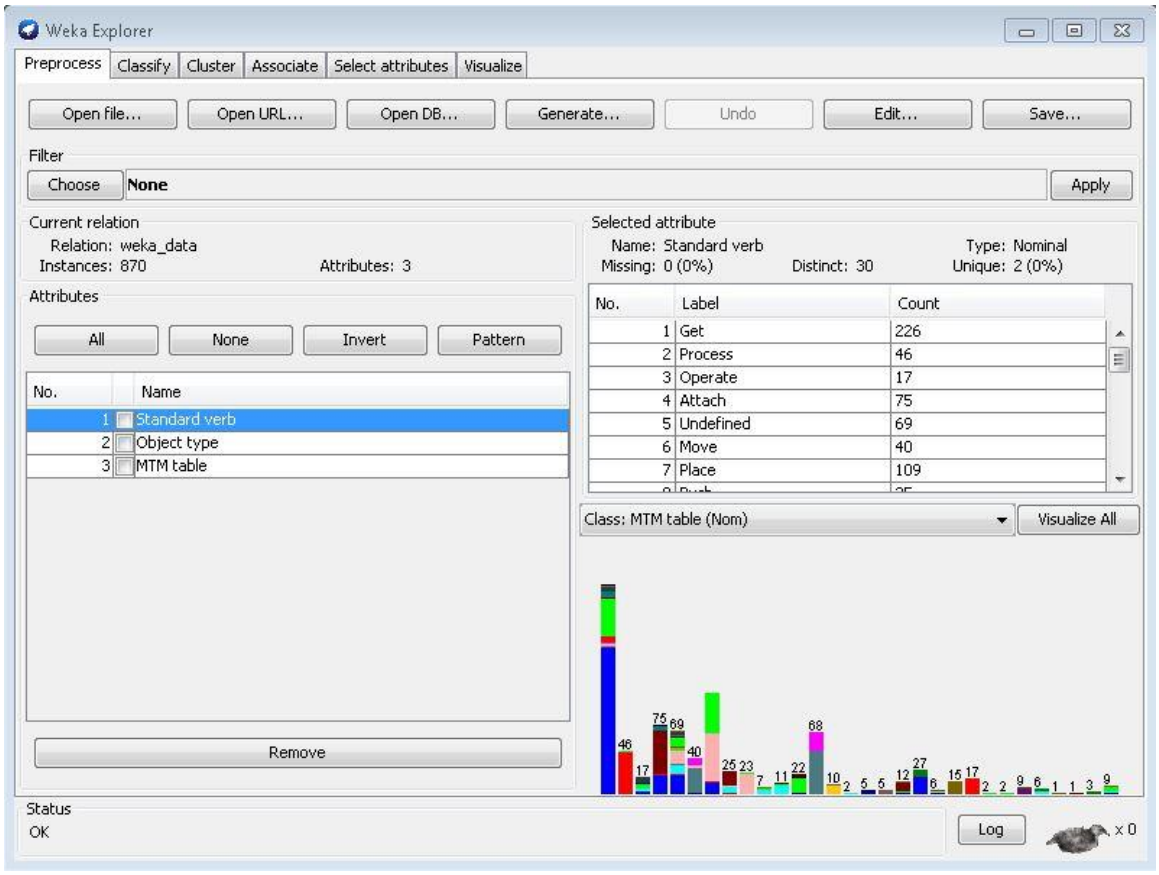

Figure 3.9: WEKA Explorer user interface

WEKA accepts the data in various formats, including ARFF (Attribute-Relation File Format) and CSV (Comma Separated Values). The ARFF format is WEKA's native file format and the preferred format used in this research. The ARFF format defines a data set in terms of relation or a table with attributes or columns of data [45]. Figure 3.10 shows a sample dataset in ARFF format. The data can be loaded from a file or from a database using an SQL query or an URL [44].

In the Preprocess panel, data is loaded and transformed using filters available. The filters perform further preprocessing on the data such as delete certain attributes or row

instances with a particular attribute value [46]. The Preprocess panel also provides a histogram of the attributes and statistics of the dataset as seen in Figure 3.9.

```
@relation details

@attribute 'Student name' {John, Mark, Harry, Tom, Amanda}
@attribute 'Location' {America, United Kingdom, France, Spain, Russia}
@attribute 'Gender' {M, F}

@data
John, America, M
Mark, Russia, M
Tom, United Kingdom, M
Amanda, Spain, F
Harry, France, M
```

Figure 3.10: Sample ARFF dataset

The second panel in WEKA Explorer interface is the Classify panel. It provides the user with access to classification and regression algorithms for analysis. The panel also provides cross-validation tools to analyze the outcome of the algorithm. The Classify panel consists of various machine learning algorithms including decision trees, rule sets, Bayesian classifiers, support vector machines, and nearest-neighbor methods [46]. The Classify panel displays the result of the algorithm used on the data set and also provides the performance of the classifier namely accuracy and confusion matrix.

Clustering is the process of grouping or organizing a set of objects or data instances such that all the members in a group are closely related or similar to each other than objects in other groups. The Association panel consists of algorithms for generating association rules used to identify the relationships between the attributes of the data. Association helps the user to identify the attribute that have the most impact on the prediction model.

WEKA provides several evaluation schemes to identify the most effective attributes in a dataset. Cross validation allows validation of the selected set of attributes. Evaluation methods involve latent semantic analysis and decision tree learner for a specific subset of attributes [44,46]. The last panel in WEKA Explorer is the Visualize panel. This panel allows the user to view the results of the analysis is various color coded matrix of scatter plots.

3.4.1 Decision Trees

As discussed in the previous sections, the MTM mapping rules are formed by extracting the verb, object and the MTM table from the time study steps and performing statistical analysis of the extracted data to find patterns. But the manual generation of rules is exhaustive and also certain implicit relationships can be easily overlooked. Also there is a need to automate the process and establish a concrete method to extend it over large set of data. The functionality of WEKA is utilized for this process.

The Classify panel in the WEKA Explorer consists of several machine learning algorithms and generates simple rules using classification and regression analysis. Decision trees are one of the most often used decision based classification algorithms for their ease of use, understandability, ability to handle both numerical and categorical data, and ability to perform well on large datasets [47–49]. Decision trees are supervised learning algorithms. The main objective of a decision tree is to generate a model to predict a target or output value based on several input variables provided. Decision tree algorithms generate a tree like structure wherein each internal node represents a test and each branch is an outcome. The leaf nodes represent the net result. Each path from the

root node to the leaf node denotes a rule. Figure 3.11 shows a sample tree graph generated by a decision tree algorithm.

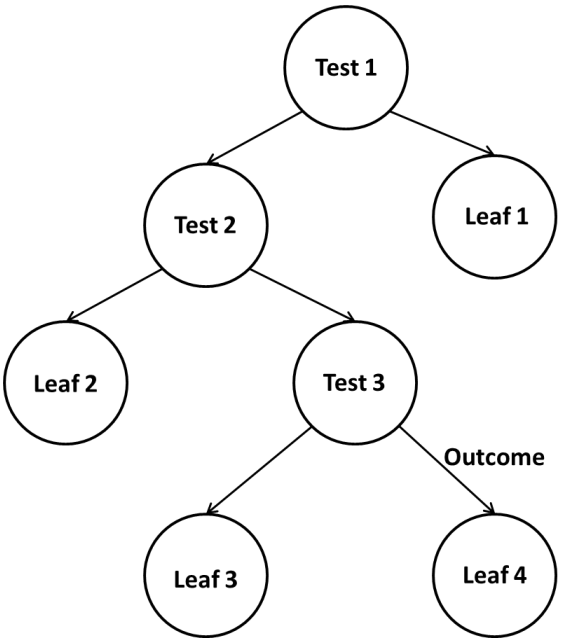

Figure 3.11: Sample tree graph

WEKA contains several decision tree algorithms including Random Tree, J48, Decision Stump, and Naïve Bayesian Tree. Zhao and Zhang [49] compared various decision trees in WEKA using data gathered from astronomical surveys. Based on their results, one of the best performing decision trees is J48 decision tree.

3.4.1.1 J48 decision tree

C4.5 is a widely used decision tree algorithm developed by Ross Quinlan [50][51]. It uses the principle of divide-and-conquer to construct a decision tree structure. The algorithm examines all tests that can split the data and selects the test that gives the best gain [49]. The C4.5 technique is one of the decision tree algorithms that is capable of generating a decision tree and produces rules that are easy to interpret. J48 classifier is

the WEKA implementation of C4.5 technique. J48 classifier is one of the most preferred and efficient decision tree classifiers in WEKA [51]. These factors establish J48 as favorable classifier for generating MTM mapping rules. Furthermore, the J48 algorithm provides the user with option to trim the decision tree to reduce noise and improve accuracy. This process is known as pruning.

Several options are available to the user to provide better control on the parameters of the algorithm. Figure 3.12 shows the options to alter the parameters of the J48 algorithm.

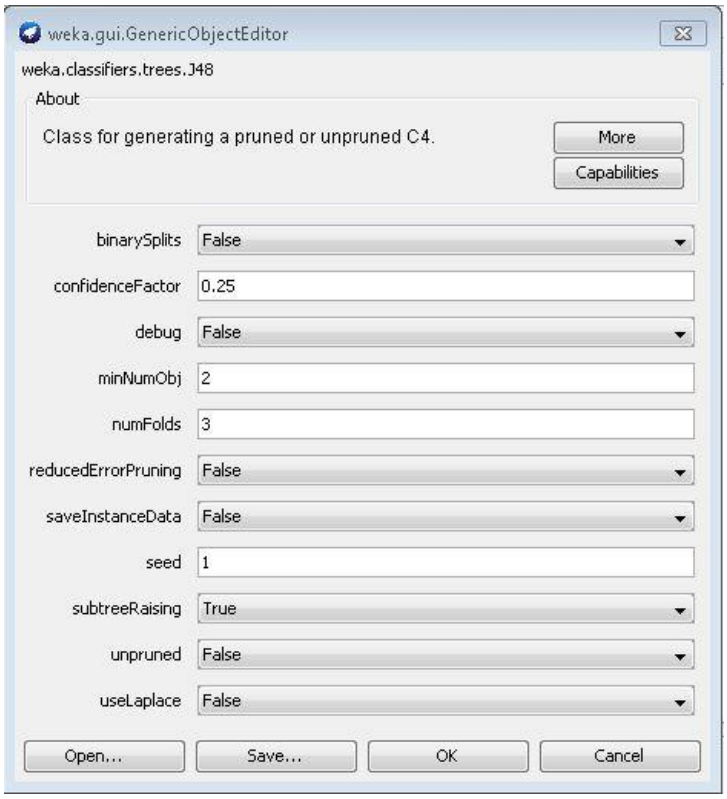

Figure 3.12: Options window to alter parameters of the J48 algorithm

During the construction of a decision tree, the size of the tree is dependent on the dataset supplied. Many nodes and branches reflect the noise and outliers contained within

the dataset [47]. This results in a huge tree structure with an effect on the accuracy of the model. Therefore certain pruning measures are required to identify and eliminate such branches that do not add value and lower the overall accuracy. Pruning decisions trees is an essential step to reduce the complexity of the tree. It aids is optimizing the computational efficiency and also improves the classification accuracy of the model [48]. Also pruning is performed to avoid over-fitting of new data. The two most often used pruning methods are – Post-pruning and Online pruning.

3.4.1.2 Post-pruning

Post-pruning is generally applied to an induced decision tree and it works to remove insignificant branches and nodes. The probabilities of existing sibling leaf nodes is compared and if one leaf node is statistically dominating the other leaf, then the dominating leaf node replaces the two existing nodes. The parent node error is calculated for both cases and compared. This comparison decides if pruning is advantageous at the certain node [48]. The parameter that determines the post-pruning process in WEKA is classified as the confidence factor. Lowering or increasing the confidence factors decides the post-pruning process of the J48 classifier. At each node junction, the algorithm compares the weighted error of each child node and the misclassification error in parent node if the child nodes assigned the majority class. The misclassification error is approximation of the actual error based on incomplete data. The actual error is not an exact value and varies over a range and the confidence factor decides whether the error should lean toward the upper bound or lower bound [48]. The actual error assigned is inversely proportional to the confidence factor. Therefore a low confidence factor relates

to a high actual error assigned. The confidence factor ranges from a scale of 0 to 1. Based on the confidence factor assigned, pruning is carried out.

3.4.1.3 Online pruning

Online pruning is carried out while the decision tree is being induced unlike post-pruning. During the construction of the decision tree, a split in the parent node is made if the child node has sufficient number of data instances. If there exists a case wherein one sibling child node has fewer instances than the minimum required, the child node and the parent node are combined into a single leaf node. The parameter that decides the value for the minimum required data instances is known as minimum number of object instances (minNumObj). Higher the value of minimum number of object instances, higher the pruning and hence smaller the size of the decision tree.

Pruning methods and techniques help in reducing the complexity of the decision trees, improve the accuracy of the model, filtering out the outliers in data. But pruning can also lead to misclassification errors and can have a detrimental effect on accuracy if chosen poorly [48]. Various factors have to be considered and tested while pruning and the parameters are to be adjusted based on individual dataset.

CHAPTER FOUR: DEVELOPMENT AND IMPLEMENTATION OF THE NATURAL LANGUAGE PROCESSING (NLP) AND MACHINE LEARNING (ML) TOOLS

This chapter details the development of the methods to realize the research objectives, using the NLP tools and machine learning techniques that are reviewed in the Chapter Three. Explicitly, this chapter presents how these NLP tools and machine learning algorithms are integrated to achieve the desired outcome.

The purpose of the first research objective is to develop a method to automatically extract information from TVGs to build a standard vocabulary for a consistent structure and format of work instructions and standardizing the TVG authorship process.

4.1 Building standard vocabulary and object type classifier

The Stanford parser is capable of identifying the action verbs in an assembly instruction, but requires the sentences to be in a particular format for accurate parsing. Therefore the all the assembly instructions from the TVGs are edited to fit the desired format. To generate a standard vocabulary and sentence structure for the authorship tool, 236 TVGs consisting of 566 work instructions are analyzed. As discussed earlier, these work instructions are compounded and are thus required to be broken down to single action conveying statements. These work instructions are edited as per the desired format required for parsing and exported to a text file. The Stanford parser is available as an online tool at http://nlp.stanford.edu:8080/parser/. The work instruction text is tokenized, tagged and parsed. The tagged work instruction text is then extracted into a text file for further analysis. A function for extracting all the verb and object tokens from the text file

is developed in Python. Figure 4.1 shows the process flow of the extraction of verbs and objects from the work instructions text.

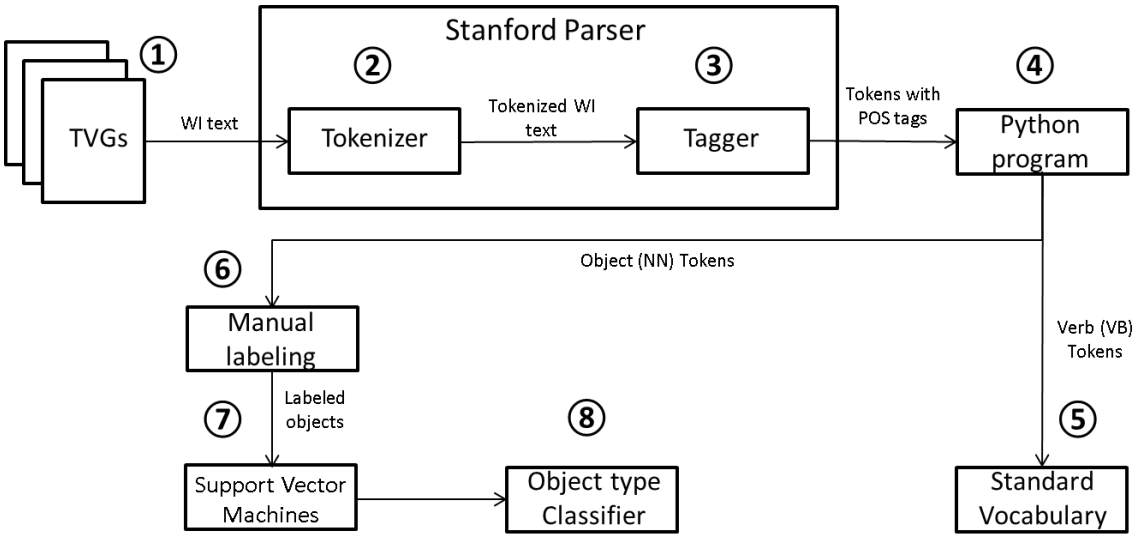

Figure 4.1: Process flow illustrating extraction of verbs and objects

A step by step discussion detailing each stage of the process is provided below.

Step 1: The work instruction text from the process sheets, TVGs, is extracted and the necessary text pre-processing is performed.

Step 2: The tokenizer splits the work instruction text to form single entities based on user specified separator, in this case the whitespace.

Step 3: The tagger assigns a PoS (Part of Speech) to each token. The PoS tag is adjoined at the end of each token separated by a forward slash (‘/’). The parsing process is complete.

Step 4: The tokens with their corresponding PoS tags is supplied as input to the python program. The program extracts the tokens with verb (‘VB’) and noun (‘NN’) tag.

Step 5: The verb tokens are manually analyzed to generate a standard vocabulary based on domain knowledge.

Step 6: The noun tokens are the primary object in each work instruction text. The object instances are manually categorized into one of the five object types.

Step 7: The labeled set of object instances are used as training set to develop a classifier using support vector machines.

Step 8: An object type classifier is developed that is capable of assigning an object type to new object instances.

Figure 4.2 illustrates the process of developing a standard vocabulary of verbs (Step 1 - Step 5) with the help of a sample work instruction – "Get bumper from rack.". The illustration highlights the core mechanism of the process and hence the pre-processing performed on the work instruction is not shown. The work instruction is tokenized and tagged in that order by the parser. The output from the parser is a list containing each entity as a token along with its tag- ['Get/VB', 'bumper/NN', 'from//IN', 'rack/NN', './.']. The entity './.' indicates the end of each work instruction statement. The python function searches the entire list and extracts the token with verb tag ('Get') and noun tag ('bumper', 'rack') and exports them into two separate csv (comma separated values) files as shown.

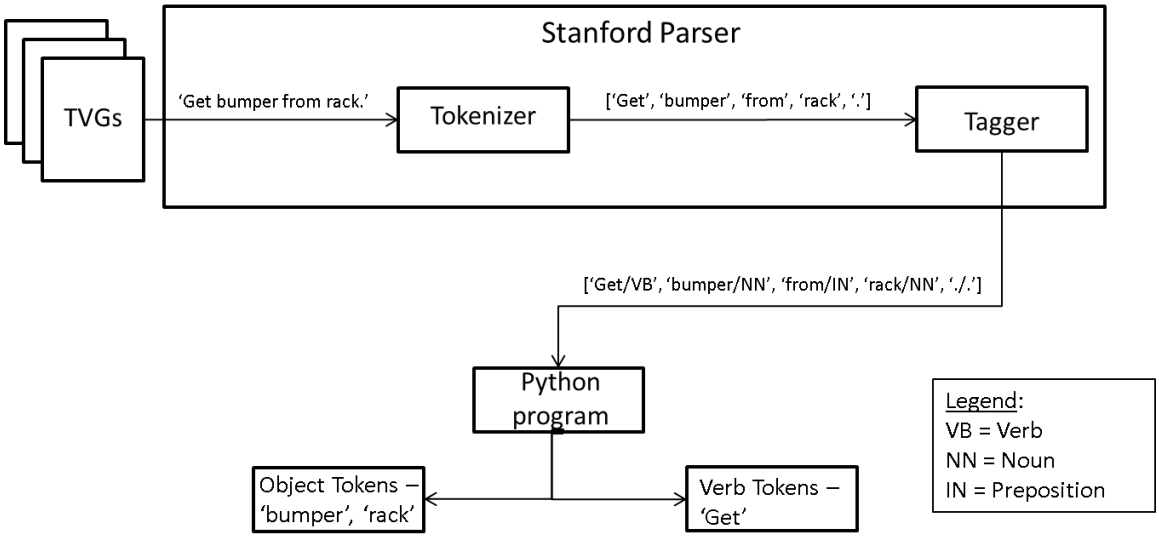

Figure 4.2: Extraction of verb and object from sample work instruction

Some of the work instructions were found to contain inadequate information for analysis. To reduce the noise and capture only work instructions statements that will aid the analysis process, it is determined that each work instruction statement must contain at least one verb and one object on which the verb acts upon. The python program discards all parsed work instruction statements that do not adhere to this condition. The reduced number of valid work instruction statements considered for analysis is 522. The code for the program is provided in Appendix B. NLTK is used as a programming tool to support the analysis. Table 4.1 provides a comprehensive view of the number of work instructions considered for analysis.

Table 4.1: Number of work instructions considered for analysis

Number of TVGs analyzed	236
Number of compound work instructions	566
Number of single-action work instructions (broken down)	697
Number of work instruction considered for analysis	522

4.1.1 Standard verb vocabulary

The verbs extracted from the work instructions are exported to a Comma-separated values (.csv) file. The list consists of 522 verbs with multiple instances of 84 unique verbs. Table 4.2 shows a sample of the most frequently used verbs.

Table 4.2: Sample set of most frequent verbs

Verb	Count	Percentage	Cumulative percentage
Get	44	8.42	8.42
Secure	44	8.42	16.85
Align	36	6.89	23.75
Place	34	6.51	30.26
Take	31	5.93	36.20
Walk	31	5.93	42.14
Fit	18	3.44	45.59
Check	13	2.49	48.08
Insert	13	2.49	50.57
Connect	12	2.29	52.87
Collect	11	2.10	54.98
Install	11	2.10	57.08
Pick up	11	2.10	59.19
Fasten	10	1.91	61.11
Push	9	1.72	62.83
Remove	9	1.72	64.55
Handstart	8	1.53	66.09
Pick	8	1.53	67.62
Press	7	1.34	68.96
Ensure	6	1.14	70.11
Snap	6	1.14	71.26
Tighten	6	1.14	72.41
Verify	6	1.14	73.56

It is observed that many verbs are synonyms of each other and describe the same activity since each planner has his/her own style of authoring process sheets and no restriction on grammar or vocabulary exists. This method introduces redundancy and hence a standard list of verbs is developed to contain only sufficient and necessary verbs. The controlled vocabulary also serves towards standardizing the process sheet authorship process. Therefore the list of 84 unique verbs is further pruned to obtain a set of 31 standard verbs that are sufficient and can distinctly describe all the work instructions that are analyzed from the 236 TVGs. The standard verb vocabulary is manually developed since it requires expert domain knowledge and is specific to the assembly activities carried out in the manufacturing plant of the OEM. The standard verbs are also assigned an OPR class. The OPR class consists of four primary categories that describe the type of process. Each standard verb is assigned one or more OPR class based on the type of physical motions the standard verb describes. The OPR classes are shown below in Table 4.3

Table 4.3: OPR classification

OPR class	Description
M	Assembly
ZH	Additional Handling
ZW	Additional Walking
PF	Functional Inspection

A sample list of standard verbs with their definitions and OPR classification is shown in Table 4.4. The complete list of standard verbs is provided in Appendix A.

Table 4.4: Sample list of standard verbs

Standard verb vocabulary				
S. No	Verb	Definition	Example	OPR class
1	Align	Accurate Positioning of a part or tool over another part	Align bumper to BIW	M
2	Apply	Putting on a medium on an object with or without the aid of a tool	Apply headlight seal initial	M
3	Attach	Setting or binding two parts with each other using only the features on each part	Attach hook to ARB	M
4	Clean	Includes all performances, to clean an object with a tool.	Clean windshield with wipe	M
5	Connect	Includes all activities to connect/ locking or unlocking a cable, with or without tool.	Connect cable to harness	M
6	Disengage	Unlocking a fixture or removing a part from the fixture or tool.	Disengage the fixture / Remove Jig	M, ZH
7	Engage	Locking a fixture or engaging a tool onto a part.	Engage a fixture or clamp.	M, ZH
8	Exchange	Involves exchanging empty bins containing parts and supplies with full bins.	Exchange container nuts	M, ZH
9	Get	Picking up a part or tool from around 1 m or does not necessitate getting up or walking from position.	Get torque tool	M, ZH
10	Handstart	Screwing in 2 rounds, the bolt or nut by hand or with the aid of tools, to set it in position.	Handstart first screw on tool holder at lift assist	M
11	Insert	Includes all activities to assemble clips with hands and/or tool	Insert clip to Y-strut	M
12	Inspect	Carrying out a check on a part or process, in order to make a decision.	Inspect bumper for damages	M, PF
13	Lay	Laying a cable by hand and/or fastening exactly	Route Bowden cable	M
14	Move	Moving with/without a	Move to front bumper	M

Standard verb vocabulary				
S. No	Verb	Definition	Example	OPR class
		part/tool around the car or actions like bending down, squatting.		
15	Open (Preparatory)	Includes all activities to handle packaging, separating layers and opening package to take contents.	Open bag with tool	ZH

The next section discusses the development of the object type classifier using the objects extracted from the work instruction text in addition to forming a standard vocabulary of verbs.

4.1.2 Object type classifier

As discussed in the previous chapter, the MTM mapping rules are generated by analyzing the historical data and formalizing the rules based on the standard verb and object type. Since, manually assigning each object with one of the five object types is tedious and labor intensive; there is a need to automate the process. Therefore, an object classifier is developed to address this issue. To build an object classifier through text classification, an initial dataset with labeled instances, in this case objects, is required to train and build. Figure 4.1 also shows the process of extracting objects (Step 6 – Step 7) from work instructions in addition to extracting the verbs. The list of objects is then manually labeled with an object type each. This dataset acts as a basis for developing a classifier to label new objects that the program encounters. The object type classifier is developed using support vector machines and an OVA (one-versus-all) approach. The

code to developing the classifier is presented in Appendix B. Figure 4.3 shows the process flow illustrating the development of the object type classifier.

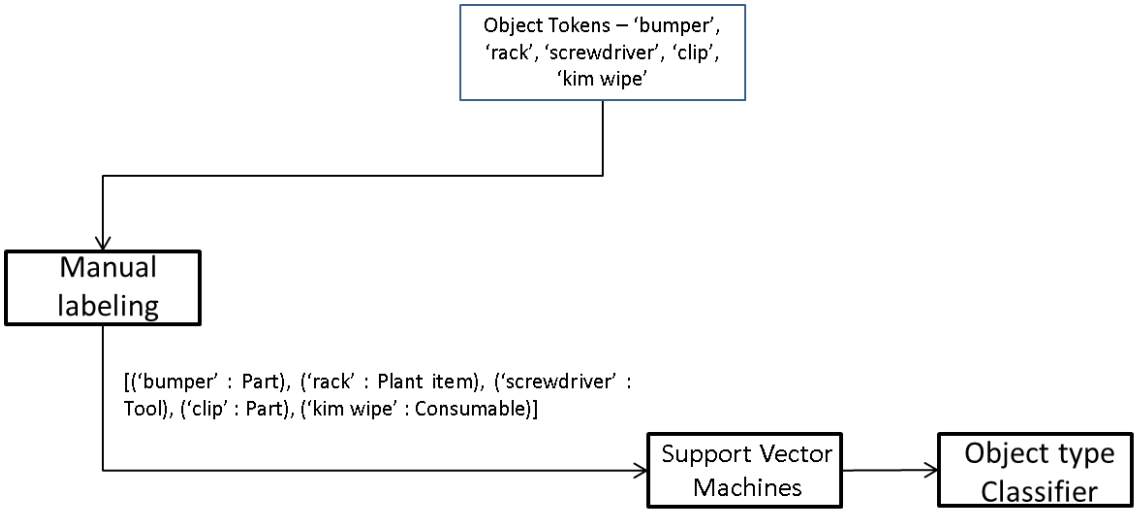

Figure 4.3: Development of object type classifier

The training data presented the problem of an imbalanced dataset. This issue is addressed by over sampling the data set, as discussed in Chapter 3. The initial dataset consists of 794 object instances with majority class being 'Part'. The dataset is oversampled to have almost equal number of instances for each label. The instances of minority classes were randomly duplicated several times keeping the majority class almost intact. Table 4.5 shows the number of part instances before and after over-sampling.

Table 4.5: Dataset before and after oversampling

	Total number of instances	**Instances with label – Part**	**Instances with label - Tool**	**Instances with label - Consumable**	**Instances with label - Plantitem**	**Instances with label - Fixture**
Before Over-sampling	794	464	148	29	122	31

	Total number of instances	Instances with label – Part	Instances with label - Tool	Instances with label - Consumable	Instances with label - Plantitem	Instances with label - Fixture
After Over-sampling	2303	498	455	441	475	434

The new dataset obtained after over-sampling is used to build the classifier. The object type classifier is developed and stored as a function, which can be invoked when required. The development of the standard vocabulary and object type classifier concludes this section.

4.2 MTM mapping rules

This section discusses the process to automatically generate the MTM mapping rules from the time study steps of the process sheets using machine learning algorithms. The MTM mapping rules are formed by analyzing the time study steps from the aforementioned 236 process sheets that are used to generate standard vocabulary and object type classifier. The process for the development of the MTM rules is shown in Figure 4.4.

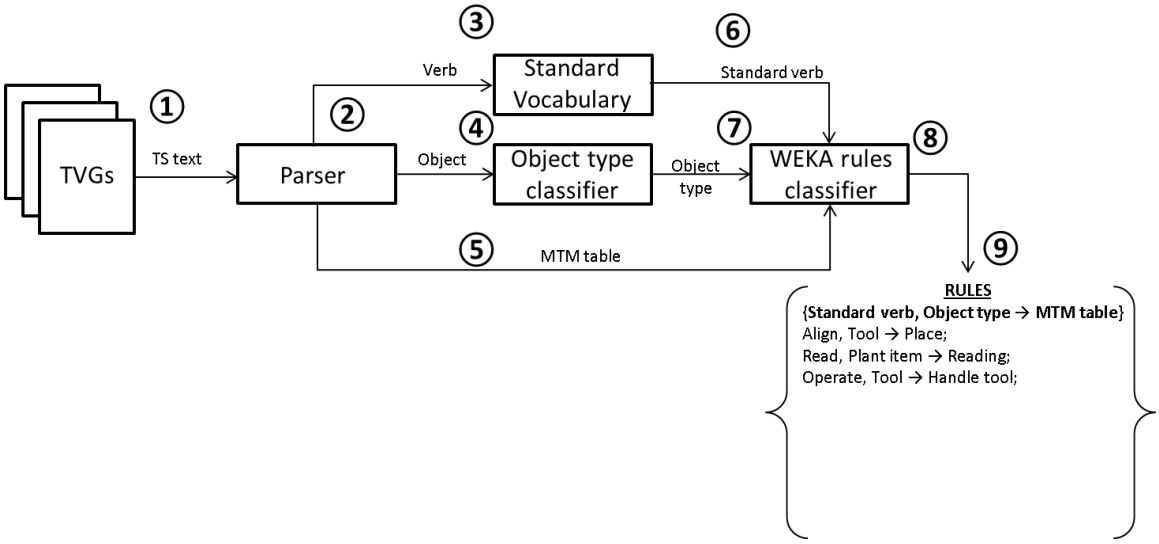

Figure 4.4: Process flow illustrating the generation of MTM rules

A step by step discussion detailing each stage of the process is provided below.

Step 1: The time study steps from the process sheets, TVGs, are extracted and the necessary text pre-processing is performed.

Step 2: The parser performs both tokenizer and tagger functionalities. The time study steps are split and tagged to acquire the PoS (Part of Speech) tags for each token in the text.

Step 3: The token with a verb tag ('VB') is extracted and checked against the standard vocabulary of verbs the equivalent standard verb is obtained

Step 4: The tokens with noun ('NN') tag is extracted. The token denotes the object. The object is supplied to the object type classifier to determine the type of object.

Step 5: The MTM table name is extracted from the corresponding time study step.

Step 6, Step 7: The {Standard verb, object type, MTM table} tuple set is supplied to WEKA workbench in an Attribute-Relation file format (ARFF).

Step 8: WEKA classifier analyzes the dataset and generates a decision tree using the J48 classifier

Step 9: The decision tree is interpreted and the MTM mapping rules are derived.

The data required to generate the MTM mapping rules is extracted from 1019 time study steps from 236 process sheets. In order to reduce the noise and eliminate insignificant data, it is determined that each time study statement must contain at least one verb, one object and a MTM code. The python program discards all parsed time study steps that do not adhere to this condition. Figure 4.5 shows examples of time study steps that are discarded by the Python program since they do not contain a verb and/ or an object.

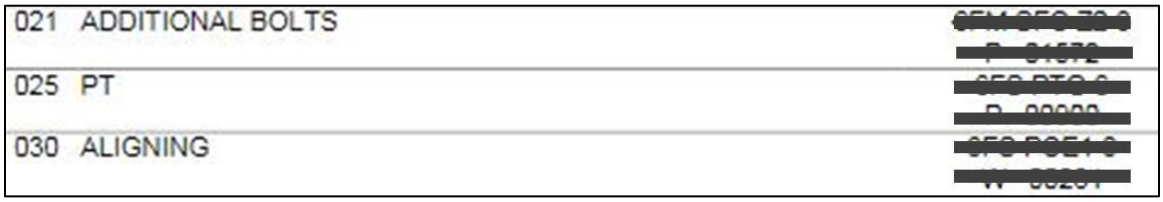

Figure 4.5: Examples of discarded time study steps

The reduced number of valid time study steps considered for analysis is 870. Table 4.6 provides a comprehensive view of the number of TVGs and time study steps considered for analysis

Table 4.6: Number of TVGs and time study steps considered for analysis

Number of TVGs analyzed	236
Number of time study steps	1019

Number of time study steps considered for analysis	870

The first step in generating MTM rules is to extract the verb, object and MTM table from each time study step. This data is analyzed to map standard verb and object type to an MTM table as shown below.

$$\{Verb, Object\ type\} \Rightarrow \{MTM\ table\}$$

Similar to the process of extracting information from the work instruction text, certain text pre-processing tasks are performed on the time study text before parsing. Each time study step is concatenated with 'The associate must' at the beginning of each sentence without altering the time study text. In addition to the general text pre-processing, the MTM information is linked to each sentence. Each time study step is associated with a MTM code. The corresponding MTM table for each code is found from the MTM charts. The MTM table name of each time study step is concatenated at the end of the sentence as shown in Table 4.7. The MTM table name is integrated to the sentence in a pair of square brackets to separate the MTM information from the time study step and to act as an identifier for the python code while extracting information hence any format can be employed.

Table 4.7: Edited time study text

Raw time study step	MTM code	Edited time study step
COLLECT SPEED NUTS AND BOLTS	S-AGHR	The associate must collect speed nuts and bolts [MTM Get and Place].
Fit speed nuts to bumper	S-ACE	The associate must fit speed nuts to bumper [MTM Get and Place].
Take screws and fit to bumper.	M-SAK E	The associate must take screws and fit to speed nuts [MTM Working

		with Screws/ Bolts].

These time study steps are edited for accurate parsing and exported to a text file. The text file is then parsed using the Stanford parser. As mentioned above, each time study step is associated with a MTM code. Therefore, the time study steps that contain more than one verb cannot be split into two separate sentences. All the verbs in a single statement that describe the activity are extracted as a single entry. It is observed that the maximum number of verbs present in a time study step is two. This results in a slight variation of the rule. The format of the MTM mapping rules is adjusted to accommodate time study steps with two action verbs as shown below.

$$\{Verb1\ \&Verb2,\ Object\ type\} \Rightarrow \{MTM\ table\}$$

The Python code extracts the verb(s), object and MTM table from the parsed time study steps using NLTK as a programming tool. However, if the Python program encounters time study steps with two verbs, it extracts both verbs and concatenates them using an ampersand – '&'. For example, consider the time study step – 'Get and Place bumper to car body'. The two verbs in this case are 'Get' and 'Place'. Therefore the Python program extracts the verbs and concatenates them into a single entry – 'Get & Place'. The verb is mapped onto a standard verb from the verb vocabulary, developed by extracting verbs from work instructions. The object type for the direct object, on which the verb acts, is generated from the object type classifier developed in the previous stage. The standard verb, object type and MTM table tuple set is supplied to the WEKA platform. The WEKA rules classifier, using the J48 decision tree algorithm, analyzes the data and outputs a set of rules.

Figure 4.6 shows the process of extracting data and formation of rules from a sample time study step. The sample time study step – 'Go to storage area' is extracted from the process sheet and the necessary text pre-processing is performed. The sentence is then concatenated with the corresponding MTM table name in a pair of square brackets. This text file is then supplied to the Stanford parser which tags and tokenizes the sentence. The parser outputs the parsed time study step in a text file. This text file is further analyzed to obtain the {verb, object type, MTM table} tuple for generating the MTM mapping rules. The python code extracts the verb token ('Go to'), object token ('storage area') and MTM table name ('Advanced Level / Car Body'), which is present within the pair of square brackets.

The verb 'Go to' is looked up against the list of standard vocabulary of verbs and replaced with a standard verb – 'Walk'. The object is supplied as an input to the object type classifier to obtain the object type class. In this case, the object type classifier assigned the type 'Plant item' to the object 'storage area'. The program then generates a tuple consisting of the standard verb, object type and MTM table – {Walk, Plant item, Advanced Level / Car Body}. The code to extract the tuple from the time study step is presented in Appendix B. The next stage in the process is to supply the tuple set to WEKA to generate the MTM mapping rules.

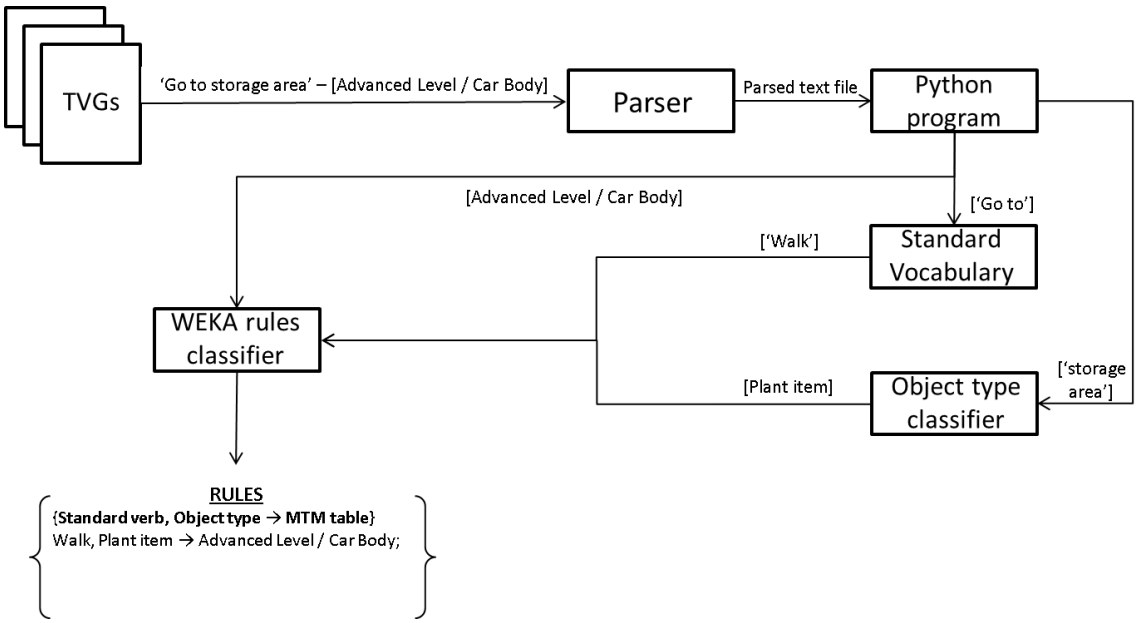

Figure 4.6: Development of MTM mapping rules using a sample time study step

4.2.1 MTM mapping rule generation through WEKA

WEKA accepts input in the form of ARFF (Attribute-Relation File Format). The three attributes of the input file are standard verb, object type and MTM table name. The {standard verb, object type, MTM table name} tuple from each of the 870 time study steps that are analyzed is supplied as input to WEKA.

J48 is used to generate the rules. As mentioned earlier in Chapter Three, the parameters for the pruning process is based on the individual dataset and preliminary tests have to be performed to understand the effect of each pruning process on the decision tree. There several decision trees are generated varying both confidence factor and the number of object instances to determine the best conditions.

Table 4.8 shows the percentage of correctly classified instances, relative absolute error and size of tree for five decision trees generated by increasing the minimum number of object instances from 1 to 5 in steps of 1, while maintaining the confidence factor

constant at a value of 0.25. The decision trees are evaluated with a cross-validation number of 10. This analysis compares the variations in online pruning without performing post-pruning.

Table 4.8: Decision trees with varying minimum number of object instances

Decision tree	Confidence factor	Minimum number of Object instances	Correctly classified instances (%)	Relative absolute error (%)	Size of tree
#1	0.25	1	71.72	43.05	103
#2	0.25	2	71.37	43.48	83
#3	0.25	3	71.26	43.58	78
#4	0.25	4	71.26	43.58	73
#5	0.25	5	70.68	44.32	73

In the second analysis, the minimum number of object instances is kept constant while varying the confidence factor from 0.1 to 0.5 in incremental steps of 0.1. Table 4.9 shows the five decision trees with the correctly classified instances, relative absolute error and size of tree.

Table 4.9: Decision trees with varying confidence factor

Decision tree	Confidence factor	Minimum number of Object instances	Correctly classified instances (%)	Relative absolute error (%)	Size of tree
#1	0.1	1	70.80	44.05	88
#2	0.2	1	71.91	43.79	103
#3	0.3	1	71.60	42.60	103
#4	0.4	1	71.95	42.05	108
#5	0.5	1	71.95	41.92	113

The first analysis shows that as the minimum number of object instances increases, the absolute relative error also increases thereby affecting the accuracy of the

model. The second analysis proves that as the confidence factor increase, the relative absolute error decreases thereby having a positive effect on accuracy. Also, the accuracy of each decision tree is relatively constant throughout. However, a noticeable difference is observed in the size of tree. The size of the decision trees greatly differs without any significant change in accuracy of the model. Therefore the deciding factor in choosing the parameters is the size of the decision tree. The number of rules generated is directly proportional the number of rules. Therefore, a smaller size tree generates fewer rules. Based on the above analysis, the parameters for both post-pruning and online pruning is determined for the decision tree and is shown in Table 4.10.

Table 4.10: Parameters for decision tree pruning

Pruning process	**Parameter**	**Value**
Post-pruning	Confidence factor	0.3
Online pruning	Minimum number of object instances	3

Table 4.11 show the accuracy of the decision tree along with the size of tree. The cross –validation for the algorithm is set at 10.

Table 4.11: Statistics of accuracy and size of decision tree

Confidence factor	**Minimum number of Object instances**	**Correctly classified instances (%)**	**Relative absolute error (%)**	**Size of tree**
0.3	3	71.14	43.14	78

A decision tree is generated using the above mentioned parameters as shown in Figure 4.7. The output window in the classifier panel displays the pruned tree in text format.

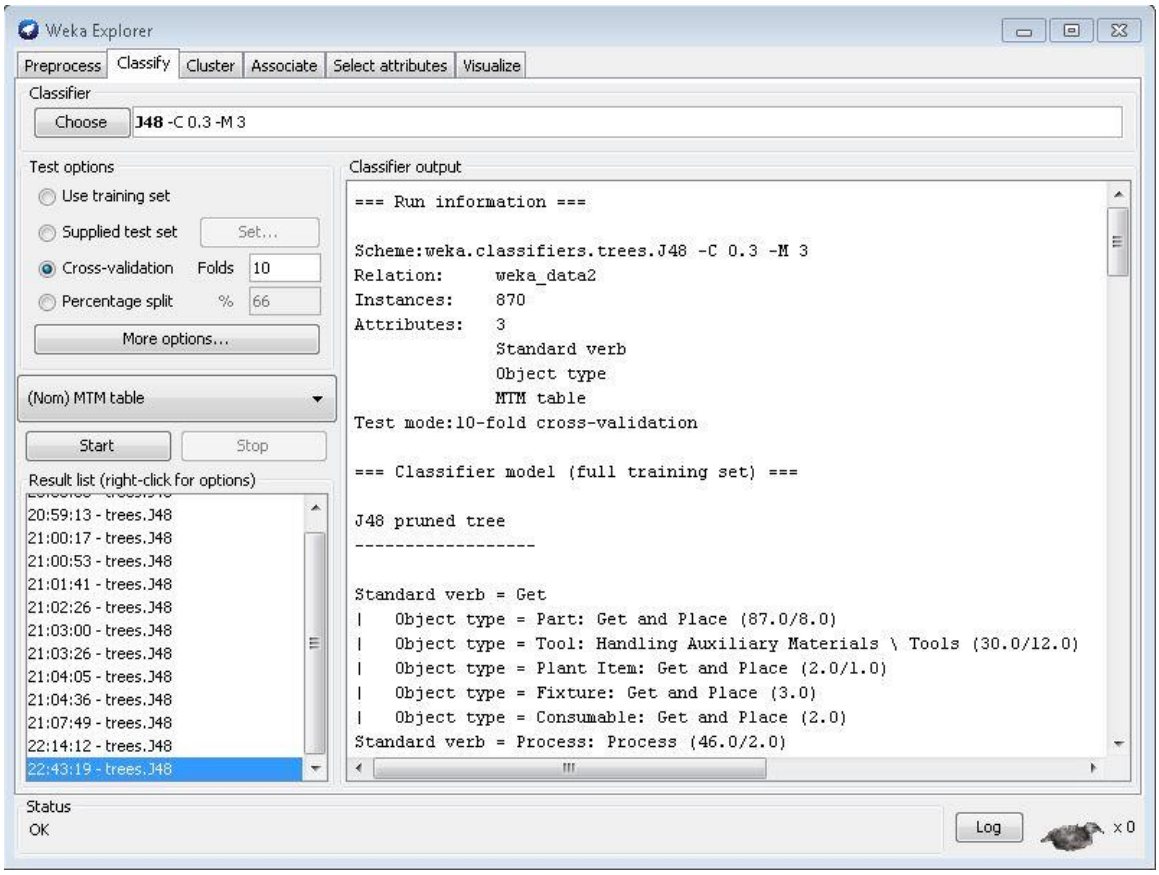

Figure 4.7: J48 decision tree with output

WEKA outputs the decision tree of the J48 algorithm in a rule format along with the number of instances encountered as shown in Figure 4.8.

```
Standard verb= Get
                | Object type= Part: Get and Place (87.0/8.0)
```

Figure 4.8: Sample rule format

The rule implies, IF <Standard verb= 'Get'> AND <Object type= 'Part'> THEN <MTM Table= 'Get and Place'>. The first number in the bracket indicates the number of instances that follow the particular rule in the dataset supplied and the second indicates

the number of incorrectly classified instances as a result of the rule. The MTM mapping rules generated from the decision tree present three different types of rules.

In Level 1, the standard verb directly maps onto the MTM table without requiring the object type information. This means that the standard verbs always maps to a specific MTM table irrespective of the object it acts on. An illustration of Level 1 rule is shown below.

$$\{Verb\} \Rightarrow \{MTM\ table\}$$

The Level 1 MTM rules are presented in Table 4.12

Table 4.12: MTM rules - Level 1

MTM mapping rules – Level 1	
Standard verb	**MTM table**
Align	Place
Apply	Motion Cycles
Connect	Laying Cables
Clean	Cleaning
Disengage	Operate
Engage	Operate
Exchange	Handling Containers
Handstart	Working with Screws\ Bolts
Insert	Working with Clips
Inspect	Visual Control
Lay	Laying Cables
Move	Body Motions
Press	Operate
Read	Read
Remove (Preparatory)	Preparatory Activities
Restock	Parts Supply
Scan	Marking and Documenting
Secure	Handling Auxiliary Materials\ Tools
Tighten	Handling Auxiliary Materials\ Tools
Unscrew	Motion Cycles
Walk	Body Motions

Certain standard verbs map to several MTM tables and therefore require object type information to further narrow down the mapping. Therefore the standard verb and object type together drive the user to a particular MTM table. This is represented as Level 2 rules and is shown below.

$$\{Verb, Object\ type\} \Rightarrow \{MTM\ table\}$$

The Level 2 MTM rules are presented in Table 4.13

Table 4.13: MTM rules - Level 2

MTM mapping rules – Level 2		
Standard verb	**Object type**	**MTM table**
Get	Part	Get and Place
Get	Tool	Handling Auxiliary Materials\ Tools
Get	Plant item	Get and Place
Get	Fixture	Get and Place
Get	Consumable	Get and Place
Operate	Part	Operate
Operate	Tool	Handle Tool
Place	Part	Place
Place	Tool	Handling Auxiliary Materials\ Tools
Place	Plant item	Place
Place	Fixture	Place
Push	Part	Working with Clips
Push	Tool	Operate
Attach	Part	Working with Clips
Attach	Tool	Working with Clips
Attach	Plant item	Get and Place
Attach	Consumable	Working with Adhesives
Remove	Part	Get and Place
Remove	Tool	Get and Place
Remove	Plant item	Get and Place
Remove	Fixture	Get and Place
Remove	Consumable	Preparatory Activities

The final type, Level 3, corresponds to the compound time study steps that contain two verbs. The two standard verbs and the object type directly map onto a MTM table.

$$\{Verb1\ \&\ Verb2, Object\ type\} \Rightarrow \{MTM\ table\}$$

The Level 2 MTM rules are presented in Table 4.14

Table 4.14: MTM rules - Level 3

MTM mapping rules – Level 3		
Standard verbs	**Object type**	**MTM table**
Get & Attach	Part	Working with Clips
Get & Attach	Tool	Get and Place
Get & Attach	Fixture	Get and Place
Get & Operate	Tool	Handle Tool
Get & Connect	Part	Laying Cables
Get & Insert	Part	Get and Place
Get & Apply	Part	Get and Place

4.3 MTM table generator - GUI to generate MTM table for work instructions

This section discusses the development of a GUI to generate MTM tables for the work instructions authored by the planner. The GUI is developed using NLTK as a platform to and utilizes the Stanford parser, object type classifier and the MTM rules developed through WEKA. It aids the user in suggesting the appropriate MTM table and reduces the cognitive load and ambiguity.

The GUI is written in Python using the library of functions provided by NLTK. The tools and decision support generated in Chapter 4 -Stanford parser, object classifier and MTM mapping rules are integrated within the GUI which provides the functionality

to author work instruction in free form and generate MTM table for each work instruction authored. The process flow for generating MTM tables for a set of work instruction authored by the planner is shown Figure 4.9.

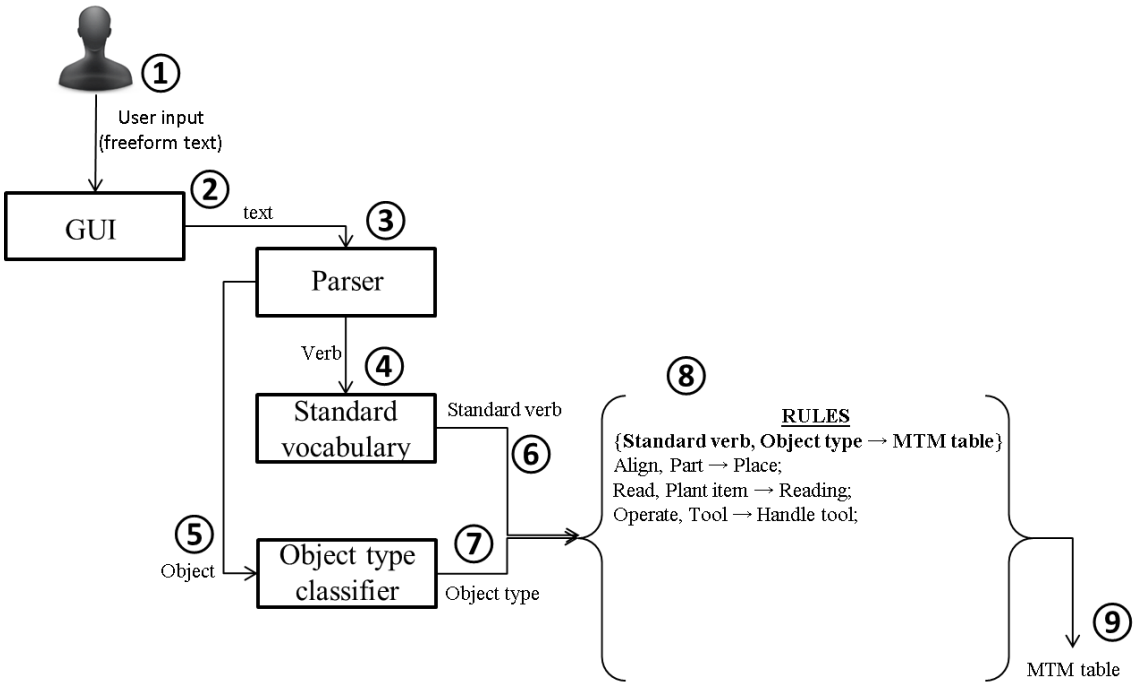

Figure 4.9: Process flow illustrating the generation of MTM tables

A step by step discussion detailing each stage of the process is provided below.

Step 1: The user inputs a work instruction, in free form text, in the input box of the GUI.

Step 2: The program performs the necessary text pre-processing on the work instruction text. The phrase “The associate must” is concatenated at the start of the sentence. The work instruction statement is edited to the desired format.

Step 3: The parser performs both tokenizer and tagger functionalities. The time study steps are split and tagged to acquire the PoS (Part of Speech) tags for each token in the text.

Step 4: The token with a verb tag (‘VB’) is extracted and checked against the standard vocabulary of verbs and the equivalent standard verb is obtained

Step 5: The tokens with noun (‘NN’) tag, denoting the objects, is extracted. The object is supplied to the object type classifier to determine the type of object.

Step 6, Step 7: The standard verb and object type is gathered from the standard vocabulary and object type classifier.

Step 8: The {Standard verb, object type} pair is checked against the existing MTM mapping rules.

Step 9: The appropriate MTM table is determined and displayed in the output box of the GUI.

The planner input the desired work instruction in free text in the upper input window of the GUI as shown in Figure 4.10. Multiple work instruction can be written at one instance. The work instructions must be input subject to the following rules.

1. The work instruction should start with a valid standard verb.
2. The work instruction should contain at least one object on which the standard verb acts on.
3. A period at the end of each work instruction to indicate that the sentence is complete.

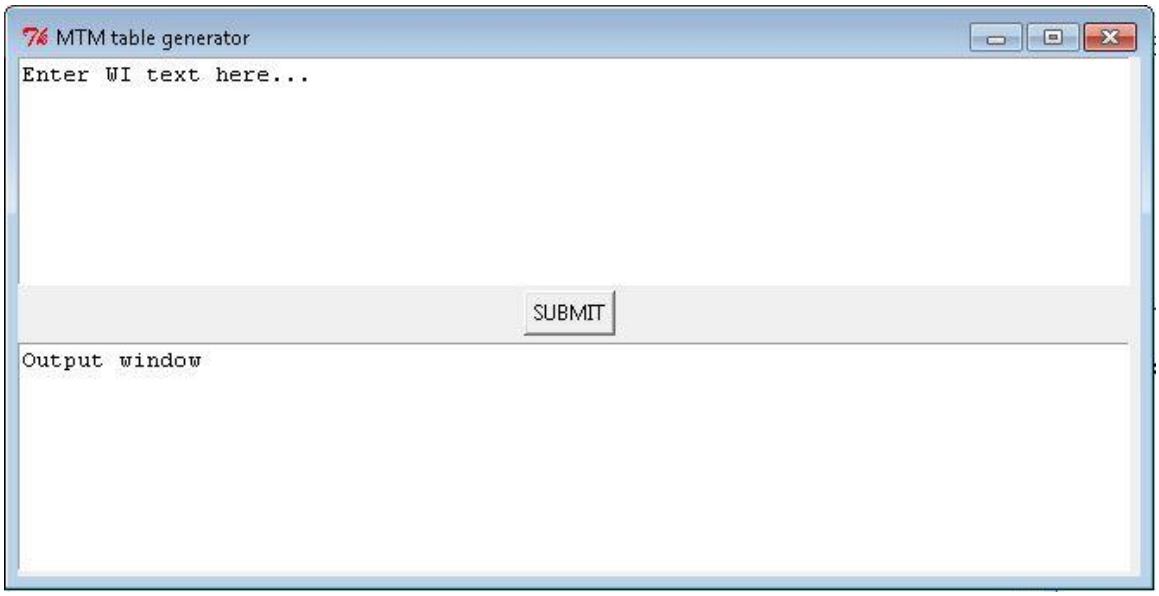

Figure 4.10: Screenshot of the GUI

The GUI collects the work instructions and passes it to the Python program. The Python program performs text pre-processing before parsing. Pre-processing the work instruction is essential since the parser can only analyzed complete and grammatically correct sentences. The phrase "The associate must" is concatenated at the start of each sentence to provide contextual meaning to the sentence. The pre-processing does not alter the intent of the work instruction.

The Stanford parser tags and tokenizes the processed work instructions. Once the work instructions have been parsed, the Python program extracts the verb and object from the sentences. The object is classified and assigned an object type using the object type classifier. This results in the formulation of verb and object type information pair. The {standard verb, object type} is searched against the MTM rules and the appropriate MTM table is displayed along with the work instructions in the output window of the GUI as shown in Figure 4.11.

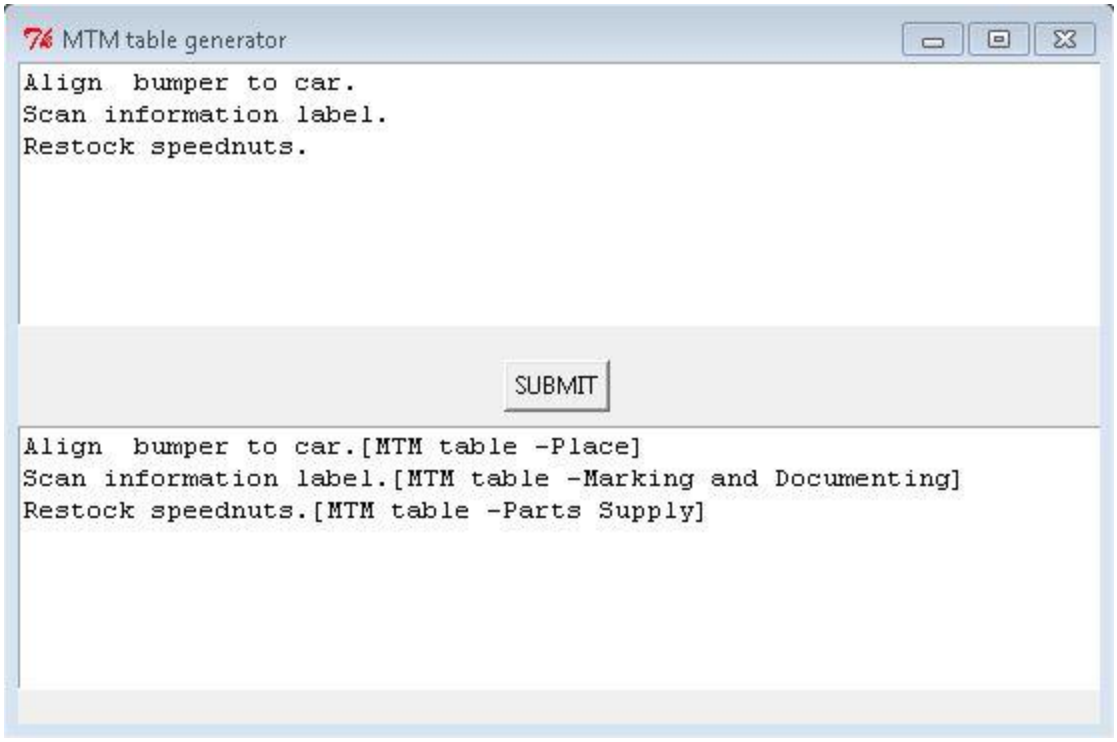

Figure 4.11: Screenshot illustrating the MTM tables generated for sample work instructions

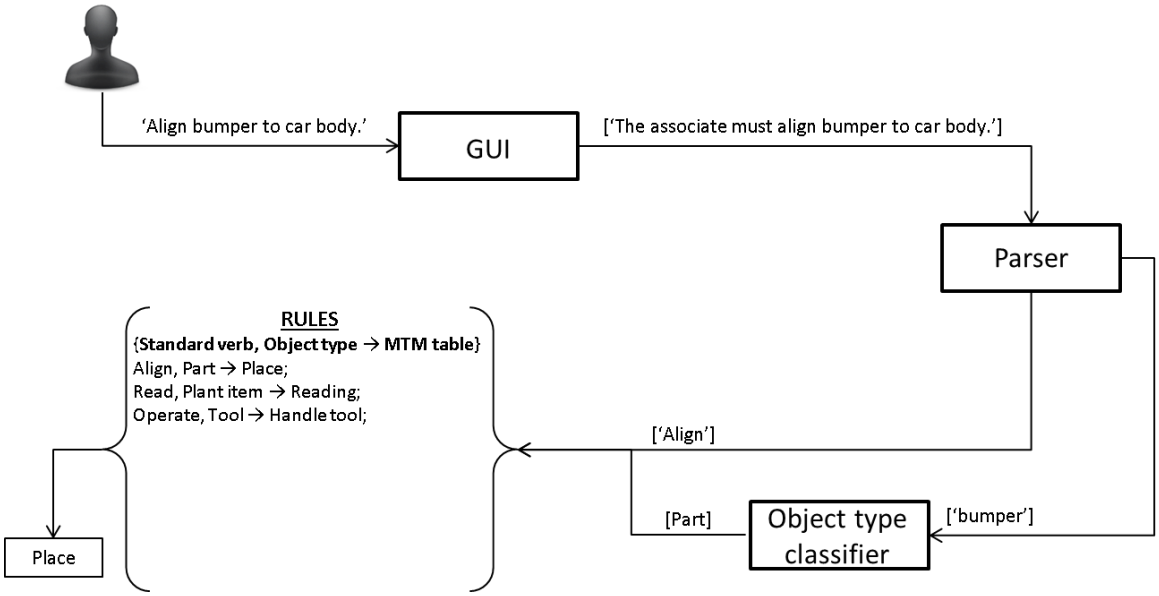

Figure 4.12: MTM table generation for sample work instruction

Figure 4.12 shows the generation of the MTM table for a sample work instruction. The user inputs the sample work instruction “Align bumper to car body.” The GUI supplies the work instruction text to the Python program, wherein the sentence is restructured to meet the requirements of the parser. The parser tokenizes and tags the work instruction. The verb (VB) token in the sample work instruction is ‘Align’ and the primary object (NN) token is ‘bumper’. The object is classified by the object type classifier and assigned the label ‘Part’. The standard verb and object type pair is then checked against the MTM rules. The {standard verb, object type} maps on to the MTM table ‘Place’. The MTM table is coupled to the work instruction and displayed in the output window of the GUI. The code for the development of the MTM table generator is provided in Appendix B.

CHAPTER FIVE: TESTING AND VALIDATION OF TOOLS DEVLOPED

This chapter discusses the validation of the object type classifier and MTM mapping rules developed. The MTM mapping rules are tested against a set of time study steps (from TVGs that are not considered for the initial analysis) to obtain the accuracy of mapping.

5.1 Validation of Object type classifier

The Object type classifier is developed using Support Vector Machines (SVM) with a linear kernel, a supervised machine learning algorithm. The accuracy of the classifier is tested using random split method and cross-validation.

5.1.1 Random split

In random split testing method, the training set is randomly split into two sets based on an attribute value supplied by the user. One set is used to train the model and the other set is used to test the model. The criterion to split the data is based on a percentage split ratio, established by the user. If the percentage to test the object type classifier is set at 40%, then 60% of the dataset is used to train the model and the remaining 40% is reserved to test the classifier. The accuracy of the classifier is tested using 30%, 40% and 50% split ratios. Table 5.1 shows the accuracy for each split ratio employed and the average accuracy when tested using random split. The average accuracy of the classifier when tested using random split method is determined to be 94.3%

Table 5.1: Validation of object type classifier using random split method

Percentage split ratio	Accuracy (%)
30 %	95.0
40 %	94.0
50 %	94.0

5.1.2 Cross-validation

By portioning the dataset to create train and test data, the available dataset to train the model is considerably reduced. To counter this, cross-validation of the classifier is performed and compared against the result from random split. In k-fold cross-validation, the dataset is first divided into k smaller but equal datasets. Of these k sets, k-1 sets are used to train the model and tested on the remaining set. This procedure is repeated k times (number of smaller datasets). The accuracy from each test is then averaged. The classifier is tested using 5-fold, 10-fold, and 15-fold cross validation. The accuracy results for each k value is presented in Table 5.2

Table 5.2: Validation of object type classifier using k-fold cross validation

Value of k	Accuracy
5	94% +/- 2%
10	94% +/- 3%
15	94% +/- 4%

Table 5.3 shows the average accuracy results obtained from each test. Both testing methods prove that the object type classifier has a very high accuracy of 94%.

Table 5.3: Comparison of accuracy - Random split vs. Cross validation

Testing method	Average accuracy
Random split	94.3 %
Cross-validation	94% +/- 3%

5.2 Comparisons of WEKA classifiers – JRip, PRISM, and J48

WEKA contains several machine learning algorithms to classify data and generate rules. These classifiers are divided into groups based on the technique employed to classify data. The J48 classifier, used to generate the MTM mappings, is a decision tree algorithm. This section compares the results from J48 with two rule based classifiers – JRip and PRISM.

JRip is an inference and rule based learner which implements a propositional rule learner. JRip and PRISM can be accessed from the classify panel, under the rules sub-category. PRISM is also a rule based learned which is closely based on ID3 algorithm. The rule based classifiers generate rules directly from the data when compared to J48 which is an indirect approach to generate rules since the rules are derived from the decision tree.

To determine the performance of each classifier the number of correctly classified instances and relative absolute error is used as criteria. It is found that the J48 classifier performs better than the two rule based classifier. JRip classifier generated 26 rules whereas PRISM produced 132 rules. The summary of results from the classifiers is shown in Table 5.4.

Table 5.4: Summary of results - JRip, PRISM, and J48

Classifier	Correctly classified instances	Relative absolute error	Number of rules
JRip	68.50	52.93	26
PRISM	53.33	53.35	132
J48	71.14	43.14	50

JRip produced far too few rules whereas PRISM generated far too many. It can also be observed that J48 performs better than JRip and PRISM by comparing the relative absolute error and correctly classified instances.

5.3 Validation of MTM mapping rules generated through WEKA

The accuracy of the MTM mapping rules generated through WEKA are tested using the time study steps extracted from TVGs, that were not used for the initial analysis. The MTM table names generated by the MTM table generator are checked against the MTM table information associated with each time study step contained in the TVGs.

The time study steps extracted from the TVGs did not contain the standard vocabulary. Hence the sentences are restructured to meet the requirements of the MTM generator. The verbs from the time study steps are replaced with a Standard verb that closely matches the original verb used to describe the activity. A total of 71 time study steps, extracted from 17 TVGs, are used to test the accuracy of the MTM rules. These 71 time study steps are inputted to the MTM generator. The MTM tables generated for each time study step is then checked against the MTM table information from the TVGs and the number of accurately estimated time study steps is obtained. Table 5.5 shows a sample list of time study steps used for the testing purpose along with the original MTM table information as found in the TVG and also the MTM table estimated by the MTM table generator. The complete list of time study steps used for the validation of MTM generator is provided in Appendix A.

Table 5.5: Results from testing MTM table generator

S. No.	Time study step	MTM table (from TVG)	MTM table (from MTM table generator)	Check
1	READ VEHICLE INFORMATION AS REQUIRED ALL PROCESSES MODEL, 4UBA.	Read	Read	Y
2	PLACE PROTECTOR CARRIED FROM CAR TO CAR DURING WALK TO NEXT CAR REMOVAL SEPARATE.	Place	Place	Y
3	GET AND PLACE PLUG.	Get and Place	Get and Place	Y
4	GET AND PLACE SEAT BELT RECEIVER.	Get and Place	Get and Place	Y
5	WALK FROM CAR TO PARTS AND THEN BACK.	Body Motions	Body Motions	Y
6	GET REAR SEAT AND PLACE IN CAR INITIAL.	Get and Place	Get and Place	Y
7	GET AND TURN SEAT UP AND PLACE UNDER BRACKETS.	Get and Place	Get and Place	Y
8	GET REAR SEAT AND PLACE IN CAR INITIAL.	Get and Place	Get and Place	Y
9	GET AND TURN SEAT UP AND PLACE UNDER BRACKETS.	Get and Place	Get and Place	Y
10	WALK TO CART THEN CAR.	Body Motions	Body Motions	Y
11	REMOVE BAGS FROM BETWEEN SEATS / SEPARATE AND DISCARD.	Get and Place	Get and Place	Y
12	PRESS BUTTON ON LIFT ASSIST FOR 3RD ROW SEATS / THEN BACK WHEN DONE.	Operate	Operate	Y
13	PRESS FORWARD SWITCH /GRAB TRIGGER UNDER HANDLE.	Operate	Operate	Y
14	PLACE LIFT TO SEAT / THEN MOVE ACROSS TO FINAL POSITION.	Place	Handling Auxiliary Materials \ Tools	N
15	APPLY PRESSURES TO STOP LIFT AND THEN PUSH OVER.	Motion Cycles	Motion cycles	Y
16	PRESS SWITCH FOR DOWN	Operate	Operate	Y

	AND SWITCH FOR CLAMP.			
17	PT (TIME FOR CLAMPS TO CLOSE).	Process	Verb does not exist	Rule does not exist

Table 5.6 presents a relationship matrix between the MTM tables identified from the TVGs and the MTM tables estimated by the MTM table generator for the test time study steps. The number in each cell denotes the number of time study steps that relate to the particular MTM table in the corresponding row and column. The presence of a linear relationship between the MTM tables from TVGs and MTM tables estimated through the mapping rules indicates that a high number of time study steps have been accurately estimated by the MTM generator. It can be observed that 55 time study steps, covering 7 MTM tables, have been accurately estimated by the MTM generator. 6 time study steps have been incorrectly mapped and MTM generator did not provide a MTM table suggestion for the remaining 10 time study steps since a mapping rule for the particular {Verb, object type} information pair does not exist.

Table 5.6: Relationship matrix between MTM tables identified from TVGs and MTM tables estimated by MTM generator for test time study steps

MTM table (Rules) / MTM table (TVG)	Get and Place	Place	Body Motions	Read	Operate	Motion Cycles	Process	Visual Control	Preparatory Activities	Handling Auxiliary Materials/Tools	Handling Containers	Working with Clips	Rule does not exist
Get and Place	17											2	1
Place		1								1			
Body Motions			10										
Read				3									
Operate			1		17						2		2
Motion Cycles						5							
Process													6
Visual Control								2					
Preparatory Activities													1
Handling Auxiliary Materials/ Tools													
Handling Containers													
Working with Clips													

Some of the time study steps mapped on to the MTM table ‘Process’. The MTM table ‘Process’ relates to the time elapsed during a multitude of activities such as wait

time for a lift assist to move to place, time required to tighten nuts/bolts, operation of tools. The time elapsed during such activities is provided by the planner and is not derived from the MTM charts. Therefore a need is recognized to provide planners with an option to include process time where they seem fit.

Out of the 71 time study steps used for testing, 6 time study steps mapped on to the MTM table 'Process'. Therefore only 65 valid time study steps are considered to determine the accuracy of the MTM mapping rules. From these 65 time study steps, 6 instances are incorrectly mapped and the remaining 4 time study steps do not have a rule yet and therefore have also been considered as a negative outcome. The summary of the results is presented in Table 5.7.

Table 5.7: Summary of results

Total number of time study steps analyzed	71
Number of valid time study steps	65
Number of accurately estimated time study steps	55
Incorrectly estimated time study steps	10
Accuracy	84.6 %

5.4 Chapter Summary and Conclusions

This chapter presents the validation of the object type classifier and the MTM mapping rules. The MTM mapping rules are tested using 71 test time study steps. These time study steps are gathered from 17 TVGs that have not been used for the initial analysis.

The object type classifier is validated using random split method and k-fold cross validation. The average accuracy is found to be 94%. This high accuracy could be the result of oversampling the data pool. Therefore, to further validate the classifier,

additional TVGs must be analyzed to observe if there is a considerable change in accuracy. However, the object type classifier performs better when trained on a larger dataset. Therefor accuracy will also increase.

The MTM mapping rules have significant accuracy of 84.6% but they do not cover all valid time study steps, thus requiring further analysis to generate rules that will encapsulate all time study steps authored. Also, it is observed that the time study steps used for testing the rules mapped onto 12 MTM tables out of the 22 MTM tables present. This indicates that only a subset of the MTM mapping rules has been tested. Therefore, further testing of time study steps, covering a wide range of activities, is required to determine the overall accuracy of the rules.

CHAPTER SIX: CONCLUSIONS AND FUTURE WORK

This chapter provides a summary of the thesis by reviewing the research objective and the tools developed to address them. The broader reach of the research work is presented. This chapter also identifies certain limitations of the developed tools and provides a brief discussion on future work.

6.1 Summary of tools developed to address the research objectives

This thesis presents the development of tools to extract information from assembly process sheets and transform the information into knowledge to support decision making. The tools address each of the research objectives.

6.1.1 Research Objective One: Automated extraction of knowledge to develop Standard vocabulary

The first tool extracts the information from process sheets using tools and techniques from Natural Language Processing (NLP) and Machine Learning (ML). The tool integrates techniques from NLP and ML to extract information; in this case verbs contained in work instruction text, and generate a standard vocabulary for authoring work instructions. A standard vocabulary of thirty one verbs is developed. Along with the standard vocabulary, an object type classifier is developed that assigns an object type to the objects. The object type classifier is validated using random split method and cross validation. The accuracy is found to be 94%. The development of the tools is discussed in Chapter Four and the necessary background to the NLP and ML techniques is presented in Chapter Three.

6.1.2 Research Objective Two: Automated generation of MTM mapping rules

The second research objective is addressed by the development of the tool to automatically generate rules that map process descriptions to MTM tables. The MTM mapping rules provide decision support to the planner while estimating assembly times. The MTM mapping rules are developed using time study information from existing process sheets. The machine learning platform, WEKA, is employed to generate the rules using decision tree classifiers. The development of the tool to generate MTM mapping rules is presented in Section 4.2**Error! Reference source not found.**.

The accuracy of the MTM rules are validated, in Chapter Five, using 71 time study steps and the accuracy of mapping is found to be 84.6%.

6.1.3 Research Objective Three: MTM table generator

The tools developed to address the first two research objectives are integrated and a decision support system is developed that allows the planners to author work instructions in free form text and provides MTM tables suggestions for each work instruction. The decision support system is developed to enable testing of the MTM mapping rules. The tool also demonstrates how NLP techniques can be used to read work instructions and provide MTM table suggestions to the planner.

6.2 Broader impact

This research lays a framework to show how Natural Language Processing (NLP) tools and techniques can be used to extract information from unstructured text data. The use of NLP techniques presented in this thesis to extract information regarding verbs and objects from process sheets can be extended to obtain any information contained within

the process sheets. NLP tools and techniques provide an opportunity to automate the process of extracting textual information from technical documents written using natural language. This automation will considerably reduce the amount of effort to generate knowledge required to develop decision support systems. In many multinational organizations, a large number of technical documents are hand written using natural language thereby requiring techniques that are capable of analyzing and interpreting the information. This thesis addresses one such issue encountered for a specific OEM. The use of NLP can also be leveraged to translate process descriptions into other natural languages.

The application of Machine Learning (ML) to develop MTM mapping rules demonstrates the use of Artificial Intelligence (AI) in flexible manufacturing systems. ML is capable of replicating the domain knowledge of an expert by analyzing historical data and developing models that mimic the decision making process of a human. Systems have access to a large network of other systems and data. In a global organization, each member is connected to every other member through a network of systems. Utilizing the accessibility to information from various sources, intelligent systems can be developed to support decision making process

Peterson [4] standardized the TVG authorship process through the use of text element structures in the controlled language. This methodology minimizes human error and regulates a set format, but it does so at the cost of restricting the planner's input. The planners cannot freely author work instructions. Also, controlled language for authoring of process sheets requires additional training for planners and frequent updating of the

system to accommodate variations. The GUI of the MTM table generator presented in this research allows planners partial, if not fully, free-form authorship of work instructions. This approach attempts to reduce the gap between a restricted controlled language and unrestricted free-form syntax, while still restricting the planner from ambiguous and inconsistent work instruction authoring.

6.3 Future Work

The work instruction text and time study steps, required to develop the tools, is obtained from process sheets that are present in Portable Document Format (PDF). Since Natural Language Toolkit (NLTK) does not support PDF files, the information is extracted from the process sheets, pre-processed, and exported to a text file. This is performed manually. To move towards a more automated process of extracting information, the system should be capable of obtaining the required information from a database containing process sheets and pre-processing it to the desired format.

The standard vocabulary presented in this thesis is developed by extracting the verbs from existing process sheets. The list of verbs is further pruned to generate a standard vocabulary of verbs that is sufficient to describe all the work instructions. The pruning of the verbs is performed manually. Latent Semantic Analysis (LSA) is a NLP technique that deals with grouping concepts that are similar to each other. This functionality can be employed to group verbs that are synonyms of each other or convey similar meaning.

During validation of MTM mapping rules, it is observed that certain mapping between the standard verbs and MTM tables do not exist. To encapsulate all existing

relationships, additional process sheets must be analyzed. Also, additional sources of information regarding objects used during the assembly must be analyzed to improve the accuracy of the object type classifier. The system should be dynamic in nature, such that as new process sheets are authored, MTM mapping rules and object type classifier are automatically generated and updated.

The MTM table generator only provides suggestion regarding the MTM table to the planner based on the work instruction authored. One area of future work is to further augment the tool to provide the planner with complete MTM information including MTM code and time units. To estimate the assembly time further information regarding the part attributes such as weight and size, the quantity of parts required, the distance travelled by the associate, and the motion of the associate is required. The first step towards developing an integrated system is to identify the sources of information and extract the required data to further narrow down the selection to a single MTM code.

REFERENCES

[1] Rychtyckyj N., 1999, "DLMS : Ten Years of AI for Vehicle Assembly Process Planning," National Conference on Artificial Intelligence, pp. 821–828.

[2] Rychtyckyj N., Standard Language at Ford Motor Company : A Case Study in Controlled Language Development and Deployment, Dearborn, MI.

[3] Rychtyckyj N., 2005, "Intelligent manufacturing applications at Ford Motor Company," NAFIPS 2005 2005 Annual Meeting of the North American Fuzzy Information Processing Society, pp. 298–302.

[4] Peterson M. G., 2012, "Standardization of Process Sheet Infomration to Support Automated Translation of Assembly Instructions and Product-Process Coupling," M.S. thesis, Department of Mechanical Engineering, Clemson University, SC.

[5] Renu R. S., 2013, "Decision Support Systems for Assembly Line Planning Modular Subsystems for a Large-Scale Production Management System," M.S. thesis, Department of Mechanical Engineering, Clemson University, SC.

[6] Maynard H. B., Stegemerten G. J., and Schwab J. L., 1948, Methods-time measurement, McGraw-Hill, New York, NY, USA.

[7] Meziane F., Vadera S., Kobbacy K., and Proudlove N., 2000, "Intelligent systems in manufacturing: current developments and future prospects," Integrated Manufacturing Systems, **11**(4), pp. 218–238.

[8] Feldmann K., and Slama S., 2001, "Highly flexible Assembly – Scope and Justification," CIRP Annals Manufacturing Technology, **50**(2), pp. 489–498.

[9] Huang Y. F., and Lee C. S. G., 1989, "Precedence knowledge in feature mating operation assembly planning," Proceedings of the 1989 International Conference on Robotics and Automation.

[10] Mantegh I., and Darbandi N. S., 2010, "Knowledge-based Task Planning Using Natural Language Processing for Robotic Manufacturing," Proceedings of the ASME/IDETC CIE 2010, pp. 1–8.

[11] Rychtyckyj N., 2002, "An assessment of Machine Translation for Vehicle Assembly Process Planning at Ford Motor Company," Association for Machine Translation in the Americas, pp. 207–215.

[12] Abdullah T. A., Popplewell K., and Page C. J., 2003, "A review of the support tools for the process of assembly method selection and assembly planning," International Journal of Production Research, **41**(11), pp. 2391–2410.

[13] Rychtyckyj N., 2005, "Ergonomic Analysis for Vehicle Assembly Using Artificial Intelligence," AI Magzine, **26**(3), pp. 41–50.

[14] Zhao J., and Masood S., 1999, "An Intelligent Computer-Aided Assembly Process Planning System," The International Journal of Advanced Manufacturing Technology, **15**(5), pp. 332–337.

[15] Miller M. G., 2011, "Product and Process based Assembly Time Estimnation In Engineering Design," M.S. thesis, Department of Mechanical Engineering, Clemson University, SC.

[16] Boothroyd G., Knight W., Inc B., and Wakefield R., 1993, "Design for assembly," IEEE Spectrum, **30**(9), pp. 53–55.

[17] Tan A., 1999, "Text Mining : The state of the art and the challenges," Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases, **8**, pp. 65–70.

[18] Huai Y., 2011, "Study on ontology-based personalized user modeling techniques in intelligent information retrievals," 2011 IEEE 3rd International Conference on Communication Software and Networks, pp. 204–207.

[19] Monostori L., Kumara S. R. T., and Váncza J., 2006, "Agent-Based Systems for Manufacturing," CIRP Annals - Manufacturing Technology, **55**(2), pp. 697–720.

[20] Jurafsky D., and Martin J. H., 2000, Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics, Prentice Hall.

[21] Choudhary A. K., Harding J. A., and Tiwari M. K., 2008, "Data mining in manufacturing: a review based on the kind of knowledge," Journal of Intelligent Manufacturing, **20**(5), pp. 501–521.

[22] Chowdhury G. G., 2005, "Natural language processing," Annual Review of Information Science and Technology, **37**(1), pp. 51–89.

[23] Nadkarni P. M., Ohno-Machado L., and Chapman W. W., 2011, "Natural language processing: an introduction.," Journal of the American Medical Informatics Association, **18**(5), pp. 544–551.

[24] Chowdhury G., 2005, “Natural language processing,” Fifth International Conference on Hybrid Intelligent Systems HIS05, **37**(1), pp. 460–471.

[25] Lash A. V., 2013, “Computational Representation of Linguistics Semantics for Requirement Analysis in Engineering Design,” M.S. thesis, Department of Mechanical Engineering, Clemson University, SC.

[26] Klein D., and Manning C. D., 2003, “Accurate Unlexicalized Parsing” *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pp. 423-430.

[27] Bird S., Klein E., Loper E., and Bird C. S., 2009, Natural Language Processing with Python, O’Reilly Media, Incorporated.

[28] Petrov S., and Klein D., 2007, “Improved Inference for Unlexicalized Parsing,” North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2007), pp. 404–411.

[29] Marcus M. P., Santorini B., and Marcinkiewicz M. A., 1993, “Building a Large Annotated Corpus of English : The Penn Treebank,” Computational Linguistics, **19**(2), pp. 313–330.

[30] Marneffe M. De, Maccartney B., and Manning C. D., 2006, “Generating Typed Dependency Parses from Phrase Structure Parses,” Linguistics in the Netherlands, Citeseer, pp. 449–454.

[31] Perkins J., 2011, Python Text Processing with Nltk 2.0 Cookbook: LITE Edition, Packt Publishing.

[32] Manning, Christopher D., Raghavan P., and Schütze H., 2008, “Introduction to information retrieval”, Cambridge: Cambridge University Press.

[33] Ikonomakis M., Kotsiantis S., and Tampakas. V., 2005, “Text Classification Using Machine Learning Techniques,” *WSEAS Transactions on Computers*, **4**(8), pp. 966–974.

[34] Pedregosa F., Weiss R., and Brucher M., 2011, “Scikit-learn : Machine Learning in Python,” *The Journal of Machine Learning Research*, **12**, pp. 2825–2830.

[35] Hsu C., Chang C., and Lin C., 2010, “A Practical Guide to Support Vector Classification”.

[36] Joachims T., 1998, “Text Categorization with Support Vector Machines: Learning with Many Relevant Features” pp. 137-142, Springer Berlin Heidelberg

[37] Alpaydin E., 2004, Introduction to machine learning, MIT press.

[38] Cortes C., and Vapnik V., 1995, "Support-Vector Networks," *Machine learning*, **297**, pp. 273–297.

[39] Wang B. X., Japkowicz N., Ave K. E., and A P. O. B. S., "Boosting Support Vector Machines for Imbalanced Data Sets," *Knowledge and Information Systems*, **25**(1), pp. 1–10.

[40] Akbani R., Kwek S., and Japkowicz N., 2004, "Applying Support Vector Machines to Imbalanced Datasets," *Machine Learning: ECML 2004*, Springer Berlin Heidelberg, pp. 39–50.

[41] Phung S. L., 2009, "Learning pattern classification tasks with imbalanced data sets," pp. 193–208.

[42] Holmes G., Donkin A., and Witten I. H., 1994, WEKA: a machine learning workbench, *Intelligent Information Systems, Proceedings of the 1994 Second Australian and New Zealand Conference on* IEEE, pp. 357–361.

[43] Witten I. H., Frank E., Trigg L., Hall M., Holmes G., and Cunningham S. J., 1999, "Weka : Practical Machine Learning Tools and Techniques with Java Implementations," Proceedings of ANNES'99 International Workshop on emerging Engineering and Connectionnist-based Information Systems, pp. 192–196.

[44] Hall M., Frank E., Holmes G., Pfahringer B., Reutemann P., and Witten I. H., 2009, "The WEKA data mining software: an update," SIGKDD Explorations, **11**(1), pp. 10–18.

[45] Garner S. R., 1995, "WEKA: The Waikato Environment for Knowledge Analysis," Proceedings of the New Zealand computer science *research students conference*, pp. 57–64.

[46] Frank E., Hall M., Trigg L., Holmes G., and Witten I. H., 2004, "Data mining in bioinformatics using Weka," Bioinformatics, **20**(15), pp. 2479–2481.

[47] Entezari-maleki R., Rezaei A., and Minaei-bidgoli B., "Comparison of Classification Methods Based on the Type of Attributes and Sample Size," *JCIT*, ***4***(3), pp. 94-102.

[48] Drazin S., 2010, "Decision Tree Analysis using Weka," Machine Learning-Project II, University of Miami, pp. 1–3.

[49] Zhao Y., and Zhang Y., 2008, "Comparison of decision tree methods for finding active objects," Advances in Space Research, **41**(12), pp. 1955–1959.

[50] Quinlan J. R., 1993, C4.5: Programs for Machine Learning, Morgan Kaufmann.

[51] Rajput A., Aharwal R. P., Dubey M., Saxena S. P., and Raghuvanshi M., 2011, "J48 and JRIP Rules for E-Governance Data," International Journal of Computer Science and Security (IJCSS), **5**(2), p. 201.

APPENDICES

Appendix A: Standard verb vocabulary and MTM mapping rules validation

The following table presents the entire standard verb vocabulary along with definition, examples and OPR classification.

S. No	Verb	Definition	Example	OPR class
1	Align	Accurate Positioning of a part or tool over another part	Align bumper to BIW	M
2	Apply	Putting on a medium on an object with or without the aid of a tool	Apply headlight seal initial	M
3	Attach	Setting or binding two parts with each other using only the features on each part	Attach hook to ARB	M
4	Clean	Includes all performances, to clean an object with a tool.	Clean windshield with wipe	M
5	Connect	Includes all activities to connect/ locking or unlocking a cable, with or without tool.	Connect cable to harness	M
6	Disengage	Unlocking a fixture or removing a part from the fixture or tool.	Disengage the fixture / Remove Jig	M, ZH
7	Engage	Locking a fixture or engaging a tool onto a part.	Engage a fixture or clamp.	M, ZH
8	Exchange	Involves exchanging empty bins containing parts and supplies with full bins.	Exchange container nuts	M, ZH
9	Get	Picking up a part or tool from around 1 m or does not necessitate getting up or walking from position.	Get torque tool	M, ZH
10	Handstart	Screwing in 2 rounds, the bolt or nut by hand or with the aid of tools, to set it in position.	Handstart first screw on tool holder at lift assist	M
11	Insert	Includes all activities to assemble clips with hands and/or tool	Insert clip to Y-strut	M
12	Inspect	Carrying out a check on a part or process, in order to make a decision.	Inspect bumper for damages	M, PF
13	Lay	Laying a cable by hand and/or	Route Bowden cable	M

S. No	Verb	Definition	Example	OPR class
		fastening exactly		
14	Move	Moving with/without a part/tool around the car or actions like bending down, squatting.	Move to front bumper	M
15	Open (Preparatory)	Includes all activities to handle packaging, separating layers and opening package to take contents.	Open bag with tool	ZH
16	Operate	Operating is to getting control over adjusting elements with a hand or foot and performing a single operation or a combined operation.	Operate to lower EMS onto hook	M, ZH
17	Place	Position a part or tool that is already in hand and requires no additional walking	Place ems hanger on third coil	M
18	Press(Switch/button)	Pushing a button or switching on a control to operate a tool.	Press button to release	M, ZH
19	Push	Manipulating a tool or part to align or start motion.	Push seat into place	M
20	Read	Reading information carrier, data cards to comprehend the information.	Read option list	M
21	Remove (Preparatory)	Includes all activities to handle packaging, separating layers and opening package to take contents.	Remove flex layer	M, ZH
22	Remove	Take a part off an assembly or piece of a part.	Remove a round cut out	M
23	Restock	Refilling storage containers, toolboxes and/or containers.	Restock rivets to carts	ZH
24	Restrict	Bind or guard cables, wires, electrical components etc.	Restrict cables.	M
25	Scan	Includes all activities to mark an object with a marking device or to document an object with a scanner.	Get scanner and scan label on IP skin	M
26	Screw in	Involves screwing in a bolt or nut completely with hand.	Screw in by hand total depth	M
27	Secure	Securing a cable with	Secure cable for	M

S. No	Verb	Definition	Example	OPR class
		stationary or moveable fastening elements. With or without tools.	foglight	
28	Snap	Clipping in parts with clips and onto other parts	Snap I- Panel Finisher into console stack	M
29	Tighten	Fastening screws and bolts with manual tools or torque tools.	Tighten 4 off screws with torque tool.	M
30	Unscrew	Unscrewing bolts/nuts manually or with help of a tool.	Unscrew adjuster 3 half turns 3mm gap	M
31	Walk	Walk from car body to car body or supply area without picking up part or any action. (and) Walk to supply area to pick up a part.	Walk to cart and back	ZW

The seventy one time study steps extracted from TVGs to test the accuracy of the MTM mapping rules are shown below in tabular format.

S. No.	Time study step	MTM table (from TVG)	MTM table (from MTM table generator)	Check
1	READ VEHICLE INFORMATION AS REQUIRED ALL PROCESSES MODEL, 4UBA.	Read	Read	Y
2	PLACE PROTECTOR CARRIED FROM CAR TO CAR DURING WALK TO NEXT CAR REMOVAL SEPARATE.	Place	Place	Y
3	GET AND PLACE PLUG.	Get and Place	Get and Place	Y
4	GET AND PLACE SEAT BELT RECEIVER.	Get and Place	Get and Place	Y
5	WALK FROM CAR TO PARTS AND THEN BACK.	Body Motions	Body Motions	Y
6	GET REAR SEAT AND PLACE IN	Get and Place	Get and Place	Y

S. No.	Time study step	MTM table (from TVG)	MTM table (from MTM table generator)	Check
	CAR INITIAL.			
7	GET AND TURN SEAT UP AND PLACE UNDER BRACKETS.	Get and Place	Get and Place	Y
8	GET REAR SEAT AND PLACE IN CAR INITIAL.	Get and Place	Get and Place	Y
9	GET AND TURN SEAT UP AND PLACE UNDER BRACKETS.	Get and Place	Get and Place	Y
10	WALK TO CART THEN CAR.	Body Motions	Body Motions	Y
11	REMOVE BAGS FROM BETWEEN SEATS / SEPARATE AND DISCARD.	Get and Place	Get and Place	Y
12	PRESS BUTTON ON LIFT ASSIST FOR 3RD ROW SEATS / THEN BACK WHEN DONE.	Operate	Operate	Y
13	PRESS FORWARD SWITCH /GRAB TRIGGER UNDER HANDLE.	Operate	Operate	Y
14	PLACE LIFT TO SEAT / THEN MOVE ACROSS TO FINAL POSITION.	Place	Handling Auxiliary Materials \ Tools	N
15	APPLY PRESSURES TO STOP LIFT AND THEN PUSH OVER.	Motion Cycles	Motion cycles	Y
16	PRESS SWITCH FOR DOWN AND SWITCH FOR CLAMP.	Operate	Operate	Y
17	PT (TIME FOR CLAMPS TO CLOSE).	Process	Verb does not exist	Rule does not exist
18	PRESS SWITCH TO RAISE SEAT OFF LIFT TABLE.	Operate	Operate	Y
19	PT (TIME TO RAISE SEAT UP TO CLEAR TABLE).	Process	Verb does not exist	Rule does not exist
20	PRESS LATCH SWITCH / PRESS REVERSE SWITCH.	Operate	Operate	Y
21	PRESS ROTATE SWITCH.	Operate	Operate	Y

S. No.	Time study step	MTM table (from TVG)	MTM table (from MTM table generator)	Check
22	APPLY PRESSURE TO TURN LIFT.	Motion Cycles	Motion cycles	Y
23	PT (TIME TO ROTATE SEAT).	Process	Verb does not exist	Rule does not exist
24	PRESS BRAKE BUTTON.	Operate	Operate	Y
25	GET AND PLACE PROTECTOR FROM LIFT TO C-PILLAR ON CAR.	Get and Place	Get and Place	Y
26	PRESS CLAMP SWITCH AND DOWN BUTTON.	Operate	Operate	Y
27	PT (UNCLAMP).	Process	Verb does not exist	Rule does not exist
28	APPLY PRESSURE TO START AND STOP LIFT.	Motion Cycles	Motion cycles	Y
29	PRESS UP BUTTON.	Operate	Operate	Y
30	PT (TIME FOR SEAT TO RAISE).	Process	Verb does not exist	Rule does not exist
31	PRESS LATCH SWITCH.	Operate	Operate	Y
32	APPLY PRESSURE TO SWING LIFT AROUND.	Motion Cycles	Motion cycles	Y
33	PRESS FORWARD SWITCH.	Operate	Operate	Y
34	PRESS BRAKE BUTTON.	Operate	Operate	Y
35	READ SEQ NUMBER ON RACK TO ENSURE IT IS THE CORRECT ONE.	Read	Read	Y
36	MOVE TO PRESS CYCLE BUTTON AND BACK.	Body Motions	Body Motions	Y
37	EXCHANGE CARTS PUSH CYCLE BUTTON.	Operate	Handling Containers	N
38	OPEN LATCH HOLDING PALLET WITH SEAT.	Operate	Verb does not exist	Rule does not

S. No.	Time study step	MTM table (from TVG)	MTM table (from MTM table generator)	Check
				exist
39	PRESS BUTTON TO RAISE LIFT TABLE.	Operate	Operate	Y
40	GET AND PULL PALLET WITH SEAT ONTO TABLE.	Get and Place	Get and Place	Y
41	PUSH BUTTON TO ACTIVATE SEAT STOP ON LIFT TABLE.	Operate	Operate	Y
42	PUSH EMPTY PALLET BACK ONTO CART AFTER SEAT REMOVED.	Get and Place	Working with Clips	N
43	APPLY PRESSURE TO HELP GUIDE SLIDES OFF AND ON SEAT RACK.	Motion Cycles	Motion cycles	Y
44	PUSH BUTTON TO RELEASE SEAT STOP ON LIFT.	Operate	Operate	Y
45	PRESS BUTTON TO LOWER TABLE.	Operate	Operate	Y
46	PT (TIME FOR TABLE TO LOWER).	Process	Verb does not exist	Rule does not exist
47	INSPECT PARTS.	Visual Control	Visual Control	Y
48	WALK TO GET BAG ON BACK OF RACK AND BACK AVERAGE 1 TIME PER RACK.	Body Motions	Body Motions	Y
49	GET AND HOLD BAG WITH ONE HAND.	Get and Place	Get and Place	Y
50	GET AND PULL VELCRO OPEN WITH OTHER HAND .	Get and Place	Get and Place	Y
51	WALK TO CAR WITH PARTS.	Body Motions	Body Motions	Y
52	GET AND PLACE TO CARRY FROM CAR TO CAR .	Get and Place	Get and Place	Y
53	MOVE BRACKETS ON SEAT UP.	Operate	Body Motions	N
54	GET AND PLACE PROTECTOR FROM LIFT TO C-PILLAR ON CAR.	Get and Place	Get and Place	Y
55	INSPECT PART.	Visual	Visual Control	Y

S. No.	Time study step	MTM table (from TVG)	MTM table (from MTM table generator)	Check
		Control		
56	WALK TO GET BAG FROM LIFT AVERAGE AND BACK TO FRONT OF RACK (1 TIME PER PACK).	Body Motions	Body Motions	Y
57	OPEN VELCRO FLAP ON BAG.	Get and Place	Verb does not exist	Rule does not exist
58	GET AND PLACE SECOND SET TO SEAT ON RACK TEMPORARILY (1 TIME PER PACK).	Get and Place	Get and Place	Y
59	GET BOTH BOLSTER AND PLACE IN CAR.	Get and Place	Get and Place	Y
60	WALK TO CAR TO PLACE PARTS.	Body Motions	Body Motions	Y
61	OPEN PACK.	Preparatory Activities	Verb does not exist	Rule does not exist
62	EXCHANGE CARTS PUSH CYCLE BUTTON.	Operate	Handling Containers	N
63	OPEN LATCH HOLDING PALLET WITH SEAT.	Operate	Verb does not exist	Rule does not exist
64	PRESS BUTTON TO RAISE LIFT TABLE.	Operate	Operate	Y
65	GET AND PULL PALLET WITH SEAT ONTO TABLE.	Get and Place	Get and Place	Y
66	PUSH BUTTON TO ACTIVATE SEAT STOP ON LIFT TABLE.	Operate	Operate	Y
67	PUSH EMPTY PALLET BACK ONTO CART AFTER SEAT REMOVED.	Get and Place	Working with Clips	N
68	MOVE WITH LIFT ONCE SEAT IS LOADED AND TURN.	Body Motions	Body Motions	Y
69	MOVE TO CAR AND BACK TO	Body Motions	Body Motions	Y

S. No.	Time study step	MTM table (from TVG)	MTM table (from MTM table generator)	Check
	PLACE PROTECTOR.			
70	MOVE SEAT INTO CAR.	Body Motions	Body Motions	Y
71	READ SEQ NUMBER ON RACK TO ENSURE IT IS THE CORRECT ONE.	Read	Read	Y

Appendix B: Python program scripts

This appendix contains the entire code to develop the NLP tools and techniques.

1. Code for extraction of verbs and objects from parsed work instruction text

```
import nltk
from nltk.tokenize import *

text=open('wi_parsed.txt','r').read()

tokenizer = RegexpTokenizer('\s+', gaps=True)
text_token = tokenizer.tokenize(text)
j = [item for item in range(len(text_token)) if text_token[item] == './.']
verbs = []
verb = ''
space=' '
obj_join=''
objects = []
for current_index in j[0:]:

  if (text_token[current_index+1]== 'The/DT') & (text_token[current_index+2]==
       'associate/NN') & (text_token[current_index+3]== 'must/MD'):

    master_index=current_index+4
    for vb in text_token[master_index:] :
      if vb.endswith('/VB'):
        verb += vb.split('/')[0]
        if text_token[master_index+1].endswith('/RP') :
          verb += space
          verb += text_token[master_index+1].split('/')[0]
        break


      else :
        break

    if verb == '':
      continue
```

```
        else :
            verbs.append(verb)
            verb = ''

            found = False

            for obj in text_token[master_index:] :

                if obj.endswith('/JJ') or obj.endswith('/NN') or obj.endswith('/NNS') :
                    found = True
                    obj_join += obj.split('/')[0] + space
                    continue


                else :
                    if found :
                        obj_join=obj_join.rstrip()
                        objects.append(obj_join)
                        obj_join=''
                        break
                    else :
                        continue




    else :
        print 'error'


results=[]
results.append(verbs)
results.append(objects)
print results

import csv

item_length = len(results[0])

with open('verb_obj2.csv', 'wb') as test_file:
  file_writer = csv.writer(test_file)
```

```
 for i in range(item_length):
   file_writer.writerow([x[i] for x in results])
```

2. Code for developing object type classifier

```
import numpy as np
import pandas as pd
import sklearn
from sklearn import cross_validation
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm.sparse import LinearSVC
import csv
import pickle

labeleddata = pd.read_csv("training_data_oversampling.csv")
target = labeleddata["Object type"]
data = labeleddata.ix[:,:-1]
x_train, x_test, y_train, y_test = cross_validation.train_test_split(data,target,
        test_size=0.4, random_state=17)

x_train = [item for sublist in x_train for item in sublist]
x_test = [item for sublist in x_test for item in sublist]

ngram_vectorizer = CountVectorizer(analyzer='char_wb',ngram_range=(5,5),min_df=1)
x_train_count = ngram_vectorizer.fit_transform(x_train)
x_train_vector = ngram_vectorizer.transform(x_train).toarray()
x_test_count = ngram_vectorizer.fit_transform(x_train)
x_test_vector = ngram_vectorizer.transform(x_test).toarray()
base_clf = sklearn.svm.LinearSVC(class_weight= 'auto')
clf = OneVsRestClassifier(base_clf).fit(x_train_vector, y_train)

f = open('my_classifier.pickle', 'wb')
pickle.dump(clf,f)
f.close
```

3. Code for extraction of verbs, objects and MTM table name from parsed time study steps

```
import nltk
from nltk.tokenize import *
```

```
import csv

text= open('ts_parsed.txt','r').read()

tokenizer = RegexpTokenizer('\s+', gaps=True)
text_token = tokenizer.tokenize(text)
j = [item for item in range(len(text_token)) if text_token[item] == './.']
verbs = []
table = []
space=' '
verb = ''
obj_join=''
tbl_join=''
objects = []
for current_index in j[0:]:

  if  (text_token[current_index+1]==  'The/DT')  &  (text_token[current_index+2]==
       'associate/NN') & (text_token[current_index+3]== 'must/MD'):

    master_index=current_index+4
    for vb in text_token[master_index:] :
      if vb.endswith('/VB'):
        verb += vb.split('/')[0]
        if text_token[master_index+1].endswith('/RP') :
          verb += space
          verb += text_token[master_index+1].split('/')[0]

        temp_index=master_index
        while (text_token[temp_index] != 'MTM/NNP'):
          temp_index +=1
          if  (text_token[temp_index]==  'and/CC')  &
      (text_token[temp_index+1].endswith('/VB')):
            verb += space
            # check if its verb or not
            verb += text_token[temp_index+1].split('/')[0]
            break

          else :
            continue
```

```
else :
   break
verbs.append(verb)
verb = ''

found= False
for obj in text_token[master_index:] :
   if obj.endswith('/JJ') or obj.endswith('/NN') or obj.endswith('/NNS') or obj
 =='of/IN' :
      found = True
      obj_join += obj.split('/')[0] + space
      continue


   else :
      if found :
         obj_join=obj_join.rstrip()
         objects.append(obj_join)
         obj_join=''
         test = 'lrb not encountered'
         break
      else :
         if (obj !='-LRB-/-LRB-'):
            test = 'lrb not encountered'
            continue
         else :
            test = 'lrb encountered'
            del verbs[-1]
            break
if (test == 'lrb encountered'):
   continue
else :
   count=0
   for tbl in text_token[master_index:] :
      count+=1
      if (tbl == 'MTM/NNP'):
         break

   found2 = False
   new_index=master_index+count
   for tbl in text_token[new_index:] :
```

```
            if (tbl !='-RRB-/-RRB-') :
               found2 = True
               tbl_join += tbl.split('/')[0] + space
               continue

            else :
               if found2 :
                  tbl_join=tbl_join.rstrip()
                  table.append(tbl_join)
                  tbl_join=''
                  break



   else :
      print 'error'

results=[]
results.append(verbs)
results.append(objects)
results.append(table)
print results

item_length = len(results[0])
with open('ts_full2.csv', 'wb') as test_file:
  file_writer = csv.writer(test_file)
  for i in range(item_length):
    file_writer.writerow([x[i] for x in results])
```

4. Code for developing MTM table generator

```
import nltk
from nltk.tokenize import *
import Tkinter
from Tkinter import *
import stanford_parser
from stanford_parser.parser import Parser
import numpy as np
import pandas as pd
import sklearn
from sklearn import cross_validation
```

```
from sklearn.feature_extraction.text import CountVectorizer
#from sklearn.multiclass import OneVsRestClassifier
#from sklearn.svm.sparse import LinearSVC
import csv
import pickle
root = Tkinter.Tk()
root.title("MTM table generator")
root.geometry('650x300+200+200')

def restructure_wi():
   raw_wi = input_wi.get('0.0', END)
   restructured_wi = "The associate must "+raw_wi.lower()
   restructured_wi = restructured_wi.replace('\n','')
   restructured_wi = restructured_wi.replace('.','. The associate must ')
   restructured_wi = restructured_wi[:-21]
   parsing(restructured_wi)

   return


def parsing(restructured_wi):

        stanford_parser = Parser()
        parsed_wi = stanford_parser.justTags(restructured_wi)
        extract_verb_object(parsed_wi)
        return


def extract_verb_object(parsed_wi):
        text = parsed_wi
        tokenizer = RegexpTokenizer('\s+', gaps=True)
        text_token = tokenizer.tokenize(text)
        text_token.insert(0, './.')
        print text_token

        j = [item for item in range(len(text_token)) if text_token[item] == './.']
        verbs = []
        verb = ''
        space=' '
        obj_join=''
        objects = []
```

```
for current_index in j[0:]:

   if (text_token[current_index+1]== 'The/DT') & (text_token[current_index+2]==
'associate/NN') & (text_token[current_index+3]== 'must/MD'):

      master_index=current_index+4
      for vb in text_token[master_index:] :
         if vb.endswith('/VB'):
            verb += vb.split('/')[0]
            if text_token[master_index+1].endswith('/RP') :
               verb += space
               verb += text_token[master_index+1].split('/')[0]
            break


         else :
            break

      if verb == '':
         continue
      else :
         verbs.append(verb)
         verb = ''

         found = False

         for (e,obj) in list(enumerate(text_token[master_index:])) :


            if obj.endswith('/JJ') or obj.endswith('/NN') or obj.endswith('/NNP') or
obj.endswith('/NNS') or obj =='of/IN' :
               found = True
               obj_join += obj.split('/')[0] + space

               if (e+1) == len(text_token[master_index:]) :
                    obj_join=obj_join.rstrip()
                    objects.append(obj_join)
                    obj_join=''
                    break

               else :
```

```
                    continue


                else :
                    if found :
                        obj_join=obj_join.rstrip()
                        objects.append(obj_join)
                        obj_join=''
                        break
                    else :
                        continue



            else :
                print 'error'


        results=[]
        results.append(verbs)
        results.append(objects)
        print results
        object_classifier(results)
        return


def object_classifier(results):
        labeleddata = pd.read_csv("training_data_oversampling.csv")
        target = labeleddata["Object type"]
        data = labeleddata.ix[:,:-1]
        x_train, x_test, y_train, y_test = cross_validation.train_test_split(data,target,
        test_size=0.4, random_state=17)

        x_train = [item for sublist in x_train for item in sublist]

        ngram_vectorizer                                                    =
        CountVectorizer(analyzer='char_wb',ngram_range=(5,5),min_df=1)


        f= open('my_classifier.pickle', 'rb')
        clf = pickle.load(f)
```

```
        f.close()

        test_list = results[1]
        print results[1]
        test_set = np.array(test_list)
        test_set_count = ngram_vectorizer.fit_transform(x_train)
        test_set_vector = ngram_vectorizer.transform(test_set).toarray()
        list_obj_type = []
        list_obj_type = clf.predict(test_set_vector)
        MTM_rules(results,list_obj_type)
        return

def MTM_rules(results,list_obj_type):
        sverb = results[0]
        list_obj = results[1]
        obj_type = list_obj_type
        print list_obj_type
        MTM_table_list = []
        print sverb

        for count in range(len(sverb)):

                if sverb[count] == 'get' :
                        if obj_type[count] == 'Part' or obj_type[count] == 'Plant item' or
        obj_type[count] == 'Fixture' :
                                MTM_table = 'Get and Place'
                        elif obj_type[count] == 'Tool':
                                MTM_table = 'Handling Auxiliary Materials \ Tools'
                        elif obj_type[count] == 'Consumable':
                                MTM_table = 'Working with Adhesives'
                        else :
                                MTM_table = 'No MTM table found / MTM rule does not
        exist'

                elif sverb[count] == 'operate' :
                        if obj_type[count] == 'Part' :
                                MTM_table = 'Operate'
                        elif obj_type[count] == 'Tool' :
                        MTM_table = 'Handle Tool'
                else :
                        MTM_table = 'No MTM table found / MTM rule does not exist'
```

```
        elif sverb[count] == 'attach' :
                if obj_type[count] == 'Part' :
                        MTM_table = 'Working with Clips'
                elif obj_type[count] == 'Plant item' or obj_type[count] == 'Fixture'
or obj_type[count] == 'Tool' :
                        MTM_table = 'Get and Place'
                elif obj_type[count] == 'Consumable':
                        MTM_table = 'Working with Adhesives'
                else :
                        MTM_table = 'No MTM table found / MTM rule does not
exist'


        elif sverb[count] == 'move' :
                MTM_table = 'Body Motions'

        elif sverb[count] == 'place' :
                if obj_type[count] == 'Part' or obj_type[count] == 'Plant item' or
obj_type[count] == 'Fixture' or obj_type[count] == 'Consumable' :
                        MTM_table = 'Place'
                elif obj_type[count] == 'Tool':
                        MTM_table = 'Handling Auxiliary Materials \ Tools'
                else :
                        MTM_table = 'No MTM table found / MTM rule does not
exist'


        elif sverb[count] == 'push' :
                if obj_type[count] == 'Part' or obj_type[count] == 'Plant item' or
obj_type[count] == 'Fixture' or obj_type[count] == 'Consumable' :
                        MTM_table = 'Working with Clips'
                elif obj_type[count] == 'Tool':
                        MTM_table = 'Operate'
                else :
                        MTM_table = 'No MTM table found / MTM rule does not
exist'


        elif sverb[count] == 'align' :
                MTM_table = 'Place'
```

```
        elif sverb[count] == 'disengage' :
                MTM_table = 'Operate'

        elif sverb[count] == 'press' :
                MTM_table = 'Operate'

        elif sverb[count] == 'apply' :
                MTM_table = 'Motion Cycles'

        elif sverb[count] == 'walk' :
                MTM_table = 'Body Motions'

        elif sverb[count] == 'inspect' :
                MTM_table = 'Visual Control'

        elif sverb[count] == 'engage' :
                MTM_table = 'Operate'

        elif sverb[count] == 'clean' :
                MTM_table = 'Cleaning'

        elif sverb[count] == 'read' :
                MTM_table = 'Read'

        elif sverb[count] == 'insert' :
                MTM_table = 'Working with Clips'

        elif sverb[count] == 'remove (preparatory)' :
                MTM_table = 'Preparatory Activities'

        elif sverb[count] == 'remove' :
                if obj_type[count] == 'Part' or obj_type[count] == 'Plant item' or
obj_type[count] == 'Fixture' or obj_type[count] == 'Tool' :
                        MTM_table = 'Get and Place'
                elif obj_type[count] == 'Consumable':
                        MTM_table = 'Preparatory Activities'
                else :
                        MTM_table = 'No MTM table found / MTM rule does not
exist'
```

```
        elif sverb[count] == 'connect' :
                MTM_table = 'Laying Cables'

        elif sverb[count] == 'handstart' :
                MTM_table = 'Working with Screws \ Bolts'

        elif sverb[count] == 'tighten' :
                MTM_table = 'Handling Auxiliary Materials \ Tools'

        elif sverb[count] == 'unscrew' :
                MTM_table = 'Motion Cycles'

        elif sverb[count] == 'restock' :
                MTM_table = 'Parts Supply'

        elif sverb[count] == 'lay' :
                MTM_table = 'Laying Cables'

        elif sverb[count] == 'scan' :
                MTM_table = 'Marking and Documenting'

        elif sverb[count] == 'exchange' :
                MTM_table = 'Handling Containers'

        elif sverb[count] == 'secure' :
                        MTM_table = 'Handling Auxiliary Materials \ Tools'
                else :
                        MTM_table = 'No MTM table found / MTM rule does not
exist'

        else :
                MTM_table = 'Verb does not exist'


        MTM_table_list.append(MTM_table)
        MTM_table = ''
        print MTM_table_list
        display_ts(MTM_table_list)


return
```

```
def display_ts(MTM_table_list):
        output_ts.delete('0.0', END)
        a = input_wi.get('0.0', END)
        output_ts.insert('0.0', a)

        for i in range(len(MTM_table_list)):
                output_ts.insert('%d.end' %(i+1), "[MTM table -")
                output_ts.insert('%d.end' %(i+1), MTM_table_list[i])
                output_ts.insert('%d.end' %(i+1), "]")
        return

input_wi = Text(height = 8, wrap = WORD)
input_wi.insert(INSERT, "Enter WI text here...")
input_wi.place(relx= 0, rely = 0)

submitbutton = Button(text="SUBMIT", fg="black", activebackground = "blue",
        command = restructure_wi)
submitbutton.place(relx= 0.45, rely= 0.45)

output_ts = Text(height = 8, wrap = WORD)
output_ts.insert(INSERT, "Output window")
output_ts.place(relx = 0, rely =0.55 )

root.mainloop()
```