



GRADUATE SCHOOL
EAST TENNESSEE STATE UNIVERSITY

East Tennessee State University
Digital Commons @ East
Tennessee State University

Electronic Theses and Dissertations


Student Works

5-2021

PLPrepare: A Grammar Checker for Challenging Cases

Jacob Hoyos
East Tennessee State University

Follow this and additional works at: <https://dc.etsu.edu/etd>

 Part of the [Computational Linguistics Commons](#), [Computer Sciences Commons](#), [Morphology Commons](#), and the [Slavic Languages and Societies Commons](#)

Recommended Citation

Hoyos, Jacob, "PLPrepare: A Grammar Checker for Challenging Cases" (2021). *Electronic Theses and Dissertations*. Paper 3898. <https://dc.etsu.edu/etd/3898>

This Dissertation - unrestricted is brought to you for free and open access by the Student Works at Digital Commons @ East Tennessee State University. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Digital Commons @ East Tennessee State University. For more information, please contact digilib@etsu.edu.

PLPrepare: A Grammar Checker for Challenging Cases

A thesis

presented to

the faculty of the Department of Computing

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Science in Computer Science, Applied Computer Science

by

Jacob Hoyos

May 2021

Dr. Ghaith Husari, Chair

Dr. Brian Bennett

Dr. Christopher Wallace

Keywords: natural language processing, dependency parsing, foreign language education,
arbitrary cases, rule-based morphology detection, hybrid grammar checking

ABSTRACT

PLPrepare: A Grammar Checker for Challenging Cases

by

Jacob Hoyos

This study investigates one of the Polish language's most arbitrary cases: the genitive masculine inanimate singular. It collects and ranks several guidelines to help language learners discern its proper usage and also introduces a framework to provide detailed feedback regarding arbitrary cases. The study tests this framework by implementing and evaluating a hybrid grammar checker called PLPrepare. PLPrepare performs similarly to other grammar checkers and is able to detect genitive case usages and provide feedback based on a number of error classifications.

DEDICATION

This thesis is dedicated to the memory of my friend Christopher Nitzband, who was the first person to introduce me to programming.

Copyright 2021 by Jacob Hoyos

All Rights Reserved

ACKNOWLEDGEMENTS

I would like to thank my amazing Polish professors for teaching me this difficult and beautiful language and ultimately equipping me to pursue this study. I offer my heartfelt gratitude to Pani Jola (Dr. Jolanta Lion – Carnegie Mellon) and Pani Agnieszka (Dr. Agnieszka Jezyk – University of Toronto) for patiently guiding me through the language. I also would like to thank Dr. Oscar Edgar Swan – University of Pittsburgh, whose interest in Polish education paved the way for people like me to learn.

I would also like to thank my committee and especially Dr. Ghaith Husari for helping me out and offering guidance through these two years of hard work and Dr. Brian Bennett for being a great friend throughout my academic career.

Finally, I would also like to thank both Herr Brown (Ken Brown – Alcoa High School) and Frau Negrisanu (Raluca Negrisanu – ETSU) for jump-starting my interest in foreign languages.

TABLE OF CONTENTS

ABSTRACT.....	2
ACKNOWLEDGEMENTS.....	5
LIST OF TABLES.....	8
LIST OF FIGURES.....	9
Chapter 1. Introduction.....	10
1.1. Genitive Case and the Genitive Masculine Inanimate Singular.....	10
1.2. Linguistic Vocabulary and Related Concepts.....	11
1.1.1. Some assorted definitions:.....	13
1.3. Natural Language Processing.....	13
1.4. Significance of the Study.....	14
1.5. Overview of the Study.....	16
Chapter 2. Related Research in Linguistics, Polish NLP, and Education.....	17
2.1. Linguistic Significance.....	17
2.1.1. Challenges from a Linguistic Perspective.....	17
2.1.2. Implication of Related Work.....	18
2.2. NLP Tools and Other Resources in the Polish Language.....	22
2.2.1. Dictionaries.....	22
2.2.2. Machine Learning Models.....	25
2.2.3. Dependency Parsers and SpaCy’s Polish NLP Suite.....	30
2.2.4. Feedback Systems.....	32
2.3. Statement of Research.....	34
Chapter 3. Design and Implementation of PLPrepare: A Grammar Checker Specializing in Providing Feedback to Language Learners.....	35
3.1. Guidelines.....	35
3.1.1. Selecting Guidelines and Finding the Words They Govern.....	35
3.1.2. Finding Word Lists.....	36
3.2. Filtering the Wordlists.....	37
3.2.1. PLList.....	37
3.2.2. Processing Word Lists Using PLList.....	39
3.3. Finding the Genitive in a Sentence using Grammar Rules and Dependency Parsing ..	42
3.3.1. PLPrepare Pipeline.....	42

3.3.2. Collecting Grammar Rules	43
3.3.3. Dependency Parsing.....	47
3.3.4. Mapping Syntactic Dependency Tokens to Grammar Rules	49
3.3.5. Confirming the Candidates	51
3.3.6. Grammar Rule Edge Cases and Limitations	53
3.4 Generating Automated Feedback Using PLPrepare	55
3.4.1. Detecting Grammatical Errors	56
3.4.2. Classification of Grammatical Errors	56
3.4.3. Generating the Feedback.....	60
Chapter 4. Performance and Evaluation	63
4.1. Gathering and Generating the Test Data.....	63
4.1.1. Sentence Collection Using NKJP	63
4.1.2. Sentence Preprocessing.....	66
4.1.3. Grammatical Error Injection	67
4.2. Evaluation Experiments	68
4.3. Results.....	69
4.3.1. Genitive Detection Results	70
4.3.2. Grammatical Error Classification Results	72
4.3.3. Generated Feedback Results	78
4.3.4. Results from the Perspective of the User	79
Chapter 5. Conclusion and Future Work	81
References.....	82
VITA.....	87

LIST OF TABLES

Table 1. Genitive Case Declension with the Masculine Inanimate Singular in Bold	11
Table 2. PLPrepare Compared with Other Language Learning Tools.....	15
Table 3. PLPrepare Compared with Recent Grammar Checkers.....	15
Table 4. SpaCy’s Polish Model Performance [32].....	31
Table 5. Total Number of Examples Collected for each Category	36
Table 6. A Simplified Example of a PoliMorf Entry	40
Table 7. The -a vs -u Accuracy Scores for the Guidelines Selected for this Study	41
Table 8. List of Rules Governing the Genitive Case in Iwona Sadowska’s Polish: A Comprehensive Grammar [1]	45
Table 9. Example of Contents of a Rule’s Word List	45
Table 10. Genitive Rules in the Database Requiring Word Lists	47
Table 11. SpaCy Lemmatizer Performance with Incorrect Entries. Correct Results in Bold.....	57
Table 12. NKJP Test Entry Preprocessing	66

LIST OF FIGURES

Figure 1. High Level Architecture for PLList.....	38
Figure 2. PLPrepare Pipeline	43
Figure 3. Simplified Pipeline	43
Figure 4. Example Output of SpaCy’s Dependency Parser – Figure Rendered by DisplayCy ..	48
Figure 5. Grammar Tree: Grammar Rules Requiring Additional Parsing in Red	50
Figure 6. Comparison of a Negated and Non-Negated Verb That Does Not Take an Accusative Compliment	54
Figure 7. Grammatical Error Classification Decision Tree	60
Figure 8. Displaying Feedback from the Rule Verification Stage Regarding the Use of ‘Z’	60
Figure 9. Reporting a Case Error Where ‘Wodorem’ Should be in the Genitive Case Due to Possession by ‘Nadtlenku’	61
Figure 10. Informing the User that a Word is Misspelled	61
Figure 11. Displaying Feedback for a Postfix Error with a Guideline Suggestion.....	61
Figure 12. Group 1 Gender Distribution – For Testing Genitive Usage Detection – Corrected for Dictionary Retrieval Ambiguations	65
Figure 13. Group 2 Gender Distribution – For Testing Error Classification – Corrected for Dictionary Retrieval Ambiguations.....	65
Figure 14. Genitive Usage Detection Performance by Test Set in Group 1	71
Figure 15. Error Distribution for the Error Classification Stage.....	73
Figure 16. Recall for each Test Set	73
Figure 17. Precision for each Test Set	74
Figure 18. Total -a vs -u Feedback Found Per Test	78

Chapter 1. Introduction

Learning languages is a difficult process. Languages consist of a broad spectrum of grammatical constructs that are not always very intuitive. Several languages have seemingly arbitrary grammar rules that make learning incredibly challenging. In particular, the Polish language is rife with these grammar rules, the most common of which is the masculine inanimate genitive singular case. This study proposes a language learning tool that uses natural language processing (NLP) techniques to detect mistakes in this case and offers feedback using guidelines to help language learners and their instructors navigate this difficult case.

1.1. Genitive Case and the Genitive Masculine Inanimate Singular

Iwona Sadowska describes case as a “grammatical concept that through a set of different endings attached to the normal forms explains who does what to whom without heavily relying on word order” [1]. These endings can give the same noun many different meanings in the Polish language and have an almost immeasurable impact on interpreting the semantics of the language. As such, it is a vital aspect of the morphology of the Polish language. Case shows the relationship between words and their roles in the sentence. Each case has a set of possible endings belonging to that case. The ending used is based on a series of qualities including etymology, gender, animacy, and the pronunciation of the stem. One can often predict case endings in Polish based on phonology. The masculine inanimate singular in the genitive case, however, proves to be a troublesome exception. Table 1 shows this particular declension pattern in relation to the rest of the genitive case declension patterns.

Table 1. Genitive Case Declension with the Masculine Inanimate Singular in Bold

Genitive Case Declension [2]				
	Singular		Plural	
Gender	Ending	Change	Ending	Change
Masculine Men	not in -a in -a	-a -y/i	Most Nouns	-ów
Masculine Animate	Most Nouns	-a		
Masculine Inanimate	Most Nouns	-u/-a		
Feminine	Most Nouns	-y/-i	After Hard Stems After Soft Stems	-∅ -y/-i
Neuter	Most Nouns	-a	Most Nouns After Some Soft Stems After -um	-∅ -y/-I No Change

The genitive case in Polish shows a lack, an absence, a possession, or a quantity in addition to several other uses. The genitive case in Polish roughly corresponds to the use of ‘of’ in English. The masculine inanimate singular in the genitive case can take a few different endings, but, for the vast majority of nouns, the ending will either be -a or -u [1][2][3]. Stems could take other endings that include the adjectival ending -ego (for nouns that decline like adjectives), or a -∅ ending (for nouns that the Polish inflectional system cannot accept). The last two cases are much easier to predict than the first two. Regarding the -a and -u endings, choosing between the two seems incredibly arbitrary [3], but it does follow some very specific rules that vary widely in their consistency [1]. The result is a grammatical case in which the inflection is nearly impossible to predict using concrete grammar rules.

1.2. Linguistic Vocabulary and Related Concepts

This study refers to various linguistic concepts, explaining some specific concepts as the need arises. However, the general concepts are important to understand.

The most referenced area of linguistic study in this study is morphology. Morphology is the “study of the forms of words, and the ways in which words are related to other words of the same language” [4]. While there are many types of morphology, this study refers to inflectional morphology when using the general term. Inflectional morphology is the study of the formation of words that relate to inflectional categories. An inflectional category could be gender, number, case, or anything that helps one understand the grammatical structure of a word in relation to others. This concept is not to be confused with declension. Declension constitutes a form of inflection; inflection is a broader term. For example, conjugation is inflection that applies to verbs, but declension does not apply to verbs but to nouns, pronouns, adjectives, etc.

Inflectional morphology collects and binds different forms of a word together. These collections are called lexemes. In other words, a lexeme is a set of forms related by inflection [5]. These lexemes have an agreed-upon name that is called a lemma. An easy example is the English lexeme ‘run.’ The lexeme contains the forms ‘run,’ ‘runs,’ ‘ran,’ and ‘running.’ All of these forms are related by inflection, and the lexeme chooses ‘run’ as the lemma. The lemma will almost always correspond to the nominative (subject/dictionary) form of the lexeme regarding Polish nouns. The notion of a lemma and a stem are often confused. A lemma is a member of the lexeme that linguists choose to represent the whole lexeme, but a stem is a part of a word with no affixes or postfixes. In English, it is common that a word’s stem shares the same spelling as the lexeme (e.g. ‘runs’ → ‘run,’ stemming reveals the lemma and the stem). The distinction is very important in Polish because this is not often the case (e.g. ‘końca’ → ‘koniec,’ the word must be lemmatized).

Finally, this study examines sentences using a morphosyntactic paradigm. Meaning the syntax and morphology of the words in the sentence are both important. Where inflectional

morphology relates words in the context of a lexeme, syntax relates them in the context of a broader sentence structure [6]. The specific syntactic dependency relationships that this entails are discussed where relevant, as there are far too many to discuss them all here.

1.1.1. Some assorted definitions:

- Clitic – A morpheme that has its own syntactic characteristics, but it relies on a ‘host’ for its pronunciation [13].
- Digraph – A pair of characters that represent a single sound.
- Lexicon – The set of all of the lexemes in a language [5].
- Modal Verb – A verb that expresses modality and requires the infinitive form of the verb it is connected to.
- Morpheme – The smallest set of characters that has any meaning in a language.
- Postfix – An attachment to the stem of a word, usually indicates some inflectional information.
- Phoneme – The smallest phonetic unit that can produce a different word.

1.3. Natural Language Processing

Jacob Eisenstein defines Natural Language Processing (NLP) as “... the set of methods for making human language accessible to computers.” [8]. NLP is a computer science field that is closely intertwined with linguistics and has applications spanning the spectrum of daily life. There is a difference between NLP and computational linguistics, however. NLP is most concerned with the analysis of algorithms and the design of systems that can help computers process natural languages. In contrast, computational linguistics uses algorithms and systems to learn more about natural languages and their constructs. This study uses a pipeline of NLP techniques, most importantly, stemming, lemmatization, and dependency parsing.

Stemming and lemmatization are extremely similar. Stemming reduces a form of a word to the bare stem as described in section 1.2. Lemmatization, on the other hand, reduces a form of a word to the lemma. In NLP, lemmatization is often preferred because it guarantees that a real word will be returned (ex. ‘Chłopca’ → ‘Chłopiec’), whereas a form’s stem may not always be a member of a lexeme (ex. ‘Chłopca’ → ‘Chłop’ or ‘Chłopc’ depending on the algorithm). Stemming techniques can involve algorithms that use statistical models, or they can use finite-state transducers to describe character-level rules, like the Porter Stemmer [9]. Lemmatization techniques can vary between statistical models that try to guess the lemma and simple lookups. The statistical models can predict lemmas from unseen lexemes, whereas a lookup is limited by the number of lexemes it can access.

The third technique used extensively in this study is dependency parsing. Dependency parsing is the act of making attachments between words in a sentence in the form of a directed graph [8]. A dependency parse will result in a spanning tree where the edges describe a syntactic dependency. The head of this syntactic dependency determines what type of dependency the relationship will be. Eisenstein uses the example of nouns as being the head of noun phrases and verbs at the head of verb phrases. The tail (or dependent) describes the subtree of dependencies that have the head as its parent. This technique is incredibly important for resolving the syntactic information necessary to discern where grammar structures ought to be used. For more discussion regarding dependency parsers, see sections 2.2.3 and 3.3.3.

1.4. Significance of the Study

The ultimate goal of this study was to introduce a framework for helping students learn arbitrary cases in natural languages. The study focused on the Polish genitive masculine inanimate singular because of the case’s prevalence and the amount of research available on the

case. To test the framework, this study introduced two tools: PLList – a word list management tool, and PLPrepare – a grammar checker designed to give grammar feedback to students when using the Genitive Case.

The framework that PLPrepare embodies serves to fill in the gaps left by other language learning tools. Table 2 shows a comparison between PLPrepare and other tools. The most common problem is that these language learning tools rely on user configuration. Both Quizlet and DuoLingo offer feedback in real time, but they do not explain grammar mistakes in real time. Grammar Checkers, on the other hand, do. The problem is that grammar checkers neglect guideline-based feedback for arbitrary cases, and they do not offer grammar feedback that targets foreign language learners. PLPrepare offers both. Table 3 differentiates between PLPrepare and some recent grammar checkers.

Table 2. PLPrepare Compared with Other Language Learning Tools

PLPrepare As a Language Learning Tool			
Service	Grammar Feedback?	Leverages Guidelines?	Coverage
Quizlet[10]	User-Configured, Real-Time	User-Configured	User-Dependent
DuoLingo[11]	Determined By Course Managers	Determined by Course Managers	Low
PLPrepare	Yes	Yes	Over 2 Million Supported Words

Table 3. PLPrepare Compared with Recent Grammar Checkers

PLPrepare As a Grammar Checker[12]					
Language	Year	Type	Error Explanation?	Leverages Guidelines?	Comprehensive?
Afran Oromo Amharic	2011	Rule-Based	NO	NO	YES
	2013	Rule-Based and Statistical Modes	NO	NO	YES
Nepali	2007	Rule-Based	NO	NO	YES
Portugeese	2003	Rule-Based	NO	NO	YES
Bangla	2006	Stastical	NO	NO	YES
Polish (PLPre-prepare)	2021	Hybrid	YES	YES	NO

PLPrepare is not a comprehensive tool, but unlike other tools, it aims to give real-time feedback to language learners struggling with arbitrary cases.

1.5. Overview of the Study

Chapter 1 introduced the problem and its importance. It also gave background information for easier understanding of the techniques that this study employs. Chapter 2 reviews the literature related to the problem. Chapter 3 presents the methodology used in the study and Chapter 4 reports and interprets the results. Chapter 5 offers a discussion regarding the researcher's conclusions and directions for future research.

Chapter 2. Related Research in Linguistics, Polish NLP, and Education

2.1. Linguistic Significance

The incredible linguistic strides that were taken to understand the genitive masculine inanimate singular, unfortunately, produce controversy. Within the field of linguistics, there are arguments regarding which model of childhood acquisition best describes reality. This section will explore some of the experiments performed by linguistic researchers to try to understand how Polish children end up learning this case. This will highlight the challenges that this case pose.

2.1.1. Challenges from a Linguistic Perspective

To illustrate the problem, consider the examples of different forms below:

Nominative Singular: Młot (Hammer)	-	Genitive Singular: Młota (-a)
Nominative Singular: Płot (Fence)	-	Genitive Singular: Płotu (-u)
Nominative Singular: Fotel (Armchair)	-	Genitive Singular: Fotela (-a)
Nominative Singular: Hotel (Hotel)	-	Genitive Singular: Hotelu (-u)

These pairs of words share final and penultimate consonants, they rhyme, and they are in the same grammatical gender. However, the pairs still possess different endings when considering the genitive singular. Because of this, there must be some model by which Polish speakers acquire this case. In this example, the choice between -u and -a appears completely arbitrary.

The arbitrariness involved in the genitive masculine inanimate singular takes its toll on the native speakers of the language. According to an experiment done by inter-lingual researchers on Polish, Estonian, and Finnish speaking children, Polish children made the most mistakes [13]. Of these mistakes, using an -a ending where a -u ending was required was the second most frequent mistake of those involving leaving the lemma in the Nominative Case (in other words, mistakes where the child did not attempt an inflection at all) were not counted. This

study paints a bleak picture regarding the acquisition of the genitive case until the actual ratio for correct vs incorrect inflections is revealed.

Another study focusing strictly on genitive case acquisition in children measured the overgeneralization error rate [3]. An overgeneralization error occurs when a child takes an inflectional pattern and applies it to a different lemma that produces an incorrect result [13]. An example using the forms listed above would be a child trying to apply an -a ending to ‘Młot’ because the child is familiar with the inflectional pattern of “Płot.” The overgeneralization error rate was an average of 2.02% when children formed the Genitive Case Masculine Inanimate [3]. This error rate is extraordinarily impressive especially for children, but there is a caveat that within this error rate. The majority of the overgeneralization errors involving the genitive case were on masculine inanimate singular nouns. This result shows the extent to which young children have mastered this form, but it also shows that there are still some improvements needed.

2.1.2. Implication of Related Work

These results suggest that children gain understanding by sub-consciously manipulating data, keeping track of frequency, pattern matching, and the intersection of all of them [13]. First, the exposure to the inflectional pattern is measured in Surface Form Frequency (SFF). The similarity of the patterns (and the forms) is measured in the Phonological Neighborhood Density (PND), sub-divided into two related measures: Form-Based and Class-Based PND. Form-Based PND references a specific form, whereas Class-Based PND looks at all forms across the whole morphological spectrum. An important element of Class-Based PND is that the similarity tracked also considers various criteria including phonological aspects, animacy and gender. The diversity

of the criteria used makes Class-Based PND a potent tool for children rapidly searching for patterns.

To contrast this finding for the first experiment, Dąbrowska suggests using the Dual Mechanism Theory to model this acquisition [3]. This theory describes that, when given an unfamiliar form, children will search their memories to find it. When they fail, they will try to respond with a default. This default is formed by examining clusters of irregular words in the memory and finding the most frequent form that works for that specific, often phonologically based, cluster. Dual Mechanism Theory has gained an enormous amount of traction when representing English past tense acquisition, but this is because transformations in the English past tense are phonologically based [3]. The morphology of the genitive masculine inanimate singular is not phonologically based, so additional factors must be considered.

Given the mental tools described by these different case acquisition models, children still navigate the arbitrary Masculine Inanimate Genitive Singular with great difficulty. However, they eventually eliminate the error through the combination of Class-Based PND and Surface Form Frequency. This interaction works based on labeling lemmas as friends or enemies based on the inflection pattern. Using the example of *Młot-Płot* as shown before, a child, knowing that *Płot* ends in an -u, would attempt to label *Młot* as an -u because of the similarity denoted in Class-Based PND. This error would be reflected in a sub-conscious classification of *Młot* as an enemy of that pattern in Form-Based PND. Once this process continues throughout a child's early development, the Surface Form Frequency reinforces the learned patterns and enemies of those patterns so a child will either know the form or could predict the form.

Despite this insight, there are still complications when predicting this case. One complication is mobile e (often denoted -e-) and that children innately search for a default ending

in inflectional systems [3]. The mobile e can be predicted in most forms, so it does not pose much of a barrier in the acquisition process. The search for a default, on the other hand, can lead to overgeneralization in favor of the -a ending. This generalization occurs because, when the genitive case is examined with Masculine Inanimate in addition to Masculine Animate and Masculine personal, the -a ending is overwhelmingly more common. The commonality causes children to overgeneralize patterns from the other parts of the masculine genitive into the masculine inanimate.

Despite this fact, Dąbrowska states that there was no evidence to suggest that any ending was treated as a default. This finding, along with each circumstance not being tied to a single ending, contradicts some central tenants in the Dual Mechanism Theory. As noted above, DMT suggests that overgeneralization errors are made when the irregular form is unknown, and the default was used in an attempt to guess the correct form. This proves that DMT's ability to describe Polish is somewhat diminished from its prevalence in the English language.

A third experiment conducted by Ewa Dąbrowska [14] gives some interesting results regarding this case. Dąbrowska [14] notes that around eighty percent of substance nouns take their endings in -u while object nouns take the -a endings at around the same rate. This finding gives another rule to help predict this case in a rule-based fashion. Dąbrowska's [14] experiment was designed to highlight this particular rule, asking children to produce target forms in both -u and -a in a conscious effort to see if the rule was taking hold in the children's minds. It is necessary to note that, as children develop, their skills in producing the correct forms increase dramatically as their brains produce rules that solidify. The results defied all expectations. The children avoided acquiring the abstract pattern that would improve prediction, and continued to make errors in choosing between -a and -u. To make matters worse, older children should have

been able to take advantage of these rules, but they still produced incorrect forms when even adapting according to the rules would have improved their performance. Despite all of this, the output was generally correct. The author argues that this indicates that correct usage is tied to memorization.

Regarding the problems with the models addressed so far, it is important to recognize that the acquisition of these cases does not occur all at once but is rather a gradient [15]. The rules vs. schemas debate at the heart of the issue regarding models has had an incredibly difficult time pinning down Polish case acquisition for some time. The debate is not new in the linguistics community. Krajewski [15] states there has not been a model that accounts for the distinctions between regular and irregular nouns, the contexts that create them, and how they can help choose the correct form, or even whether or not predicting the correct forms is even possible given the context. According to Krajewski [15], DMT is insufficient to model Polish correctly, but it does offer a way to navigate the irregularity of Polish. Navigation is possible because of how DMT models the storage of irregular inflections. The genitive case is part of why this compromise has to be made, as it is difficult to work out the criteria for inflection. Krajewski [15] does note that a schema's productivity is based on the variability of what has been produced in the past. The more a form is produced, the easier it is to adapt to new patterns.

2.1.3. Applications

In practice, the experiments discussed in the previous section reveal fundamental findings about this case. First, students cannot rely on phonetic rules to learn this case. Second, teachers cannot rely on a default ending to teach students adequately. Third, students must practice reinforcing knowledge and identifying the 'enemies' that Krajewski [15] mentions. Fourth, rules,

while not as important as exposure, are important for acquisition. Finally, it is indeed possible to acquire the case.

PLPrepare is a reaction to the truths revealed in these experiments. While this study makes no claims regarding a linguistic model for case acquisition, PLPrepare provides a framework for students to get feedback for essays, grammar drills, and other exercises to increase their exposure to new forms. The feedback itself addresses the need for rules and does not attempt to identify any default ending for the entire case.

2.2. NLP Tools and Other Resources in the Polish Language

Polish is a unique language that requires special treatment when it comes to using NLP. There are too many Polish grammatical structures that are not shared with more common languages, requiring additional NLP constructs [16]. The issue of transformations and the commonality of passive voice, adjectival participles, inflection of negated direct objects, valid double negatives, and more virtually guarantee NLP systems' inability to be adapted to Polish from any non-Slavic language.

Despite its particular needs, the Polish language has a multitude of parsers, spell checkers, valence dictionaries, and other tools that prove that researchers in the NLP field have not neglected Polish. There are limits to these tools concerning the focus of this study, however. As these tools are surveyed, their usefulness and shortcomings are examined.

2.2.1. Dictionaries

Two Polish mainstream language parsers integrate a valence dictionary called Walenty, so it is extraordinarily relevant to the field as a whole [17][18]. A valence dictionary produces a massive amount of inflectional information for each lemma passed into it [19][20]. Parsers can use this to evaluate the lemmas that they come across and produce productive output. Walenty

can map verbs to phrases, and it makes an extraordinary attempt at untangling complex Polish idioms. There are some problems, however. Linguistically speaking, Walenty is theory-neutral, which means that it does not adapt itself to modeling the output according to any particular theory. This neutrality improves its universality but reduces its effectiveness regarding this problem [18]. The output is both machine- and human-readable, but this output is not ideal. Additionally, the focus is on verbs and their roles in the sentence. Because of this, the inner workings of Walenty are not pertinent to the methodology of this project. It does, however, shed some light on one of the necessary components of a language parser. For example, dictionaries like Walenty can be used as an input source for several other components along the pipeline, including a formal grammar, which can then be used to generate a structure bank for referencing various syntactical elements and structures. All of these resources combined can be useful for morphological analysis [18].

Despite these shortcomings, Walenty also provides some insight into the case inflections that are common throughout Polish. Grammatical case is usually directly specified by some external factor which could be a verb, a sentence structure, adjectives, or even another noun [17]. As a consequence, the syntax surrounding grammatical cases is extraordinarily complex, which makes valence dictionaries very useful for analyzing the circumstances under which certain cases appear. The context and its syntactical impact may prove to be valuable in approaching the problem of the Polish genitive case. Unfortunately, the valence dictionary itself cannot tackle this problem alone, but it can work with a formal grammar model to navigate the difficulty of Polish syntax.

The core of this study is the morphological aspect. As such, an examination of Morphological Analyzers will shed light on how the existing tools tackle the problem of the

Masculine Inanimate Genitive Singular. Of the twelve morphological tools found by the PoliMorf developers, only a select few were of a quality fit for research, and of those, even fewer were large enough to provide sufficient coverage of the Polish lexicon [21]. This led these developers to create a new tool that merged SGJP and Morfologik, which are both morphological dictionaries developed using data from an eleven-volume dictionary of the Polish language. Scholars of the language manually annotated this data to make it fit for a morphological dictionary. These enormous dictionaries, when combined, could produce the morphological information for all cases – including the masculine inanimate genitive singular. This result did not come through a language acquisition model, machine learning, or any dynamically produced methodology [22]. Instead, it utilized a collection of around a thousand unique morphological patterns [21].

While it is useful to parse large groups of data that only require surface-level information [22], relational models used to generate inflectional patterns mean it is difficult for a human to determine causal relationships between a form and the grammar information that created it. What these models lead to is a dictionary that can decompose the morphological rules but cannot explain anything about reaching its conclusions. Also, it does not truly acquire the rules, but it stores patterns in a database. Thus, it cannot show why particular patterns are chosen while other seemingly valid patterns are less correct. To language learners, linguists, and students, the relational models used cannot reveal any useful patterns that improve their understanding of the language. PoliMorf is, however, an invaluable tool for generating the underlying morphological information needed to make sense of sentences. This study makes extensive use of the PoliMorf dictionary, which serves as the underlying grammar dictionary in the PLList system introduced in section 3.2.1.

2.2.2. Machine Learning Models

The actual process children use to acquire the Masculine Inanimate Genitive Singular is, conceptually, very close to the process of training a computer to perform a task using a neural network. Several attempts have been made, however, to predict various morphological elements based on a single word. The richness of the field of machine learning for re-inflection is astounding. These machine learning techniques provide insight into how making context-informed case predictions might be possible.

To begin, recently, recurrent neural networks have been a popular tool to perform morphological analysis [23]. There is also the possibility of using a linear model on untrained data sets, which offers cheaper computation at the cost of a slight amount of accuracy. The model described in Ç. Çöltekin, J. Barnes [23] utilized a linear support vector machine (SVM) to predict lemmas, the part-of-speech, and other morphological information. They accomplished this by using a pipeline structure that included an encoder that retained other models' recurrent bidirectional networks and placed the input into an embedding layer. Eventually, that information reached a part-of-speech (PoS) classifier that took advantage of two hidden layers with rectified linear units. From there, a softmax classifier was used to export a probability model. The morphological prediction involved the same basic steps as the PoS classifier. It made use of the same two-layer setup with rectified linear units. The difference is that it involved separating the analysis into distinct features. Each morphological feature was placed into several feed-forward networks.

This linear model was compared to a neural model that used 64 as the embedding size and utilized forward and backward gated recurrent units. This model learned 512-dimensional representations, which led it to have great performance [23]. The linear model performed

admirably and produced more accurate lemma predictions than the neural model, but it was still outperformed in almost every other avenue.

There is a difference between morphological analysis and morphological reinflection, however. This difference is best explained in two steps. The first step is morphological analysis and the second step is morphological generation [24]. In the model to be examined in this section, convolutional architectures were used. These architectures are powerful tools that are used to great effect in image processing, but they can be applied re-inflection. Typically, a well-performing convolutional neural network model is created by using a single decoder to avoid data sparsity. They also tend to use letter sequences in addition to convolution for input. Finally, some form of long short-term memory is used very deep in the pipeline to decode. Long short-term memory is used for processing sequences of data, which makes its use for strings of characters extremely common.

The model described above is an actual system developed by Robert Östling [24]. His experiments used a system that had two Long Short-Term Memory (LSTM) architectures tied together in a pipeline along with four Convolutional Layers and 256 LSTM units at the encoder level and the decoder level. These fed into even more layers of Convolutional filters until a hidden layer was finally reached. Morphological features were used as input to the decoder in the form of binary vectors. Once the time came for the decoding step in the pipeline, a beam search was used. Finally, an output string was generated. Models like this perform well when given a small amount of information, making them extraordinarily relevant for the purposes of predicting the Polish Genitive.

To introduce the enormous array of other models used in morphological re-inflection, the CoNLL-SIGMORPHON 2018 shared task on supervised learning of morphological generation

saw the use of several models using neural networks to predict morphological information [25]. These models utilized supervised learning on data sets acquired through Wiktionary [25], which provides almost every aspect of morphological data imaginable. The data were separated into high, medium, and low conditions, amounting to different levels of difficulty. The first of the tests involved taking the source form of a word and producing a target form via the morphosyntactic description. The second test involved producing the form from a sentential context. This test is more difficult than the earlier tests and also more relevant to predicting noun's inflectional forms. The second test requires the systems to produce a lemma in a given context. The test that was used to evaluate these systems was performed for 103 languages. For the harder sets of data, the performance was much lower, especially for Polish. One of the models that performed well, notably in Polish, was the Copenhagen System [26].

The Copenhagen System, like many of the other systems used in this experiment, made use of an encoder-decoder system. Like the system described in Çöltekin, J. Barnes [23], it utilized a recurrent neural network [26]. Something that increased its performance that this system combined the training data for the languages in this test. This system also treated the morphosyntactic description as an auxiliary task.

This system provided several novel solutions to common problems of efficiency and did so on the baseline system provided at CoNLL [26]. For example, when given the list of languages for the task, this system performed tuning on the groupings of the randomly selected languages and then utilized those languages for a different type of training. The system's core value is that it assumed that morphosyntactic features of languages are similar regardless of which language they belong to. This value provides an interesting philosophy, especially considering the differences that arise when examining languages of different families. This

approach complements the combination of multilingual tuning discussed earlier with monolingual training.

Another interesting part of this system is that the weights of the monolingual training sets were less than the multilingual training. This difference reinforces the core value that these languages share morphosyntactic features. The models from these experiments trained for fifty epochs, including validation. Also, for task 1, the passed context was stored in an LSTM. Once this training was complete and predictions were ready to be made, the system relied upon five models to generate the target form rather than a single model. This prevents overfitting. The most intriguing part of this system is the notion that it utilized an additional task to improve the prediction performance by not only predicting the form of a target but also by tagging it. The addition of tagging improved the performance by tying in a separate thread to the learning and giving the system more awareness of the distinct morphological features. Finally, most of the incorrect predictions were nowhere near the target form. Target forms mostly come from substituting a few letters near the end of the source form. The authors observe that their model did not effectively learn to copy the source forms and apply the necessary transformations very often [26]. They also observe that the issue could be fixed using techniques that push the system toward encoding by data augmentation. All of these adjustments contributed to excellent overall performance [26]. This performance gives evidence to the assumptions made by the authors and to the merit of their adjustments.

Another top performer in this task was the UZH system which set itself apart from the other contenders by using a beam search for the decoding process [27]. The encoder-decoder models tend to lag in these models, so this system (and many others) attempted to learn edit operations as a sequence instead of relying on string-to-string transductions. Also, this system

used an alignment step that was completely removed from the application of the edit sequence. These factors led to the system's success in these tasks and should be noted for future attempts at morphological prediction systems.

The UZH system used a transition-based transducer system that operates by performing a sequence of edits on individual characters in a string [27]. Each of these edits is made based on the history of all of the edits and the representation of the character edits. The Recurrent Neural Network has the task of choosing the proper edit out of several possible edits. This process is modeled using several time stamps. A probability is calculated for each of the possible edits using classifier weights and LSTM at a specific time. At a specific time, a probability is calculated for each of the possible edits using classifier weights and Long Short-Term Memory. At this time, the system generates a prediction of the loss for actions that take place after the transition has taken place. The prediction step has the effect of allowing the system to look ahead and determine the proper course of action while knowing the possible consequences of a choice.

One of the problems that this particular system was attempting to solve is the problem of using character alignment algorithms, which are incredibly important to a transducer's proper functioning [27]. They do have some problems, however. For example, actions that generate a productive result are referred to as Gold Actions. Character alignment algorithms typically generate such actions, but several sequences often create probability ties. At this point, these algorithms suffer because they could choose either action without any morphosyntactic description. In other words, they are not sensitive to certain information that ought to inform the choice, leading to removing this common step from the system.

Also, this model worked at an ascending gradient that performed two distinct phases over and over again. These are the roll-in and roll-out phases: First, the roll-in phase is sampled from

the set of all of the optimal edits found or selected from the set of possible edits that exist at that particular time. Once completed, the roll-out phase will perform the roll-in step for all of the input or perform that step for a possible action to be taken and stores the effect on the sequence level loss. Finally, this model was used in a group along with other models to avoid overfitting. The UZH model had excellent performance for re-inflection because of these adjustments to the common RNN architecture that is often deployed for these tasks. Furthermore, these improvements introduced the notion of a transition-based transducer system that critically breaks down these words in terms of transitions [27]. This breakdown is a clever idea that makes these systems more transition-aware – critical in a transition-heavy environment like morphological reinflection.

2.2.3. Dependency Parsers and SpaCy’s Polish NLP Suite

There are several dependency parsers available to the NLP community. The dependency parsers often train on dependency treebanks to generate specific models, so the dependency parsers discussed in this section can be trained on any language that a treebank exists for [28]. Dependency parsers use various approaches from using dynamic oracles to word embeddings and from using transition-based paradigms to using decomposition paradigms [29]. Broadly, they can use either a greedy or non-greedy approach. According to an experiment performed by Choi, Tetreault, and Stent [29], greedy parsers were consistently faster than their non-greedy counterparts. These parsers sacrificed accuracy for speed, however, as non-greedy solutions offered consistently better accuracy. In this experiment, the non-greedy Mate system had the best performance, but it was among the slowest competitors. The greedy SpaCy model was the fastest and had good performance for a greedy model (UAS of 89.61% in sentences with punctuation, where greedy

models averaged 88.5% UAS). From this experiment, SpaCy’s dependency parser appears to be the best of both worlds.

On top of the SpaCy dependency parser’s performance in the experiment described above, SpaCy continues to improve its dependency parser. In 2015, Honnibal and Johnson [30] improved their parser. The newer parser, like the old, was a greedy, non-monotonic transition-based dependency parser [30]. The novel approach here is creating a new non-monotonic transition set, which necessitates the use of Goldberg and Nivre’s [28] dynamic oracle. This approach was an improvement on previous greedy non-monotonic systems that improves the UAS score by 0.6% and bringing the directed accuracy measure up to 91.85%. These accuracy improvements combined with the speed of a greedy approach makes the SpaCy dependency parser an attractive choice for this project.

SpaCy offers pipelines for NLP processing in many languages, including Polish. In addition to dependency parsing, the SpaCy Polish model offers lemmatization, morphologizing, PoS tagging, and many more services [31]. This model is armed with the tools to tackle almost any problem involving NLP. The dependency parser model was trained using the dependency treebank developed by Alina Wróblewska, et. al. [32]. The pipeline itself was trained on lemmas from Morfeusz 2[33] and on material from the National Corpus of Polish (Narodowy Korpus Języka Polskiego) [34]. The SpaCy Polish pipeline reports the accuracy measurements as show in Table 4.

Table 4. *SpaCy’s Polish Model Performance [32]*

Selected SpaCy pl_core_news_lg Pipeline Performance Measurements		
Label	Description	Accuracy
POS_ACC	Part-of-Speech Tags	0.97
MORPH_ACC	N/A	0.88
LEMMA_ACC	N/A	0.89
DEP_UAS	Unlabelled Dependencies	0.89
DEP_LAS	Labelled Dependencies	0.84

2.2.4. Feedback Systems

In the realm of tools designed to give feedback, two areas that merit discussion. First is the grammar checker, which find grammar mistakes based on three general approaches [12]. The second is Kostikov's Adaptive Learning paradigm [16] [35]. This direction is very new and not widely known. Given this paradigm's goals and their similarity to those of this study, it is worth examining.

Grammar checkers can be classified into three approaches [12]. The first is the rule-based grammar checker, which provides feedback based on manually created rules. Rule-based grammar checkers involve a great deal of manual entry and configuration, but these remain the most common. The second is the data-driven grammar checker, which uses statistical techniques like PoS tagging combined with a corpus-based approach to compare the input to corpus data to determine correctness. Finally, the hybrid approach combines elements of the two. Typically, this means adding some concrete rule-based parsing to a statistical approach.

Grammar checkers will often follow a four-stage process:

1. Sentence Tokenization
2. Morphological Analysis
3. POS-Tagging
4. Parsing.

The sentence tokenization stage breaks the input into tokens and performs word segmentation to generate lexical information about each word in the sentence. Next, the morphological analysis stage finds word stems, and the PoS tagging stage determines which word in the sentence belongs to which PoS. Finally, the parsing stage uses this information and syntactic constraints to determine the structure of the sentence and mark any errors that violate the constraints. In a

survey of grammar checkers for ten languages, only three could offer a correction for an error, and none were designed for language learners. Furthermore, none were designed with arbitrary cases in mind [12].

One paradigm that is targeted toward language learners is Mykola Kostikov's Adaptive Learning approach [16]. Adaptive Grammar Learning attempts to predict nouns' inflectional forms using the lemma itself and some additional inputs [16]. The largest problem facing this model is the lack of accuracy. To make matters worse, this model has a particular weakness to grammatical exceptions. Another problem is the lack of research that surrounds this model. A few papers involve rule-based paradigms to model the transformations of grammar, but it is a model that requires more research.

The Adaptive Learning Model involves manually encoded sequences that are recorded [35]. Manual encoding presents the problem of how these sequences are checked for accuracy. An external factor must ensure that the manually encoded sequences are correct. External factors highlight this model's weaknesses regarding the necessity of user input. The author details that if sequences are entered incorrectly, they can simply be redone. Such loose error-handling does not inspire confidence regarding the correctness of the encoded sequences. However, it should be mentioned that this model's goal is to be used to aid language learners. Unfortunately, the prospects of piggybacking this model onto research-oriented models shrink with the amount of user input required. The goal is to provide visual aids to display the transformations that take place as lemmas are placed in different cases. This could prove to be very useful in determining a potential rule behind the Polish Genitive but given that these analyses do not provide any information as to why the forms change beyond a description given by a process of decomposition, its usefulness stops there.

Still, some intriguing concepts are addressed. First, the concern with the eventual transformation of a word instead of the immediate transformation is an interesting concept [35]. An eventual transformation describes that, given a word, several potential transformations could be applied to that word. This set of transformations becomes a set of building blocks to reach a description of that word's final transformations eventually. A similar idea presented is that a decomposition can take place to create a sequence of transformations and then describe what caused the previous transformation. The sequence itself does not give a complete explanation, but it instead determines the criteria surrounding a transformation. The criteria can give the learner at least some clue as to what caused the transformation. All of these extra ideas involved in producing forms can serve a purpose breaking down this problem.

2.3. Statement of Research

The motivating problem presented in this chapter was the need to provide language learners with a way to navigate and master arbitrary cases in languages. In the Polish language, the genitive masculine inanimate singular presents one of the largest arbitrary cases and one of the most difficult challenges for language learners. This study proposed an approach to construct a hybrid grammar checker that is primarily designed to provide detailed, accessible feedback to its users. This study produced the PLPrepare Hybrid Grammar Checker to test this approach.

Chapter 3. Design and Implementation of PLPrepare: A Grammar Checker Specializing in Providing Feedback to Language Learners

PLPrepare is a tool for finding and correcting errors that students make in the Genitive case. The system's end goal was to use NLP resources to mark and provide feedback for these errors. Accomplishing this goal required the system to be able to detect where the Genitive Case should be used, examine the word in that location, determine if that word has the correct morphology, and, if not, classify the error to provide feedback. Additionally, the arbitrary genitive masculine inanimate singular case required more information to produce meaningful feedback, so the system generated that feedback with the help of a guideline database.

3.1. Guidelines

Over time, Polish linguists have discovered patterns that can inform students' decisions regarding the genitive masculine inanimate singular case, so the system uses these patterns (referred to henceforth as guidelines) to arm them with the knowledge to avoid repeating those mistakes [1]. To provide feedback on the genitive masculine inanimate singular, a sample of guidelines was taken, lists of applicable words were filled, those lists were processed with a tool called PLList (Discussed in section 3.2.1.) to create a database of words that these guidelines govern.

3.1.1. Selecting Guidelines and Finding the Words They Govern

Because of the number of guidelines and the limited resources of this project, only a sample of guidelines were catalogued. The selected guidelines were those for which word lists could be filled easily. "Easily" here means that accessible, unambiguous lists could be found for those guidelines on collaborative pages such as wiki projects and other sources.

3.1.2. Finding Word Lists

The guidelines each specify a category of words describing similar things, body parts, buildings, and spices, for example. The problem with using these categories is that it is not always easy for the observer to know exactly where one category ends and another begins. To avoid merged categories, the word lists use pre-compiled lists that correspond to a category that fits the guideline’s meaning. Most of the wordlists were found on Wikipedia, where word lists were found by recording the names of pages according to Wikipedia categories. Those found on Wiktionary were already organized into a convenient list format. Table 5 lists the categories found, the raw number of found words, and the source.

Table 5. Total Number of Examples Collected for each Category

Selected Guidelines			
Guideline Category	Total Examples	Translated?	Source(s)
Body Parts	175	NO	[36]
Buildings	355	YES	[37]
Dances	216	NO	[38]
Days Of the Week	7	NO	[1]
Fruits	194	NO	[39][40]
Means of Transportation	396	NO	[41]
Measurements	211	NO	[42]
Months	12	NO	[1]
Spices	263	NO	[43][44]
Vegetables	260	NO	[45][46]

Table 5 introduces two sub-categories of word lists: those translated and those that did not originate from a wiki project. One potential problem was that these words could stray outside the bounds of the given guideline. To prevent straying outside the guideline, words were translated into English and checked against existing English word lists that did not exist on the corresponding Polish wiki page. The remaining problem is that Polish grammar does not always recognize categories like English does [1]. This lack of recognition could be considered an edge

case that is a potential subject for further research. Given the lack of a reliable Polish corroborating categorization, the English category suffices.

3.2. Filtering the Wordlists

The result of the previous stage was a selection of unfiltered word lists. In their unfiltered form, they are useless because they contained feminine and neuter nouns as well as masculine inanimate nouns. Additionally, they appeared in the nominative case, which is the subject or dictionary form [1][2]. The word lists must contain only masculine inanimate singular nouns in the Genitive case to provide the correct feedback to students and catalog the guideline' accuracy. At this stage, the word lists also contained duplicates, adjectives, and other superfluous information. A preprocessing stage was necessary to make these word lists usable.

3.2.1. PLList

The task of reviewing 2,089 words' grammatical properties is a time-consuming one, so the PLList system was developed to handle the task of preprocessing these lists. The PLList system provided a basic but user-friendly flag system that allowed for easy removal of unwanted items by allowing the user to specify filtering information in the form of a command line flag. At its core, PLList is a command-line tool that facilitates batch NLP operations on lists without requiring the user to learn PoliMorf's tagset [21] for identifying morphosyntactic tags.

The architecture of PLList is relatively simple; it stores PoliMorf's rows in an indexed SQLite database and uses the user's command and filter information to query PoliMorf for the morphosyntactic tags in each of the words in a given list. Then, the system processes the items according to the morphosyntactic tags retrieved and the specified command and filter information. PLList then outputs the results in the form of a list. Figure 1 shows an overview of PLList's architecture.

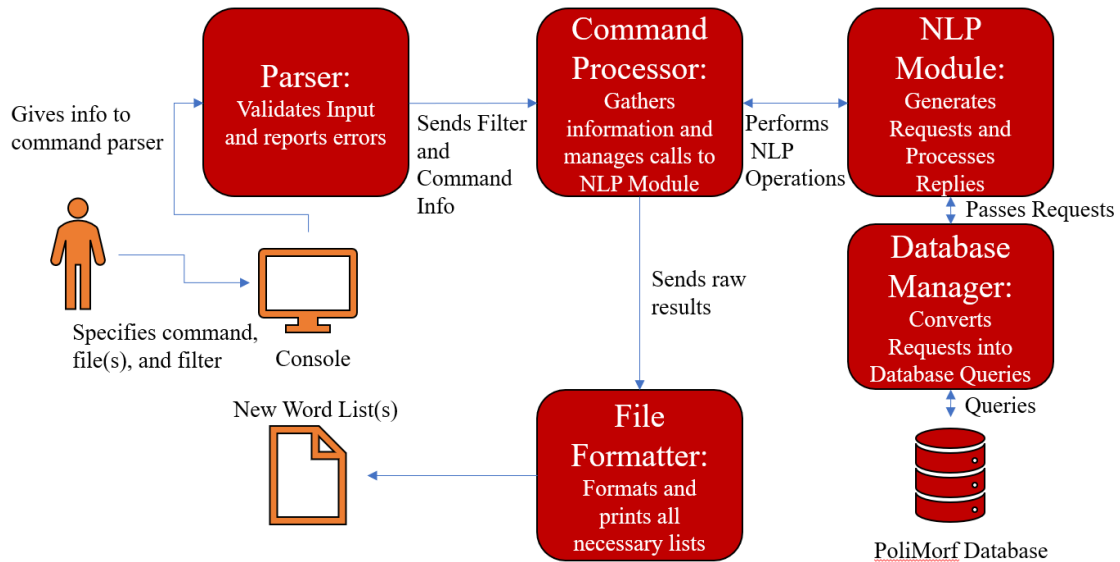


Figure 1. High Level Architecture for PLList

PLList performs the following basic tasks: simple filtering, applying a morphological case to the words in the list, retrieving a dictionary form of inflected words, retrieving the grammar information of the words, and filtering based on grammar information. These tasks are achieved by querying the PoliMorf database, which contains the grammar information necessary to accomplish these tasks. The user simply enters a command with a filter and a file to process, and the command is parsed. Once PLList recognizes a supported command, it checks the file. Then, it uses a regular expression table to translate from the user-friendly filtering information into a query compatible with PoliMorf's grammar tags. The command processor then manages all of the calls necessary to the NLP module to accomplish the task. The NLP module then gets feedback from the Database Manager and forwards that information back to the Command Processor, which sorts the output and sends the new lists to the File Formatter to be structured and written to the hard drive.

3.2.2. Processing Word Lists Using PLList

To remove all of the words that the guidelines do not govern, PLList had to remove all words in a list that are not masculine inanimate in gender or singular in number. It filtered out any word that is not a noun, and it inflected all of the words in the list into the genitive case. Also, it had to remove duplicates, classifying and descriptive adjectives, and any accompanying tokens that were present in the word list.

Before PLList could filter any words, it had to read the file correctly. The system preprocessed the word list as it scanned the file. The priority was to ensure that each entry in the list contained only one word. For this study, descriptive adjectives that inform the meaning of nouns were removed. One example is ‘okulary przeciwsłoneczne’ or sunglasses, the adjective ‘przeciwsłoneczne’ (lit. against the sun) is vitally important to the meaning of the noun but, in most cases, it does not affect morphosyntactic attributes of the lexeme ‘okulary.’ So, it was safe to remove these adjectives. Furthermore, the preprocessing stage stripped each word of things like white space and punctuation, and excess tokens are treated as separate words that the system will filter based on the user’s filter information. It was generally unsafe to remove tokens by length because several Polish prepositions and conjunctions are one letter long (a – contrasting and; i – inclusive and; o – about; w – in, at; etc.) [1].

After filtering, the word lists had to be reinflected using PLList’s ‘case’ command. PLList’s ‘case’ command accepts a target case, filter flags, and an input and output file. The command works by first generating a new word list according to the user filters for gender, number, etc. and then lemmatizing the words. PLList lemmatizes words by searching the database for the present form of the word and retrieving the lemma from the lexeme category. As a result, only the relevant words are present in the new word list. Then, the command module

sends the new word list to the NLP module which requests the rows from the database that match both the lexeme and the target case.

As an example, the vegetable list contains the word pomidor (tomato) (Table 6). If the accusative form of the word is present in the list, the lookup finds the accusative form pomidor (2nd row) and checks the morphosyntactic tags in the 3rd column of the figure against the user filters. The lexeme pomidor is then returned and used in a separate query to find the form in the 1st column that matches the “gen” or genitive field in the 3rd column (the reasons for performing this operation as two separate queries instead of one is discussed below). Once the result is returned, PLList adds the genitive form to the list and sends the list to the File Formatter to print.

Table 6. *A Simplified Example of a PoliMorf Entry*

Selected Rows of the PoliMorf Entry for Pomidor (Tomato)			
Form	Lexeme	Grammar Tags	Category
pomidor	pomidor	subst:sg:nom:m3	pospolita
pomidor	pomidor	subst:sg:acc:m3	pospolita
pomidora	pomidor	subst:sg:gen:m3	pospolita

Thus, if the user wants to put everything in the list into the genitive case and remove everything that is not a noun or masculine inanimate, then the command would look like this:

```
case VegetableList.txt VegetableListInGen.txt gen -d -p -m noun -g ma mn -n sg
```

The -m (mode) flag with the ‘noun’ argument tells PLList to filter out all rows that do not contain a common noun or a gerund (subst or ger). It is important to note that nouns that decline like adjectives (e.g. znajomy - acquaintance, Luty - February) are still considered nouns. The -g (gender) flag with the ‘ma mn’ argument filters out all rows that are not either masculine animate or masculine inanimate. The -n (number) flag with the ‘sg’ argument filters out all rows that are not singular. PLList compares this information against the morphosyntactic tags in the 3rd column in the figure below. The -d (duplicate) flag specifies that PLList should remove all duplicates and, finally, the -p (postfix) flag causes PLList to append postfix statistics at the end

of the list. PLList performed this process on each of the word lists to generate new, inflected wordlists.

Postfix statistics counted the postfix on each of the words in the new word lists and displayed a percentage breakdown of the postfixes that occurred in the list. This percentage breakdown was necessary for creating guideline entries in the database for providing feedback in future steps. Also, a further distinction was necessary in order to quantify the accuracy of the guidelines properly. These guidelines considered only a limited breadth of words, so it is inappropriate to count words governed by different guidelines. Examples included foreign nouns (e.g. kiwi - kiwi), nouns that decline like adjectives (e.g. Luty - February), and indeclinable nouns (e.g. dur – major (scale)). In other words, the results only considered those words for which the -a/-u choice is present. Table 7 shows the results of this stage.

Table 7. *The -a vs -u Accuracy Scores for the Guidelines Selected for this Study*

Selected Guidelines			
Guideline Category	Dominant Postfix	Accuracy	Number of Examples
Body Parts	-a	69.2%	52
Buildings	-u	69.9%	93
Dances	-a	71.6%	73
Fruits	-a	67.5%	42
Measurements	-a	75.0%	104
Months	-a	100.0%	11
Spices	-u	67.7%	66
Transportation	-u	51.1%	146
Vegetables	-a	58.9%	61

Multiple nouns share the same forms, for example: list (letter) in the nominative singular and list (lists) in the genitive plural. This is an ambiguous case, which is why the level of disambiguation depends on the accuracy and detoken value of the filters. Without any filter information, the system could recognize either as the correct form. The reason that the query was

split into two steps is twofold. The first reason is to reduce ambiguity that could stem from lexemes whose forms overlap. The second reason was to provide better feedback by first segmenting off the unusable words where PLList can classify them into further subcategories and print them if the user wishes.

3.3. Finding the Genitive in a Sentence using Grammar Rules and Dependency Parsing

After the sample guidelines were ready, the next stage was detecting where it was proper to use the genitive case in a sentence. For example, if a student entered the sentence “Nie ma jabłek (There are no apples),” it was easy to pick out the genitive because jabłek is the genitive plural of jabłko. The negative existential ‘nie ma’ requires a genitive predicate noun. Since this study aimed to provide students with feedback for incorrect usages, a student could just as easily have entered the sentence “Nie ma jabłkami,” which would replace the genitive plural with the instrumental plural. There is no way to parse the sentence as written natively and know that a given word should be in the genitive case. There had to be a process to break the sentence into its fundamental pieces and examine them to find the rules that require the genitive case.

3.3.1. PLPrepare Pipeline

PLPrepare is a tool that parses sentences for genitive case rule candidates, verifies them, and later compares the contents of the sentence to the requirements of the rules found. This tool was developed to provide detailed, automated feedback to students and teachers. It is the tool that determines where the genitive should be used in a sentence. Figure 2 shows the basic pipeline, which will be discussed at length in the following sections. The pipeline could be further simplified as shown in Figure 3.

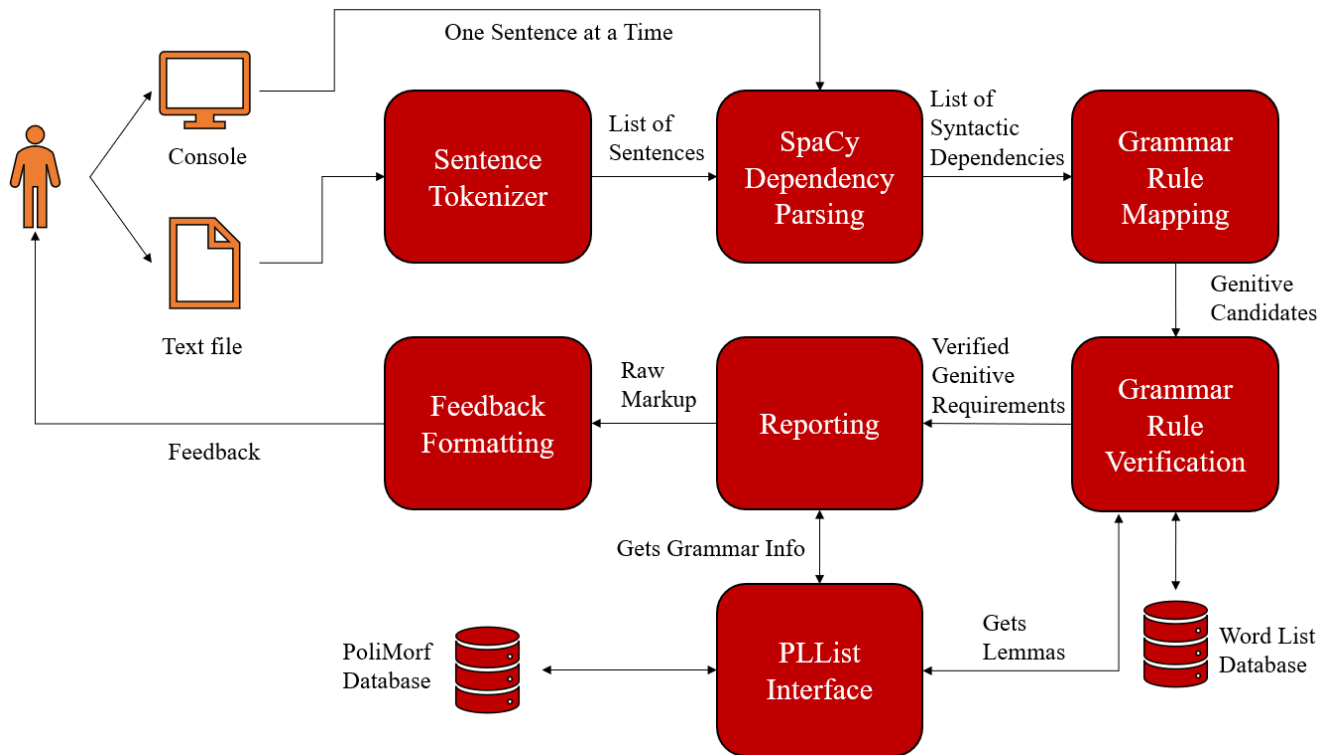


Figure 2. PLPrepare Pipeline

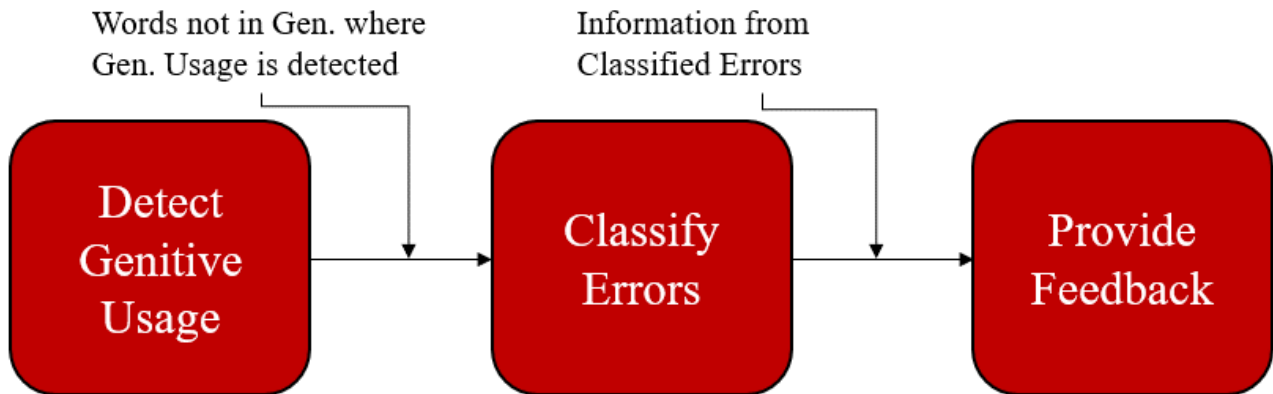


Figure 3. Simplified Pipeline

3.3.2. Collecting Grammar Rules

The first step to building this grammar checker was to collect the rules. From a design standpoint, there are two types of rules: those that require word lists and those that do not. An example of a rule that does not require a word list would be the genitive possession. This rule

states that the first noun in a pair of adjacent nouns exerts possession over the second [1]. For rules like this, there is no need to refer to outside information and can be easily included in the system. An example of a rule that requires outside information would be the rules related to preposition governance over nouns. This means that the genitive case governs several prepositions, and these prepositions should be in a database to help the system pick out where the genitive case ought to be.

Iwona Sadowska's [1] Comprehensive Grammar provides several grammar rules that the genitive case governs. Table 8 shows which of these rules are used and which of them require additional word lists to be effective. The rule coverage is incomplete because Polish numbers can be incredibly complex and require significant effort to include for completeness. In its current state, the rule database only supports finding genitive after nominative numbers not ending in 1-4. Additionally, time expressions are difficult to collect due to their volume and variety, so they are not supported. Finally, some nouns can take either -a or -u in speech depending on social preferences [1]. This case is difficult to quantify from an implementation standpoint, and it is just as difficult to provide feedback for in a written context. Due to these factors, this case was ignored and is best left to future research.

Table 8. *List of Rules Governing the Genitive Case in Iwona Sadowska’s Polish: A Comprehensive Grammar [1]*

Rules Governing the Genitive Case			
Rule Name	Requires List?	Support?	Sadowska Page Numbers
Absence (Negative Existentials)	NO	YES	pp. 76, 81
After Certain Adjectives	YES	EXTENSIVE	pp. 84
After Certain Adverbs of Quantity	YES	EXTENSIVE	pp. 82
After Certain Approximations	YES	EXTENSIVE	pp. 552
After Certain Numbers	YES	PARTIAL	pp. 83, 210, 499-500
After Certain Prepositions	YES	YES	pp. 79, 547, 565, 579-580
After Certain Measurements	NO	YES	pp. 82, 515
After Certain Verbs	YES	EXTENSIVE	pp. 78
After <i>co</i> , <i>coś</i> , etc.	YES	YES	pp. 84
As a Direct Object (Negation)	NO	YES	pp. 61
As a Subject (Masc. Numbers)	YES	PARTIAL	pp. 55, 57-58, 83, 210, 490
In Time Expressions (Negation)	YES	NO	pp. 80, 555
Linked Nouns (incl. possession)	NO	YES	pp. 74

The word lists for the rules that require them were found using lektorek.org, which is an online Polish dictionary with both inflection information and case governance information [46]. For each of the supported rules requiring a list, the PoS filter in addition to the ‘+G’ search term were used to return all words of that PoS that require the genitive case. These results were then stored with the lemma, the case(s) required, and the rule’s frequency into an SQLite database table. Table 9 below shows a few examples from the verb rule table to illustrate how these are stored.

Table 9. *Example of Contents of a Rule’s Word List*

Selected Verb Rows		
Lemma	Case	Frequency
brakować	G	aw
chcieć	G	st
zechcieć	G	st
napędzić	G	m2
napędzać	G	m2
uczyć	A + G	aw
nauczyć	A + G	aw

The verbs, for example, were ordered in imperfective-perfective pairs. The case column refers to the case(s) that the verb's predicate should take. The verb rule table considers the lemmas themselves and the different constructions available to the verbs stored there. For example, the verb *brakować* (to lack) takes a simple genitive predicate whereas *uczyć-nauczyć* (to teach) could take an accusative direct object and followed by a genitive noun. This construction has the meaning 'to teach someone something.' Finally, many verbs are either exclusively or frequently reflexive. The verb list makes a distinction between the reflexive and non-reflexive constructions of a lemma. Maintaining this construction information allowed the system to recognize and differentiate between a lemma's different forms.

The frequency column refers to how often the lemma with the construction specified in the case column is used. This information helped the system provide feedback on certain cases. The value 'aw' means always, and the value 'st' means sometimes. For example, there is a subtle difference between using an accusative predicate and a genitive predicate for the verb *chcieć* (to want). Both are correct, but the genitive implies wanting something that is not there [1]. Determining the correct usage of something of that complexity is beyond the study's scope but having the frequency table enabled the system to give this feedback to the student.

Furthermore, some words are part of several distinct constructions with distinct meanings. Consider the verb pair *przestrzegać przestrzec*. It has two distinct meanings; one is 'to warn against' which requires the preposition *przed* + I, and the second meaning is to 'observe or uphold' (as in laws and rules). The second meaning requires the genitive case, so the rows with 'm2' indicate that one of a lemma's many meanings requires the genitive and not necessarily all of them. What differentiates this from the construction information in the case column is that a significant change in meaning occurs when compared to changing the construction. For example,

consider uczyć-nauczyć + G (to teach) and uczyć-nauczyć + A + G (to teach someone something) with przestrzegać-przestrzec przed + I (to warn against something) and przestrzegać-przestrzec + G (to uphold something). These distinctions exist to better serve the student in providing detailed feedback for mistakes involving these more subtle cases.

Of course, the coverage is not wholly comprehensive. Table 10 indicates how many entries were recorded for each of the rules requiring these lists. For some (such as the preposition list), the list is nearly complete. For others (such as the verb list), the number of existing examples in the Polish language is so large that it is hard to approximate the upper limit of relevant constructions, so the coverage is extensive but by no means complete.

Table 10. *Genitive Rules in the Database Requiring Word Lists*

Genitive Rule Lists			
Rule	Number of Entries	Tracks Different Structures?	Tracks Different Meanings?
After Certain Adjectives	20	NO	YES
After Certain Adverbs of Quantity	29	NO	NO
After Certain Numbers	4*	NO	NO
After Certain Prepositions and Approximations	44	YES	NO
After Certain Verbs	262	YES	YES

3.3.3. *Dependency Parsing*

There must be a roadmap of the sentence to use as a guide for checking to apply the rules found. This roadmap is the network of relationships that the tokens in a sentence have with each other. Dependency parsing offers a fast, accurate way to provide a token valued analysis of a sentence's PoS tags and their relationships to each other through a statistical approach [29]. The result is a series of labeled syntactic dependencies that map one PoS tag to another.

Regarding the SpaCy Dependency parser that PLPrepare used, the set of labeled syntactic dependencies contain PoS tags, a relationship tag that describes that syntactic dependency, an

attempt at finding the lemma, and the original word's text [30]. The data that these syntactic dependencies provide ensures that one sentence can be broken apart into all of the pieces that make it function as a cohesive whole. The sentence can then be examined in terms of its relationships, allowing the system to track down the relationships that signal the use of the genitive case. SpaCy's tokens contain a vast swath of information that this system did not use, so PLPrepare used a different token format which contains only the head text, the head PoS tag, the token value text, the token value PoS tag, and the dep (syntactic dependency) tag that describes the relationship. Figure 4 shows an example of SpaCy's dependency parser at work.

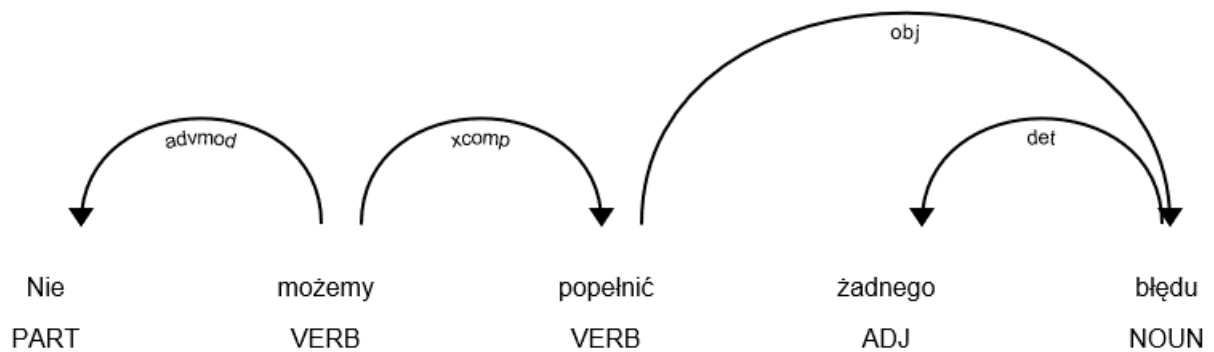


Figure 4. Example Output of SpaCy's Dependency Parser – Figure Rendered by DisplayCy

A note to prevent confusion: this figure was generated from DisplayCy, a library of SpaCy to print out dependency relationships. The arrows display the relationships between a token's head and its children NOT between the head of a token and the token value itself. The next few paragraphs describe to the relationship between the head of a token and the token itself and not between the token and its child. The head of these arrows will refer to the token itself, referred to as the token value and the tail end will refer to the head. The focus on the individual tokens rather than the relationships between head and child causes this subtle distinction.

The tags used to describe these relationships are somewhat more abstract, but they can help to identify where the genitive case is needed. Usually, rules can be detected by simply using the parts of speech of the head and the token value of the relationship, but, for some sentences, this is insufficient. For example, the sentence ‘Nie możemy popełnić żadnego błędu’ (We cannot commit an error) consists of a negated complex verb phrase with a modal verb (możemy – we can) and an infinitive main verb (popełnić – commit). There is also a direct object of the main verb with an adjective modifier (żadnego błędu – no error). Because of the negation, the verb’s direct object and its modifier are placed in the genitive case. The advmod tag indicates an adverb modifier. If an advmod has the token value text of ‘nie,’ it indicates that the verb is being negated. The xcomp tag indicates an open clausal complement, which means that it launches a new clause that is detached from a ‘higher’ subject. In this case, ‘możemy’ launches the new clause ‘popełnić żadnego błędu.’ The xcomp tag can be used to transfer the negation detected earlier from the modal verb to the main verb by tracking the head of the advmod token and matching it with the token value of the xcomp token. Finally, the obj tag indicates a direct object which links popełnić to błędu. Because popełnić was negated, the system knew that błędu ought to be in the genitive case, so the grammar rule parser could return the head of the obj token.

3.3.4. Mapping Syntactic Dependency Tokens to Grammar Rules

The section above described how a dependency parser breaks down a sentence. This breakdown included applying the previously described relationships to pick out specific grammar rules as described in section 3.3.2. To determine which of the genitive rules might apply to a given token, the system had to parse each of these tokens to flag candidates that might require the genitive case. To do this in an organized and efficient way, the system stored the information that points to the different rules in a tree structure.

PLPrepare queried the grammar tree in the form of a request, which is just the information contained in a token but with two differences. The first is that it allows the PLPrepare to adjust and swap out defective or unhelpful PoS tags. For example, the PoS tag representing an uncertain PoS is 'x' [31][48]. When injecting errors, the dependency parser had difficulty assigning a PoS tag to words with misspellings and other errors. Hence, the system would give the questionable token a 'noun' PoS tag to those it could not reliably classify. The second difference is that the token's information was stored in an ordered list format to pass through the tree faster. When the system finished processing a request, the grammar tree would evaluate the request according to the ordered list. Besides the first element, the rest of the list elements would correspond to the following order: head pos tag, token value pos tag, and dep tag. Figure 5 below illustrates the tree structure for mapping tokens to rules.

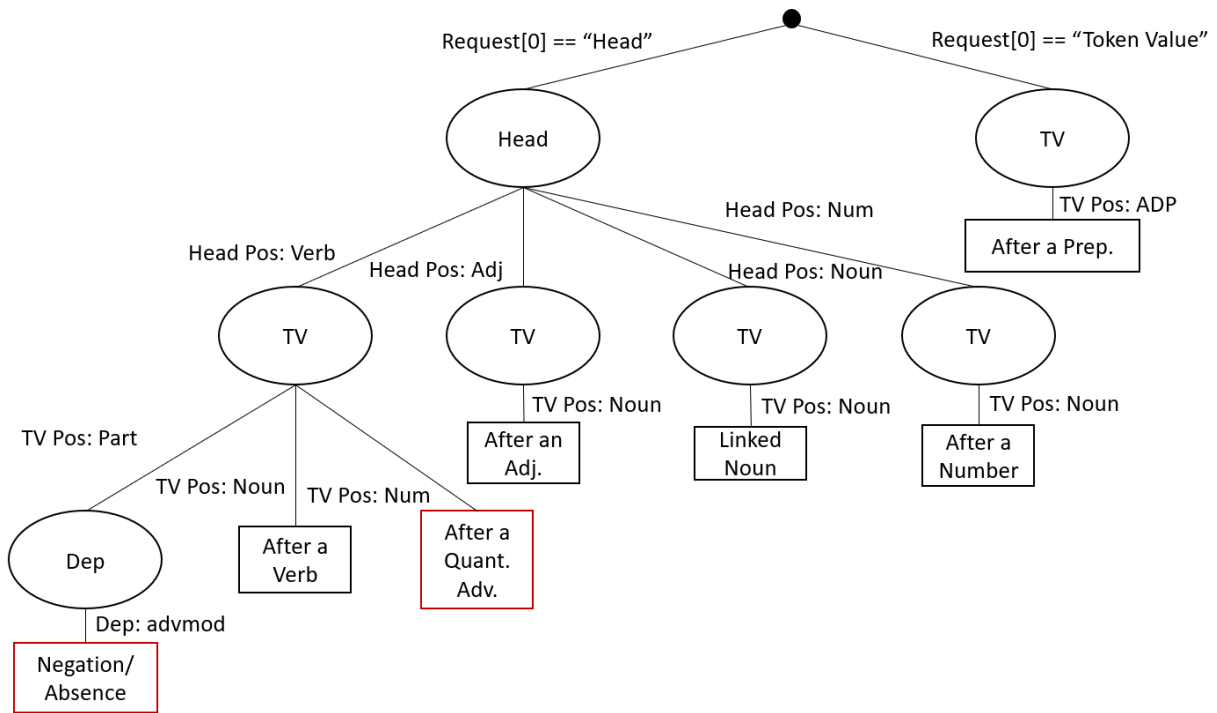


Figure 5. Grammar Tree: Grammar Rules Requiring Additional Parsing in Red

The grammar tree used the head and token value PoS to determine the possible interaction. The tree consisted of a root that organizes two subtrees. These subtrees handled cases where the system must handle the token value or the relationship's head first. For most grammar rules, the system had to examine the syntactic dependencies from the head to the token value, but prepositional phrases were the exception. The system navigated the tree by getting the *i*th element of the request's ordered list and compares it to the node's children. If a node had no children, the tree checked if a grammar rule was present in that node (denoted by rectangles in the diagram) and would return it.

To continue with the example given in section 3.3.3, PLPrepare flagged the *nie* ← *możemy* token as a candidate for the genitive following an adverb of quantity rule because the head was a verb, the token value was a participle, and the relationship was an *advmod*. Because the token did not contain a preposition, the system made element 0 of the request's list 'Head,' and would then contain the head 'verb' and the token value 'part.' This led to the Negation/Absence grammar rule. The system grouped the negation and absence rules together and the linked noun rules together because they were identical in terms of the tags that identify them. The only difference between a negation and an absence token was that an absence token's head contains a form of the lemma 'być' (to be); thus, the two could be safely combined to simplify the tree. Finally, the tree returned the grammar rule as a potential candidate for the genitive case.

3.3.5. Confirming the Candidates

After finding the candidates, most of them required additional processing to see if the selected rule applies. If the rule spanned multiple tokens (marked in Figure 3 by red rectangles), then the system had to find that candidate's child token and validate its relationship to the parent.

If the rule required a text list, the grammar rule checker had to query the database of word lists to determine if the genitive case applied to the token. Finally, once the system had verified that the token takes the genitive case, the grammar rule checker returned the part of the token that contained the noun.

The cases that required additional parsing were either a negation (like the ‘nie mogę...’ example above) or a clause with an adverb of quantity. These cases required the system to look beyond the immediate token to find the noun because the token either contained only a participle and a verb or an adverb and a verb. The system could do this by iterating through the token’s children and comparing those tokens to the required text and syntactic dependency relationship. The solution was adequate, but when handling erroneous input, the dependency parser had a more difficult time creating the parent-child relationships with the affected token. The solution was to store the relevant verb object tokens in a dictionary using the head text of the verb as a key at the beginning of the grammar rule mapping stage. The additional preprocessing eliminated the need to parse the token’s children and improves robustness, but it added a slight performance drain for tokens that did not require additional parsing.

The next case was simpler; if a token required a word list, the grammar rule checker would use PLList’s interface to lemmatize the words. The grammar rule checker would then query the word lists with the lemma and return the case required and frequency, if applicable. If the token required neither of these special cases, the grammar rule checker would simply return the noun text because the syntactic dependency token was enough to infer the rule. After the system had detected the nouns that ought to be in the genitive case, it would send the noun text along with other information to the reporting module where it would compile a noun use report.

3.3.6. Grammar Rule Edge Cases and Limitations

The grammar rule collection and verification process was complex and included edge cases and other limitations that PLPrepare did not handle at all or handled in a way that increased false positives. There were problems regarding words that use multiple cases, sentences with certain negated verbs, and dependency parsing sentences with misspellings. The system cannot handle all cases well, but it mitigated some of these problems, focusing on providing feedback where possible.

PLPrepare could not handle negated verbs that did not have an accusative complement. The negation rule for the genitive case affects only negated verbs that have an accusative complement. Consider the example: ‘Co sprawiło, że zajęłaś się aktywizmem?’ (What caused you to engage in activism?) (Top of Figure 6). Most Polish verbs take the accusative case as a complement [1], but this verb takes the instrumental case as a complement. The negation of this sentence would look like this: ‘Co sprawiło, że nie zajęłaś się aktywizmem?’ (What caused you not to engage in activism?) (Bottom of Figure 6). The sentence is a bit contrived, but the relationship between ‘zająć’ and ‘aktywizm’ when ‘zająć’ is not negated is a nominal subject (nsubj). When negated, the relationship becomes an open-clausal complement. The system would usually interpret an open-clausal complement with a verb phrase as signaling a modal verb. Hence, the system marked ‘aktywizm’ as needing the genitive case even though the rule did not apply here. The negation of these types of verbs were more rare than their accusative counterparts, but it did increase the number of false positives.

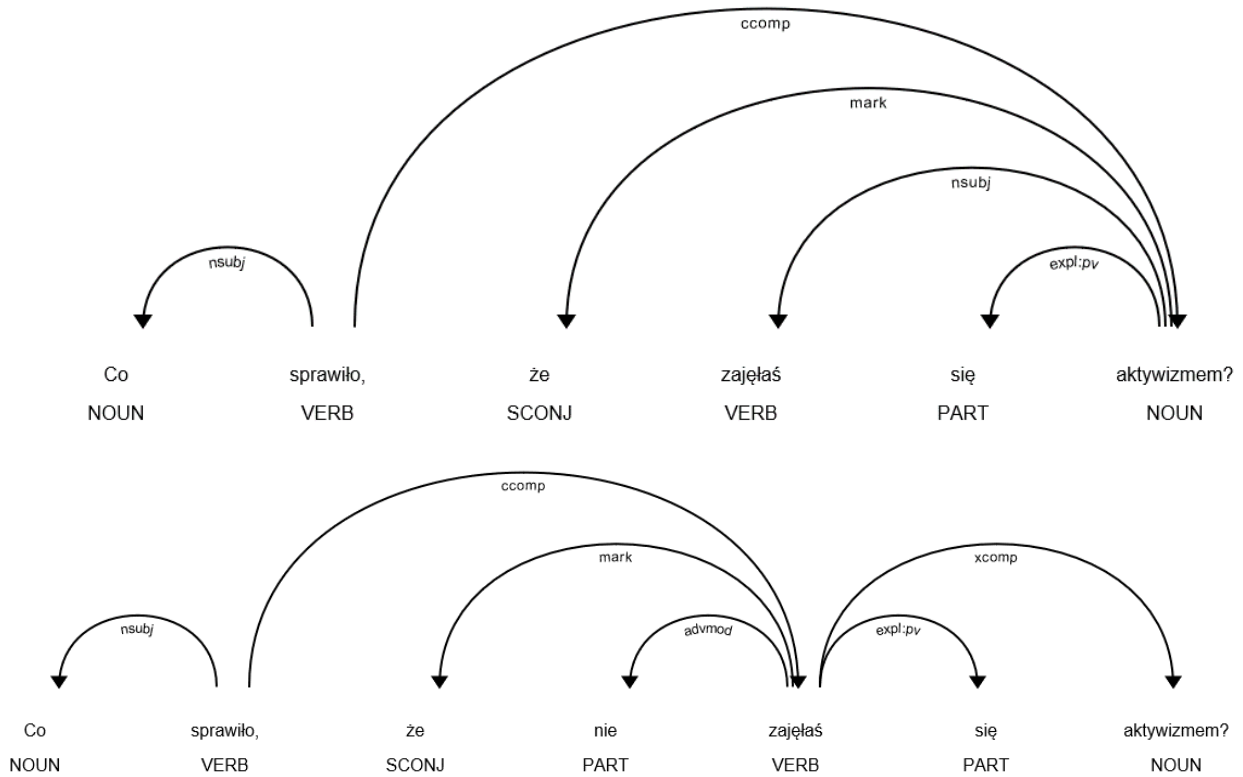


Figure 6. Comparison of a Negated and Non-Negated Verb That Does Not Take an Accusative Compliment

The handling of words that take multiple cases was a problem. Prepositions represented most of these words as most prepositions can be used in different contexts that govern different cases. For example, the preposition ‘z’ could govern the genitive case to mean ‘from, of.’ It could govern the instrumental case to mean ‘with,’ and it could govern the accusative case to act as an approximation [1]. The system did not distinguish the different uses of this preposition, so the system tagged all uses of ‘z’ as needing the genitive case. To handle this problem, the system reported that the word may take multiple cases, but this approach was far from ideal. This case increased false positives in tests where these types of prepositions are concerned. From the user’s perspective, the feedback provided may be an inconvenience, but the system, as a result, could cover many more prepositions.

One challenge that the system faced was the complexity of dependency parsing as a whole. Consider the preposition ‘z.’ It was very difficult to differentiate between ‘z’ as a simple preposition and ‘z’ as a clitic. The sentence ‘Jestem z wami’ (I am with you all) sees ‘z’ assimilate to the pronunciation with ‘wami,’ leading wami to be pronounced as ‘swami.’ The dependency changes based on this distinction. There were several different dependency relationships that could occur with the same parts of speech, so not every subtlety could be considered in the implementation.

When dealing with potentially incorrect sentences with misspellings, it was vital to mitigate both junk data and misspellings. Due to incorrect input, the SpaCy dependency parser occasionally could not determine the PoS tag and syntactic dependency of a token. When this happened, the dependency parser assigned a pos tag of ‘x’ or a syntactic dependency tag of ‘dep’ [30][48]. The system could resolve the dependency and not the pos tag or vice versa, so they did not necessarily occur together. PLPrepare did not do anything to the ‘dep’ tag because it did not affect the grammar rule parsing. It did, however, treat all ‘x’ pos tags as nouns. Polish morphology is so rich that the postfix and the context of a word give clues to the PoS [21], so junk entries or misspellings that overwrote a word’s morphology and ignored other morphological patterns were the likeliest candidates of the ‘x’ pos tag. The best way to handle this was to treat them as nouns that needed to be examined by the reporting module.

3.4 Generating Automated Feedback Using PLPrepare

After the system found the words that needed the genitive case, the next step was to compare what was written to what PLPrepare calculates should have been there. The reporting module handled this stage of the pipeline, and it generated a noun usage report based on its findings.

3.4.1. Detecting Grammatical Errors

The first step to determining whether the student made a mistake was to first determine what the target case was. Because the system focused on the genitive case, the target case would be the genitive case. The reporting module queried the PoliMorf database via the PLList interface for any morphosyntactic information in the dictionary. If morphosyntactic information existed, it stored that information and compared it to the target case. If the noun was both in the dictionary and it matched the correct case, then the system considered it correct.

PLList queries required a great deal of information to avoid ambiguity errors. The only information available to the reporting module was the noun text, the target case, and the PoS. In most cases, this was enough to avoid ambiguity errors, as it is extremely rare for two lemmas to share a genitive form. At that point, if what was passed in was recognizable as a genitive form, then the input would be considered correct regardless.

3.4.2. Classification of Grammatical Errors

If the noun was either not in the dictionary or did not match the target case, then there was an error. The system had to determine what kind of error occurred to provide specific feedback. The system tracked three primary error types: spelling errors, case errors, and postfix errors. A spelling error occurred when one or more characters inside the stem of a noun was incorrect. A ‘postfix error’ occurred when the characters beyond the stem were incorrect – this was not a true postfix error, as the entire postfix of the given form may not be affected and was to differentiate a spelling error affecting the ending of a word and a case error. A case error occurred when the system recognized that the word is in a case other than the target case.

If the noun was not correct because it was not in the dictionary, the system had to determine why it was not in the dictionary. Typically, it was either a spelling error, a postfix

error, or it was correct but the system missed it initially. After the initial comparison, the system checked if the stem or the postfix caused the error. If an incorrect postfix caused the error, then reducing that word to the lemma would reveal a form that would be in the dictionary. Therefore, the system attempted to find a match in the dictionary by a combination of statistical lemmatization and stemming.

The system first tried SpaCy’s statistical lemmatization to reduce the word [49]. Unfortunately, the SpaCy Polish model did not handle erroneous data very well. Table 11 shows the results for different misspellings of ‘system.’ The lemmatization only returned correct lemmas for those with the proper genitive endings “systemu” and “systemów.”

Table 11. SpaCy Lemmatizer Performance with Incorrect Entries. Correct Results in Bold

SpaCy Lemmatizer Results for 'System'			
Test Form of 'System'	Result	Stem Alter-ation?	Polish Ending?
systemu	systemu	YES	YES
systema	systema	YES	YES, Incorrect for this Word
systemami	systemami	YES	YES
systemu	system	NO	YES
systemów	system	NO	YES
systemiów	systemiów	NO	YES, 'i' added
systemięta	systemięta	NO	YES, 'i' added, Incorrect for this word
systemz	systemz	NO	NO
systemxyz	systemxyz	NO	NO

This lemmatizer can be helpful for picking out difficult stems, particularly for nouns with stem changes, e.g. ‘koniec’ (end – nom) -> ‘końca’ (gen), but it was not enough. A stemming step occurred after the initial lemmatization attempt to catch more postfix errors.

If the lemmatization did not return a valid noun, the algorithm removed the ending and tried again. This process was limited by the number of checks which simply divided the word length by two and cut off the decimal. The number of checks could be no greater than four. It

would almost never take more than four checks to reveal the stem of any noun and would never take more than four checks to reveal the stem of a noun in the genitive case. The smaller the word, the more likely the ending took up a more significant portion of the noun, so the algorithm removed fewer endings for smaller nouns. Additionally, removing more characters increases the risk of uncovering a new stem, which is all the more reason to reduce the number of checks where possible.

Algorithm 1 Getting the Number of Checks

```
1: procedure GETNUMCHECKS(NOUN)
2:   return floor(len(noun)/2)
```

The algorithm would remove ending the after ending checking the new noun if it was in the dictionary. It first checked if the new noun was in the dictionary via a PoliMorf look up, and then it attempted to lemmatize the noun via the SpaCy algorithm. This process continued until there was a match or the number of checks was depleted.

Algorithm 2 Noun Reduction Algorithm

```
procedure CHECKFORDICTIONARYMATCH(NOUN, NUMCHECKS)
2:   if numChecks = 0 then return 0
   if inDict(noun) then return noun
4:   newNoun ← tryLemmatize(noun)
   if inDict(newNoun) then
6:     return newNoun
   else
8:     noun ← removeFinalCharacter(noun)
     return CheckForDictionaryMatch(noun, dec(numChecks))
```

If the algorithm found the form, it was possible that it found a lemma that was unrelated to the original form. A student could enter ‘Sabaton,’ which is not a Polish word, and the algorithm would find ‘sabat’ (sabbath). Either way, if the algorithm found a lemma in the dictionary, it was safe to declare that the ending that the user typed was incorrect. This approach’s only downside was that the system could misclassify simple spelling mistakes as

postfix errors. Additionally, as a defense against ambiguation errors, the system performed a final inflection attempt with the found lemma to determine if it matched the original form. If the inflected form of the found lemma matched the original entry, then an error occurred and the dictionary did not find it the first time, so the original entry was correct all along. If the algorithm could not find a lemma by any means, then the original entry was either a valid word that was not in the dictionary or a spelling error. Either way, the system classified the entry as a spelling error.

The last case that the system considered was if the entry was in the dictionary but was not in the target case. The result was most often a case error. For example, consider the difference between ‘motel’ (motel) and ‘motela’ (a classification of fish with two dorsal fins). The genitive singular of ‘motel’ is ‘motelu,’ so a student could mistakenly enter ‘motela’ as the genitive form of ‘motel.’ The entry would be in the dictionary but was not in the target case. The question became whether an incorrect postfix led to the dictionary finding a different lemma or the correct lemma with an incorrect case inflection. The problem was rooted in lemmas of a different gender sharing different forms, so the solution to this problem was to stem the entry and reinflect it into the target case. If the entry could be stemmed and reinflected for case and number of the original entry successfully, and if it was a different gender than the original, then the entry had an incorrect postfix. On the other hand, if the entry was present in the dictionary and could not be stemmed and reinflected, then the case of the original entry was incorrect. Figure 7 shows an overview of this classification process.

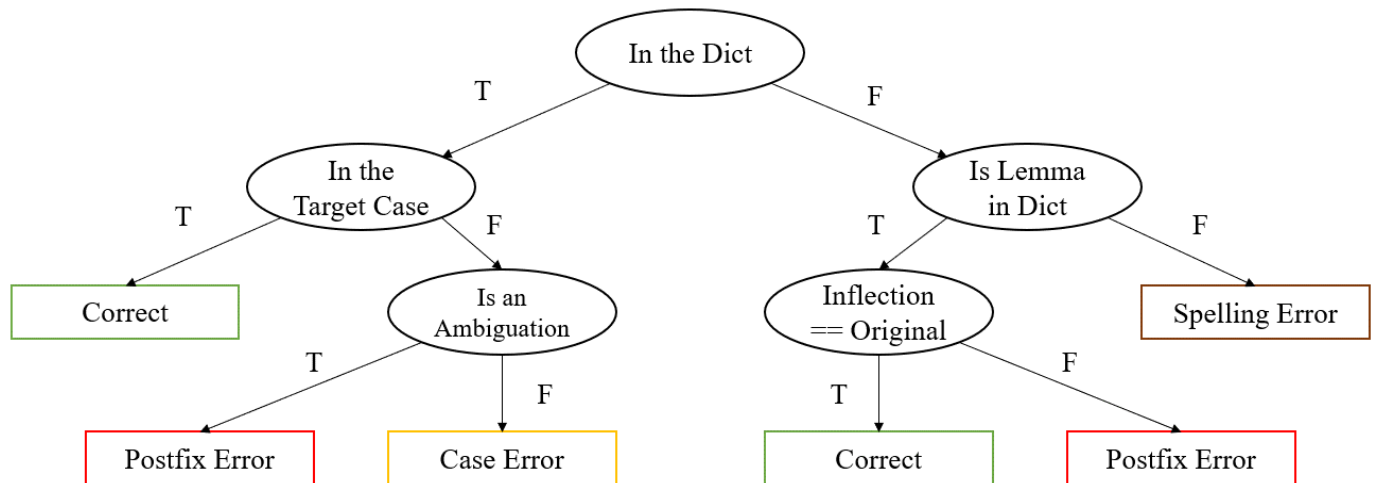


Figure 7. Grammatical Error Classification Decision Tree

3.4.3. Generating the Feedback

PLPrepare used a combination of error classification, grammar rules, and guidelines to provide detailed feedback. The system generated this information as a sentence moved through the pipeline, but PLPrepare determined what feedback to display based on the error classification. Once the system chose the feedback, it then showed it to the user.

The Grammar Rule Mapping Stage, the Rule Verification Stage, and the Reporting Stage all sent potential feedback. The process began with the Grammar Rule Mapping Stage. Each rule in the grammar tree had a description to inform the student how to spot the corresponding rule. At the Rule Verification Stage, the system added any relevant structural and frequency information discussed in section 3.3.5. This information provided the system with a variety of feedback to address most of the user’s problems when writing sentences using the genitive case.

```

41587 Feedback for sentence:
41588
41589   dyscyplinach powinno być czynione ze względu na profilaktykę
41590
41591   względ is in the nom case. It should be in the gen
41592   z can take A or G or I
41593   względ is object of a preposition that takes the genitive case
  
```

Figure 8. Displaying Feedback from the Rule Verification Stage Regarding the Use of ‘Z’

At the Reporting Stage, the system chose the feedback to display based on the error classification. If there were no errors, then the system would not produce any feedback. If the error involved an incorrect case being used, the system chose to display the grammar rule information, the incorrect case the user used, and the correct form.

```
66 Feedback for sentence:
67
68 ma większe powinowactwo do nadtlenu wodorem co sugeruje jej istotniejsza
69
70
71 wodorem is in the ins case. It should be in the gen
72
73 wodorem is is being possessed by another noun, the object of a gerund (nouns in -nie), a quantifier noun - butelka wina
(a bottle of wine)
```

Figure 9. Reporting a Case Error Where ‘Wodorem’ Should be in the Genitive Case Due to Possession by ‘Nadtlenku’

If the error involved a misspelling, PLPrepare simply displayed the word that was not in the dictionary with a message informing the user that the word was not found.

```
74 Feedback for sentence:
75
76 postępowaniu przygotowawczym nie może przekroczyć okredzu trwania dochodzenia lub śledztwa
77
78
79
80
81 okredzu is either spelled incorrectly or not in the Polimorf dictionary
```

Figure 10. Informing the User that a Word is Misspelled

If the error involved an incorrect postfix, then the system displayed the correct postfix and any guidelines found for that word. This step tied into the guideline lists from section 3.1. PLPrepare queried the guideline lists for the lemma and retrieved the guideline’s name, the postfix that the guideline recommends, the reliability of the guideline, and the number of words that the system had on file for that guideline.

```
42095 Feedback for sentence:
42096
42097 zwierzętom imion i odwoływać do językp ludzkiej psychologii
42098
42099
42100 językp should have the postfix a...
42101 językp is part of the BODYPARTS category, which take the -a ending 69.23 % of the time based on 52 examples
```

Figure 11. Displaying Feedback for a Postfix Error with a Guideline Suggestion

The feedback effectively informed the user of what he/she did wrong, what the correct form was, and how to improve in the future.

Chapter 4. Performance and Evaluation

The evaluation of PLPrepare covered a few of the possible mistakes that students might make and focused on evaluating the three most crucial stages of the pipeline, the genitive detection stage, the error classification, and the guideline assignment. The evaluation focused on recall and precision of classifying three basic error types (and the no-error), the recall and precision of the genitive detection system, and the recall of the -a vs -u feedback from the wordlists discussed in the previous chapter. The testing stage's basic outline was to collect sentences, inject errors into those sentences, and compare the injected error to the classified error.

4.1. Gathering and Generating the Test Data

To test PLPrepare's performance, there had to be a wide variety of sentences containing the genitive case to give to the system. The National Corpus of Polish (Narodowy Korpus Języka Polskiego – or NKJP) could be queried using a powerful query language to extract sentences [50], so NKJP provided an ideal corpus to extract sentences. The sentences also required preprocessing before the testing system can inject any errors.

4.1.1. Sentence Collection Using NKJP

When testing the genitive detection stage, every sentence had to contain a correct example of genitive case usage so the system can inject an error into it. This test did not focus on precision because it was difficult to manually label 6,000 sentences as either having or lacking the genitive case. Finding the recall here required certainty. One of the problems facing this goal was the issue of ambiguous cases. Several masculine inanimate nouns shared a genitive ending with the accusative case of that noun [1]. Because of this problem, a query such as "[case=="gen" & pos=="subst|ger|depr" & gnd=="m[23]" & number=="sg|pl"] within s' (select sentences where

NKJP can evaluate a noun to be masculine animate or inanimate and singular or plural) would return sentences with ambiguations where the noun would be in the accusative (or other) case(s) [50]. To mitigate this problem, a strict criterion was injected into the query to only extract words that the system was certain are in the genitive case as they appear in the sentence. The query then became '[case~="gen" & pos=="subst|ger|depr" & gnd=="m[23]" & number=="sg|pl"] within s.' Unfortunately, this seriously reduced the overall population of sentences to select from. It was difficult to find forms of a lemma where the genitive form did not share a form with another case. Using this query type virtually eliminated the feminine singular and masculine animate because the genitive case for these genders almost always shared the genitive case form with another case. These eliminations reduced the diversity of the selected words in the test. Still, it provided a measure of consistency in that the system would not have to throw out a significant portion of the test data because the words only appeared to be genitive but were in the accusative case. This type of query was ideal for testing the genitive detection stage's recall stage but ineffective for testing classification because of the exceedingly low gender diversity in them.

To test a variety of the system's capabilities, two groups of six sets containing one thousand sentences each (12,000 sentences total) were collected according to different numbers and genders of the nouns they featured. The first group used the strictest query discussed above to guarantee that genitive case examples were included. This group was used to test the genitive detection system. The second group used the looser query, which included some accusative examples. This group was used to test the error classification stage. The groups were composed as follows: The first two sets were made up of singular and plural nouns of all genders, the next two were singular and plural sets of masculine animate nouns, and the final two were singular

and plural sets of only masculine inanimate nouns. Figures 12 and 13 below show the exact gender breakdown for each group.

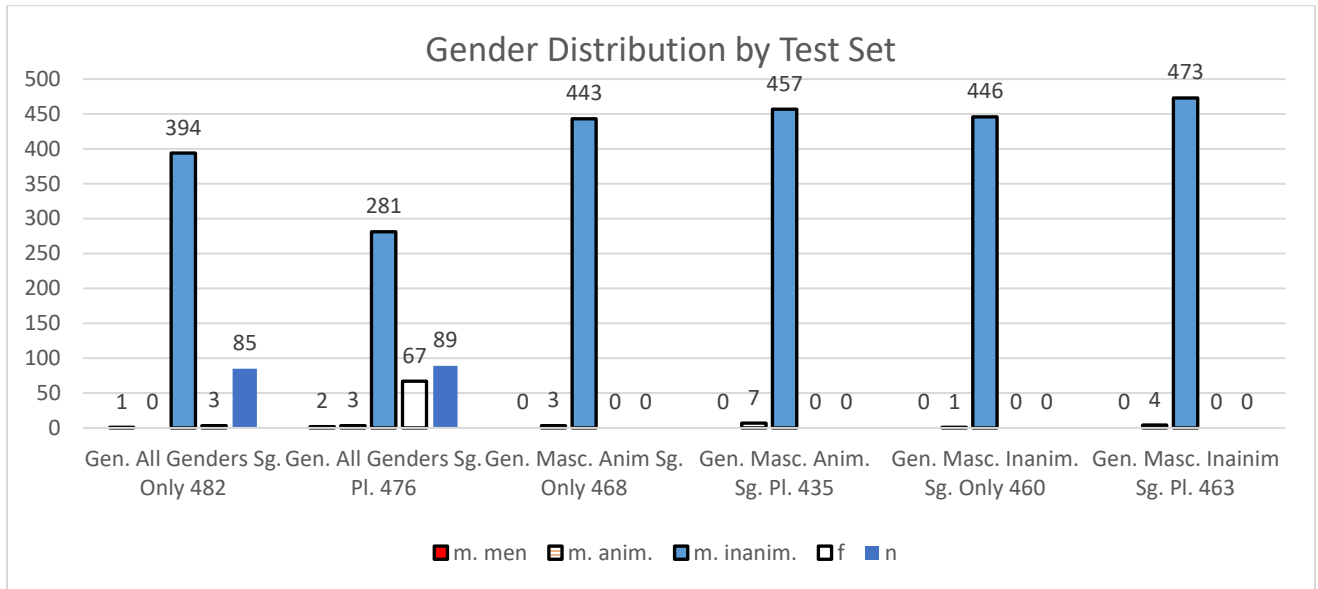


Figure 12. Group 1 Gender Distribution – For Testing Genitive Usage Detection – Corrected for Dictionary Retrieval Ambiguations

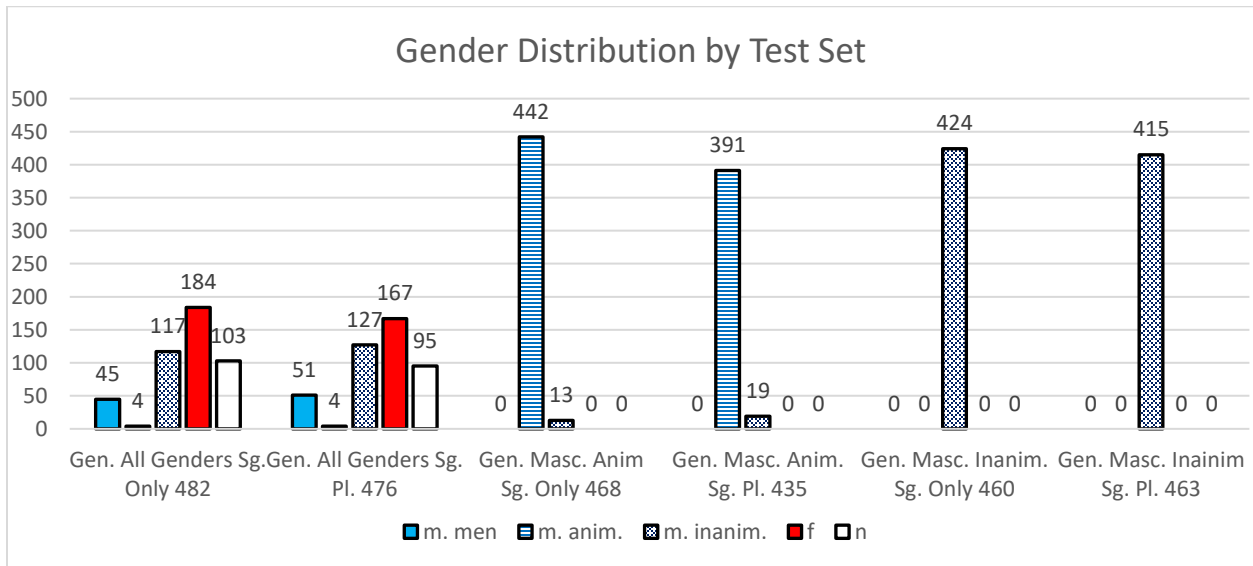


Figure 13. Group 2 Gender Distribution – For Testing Error Classification – Corrected for Dictionary Retrieval Ambiguations

4.1.2. Sentence Preprocessing

Despite the specificity of the queries used to extract the sentences from NKJP, the sentences required some cleaning to be useful. NKJP fulfills the queries in either an HTML or CSV file with a user-specified amount of context surrounding the query result. To form a testable sentence, the query result had to be re-injected into a recombined sentence. For example, the sentence ‘Jak twierdzi Hubert, do szczytu możemy iść bez liny’ appeared as Table 12 below. The bolded word was the example that had to be held as the target word but was reinjected into the sentence to give the system the necessary context. The resulting test entry is displayed below.

Table 12. NKJP Test Entry Preprocessing

NKJP CSV (Before Preprocessing)		
Jak twierdzi Hubert, do	szczytu	możemy iść bez liny.

Test Entry (After Preprocessing)	
szczytu	Jak twierdzi Hubert, do szczytu możemy iść bez liny

Because context was collected at a consistent margin throughout the query, there were a few problems regarding a few of the sentence’s testability. Some of the entries in the NKJP .csv files contained one or even two additional sentences because of the context. One example was ‘wędkowanie. W Grand Prix Wrocławia jestem w pierwszej szóstce.’ In this case, the final word of the preceding sentence was included in the entry. The entry may even have contained context from the next sentence on the rightmost cell. Also, a few entries did not contain enough context to express a complete thought. In the example ‘- np. przy sprzedaży samochody - dotyczy obecnie całego niewykorzystanego’, no punctuation was present indicating the end of the sentence or even the thought, and there was no capital letter or preceding punctuation indicating that a new sentence had begun. Other sentences were simply incomprehensible by the tool and

difficult to understand even by human standards. The entry ‘. G. (pismo Areszty Śledczego w S. z’ was one example where more context was needed. To solve these problems, a few criteria were introduced. The first was that the test sets could only include a sentence if either the first word was capitalized or the period from the preceding sentence was visible in the context. Next, the righthand context had to include some punctuation to signal a finishing thought. Finally, all enclosing punctuation had to include an opening and a closing. A complete sentence was not necessary for the detection step, but a complete thought that the dependency parser could properly examine was necessary. These criteria did not guarantee perfect test entries, but they eliminated the majority of bad candidates. Group 1 saw that, out of six initial sets of one thousand sentences each, the final testing data set contained a total of 2,835 sentences all together with around 400 sentences for each set.

4.1.3. Grammatical Error Injection

To test the system’s error classification capabilities, the testing system injected errors to simulate the different types of errors that the system attempted to classify. The error injection step involved retrieving the target word from each test entry and reinjecting a modified form of the word into the sentence. There were four possibilities that the system randomly applies to each sentence in the test set, each of which were equally likely. This error injection technique allowed for easy, unique retests, and it also tested the dependency parser’s ability to work with a variety of erroneous sentences. For a discussion of the meaning of each of the error types, see section 3.4.2.

To perform the error injection, one of four operations were used. To inject a case error, the system found the target word’s lemma and reinflected the word into a new, randomly selected case. For example, in the test entry ‘|głosu| Kto wstrzymał się od głosu?’ The word

‘głos’ (voice) would be reinflected into a random case. If the randomly selected case was the instrumental case, the test entry would be modified to read ‘|głosem| Kto wstrzymał się od głosem.’ The function disallowed any form that matched the original form. To inject a postfix error, the final character was randomly replaced by a member of a set of the Polish phonemes (including those with multiple letters, e.g., dż, rz, ch, etc.) Because this was not a true postfix error, it was intentionally not replaced with a valid postfix to differentiate a spelling error affecting the ending of a word and a case error. The distinction also necessitated that the postfix error did not randomly form a case error, so the error injection algorithm checked to see if the new noun was in the dictionary. If it was, then the algorithm removed the attempted postfix from the set and tried again. In the example sentence above, the word ‘głosu’ could become something like ‘głost’ or ‘głosrz,’ as these forms did not correspond to any forms in the lemma. But it could not have become ‘głosy’ because that form corresponds to the nominative and accusative plural form of the word. The system injected a spelling error by lemmatizing the word, selecting a character position, and injecting a random member of the Polish phoneme set discussed above to the word’s original form. Finally, the system tracked a ‘no error’ error type as a control. In this case, the error injection system simply passed the sentence to the next stage.

4.2. Evaluation Experiments

The evaluation process involved injecting errors into each of the test sets separately and then determining the system’s performance for each of them. For example, to test the genitive all genders set, the system first randomly injected errors in the manner discussed in section 4.1.3. and then the system began testing the sentences in the set. The tests were conducted on a sentence-by-sentence basis. For each sentence, the test involves recording the injected error, putting the system through the pipeline, and determining whether PLPrepare caught the injected

error. Because the system could catch multiple errors in a sentence, the test recorded only the reports on the word with the injected error. If there were no noun usage records associated with the erroneous word in the sentence, the test recorded that as a failure on the part of the genitive detection pipeline. The classification was tested by comparing the noun usage report findings to the recorded error injection type. Suppose the classification of the error found did not match that of the injected error. In that case, the system recorded that as a failure of the error classification system in the pipeline's reporting stage. A confusion matrix was constructed by recording all of the injected error types and all of PLPrepare's corresponding classifications. If the injected classification did not match the reported classification then, a false negative was recorded. If the system was testing for a particular classification and the classification was reported even though it was not present, a false positive was reported. The evaluation stage used the confusion matrix to interpret the findings in the coming sections.

After running through all of the entries, the system then reported the confusion matrix with the recall, precision, F1-Score, and the macro and weighted average. The system then displayed the number of times that it failed to detect the target genitive usage and the recall for each error type that corrects for the system missing the genitive detection. The following sections discuss these initial results and indicate the level of performance that a user could expect to see, an evaluation of the genitive detection system, and the coverage of the wordlists used to provide -a vs -u feedback.

4.3. Results

The results for each stage of the pipeline will be presented below in the order specified in section 3.3.1. The inputs, outputs, and relevant background information will be specified with graphs illustrating the relevant figures. The following subsections will summarize the results and

detail the key takeaways from the data. Finally, each subsection will discuss any data points that appear out of place and briefly describe how to improve results in future work.

4.3.1. Genitive Detection Results

The genitive detection system was tested for recall by running a normal test with the data in group 1 and with a separate test to determine precision. The conditions of the testing described above necessitated a separate precision test because those tests only work with sentences that used the genitive case, so the genitive detection system would not have the opportunity to produce a false positive. The precision test involved 92 sentences where half of the sentences did not use the genitive case and the other half did. These sets were human verified before testing. The constraining factor for the test size was the number of sentences that did not use genitive. Out of a random sample of 1000 sentences from the NKJP, the computer threw out 878 sentences for containing genitive and data cleaning purposes. The human reviewer eliminated an additional 76 sentences for containing the genitive case. The 46 non-genitive sentences were matched by 46 sentences containing the genitive case. No errors were injected into these sentences.

PLPrepare flagged 16 sentences as false positives, 46 sentences as true positives, and 30 sentences as true negatives. The test saw a recall of 1.00 and a precision of 0.74. For the most part, the system correctly flagged genitive case usage, but the cause for most of the false positives was the noun rule, which cast too large of a net. The most prevalent problem was when noun/noun tokens had child tokens. In some instances, this token contained a conjunction, which the system had to mark because of how Polish distributes case between items shared by a conjunction. In others, it was difficult for PLPrepare to determine the position of child tokens in the sentence. In some cases, the genitive noun possession rule could pass over interfering words,

especially adjectives. In others, the words interrupted the possession. Both cases required an additional level of abstraction to ascertain the role of child tokens.

More striking, however, was the comparison of this result with those of the experiment described in section 4.2. When testing the genitive detection system with the error-injected sentences, the recall plummeted as low as 70%. The results hovered between 70 and 75%, which was disappointing considering the precision test’s recall. Figure 14 below shows the recall of the Genitive detection system for each test.

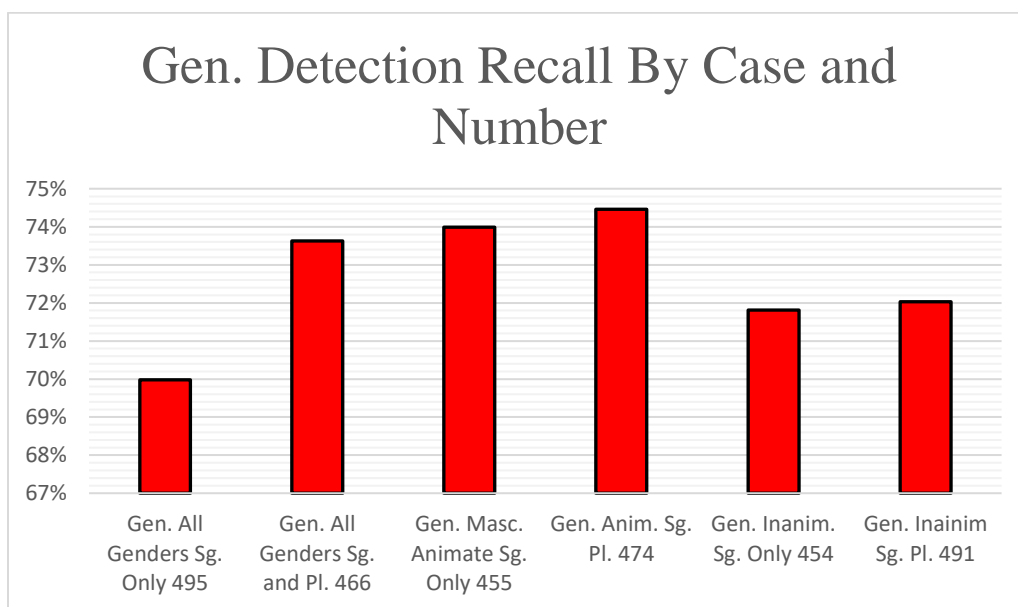


Figure 14. *Genitive Usage Detection Performance by Test Set in Group 1*

Two primary factors led to the discrepancy in the observed performance in genitive detection. The first was the sample size. It was much easier to find sentences with rarer verbs or sentences with obscure structures that the grammar tree (described section in 3.3.4.) could not find in a sample size of 2,835 sentences. The second factor was the inability of the dependency parser to make sense of some tokens with errors. The dependency parser that PLPrepare used

was trained on the Polish PUD treebank [31][32]. This treebank was built on examples spanning multiple corpora including the NKJP [32][33], which contained minimal errors. The dependency parser performed extremely well with erroneous data all things considered, but it was not explicitly optimized to handle the badly mutilated Polish words that these tests involved. The most straightforward way to improve these results could be to expand the external wordlists to cover more of the Polish lexicon. Expanding the word lists could be done by compiling grammar information from multiple sources, such as SGJP [51]. Another way to expand them would be to train a dependency parser to provide parsing with more confidence on tagging erroneous input. This enterprise would not be a trivial one, and it would require additional research. Finally, a more detailed grammar tree could help the system pick out more rules involving the genitive case by scanning more obscure sentence structures for genitive case usage.

4.3.2. Grammatical Error Classification Results

After the system detected the genitive usages, it sent those genitive usages to be checked for errors. This stage required a test that involved the test sets in group 2. These were discussed in section 4.4.1. Here, the breakdown of the error injections was important for understanding the input. Figure 15 shows the comparison of the error injections for each test. The tests were injected fairly uniformly with no error type hoarding an overwhelming majority, though some test cases could be overrepresented compared to others. It was to these injections that the classification output was compared to measure recall and precision.

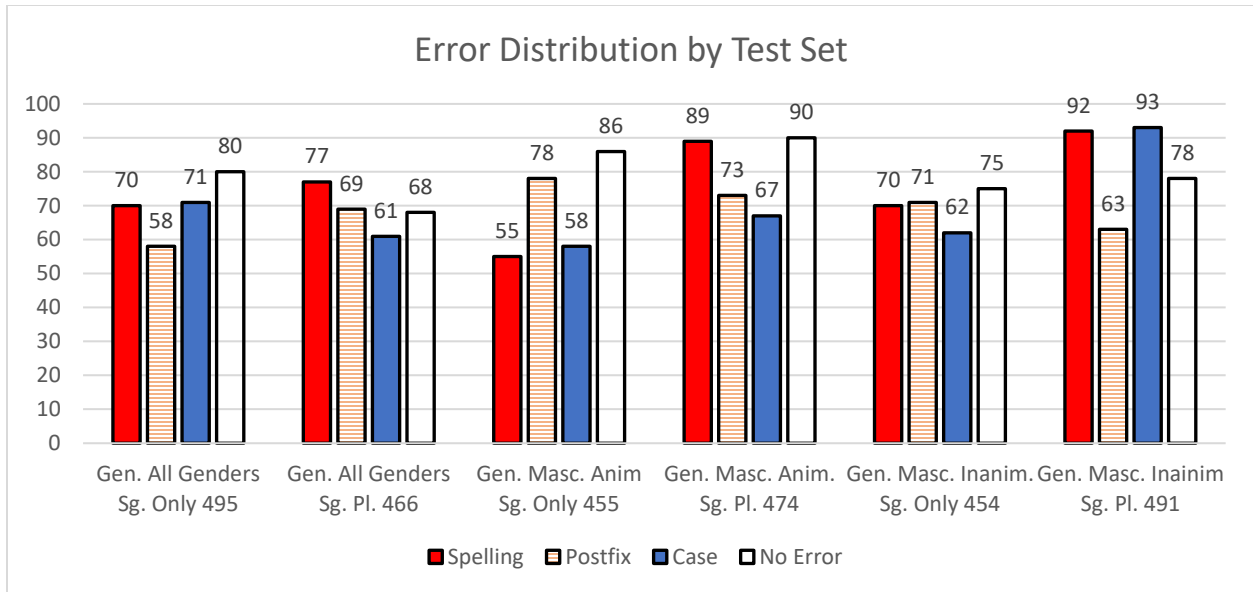


Figure 15. Error Distribution for the Error Classification Stage

This stage of the tests fared somewhat better with a few exceptions. Figures 16-17 below detail the results of this stage.

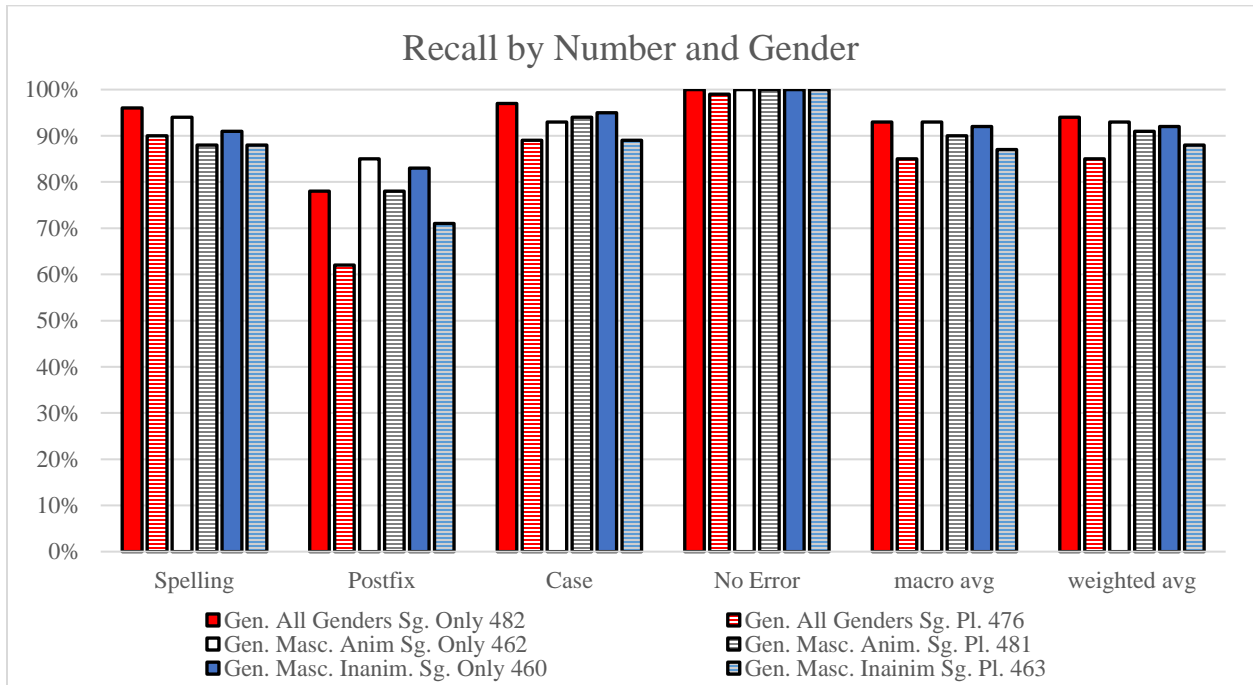


Figure 16. Recall for each Test Set

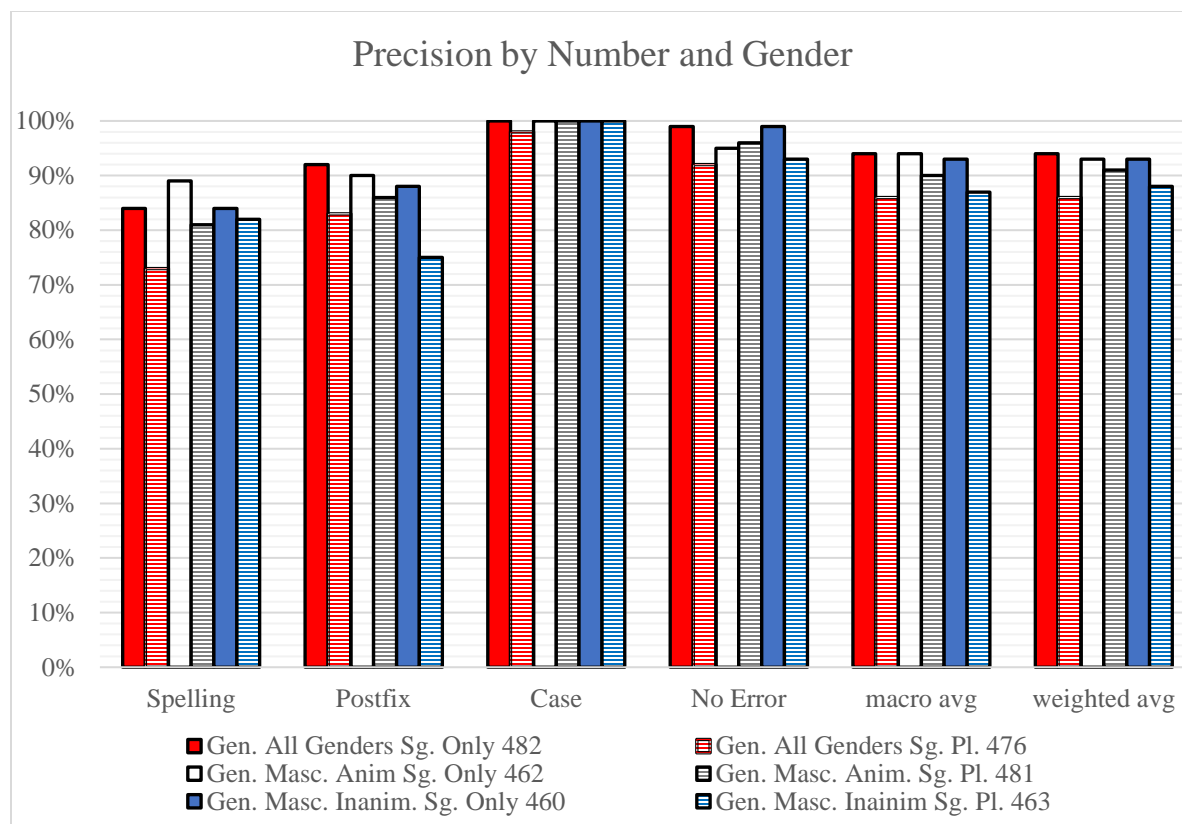


Figure 17. Precision for each Test Set

Spelling Errors (exempting non-errors) were among the simplest to detect, so it made sense that PLPrepare’s spelling error classification would hover around 90% recall. Most of the false negatives came from the noun reduction algorithm (described in section 3.4.2) checking for a postfix error. The algorithm would reduce the word into a separate lemma, which the system would recognize and declare a postfix error. This performance remained high throughout most of the tests with the recall dipping slightly for plural nouns. Regardless, the best way to improve the spelling recall would be to improve the noun reduction algorithm, which will be discussed in the next paragraph. Regarding precision, the spelling classification had lowest precision because a spelling error was essentially the default value that the classifier assigned to any word that was not in the PoliMorf dictionary. The difficulty in determining the difference between a postfix and

a spelling error was the root cause of this low precision. Simple dictionary misses, although rare, also contributed.

The Postfix Recall performed reasonably well for the masculine singular only sets, but it saw a significant reduction in recall for the all-gender plural set. This was because, when the feminine and neuter plurals were introduced, several new concepts were also introduced. The most impactful concept was the neuter verbal noun (nouns in a -nie ending). Neuter verbal nouns were very common and were difficult for the noun reduction algorithm to treat correctly because of the genitive plural, which causes most neuter nouns take a null ending. The null ending in these verbal nouns led to the diagraph 'ni' being replaced with 'ń,' altering the stem's appearance e.g. czytanie (nom sg - the reading) → czytań (gen pl. – of the readings). The noun reduction algorithm was entirely reliant on the SpaCy statistical lemmatization step of the algorithm to catch nouns with any kind of stem alterations and expansions. In these cases, the algorithm could not uncover the correct lemma by just removing characters. This problem was not exclusive to verbal nouns, but they were the most plentiful. The null ending also applies to most feminine and neuter plural nouns. The system had serious trouble detecting null endings partially because the system anticipated a final postfix and not the absence of one. This was also reflected in the number of these cases where the system wrongly identified these errors as correct.

On average, the postfix classification precision was the second-worst precision measurement due to how easy it was for the noun reduction algorithm to uncover a stem belonging to a different lemma and mistakenly label the error a postfix error. The underlying problem was that the lemmatizer used in the algorithm was part of the same model that handles the dependency parsing, so it shared the same drawbacks as the dependency parser discussed

above. The self-reported lemmatization accuracy of this lemmatizer was 89% [28], and it did not perform as well as the dependency parser for guessing a correct form. To improve accuracy for erroneous data, it could be possible to train a lemmatizer with correct lemmas and incorrect approximations to improve guessing. This innovation is a subject for future research, as producing a statistical lemmatizer with acceptable accuracy is incredibly challenging.

The recall of the postfix tests was the worst of all of the measurements. The average recall was around 76% though much of the same problems concerning neuter and feminine nouns persist. Despite this, PLPrepare performed much better at predicting the original problem area of the masculine inanimate genitive singular because the guesswork of SpaCy's lemmatizer was not required as frequently. Though there were still stem alterations, almost every noun ended in an -a or -u, so finding the stem was often as simple as removing the final character. The system performed fairly well when there were no stem changes involved in a lemma's inflection. However, these stem changes are extremely common, and the noun reduction algorithm must undergo some changes to improve recall in this test.

The case test results showed that masculine-only sets have an average recall of over 90% with the all-gender all-number set trailing behind. The problems that faced the case recall were mostly tied to the disambiguation step discussed in section 3.4.2. The existing error occurred because the system marked some injected case errors as ambiguous, meaning the system did not know if the found form of the word belonged to one lemma or another. Unfortunately, this was common enough to affect the recall, and the null endings only exacerbated this problem. There is no way to read the user's mind, so it is impossible to truly determine which lemma the user is referring to in this case. In the future, however, the system can take advantage of usage frequency information to better infer which lemma the user is referring to. The precision, on the

other hand, was stellar. The system caught the vast majority of case errors, and it was almost entirely certain when it declared that an error was a case error.

The success of case detection was interesting because case performance surpassed postfix performance since detecting a malformed postfix ought to have been easier than detecting the incorrect morphosyntactic information. The use of the stored morphological data in the PoliMorf dictionary made this task almost trivial. Whenever the system reported a noun, the PLList interface retrieved that information from the dictionary, so it was straightforward to know when a user used an incorrect case. On the other hand, the postfix relied on the system being able to differentiate between a simple misspelling of the stem and that of the postfix, which was a more complex task.

The control sentences that contained no errors were trivial for the system to detect, so the system rarely reported an error in a correct sentence. Occasionally, the system would mistakenly declare a case error as being correct on account of an ambiguity. Consider the nouns ‘obraza’ (insult) and ‘obraz’ (picture). If a case error produced ‘obraz’ in the nominative case, the classification stage would classify this as correct because the genitive plural of ‘obraza’ is ‘obraz.’ The system, therefore, labelled the entry ‘obraz’ as correct. The ambiguity made entries like these more difficult for the system to classify correctly because the initial form was both in the correct case and in the dictionary. The only solution would be to use contextual information to parse the verb agreement and other information to determine the taken number and gender of the noun. Contextual information will help resolve any ambiguities, but this would require further implementation and is not present in the current system.

The classification results were promising but underline the need for further exploration of dependency tokens to map the relationships of verbs to nouns to help with disambiguation.

Additionally, a better solution is required to resolve the difference between postfix and stem misspellings statistically. From an instructor’s perspective, this classification recall and precision could help drastically reduce the workload of grading. The additional information in the feedback can tell an instructor where to look. From a learner’s perspective, however, any mistakes in classification that the system makes could lead to an incorrect understanding of the mistake. Thus, it is recommended that these improvements be made before any students use a system derived from this study.

4.3.3. Generated Feedback Results

In a separate test, the system compared how postfix errors mapped to the word lists discussed in section 3.2.2. This test cast all test sentences as postfix errors to get a sense of how many of these words were present in the random samples taken from NKJP. This also indicated how frequently the system could give -a vs -u feedback with the current wordlists. The results of this stage are in Figure 18.

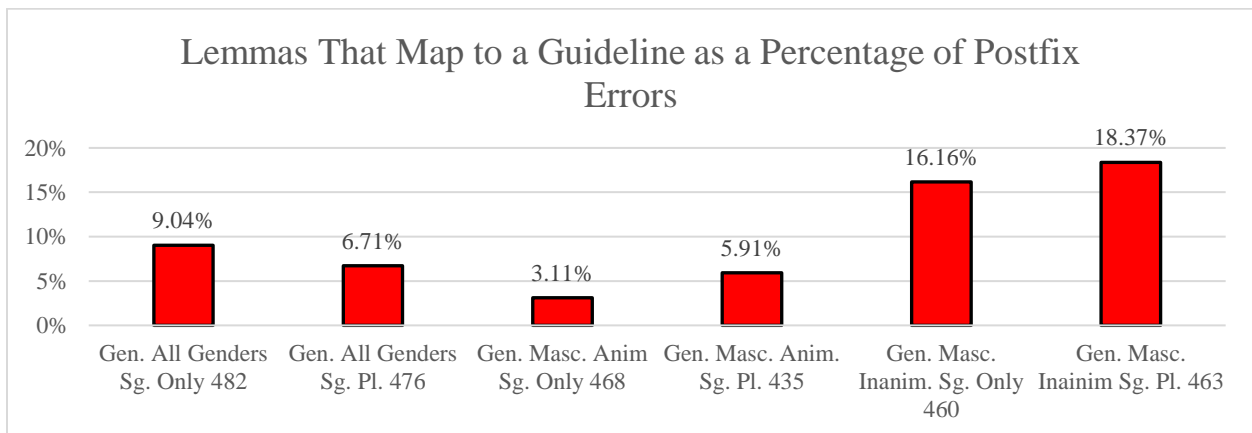


Figure 18. Total -a vs -u Feedback Found Per Test

It was expected that the all-gender sets would contain a lower density of words with a matching guideline. The animate test set contained a few nouns due to the blurred lines between inanimate nouns and grammatically alive nouns. The coverage for the masculine inanimate

singular was well above the expected rate given the word lists' small size compared to the Polish lexicon. This was because several of the words in the word lists were very commonly used, e.g., months and body parts [47].

The coverage was surprising, but a tremendous amount of work will be required to expand this coverage to be helpful to students. Collecting words that meet these categories, as expressed in section 3.1.2., is a time-intensive process. There are several possibilities for expanding the coverage including word mapping, semantic analysis, and manually categorizing words. Ultimately, drawing the lines between the different categories to match a word to a guideline will require a great deal of research and implementation, so this task was best left to future research.

4.3.4. Results from the Perspective of the User

The recall from each classification test showed that the system is very likely to pick out any error that the student makes. The classification was reasonably precise with some areas needing some additional work. The most significant point of contention in the classification was the distinction between a misspelling and a postfix error. PLPrepare has some problems addressing this distinction because of the flaws addressed in earlier paragraphs, but the problem means that a user cannot be certain of the nature of PLPrepare's output. This is compounded by the fact that the genitive detection system only catches around 70% of the genitive usage cases with a similar precision. Overall, a user can expect a correct assessment only around 60% of the time. This severely impacts the usefulness of the tool in an educational context. One saving grace is that the classification system will almost always bring attention to a noun that not used correctly whether the postfix, case, or spelling is incorrect. Additionally, when the user is only working with the masculine inanimate singular the performance increases substantially.

PLPrepare clearly needs more work to be effective in an education setting, but these early results are promising.

Chapter 5. Conclusion and Future Work

This study introduced a hybrid grammar checker that was implemented to detect and catch different types of errors concerning the usage of the Polish genitive masculine inanimate singular. The novel approach was the integration of state-of-the-art dependency parsing and the available Polish NLP resources to give specific feedback geared towards language learners and their instructors. Additionally, this study introduced a framework for developing tools to help language learners master seemingly arbitrary cases by cataloging guidelines and integrating them into a grammar checker's feedback. Finally, this study attempted to infer a deeper source of the entered errors instead of simply detecting them, and the system reacted to different types of misspellings differently.

The grammar checker itself performed in step with many similar models for the genitive case [12], but many of the approach's problems will have to be addressed in future research. Given the small amount of data, there was a surprising amount of guideline coverage due to the frequency of use in the word lists, but the implementation requires much more labor to fully complete. This study pointed to the need for some statistical NLP techniques to increase robustness when dealing with erroneous input regarding this system's more fundamental building blocks. Innovating to produce these improvements will be difficult but being able to read into spelling errors and other grammatical errors with greater detail will allow systems like this to provide even more specific feedback to language learners in the future.

References

- [1] Iwona Sadowska, *Polish: A Comprehensive Grammar*. New York, NY, USA: Routledge, 2012.
- [2] O. Swan, "Polish Grammar in a Nutshell," 2003. [Online]. Available: <http://www.skwierzyzna.net/polishgrammar.pdf>
- [3] E. Dąbrowska, "Learning a morphological system without a default: the Polish genitive," *Journal of Child Language*, vol 28 pp. 545-574. [Online]. Available: <https://pdfs.semanticscholar.org/40e6/49685e0bca0177a81f97ece10b07c53528ac.pdf> [Accessed Sep. 11, 2019].
- [4] S. Anderson, "Morphology", in *Encyclopedia of Cognitive Science*, Macmillan Reference, Ltd, (n. d.). [Online]. Available: https://cowgill.ling.yale.edu/sra/morphology_ecs.htm
- [5] *An Introduction to Language and Linguistics*, Ralph W. Fasold Ed. and Jeff Connor-Linton Ed., 6th printing, Cambridge University Press, UK, 2013.
- [6] M. Kracht, "Introduction to Linguistics," (n.d.). [Online]. Available: <https://linguistics.ucla.edu/people/Kracht/courses/ling20-fall07/ling-intro.pdf>
- [7] SIL International. "SIL Glossary of Linguistic Terms: What is a clitic?" [Online] Retrieved from "Archived copy". Archived from the original on 2004-05-10. Available: <https://web.archive.org/web/20040510110313/http://www.sil.org/linguistics/GlossaryOfLinguisticTerms/WhatIsACliticGrammar.htm>
- [8] J. Eisenstein, *Natural Language Processing*, MIT Press, 2018.
- [9] M. F. Porter, "An Algorithm for Suffix Stripping," *Program: Electronic Library and Information Systems*, Vol. 14. No3, pp 130-137. [Online]. Available: <https://www.emerald.com/insight/content/doi/10.1108/eb046814/full/html>
- [10] Quizlet, "Studying," <https://help.quizlet.com/hc/en-us/categories/360001601132-Studying>
- [11] Duolingo, "How can I use Duolingo?," <https://support.duolingo.com/hc/en-us/articles/360035932192-How-can-I-use-Duolingo->
- [12] N. Bhirud, B. R.P, and P. B.V, "A survey of grammar checkers for natural languages," 07 2017, pp. 51–62.
- [13] S. Granlund, J. Kolak, V Vihman, et al. "Language-general and language-specific phenomena in the acquisition of inflectional noun morphology: A cross-linguistic elicited-production study of Polish, Finnish and Estonian," *Journal of Memory and Language*, vol. 107, August 2019, pp. 169-194. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0749596X19300440#f0005> [Accessed Sep. 10, 2019].

- [14] E. Dąbrowska, “Productivity and Beyond: Mastering the Polish genitive inflection,” *Journal of Child Language* vol. 32, pp. 191-205. [Online], Available: <http://eprints.whiterose.ac.uk/1588/1/dabrowska.e2.pdf>
- [15] G. Krajewski, “Productivity of a Polish child’s inflectional noun morphology: a naturalistic study,” *Morphology (Dordrecht)*, Vol. 22, no. 1, pp. 9-34, February 2012, ISSN: 1971-5621.
- [16] M. Kostikov, “Decomposing Morphological Rules of Polish for Adaptive Language Learning,” *International Conference on Intelligent Information Systems, IIS2013*, Chisinau, Republic of Moldova, August, 2013. pp. 223-226.
- [17] A. Patejuk and A. Przepiórkowski, “Synergistic development of grammatical resources: a valence dictionary, an LFG grammar, and an LFG structure bank for Polish,” in *Proceedings of the Thirteenth International Workshop on Treebanks and Linguistic Theories, TLT*, 13, January, 2014, pp. 113-126.
- [18] A. Przepiórkowski, E. Hajnicz, A. Patejuk, et. al. “Walenty: Toward a Comprehensive Valence Dictionary in Polish.” *Proceedings of the Ninth International Conference on Language Resources and Evaluation*, Reykjavik, Iceland, 2014. pp. 2785-2792.
- [19] A. Wróblewska, K. Krasnowska-Kieraś, “Polish evaluation dataset for compositional distributional semantics models,” *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, July 30 - August 4, 2017. pp. 784–792.
- [20] A. Przepiórkowski, E. Hajnicz, A. Patejuk, et al. “Extended phraseological information in a valence dictionary for NLP applications,” *Proceedings of Workshop on Lexical and Grammatical Resources for Language Processing*, Dublin, Ireland, August 2014. *Association for Computational Linguistics and Dublin City University*, Dublin, 2014. pp. 83-91.
- [21] M. Woliński, M. Miłkowski, M. Ogrodniczuk, et al. “PoliMorf: a (not so) new open morphological dictionary for Polish,” *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, Istanbul, Turkey, May 2012. pp. 860-864.
- [22] M. Woliński, “A Relational Model of Polish Inflection in Grammatical Dictionary of Polish,” *Human Language Technology: Challenges of the Information Society*. Berlin, Heidelberg Springer-Verlag, 2009. pp. 96-106.
- [23] Ç. Çöltekin, J. Barnes, “Neural and Linear Pipeline Approaches to Cross-Lingual Morphological Analysis,” *Proceedings of the Sixth Workshop on NLP for Similar Languages, Varieties and Dialects*, W19-1416, Minneapolis, MN, USA, June 7, 2019. *Association for Computational Linguistics*, 2019. pp. 153-164.
- [24] R. Östling. “Morphological Re-inflection with Convolutional NNs.” In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics*,

- Phonology, and Morphology, Berlin, Germany, August 2016, Association for Computational Linguistics. pp. 23-26.
- [25] R. Cotterell, C. Kirov, J. Sylak-Glassman, et al. Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection, Brussels, Belgium, October 31, 2018. Association for Computational Linguistics, 2018.
- [26] Y. Kementchedjhieva, J. Bjerva, I Augenstein, “Copenhagen at CoNLL–SIGMORPHON 2018: Multilingual Inflection in Context with Explicit Morphosyntactic Decoding.” In Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection, Brussels, Belgium, October 31, 2018. Association for Computational Linguistics, 2018. pp 93-98.
- [27] P. Markov and S. Cematide, “UZH at CoNLL–SIGMORPHON 2018 Shared Task on Universal Morphological Reinflection Neural Multi-Source Morphological Reinflection,” In Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection, Brussels, Belgium, October 31, 2018. Association for Computational Linguistics, 2018. pp 69-75.
- [28] Y. Goldberg and J. Nivre, “A dynamic oracle for arc-eager dependency parsing,” in Proceedings of COLING 2012. Mumbai, India: The COLING 2012 Organizing Committee, Dec. 2012, pp. 959–976. [Online]. Available: <https://www.aclweb.org/anthology/C12-1059>
- [29] J. D. Choi, J. Tetreault, and A. Stent, “It depends: Dependency parser comparison using a web-based evaluation tool,” in Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Studys). Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 387–396. [Online]. Available: <https://www.aclweb.org/anthology/P15-1038>
- [30] M. Honnibal and M. Johnson, “An improved non-monotonic transition system for dependency parsing,” in Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1373–1378. [Online]. Available: <https://www.aclweb.org/anthology/D15-1162>
- [31] SpaCy, Explosion, “Available trained pipelines for Polish,” 2021. [Online]. Available: <https://spacy.io/models/pl> Github Repository: https://github.com/explosion/spacy-models/releases/tag/pl_core_news_lg-3.0.0
- [32] A. Wroblewska, “Extended and enhanced polish dependency bank in universal dependencies format,” in Proceedings of the Second Workshop on Universal Dependencies (UDW 2018), M.-C. de Marneffe, T. Lynn, and S. Schuster, Eds. Association for Computational Linguistics, 2018, pp. 173–182.

- [33] M. Woliński, *Morfeusz2: Dokumentacja techniczna i użytkowa*, 2019. [Online]. Available: <http://download.sgpj.pl/morfeusz/Morfeusz2.pdf>
- [34] A. Przepiórkowski M. Bańko, et. al., *Narodowy Korpus Języka Polskiego*, Wydawnictwo Naukowe PWN, 2012.
- [35] M. Kostikov, “A Formal Model of Polish Nouns’ Inflection”, In *Radio Electronics, Computer Science, Control*, 2015. pp. 18-21.
- [36] Wiktionary Contributors, “Indeks: Polski – Części ciała,” 2010. [Online]. Available: https://pl.wiktionary.org/wiki/Indeks:Polski_-_Cz%C4%99%C5%9Bci_cia%C5%82a
- [37] Wikipedia Contributors, “List of building types,” 2021. [Online]. Available: https://en.wikipedia.org/wiki/List_of_building_types
- [38] Wikipedia Contributors, “Kategoria: Tańce,” 2015. [Online]. Available: <https://pl.wikipedia.org/wiki/Kategoria:Ta%C5%84ce>
- [39] Wiktionary Contributors, “Indeks: Polski – Owoce,” 2020. [Online]. Available: https://pl.wiktionary.org/wiki/Indeks:Polski_-_Owoce
- [40] Wikipedia Contributors, “Owoce jadalne.” 2020. [Online]. Available: https://pl.wikipedia.org/wiki/Owoce_jadalne
- [41] Wikipedia Contributors, “Kategoria: Środki transport.” 2013. [Online]. Available: https://pl.wikipedia.org/wiki/Kategoria:%C5%9Arodki_transportu
- [42] Wikipedia Contributors, “Kategoria:Jednostki miar i wag.” 2013. [Online]. Available: https://pl.wikipedia.org/wiki/Kategoria:Jednostki_miar_i_wag
- [43] “Encyklopedia przypraw.” (n.d.) [Online]. Available: <http://www.przyprawowy.pl/encyklopedia-przypraw.html>
- [44] Wikipedia Contributors, “Kategoria: Przyprawy,” 2017. [Online]. Available: <https://pl.wikipedia.org/wiki/Kategoria:Przyprawy>
- [45] Wikipedia Contributors, “Kategoria: Warzywa,” 2021. [Online]. Available: <https://pl.wikipedia.org/wiki/Kategoria:Warzywa>
- [46] Wiktionary Contributors, “Indeks: Polski – Warzywa,” 2020. [Online]. Available: https://pl.wiktionary.org/wiki/Indeks:Polski_-_Warzywa
- [47] O. Swan, N. H. Reimer, B. L. Wolfe, “Lektorek.org,” 2021. [Online]. Available: <https://lektorek.org/polish/>
- [48] J. Nivre, M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, and D. Zeman, “Universal Dependencies v1: A multilingual treebank collection,” in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC’16)*. Portorož,

- Slovenia: European Language Resources Association (ELRA), May 2016, pp. 1659–1666. [Online]. Available: <https://www.aclweb.org/anthology/L16-1262>
- [49] Spacy, Explosion, “lemmatizer.py,” *Github*, 2021. [Online]. Available: <https://github.com/explosion/spaCy/blob/master/spacy/pipeline/lemmatizer.py>
- [50] A. Przepiórkowski A. Buczyński, J. Wilk, “The National Corpus of Polish Cheatsheet”, 2011. [Online]. Available: <http://nkjp.pl/poliqarp/help/en.html>
- [51] M. Wolinski and W. Kieraś, “The on-line version of grammatical dictionary of Polish,” in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC’16)*. Portorož, Slovenia: European Language Resources Association. Portorož, Slovenia: Association (ELRA), May 2016, pp. 2589–2594. [Online]. Available: <https://www.aclweb.org/anthology/L16-1412>

VITA

JACOB HOYOS

Education: Master of Science in Computer Science, Applied Computer
Science, East Tennessee State University, 2019-2021
Bachelor of Science in Computing, Computer Science, East
Tennessee State University, 2015-2019
Summer Language Institute, Pittsburgh, PA, 2017-2020
Public Schools, Alcoa, Tennessee, 2011-2015

Professional Experience: Graduate Teaching Assistant, East Tennessee State University,
Department of Computing
Johnson City, Tennessee, 2019-2021
APS Assistant, East Tennessee State University,
Department of Computing
Johnson City, Tennessee, 2015-2019
IT Contractor, Alcoa City Schools,
Alcoa, Tennessee, 2015-2016

Honors and Awards: Outstanding Graduate Student in Computing Award, 2021
Outstanding Senior in Computing Award, 2020
Upsilon Pi Epsilon Inductee, 2019
Foreign Language and Studies Scholarship, 2018
Dean's List 2015-2021