

THESIS APPROVAL FORM

NAZARBAYEV UNIVERSITY

SCHOOL OF SCIENCES AND HUMANITIES

BUILDING A CORPUS OF JOURNALISTIC KAZAKH LANGUAGE USING
NEURAL NETWORKS FOR AUTOMATED PART-OF-SPEECH TAGGING

BY

Nikolay Mikhailov

NU Student Number: 201345263

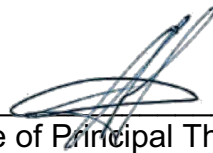
APPROVED

BY

DR. ANDREY FILCHENKO

ON

The 2 day of June, 2021



Signature of Principal Thesis Adviser

In Agreement with Thesis Advisory Committee:

Second Adviser: Dr. Benjamin Tyler

External Reader: Dr. Jonathan Washington

Building a corpus of journalistic Kazakh language using neural networks for automated part-of-speech tagging

by

Nikolay Mikhailov

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Arts in Eurasian Studies

at

NAZARBAYEV UNIVERSITY - SCHOOL OF SCIENCES AND HUMANITIES

2021

Abstract

The thesis explores the status quo of the Kazakh language in terms of corpus linguistics. The project aims to survey the currently existing corpora of the Kazakh language and contribute to the existing body through a more flexible and more automatic way of corpus building and annotation. Upon the examination of the field, it was determined that while there are some efforts to digitize the Kazakh language, those projects are largely still being developed. They are conducted on various scales — from small student projects to the projects led by Mozilla and big research groups, like Apertium. Therefore, this project set out to attempt to build a corpus of journalistic Kazakh language using neural networks for part-of-speech tagging. In order to construct the corpus, news websites were used as a source, as they provide a decent vocabulary range while remaining easily accessible. The project utilized a series of small-scale Python programs to create the body of data to be annotated via obtaining the text from the web pages. The final stage of the study involves using the neural networks in order to assign the words their respective parts of speech. Neural networks provide an automatable way of doing part-of-speech tagging that is faster compared to humans, with an accuracy that can be almost equal to that of humans. In addition, while using the neural networks is a known way to approach the tagging and annotation, it has not seen use in Kazakh corpus linguistics as of yet. The final model was able to assign the correct parts of speech to words with a reasonable degree of accuracy, which could still be improved by providing a bigger sample of training data. The project can be later utilized to build a more extensive corpus with a high degree of automation, lowering the time expenses.

Acknowledgments

I would like to express my sincere gratitude to my first advisor, Dr. Andrey Filchenko from Nazarbayev University, my second advisor, Dr. Benjamin Tyler from Nazarbayev University, my external reviewer, Dr. Jonathan Washington from Swarthmore College, and Dr. Eva-Marie Dubuisson from Nazarbayev University for their assistance in writing this thesis.

Table of Contents

Abstract	i
Acknowledgments	ii
Chapter 1: Background	1
Introduction	1
Importance of the research	4
Aims and roadmap	7
Literature review	9
Chapter 2: Theoretical foundation	14
Finite-state machines	14
Context-free grammar	17
Kazakh language	18
Chapter 3: Methodology	22
Research methods background	22
Corpus analysis software	24
Corpus analysis methodology	27
Corpus analysis in practice	30
Empirical considerations	33
Chapter 4: Practical approach	34
Practical challenges	34
Cyrillic and Latinized alphabets	35
Some practical considerations regarding the coding stage	38
Wildcard approach to morphology	41
Pronominal hardcoding	41
Advertising and suggested content	43
Classification according to the word order	45
The development process	48
Converter v.0.1.0-C — Alpha stage	49
Converter v.0.2.0-C — Second stage	50
Converter v1.0.0-Py — First stage	56
Converter v1.1.0-Py — Second stage	57
Scraper	58

Neural network approach	62
General information	62
Neural network with word percentile as a weight-influencing factor	66
Neural network based on morphological features with cross-references to training dataset as a dictionary	72
Chapter 5: Conclusion.....	73
Conclusion	73
Project outlook	77
References.....	79
Appendix A.....	84
Appendix B.....	84

Chapter 1: Background

Introduction

Corpus linguistics, as part of applied linguistics, has seen lots of development, thanks to technological development. Corpus linguistics, according to the consensus on the definition, is a method of studying languages that involves quantitative assessment of a collection of texts known as corpus¹. A corpus is a collection of machine-readable texts which are sampled to be representative of a particular language or language variety². As the possible computational and processing power of the machines grows, so grows the scientific significance of corpus linguistics. Its possible applications are rather diverse, but perhaps, two of the most outstanding fields are machine translation, and, somewhat less obviously, the judicial system – forensic linguistics is a field that relies on corpus, among other options. The former is discussed in an article by Bekbulatov et al. (2014), titled “A Study of Certain Morphological Structures of Kazakh and Their Impact on the Machine Translation Quality”. While the application of corpus linguistics is not the main topic of the paper, it is shedding light on one of the applications of corpus linguistics, which is machine translation — using content analysis to translate the content automatically. Still, the authors do highlight that improving the existing systems for Kazakh language translation could bring significant benefits, such as better software adaptation, etc. The benefits to the judicial system, which might not be as evident, have been described by several authors. The article “Advancing Law and Corpus Linguistics: Importing Principles and Practices from Survey and Content Analysis Methodologies

¹ Norbert, 2010

² McEnery et al, 2006

to Improve Corpus Design and Analysis”³ showcases precisely that — the application and methodology for use of corpus linguistics. The authors argue that the introduction of broadly social science and humanities’ methods, including linguistics, into the judicial system will allow for greater clarity alongside better rigor and improved transparency. They start out with the general application of HSS methodologies and then move to the next chapter which talks about content analysis and coding methodologies, which is what corpus linguistics uses as well. Alongside its practical point of view, the article also offers a more theoretical part, which reveals the appropriate methodologies when applying corpus linguistics. The authors select four approaches to code the information — minimalist, dictionary-driven approach, grounded theory, and register selectivity, and each of those is different in terms of goals, and methods used: while some concentrate on one particular form of a verb, others attempt to analyze the corpus and come up with a theory based on the results

Still, the theory outlined above falls short of providing a real-life example for the use of corpus linguistics, but there are still studies that showcase specific court cases and the ways corpus linguistics helped the judicial system. One of those articles, called “Ordinary Meaning and Corpus Linguistics”, explores the legal cases within the United States jurisdiction that saw the impact of corpus linguistics. One of those is defining what an ordinary meaning is based on a corpus⁴, which could help define if a person means something less obvious with the words based on what the ordinary meaning is. Another set of examples is given in a different book, “Advancing Law and Corpus Linguistics: Importing Principles and Practices from Survey and Content-Analysis Methodologies to Improve Corpus Design and Analysis”⁵, which

³ Philips et al, 2015

⁴ Gries et al, 2017

⁵ Phillips, 2017

goes into the details on where the corpus linguistics could help with the evidence verification and testimony assessment. Starting from determining whether a particular person read off of a script or said the words themselves, and whether or not the words belong to a particular person. The society in Kazakhstan can benefit from those advancements greatly, with increased transparency in the judicial system.

In addition, the lack of development in the corpus linguistics area creates a technological void in terms of voice-assistive technologies in Kazakh language. Some of the voice-assistive technology applications rely heavily on speech synthesis – using the vocabulary according to the situation. Granted, it is possible to create pre-recorded responses for some situations (for example, the currently implemented voice notification system at the traffic lights for vision-impaired people), yet when it comes to different situations (such as reading messages out loud) that require a wider range of responses, the technologies are falling short still. Kazakhstan is a new prospective market for the voice assistants. In 2013, Kazakhstan had 6 major smartphone OS companies⁶, featuring iOS and Android as some of the most popular operating system choices. Both Apple and Google have their own voice assistants — Siri and Google Assistant, respectively. Therefore, there is a ready foundation of the devices that can be used for the deployment of voice assistants. However, currently neither Siri or Google Assistant offers a solution for Kazakh language-speaking voice assistants. The efforts have been made by the Russian company Yandex⁷, yet they are still to come to fruition. While Kazakhstan might not constitute a significant portion of the market share for the big IT companies in the sphere, there can be other reasons that make the Kazakh language a less attractive option than others.

⁶ Galiev, Alexander. June 25, 2013.

⁷ Kodachigov, Valerii September 2, 2018.

Importance of the research

My research project is going to focus on the adaptation of the existing corpus analysis methodology to the Kazakh language and attempting to bring neural networks into a set of approaches that have been used. Kazakhstan is a particularly good setting for this type of research. There are several reasons for that, however, the most important one is that while the Kazakh language appears to be largely on track in terms of being digitized, it is still somewhat understudied in terms of the corpus methodology. This is a rather serious hindrance both in terms of applied technology and scientific research. In the scope of this thesis, language being “digitized” essentially means that it has been sufficiently covered by the corpus research, enabling both scientific research via corpus studies to be conducted, and commercial projects to be carried out.

Corpus linguistics has various applications in daily life. The first application of the methodology is the voice processing technology. Programs like Cortana, Siri, and others rely on corpora, among other things, to “understand” human speech and respond to it correctly. As a side note, speech synthesis directly benefits from the advances made in corpus linguistics. Voice-assistive technology can help various groups of the society: for example, visually impaired people will be able to access much more than they are currently able to do; smartphone and laptop users will be able to interact with their devices more efficiently, and those are just 2 examples. A particularly good example of the application of corpus methodology outside of the academic research is the use of corpus in the judicial practices, as outlined in “Advancing Law and Corpus Linguistics: Importing Principles and Practices from Survey and Content Analysis Methodologies to Improve Corpus Design and

Analysis”⁸. With the sufficient development of the corpora, it is possible to assess whether the witness is reading off a script, or speaking for themselves, which opens a new way of forensic analysis. Granted, this type of evidence will still need to gain legal trust before being admitted in court in Kazakhstan, however, it has to start at some point.

The status quo of the corpus methodology for the Kazakh language is that not much has been done for it. First, overall the research on the Kazakh language using corpus methodology has not been particularly advanced. There is a corpus of Kazakh language that has been built several years ago, and the “unofficial” headquarters of the corpus is located in Almaty⁹. The corpus is in digital form, and the search tools are available to the public for use; however, the body of the corpus cannot be accessed by the public freely through the website. The authors of the corpus claim to have gathered millions of words across various texts. However, it appears that the corpus is mostly focusing on the literary Kazakh language. While such a corpus is useful to a certain extent, it falls short of showing real-life examples of Kazakh language use patterns. The corpus can be used for researching literary Kazakh language. Still, another problem with the Almaty corpus is that it is not being maintained regularly. The website indicates that it is receiving some updates from a group of researchers, however, the latest changes date back to 2015, which is a relatively long time when it comes to the corpus studies. While the older text will not change, and thus don’t require updating, a corpus that is not being updated does not include newer texts, and therefore might not be representative of the newer trends, changes in the language. In addition, given that there were no updates to the corpus

⁸ Philips et al (2015)

⁹ http://web-corpora.net/KazakhCorpus/search/?interface_language=ru

in 6 years, it is highly likely that the corpus search system has not been updated either, which makes the research all the more troublesome. There is also another corpus, affiliated with Eurasian National University, and National Laboratory Astana, called Kazcorpus¹⁰; it contains 135 million words from 445078 annotated documents – to quote, “KLC is designed to be a large scale corpus containing over 135 million words and conveying five major stylistic genres (domains): literary, publicistic, official, scientific and informal”. Finally, a Universal Dependencies corpus¹¹ is currently in the works for the Kazakh language. All the projects mentioned above demonstrate that while the Kazakh language is definitely making a strong case for becoming a digitized language, there is still a lot to be done. In addition to that, there have been several attempts to build a corpus by some universities as part of their students’ projects, however, those have either not resulted in a finished corpus or one that is readily available.

At the moment, there is no ready Kazakh language corpus that focuses on the spoken language that is available to the public, but some are in the works at the moment. Some of the more recent developments in this area include QazCorpus¹², which is a new corpus, available online, that does have the subcorpus of spoken Kazakh. Another effort comes from the CommonVoice¹³ project by Mozilla – it has recently gathered enough oral data in the Kazakh language to start working. Finally, ISSAI¹⁴ has launched a Kazakh speech corpus project quite recently, and the project is currently in the data collection stage. While all those projects are a welcome contribution to the studies of the Kazakh language using corpus methodology, at the

¹⁰ <http://kazcorpus.kz/klcweb/en/>

¹¹ https://github.com/UniversalDependencies/UD_Kazakh-KTB

¹² qazcorpus.kz

¹³ Mozilla CommonVoice

¹⁴ ISSAI, Kazakh Speech Corpus

moment, none of those offer an opportunity for the researchers to download the corpus data and work on it; some of the projects are still in the early stages and do not have enough data yet. Yandex, a Russia-based web technology company, has announced its plans to develop voice recognition software for the Kazakh language as early as 2019¹⁵, however, at this point nothing has been released.

Aims and roadmap

The project aims to answer several questions. The first question can partially be answered within the course of this literature review: what is the current state of Kazakh language corpora? What kinds of corpora exist, and how could one approach the creation of a Kazakh language corpus? In addition, how could one go about analysis of such a corpus? What kinds of tests and methodologies need to be employed? The next question concerns the software available for corpus analysis — what are the tools currently available for the corpus analysis? Following that, I am going to ask how well do those tools work with Kazakh language, and survey those that are able to do so, providing a general idea of the possible tools one could use even for future research in this area. I will take a particularly close look at Apertium, since it is freely available open-source, which means it can be analyzed and improved easily, unlike commercial products or corpus-specific tools, so the practical outcome of the project can be immediately implemented with no costs associated.

The research concludes with the implementation of a neural network, the main task of which is to classify the input (the words) into the parts of speech after the training. The neural network is built as a multi-layered perceptron, which allows it to complete the classification as accurately as possible given appropriate training

¹⁵ Kodachigov, Bryzgalova, 2019

data. In this case, the training data is the corpus prepared beforehand - an actual annotated body of text; for this research, it is a dictionary-styled body. Instead of using full sentences, I opted for sentences decomposed into words that have been annotated. The dictionary is a combination of the original data, which have been assigned their corresponding parts of speech semi-automatically, with the corpus data provided by Apertium, an open-source software that was already described in the paper.

There is hardly a fixed size that can be deemed “exactly right” for the task - the more there is of the training data, the higher is the accuracy on the sample of test data. Still, there is a high possibility of diminishing returns past a certain point, when more words added to the training data will not greatly improve the accuracy, but will instead increase the training time. For example, a thousand words added to the corpus of two thousand will definitely improve the accuracy of the algorithm; while a thousand words added to a hundred million words worth of training data will not contribute significantly.

Granted, the size of the training dataset is not a sole defining factor of accuracy – the quality of the data, its diversity, and the number of epochs all contribute to the output. Epoch in the context of machine learning and neural networks refers to the number of times the algorithm has been trained with a particular dataset – the higher the number of epochs, the more times the algorithm has been allowed to run, which means more precise tuning of the weight, which leads to more accurate outcomes.

Literature review

Norbert et al. (2010), in the publication “Introduction to applied linguistics”, takes a brief detour into the history of corpus linguistics. Corpus linguistics, as a branch of applied linguistics, has developed in its current sense in the middle of the 20th century; however, it has been used in its more rudimentary sense even before that, particularly in religious studies, by various scholars. However, for this project, I would refer to the modern sense of corpus. In its current meaning, according to McEnery et al. (2006:5), given in the book “English Language: Description, Variation and Context”, it is a collection of machine-readable texts which are sampled to be representative of a particular language or language variety¹⁶. Accordingly, corpus linguistics is a branch of science that studies the languages using the corpus. A detailed analysis of the definition is a good starting point towards developing the methodologies. First, a corpus should be readable by a machine. This outright rules out the use of printed materials as a corpus — unless they have been transcribed electronically, they cannot be used as a corpus. Authenticity means that the text should not be done specifically for corpus, but instead should be real-life texts. Finally, the texts should be sampled to be representative of the language or language variation. Corpus linguistics focuses on processing large amounts of data and using systems to break the language down into basic patterns that can be used later. In a sense, modern corpus linguistics draws on the notions introduced by Noam Chomsky in the mid-twentieth century.

Chomsky advanced the idea of grammar as a set of rules that can be combined and changed using more rules, in his article “Three Models for the Description of Language”, published in 1956. The idea of describing languages via

¹⁶ McEnery et al, 2006

what is called a “finite-state machine”, or a more abstract model, context-free grammar, was not revolutionary in and of itself, but it opened the ways to use the natural language in a different way. There is a misconception¹⁷ regarding Chomsky’s approach to language – initially, it was not the intent of Chomsky to use the language as a set of fixed rules, as he noted himself, saying that many languages do not adhere to a set of rules, no matter how complex. What he did, though, was laying the foundation for the study of language based on the patterns. Granted, finite-state machines are not the only way of describing language.

Corpus linguistics does not limit itself purely to pattern discovery for the sake of pattern discovery. It has some practical applications as well, one of the most notable ones being its application to judicial practices, as forensic linguistics is a field that can definitively be aided by the developments made in corpus linguistics. While this is still largely a work in progress, it has been demonstrated that in many cases, corpus linguistics can determine certain patterns and help the experts decide whether or not a particular text exhibits features of a particular genre — for example, a terrorist threat, as was described by Gries (2017) and Phillips (2017). The same approach could also help detect if a particular speech excerpt is likely to be someone simply talking, or reading off of a script, or if those words belong to that person at all. The general approach employed by corpus linguistics, when combined with biology, psychology, criminal studies, etc, could change the way the justice system works, and not only the justice system, but also many different areas, like machine translation, voice assistive technologies, and others.

The research project is on corpus research in Kazakh language. Kazakh language has not seen much attention in terms of computational analysis, and the

¹⁷ Horgan, 2016

data on the language is not up to date with the modern standards. In addition, the corpora on Kazakh language are not systematized well¹⁸. While there are efforts to bring the corpora to a unified structure, ultimately, as of currently, the goal remains unachieved. In addition, there is no unified program that could deal with the Kazakh language: while the most popular Kazakh corpora do have their own tools¹⁹, they are largely unavailable outside the field, and they are not uniform in their nature, which makes it tough for scholars to use the resources available.

Machine translation and speech synthesis/recognition are currently one of the bigger topics in computational linguistics — as the technology develops, the demand for voice-assistive technology will likely rise. However, not all languages enjoy equal representation in the field — some, like English, are on the forefront, while others, like Kazakh, are in the very early stages. Such a discrepancy can hardly be attributed to one particular issue — it's likely there are several factors that make a certain language more prospective for voice-assisted technology. Aside from the market and popularity factors, the languages can suffer negligence due to the available data and approaches to the analysis, which hinders the effective development of technology. Such a drastic difference in terms of digitalization has even earned itself a name — “digital divide”, which refers to the unevenness of the world languages in terms of digital coverage. As mentioned before, some of the world languages with the largest number of speakers, or those located within a particular geopolitical border, enjoy cutting-edge technology (e.g. English, Russian, French), while others, smaller regional languages, are unavailable, or limited, within the currently present software. Kazakhstan is a new prospective market for voice

¹⁸ <http://kazcorpus.kz/klcweb/en/primary/>

¹⁹ [Алматинский корпус казахского языка \(АККЯ\)](#)

assistants. This is a rather problematic perspective for the development of corpus linguistics, especially for minor languages – if the scientific studies of language are dictated solely, or at least for the most part, by its marketability, the commercial interest may outweigh the scientific one, showing the bias towards more “popular” languages. This may also lead to the “digital death” of a language - lack of representation in the digital domain.

In 2013, Kazakhstan had 6 major smartphone OS companies, featuring operating systems by Apple and Google as some of the most popular options. Both of those companies have their own voice assistants — Siri and Google Assistant, respectively. Therefore, there is a ready foundation for the devices that can be used for the deployment of voice assistants. However, currently neither Siri or Google Assistant offers a solution for Kazakh language-speaking voice assistants. Moreover, even if there was a corpus of the Kazakh language that is comprehensive enough to work well with voice-assistive software, there is very little that can be done to expedite the process if the software companies mentioned above do not actively take the steps. Given that, some scientists have pushed for “Open Language Technology”²⁰ – essentially, having equal access to the standards, interfaces and resources for the languages. Some efforts have been made by the Russia-based company Yandex, yet they are still to come to fruition²¹.

One software that is capable of working with the Kazakh language is called Apertium (the module that works specifically with the Kazakh language is called Apertium-kaz). Apertium is a “free/open-source platform for rule-based machine translation”, and anyone can contribute to it, use it, study how it works, or modify it,

²⁰ Sjur et al, 2019

²¹ Kodachigov, Valerii, September 2, 2018.

which is why this platform was chosen as the basis of research. The philosophy of the open-source, as laid out by GNU, is that for a program to be considered truly free and open-source, the program has to be able to be run by anyone, has to be able to be modified by anyone (having an open-source code is a natural condition for it), its copies have to be able to be freely distributed, as well as the copies of the modified versions²². By analyzing the current data sources of Apertium and the methodology of morphological transducers used, this research project will attempt to create a more optimal Kazakh language morphologizer, and improve the database used by Apertium-kaz. Apertium is free software, currently available online, which is capable of machine translation and morphological analysis of words. There are 2 versions of any given Apertium package — a stable, non-beta version, and a beta version. The Kazakh language is represented in both of those categories — while currently only the Kazakh-Tatar version has been deemed “stable”, Kazakh is nonetheless represented in other translation pairs as well. While only 1 language pair with Kazakh is stable at the moment, more are currently being worked on, and given the attention Turkic languages receive²³, it is a matter of time before the coverage of Kazakh language becomes substantial in the open-source domain. Given the current tools, the Kazakh analyzer used in Apertium is capable of breaking down most words and indicating the possible part of speech it belongs to based on the morphology, even though there may be ambiguous cases.

However, while Apertium is a great open-source tool, it also has its shortcomings, which, while not immediately visible to the end user, are still present and can affect the overall quality of the tool. Turning back to the lexc file, some of the

²² <https://www.gnu.org/philosophy/free-sw.en.html>

²³ Washington et al, 2019

areas that can be improved are immediately noticeable. The software creators acknowledge the fact that some parts of the parser are still not done, and the ways to implement the solutions are not feasible as of yet. For example, at the moment the parser is unable to deal with the superlatives that have been created by partial reduplication of the first syllable, such as “qap-qara”, it does not differentiate between the negative pronouns, instead putting them all into a single category. By and large, while the integral components of the system are functioning very well, even as a beta/non-stable version, there are still some points that would require work, such as the ones described above.

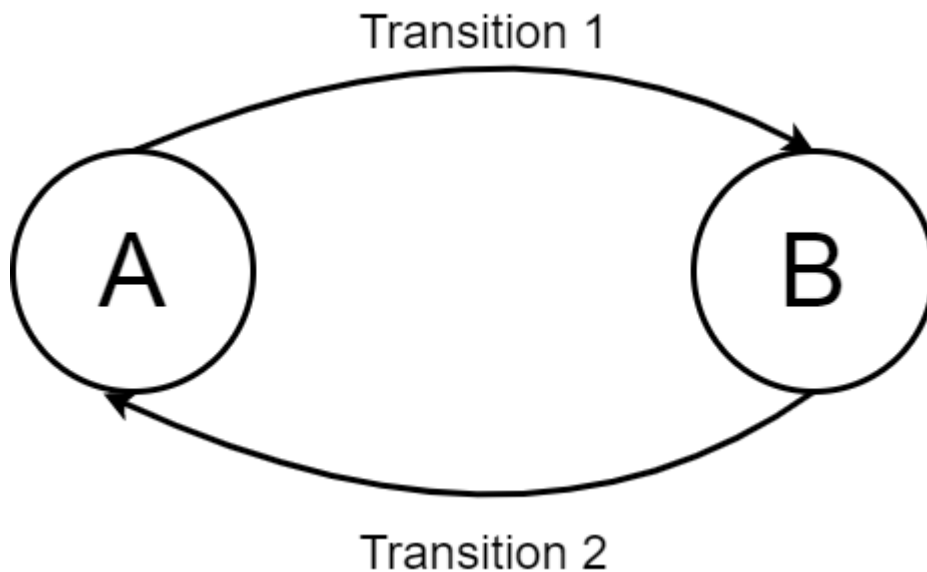
Chapter 2: Theoretical foundation

Finite-state machines and context-free grammar are rather important concepts in the context of natural language processing. Essentially, they are abstract models of language — it is through them that many languages can be broken down into a set of generalizations for rule-based parsing, among other tasks.

Finite-state machines

The first notion to be discussed is a finite-state machine (FSM). A finite-state machine is a model of computation that can be represented as a graph with the possible states of the machine as its nodes, and the transitions between the states as the vertices connecting the corresponding states²⁴. The number of states is also limited, hence the name “Finite-State Machine.” At any given moment in time, the model can only be in one state, so there is no ambiguity in that regard. The machines may stop at any state which is deemed to be a “final state”.

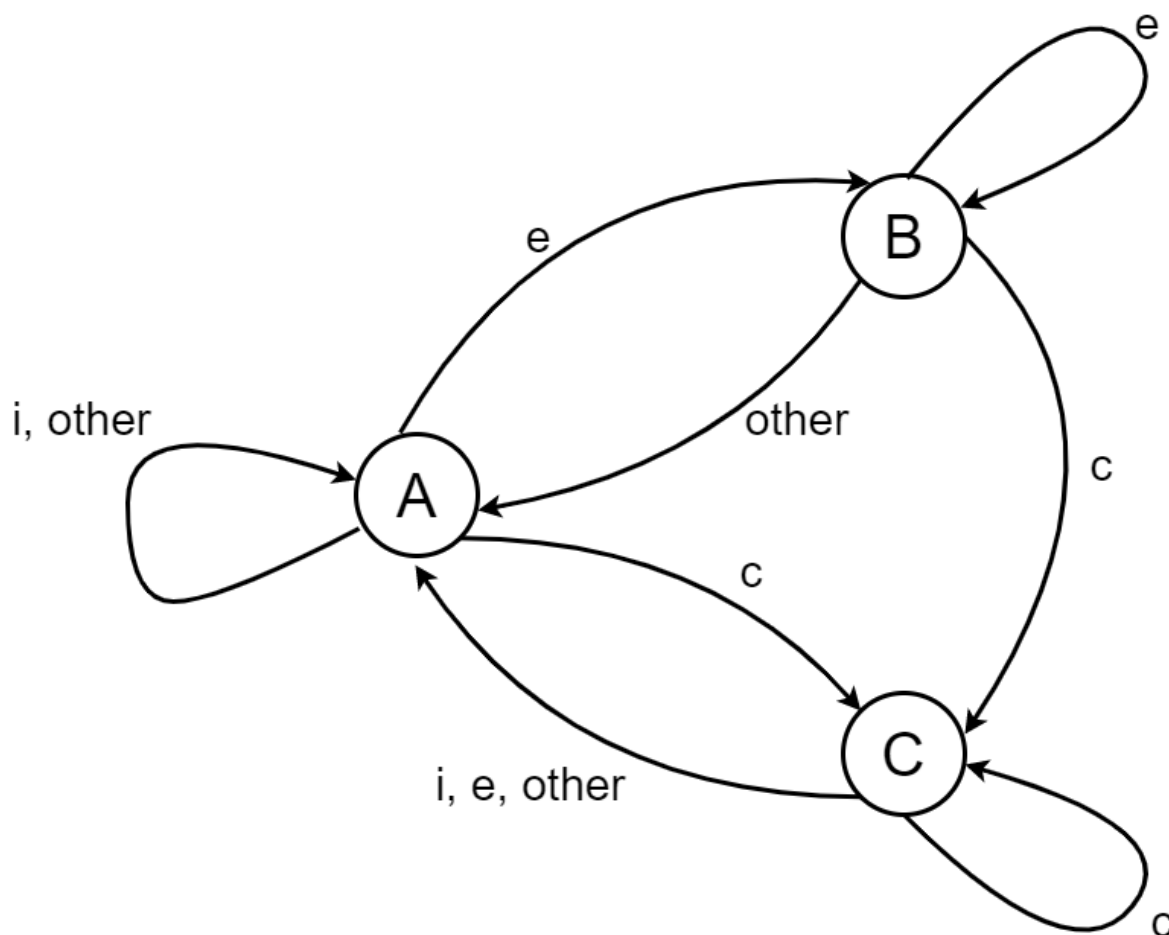
²⁴ James et al, n.d.



Let's take the figure above as an example of a finite state machine, with a simple example — a door. According to the scheme above, a door can only be in a state A (closed) or state B (open). Those two states cannot be manifested at the same time within the same finite state machine (the same door) — the door is either open or closed, not both or neither at the same time. The transitions between the two states are the actions — pushing and pulling, for example. In order to change the state of the door from closed to open (from A to B), someone has to pull the door — the action of pulling is Transition 1, and the door changes state, as reflected by the finite-state machine. To make the door closed again, Transition 2 has to take place — pushing the door makes it close, and the finite-state automaton changes its state once again from B to A.

Now, let's consider a more intricate example that is related more to linguistics rather than a physical object. One of the mnemonic devices of English language states “i before e except after c”, describing the distribution of ie/ei digraphs in English spelling. This can also be represented with a finite-state machine, which would look more complicated²⁵.

²⁵ Stanford, n.d.



The state A in the machine symbolizes “not after c or e — anything but ei”.

This is the very first state, and nothing special has occurred yet to change the machine to a different state. As long as neither c nor e occurs, but instead something else (the “*i, other*” transition in the picture, the machine will stay in that state.

However, as soon as there is e in a word, the machine changes its state to state B, which is “no i, we have an e”. If e keeps occurring, the machine will keep staying in that state, which is indicated by the “e” transition in the picture. From this state, the machine may either return to state A (if anything other than c, e, or i occurs) or shift to state C, which is what happens when the machine encounters the letter c in a word. Once it does, the machine changes its state via “c” transition to state C, which is “anything goes after c”. The machine may stay in this state as long as more c’s are encountered right after the initial one; it may return to state A once it encounters

anything else, however, it may not return to state B, which is “we have an e, so we cannot have i”, because the rule permits it, therefore, the transition from C to B is unavailable. Returning to state A, the system may immediately change back to state C instead of B, which is the first route described above, if the machine encounters c.

Finite state machines are used in computational linguistics for analyzing morphology and phonology. The same concepts can be applied also to syntactic analysis, however, their use in this domain is more limited.

In this project, finite-state machines are used for morphological analysis. They are the most optimal tool for this kind of task — morphemes, or a group of morphemes, can all be represented as nodes, and the rules, like the vowel harmony, can be represented accordingly on the transitions. This way, it is possible to represent the vocabulary of the Kazakh language via a finite set of models.

Context-free grammar

A context-free grammar is a tool often used to describe formal languages, such as a programming language. A context-free grammar is a set of rules that can describe all the grammatically possible sentences in a language; the language generated via context-free grammar is called a context-free language. In its simplest form, context-free grammar describes how a language can be constructed from blocks, and if those simple blocks can be deconstructed further to the level of individual words.

For example, let's take a simple unit - a sentence, and represent it by S. S can be broken down into 2 S's (when 2 sentences are combined into one complex sentence). This can be represented as $S \rightarrow S+S$. S can also be a noun phrase+verb phrase- a very simple sentence. This is $S \rightarrow NP+VP$. A noun phrase can be a single

noun ($NP \rightarrow N$), or an adjective+noun phrase ($NP \rightarrow Adj+NP$). A verb phrase, in turn, can be a single word ($VP \rightarrow V$), or 2 verbs, if one of them is auxiliary ($VP \rightarrow V+V$). It can also be an adverb plus verb phrase ($VP \rightarrow Adv+VP$). Therefore, a brief example of the context-free grammar from this paragraph would be

$$S \rightarrow S+S$$

$$S \rightarrow NP+VP$$

$$NP \rightarrow N$$

$$NP \rightarrow Adj+NP$$

$$VP \rightarrow V$$

$$VP \rightarrow V+V$$

$$VP \rightarrow Adv+VP$$

Using those “rules”, we could construct all the possible “sentences” allowed by this context-free grammar. This is helpful when it’s needed to describe a language to the machine as a set of production rules on what is acceptable and what is not on a sentence level.

Kazakh language

Before any attempts to study the language can be made, it’s vital to gather some background information on the language. For this, I am going to use the previously mentioned WALS.info website alongside a book by Muhamedowa (2018) to establish the relevant background for Kazakh language. Such a background is necessary to advance further in terms of machine recognition. “World Atlas of Language Structures” can give a good general starting idea of the language family Kazakh language belongs to. WALS is a database that accumulates information about 2,679 languages and their features, such as grammar, phonetics, morphology

and even lexical categories. Still, it is not written by a native Kazakh speaker, and the aspects of language it lists are not exhaustive. To be precise, WALs, which relies on sources between 1962 and 1998, which indicates it's rather outdated, lists 9 features of Kazakh language: three of those relate to nominal categories, 3 are lexical, 1 phonological feature, 1 feature of simple clauses and 1 verbal category. While all those categories are useful for typological perspective, they would serve very little purpose for corpus studies. In particular, a feature like the word "tea" being derived from a particular Chinese dialect would do very little for this project. On the other hand, a feature like "Associative same as additive plural" could help in differentiating between various structures in Kazakh language. While it does not excel in describing Kazakh language specifically, it gives a good overview of Turkic language family. Overall, though, the quality of descriptive literature on Kazakh language is not on par with the literature available, that is written by native Kazakh speakers.

Given the state of WALs on details regarding the Kazakh language, it would be appropriate to study some sources coming from native Kazakh speakers; the body of literature on the Kazakh language written by Kazakh authors in English is not particularly rich, but there are still some books that are available. One of the best overviews of Kazakh language has been written by Raihan Muhamedowa, titled: "Kazakh: A Comprehensive Grammar". The author provides a thorough overview of several aspects of Kazakh language, which are: phonology, morphology, with an emphasis on verbs, and phonology. Still, some aspects of the Kazakh language are overlooked in this book: for example, the superlative via initial syllable reduplication is not listed as a possible option.

Phonology is not particularly useful for the purposes of corpus research, as it is unlikely to be reflected anywhere; therefore, I will only use the book for its description of Kazakh morphology and syntax, as those will be helpful in later stages of the project. However, phonology should not be dismissed outright; the author mentions that it could be used during transcribing speech. While this project does not deal with the transcription issues, phonetics could help make the data more uniform by encoding it using some phonological features. The encoding will not constitute a transcription, but rather a piece of metadata. In terms of metadata, it could prove to be a useful tool for the creation of metadata to help with the analysis. While transcription using phonetics could be disputable, since the same text can be read differently by the speakers with different accents, thus producing conflicting data, if the phonetic transcription can be treated as “metadata”, those disputes can be avoided.

For the purposes of this project, phonological classification and features of Kazakh language present little value, as they are unlikely to be reflected during the creation of corpus. On the other hand, syntactic and morphological features would be important here. Muhamedowa places an emphasis on the fact Kazakh is an agglutinative language; this same fact has been emphasized by Bekbulatov et al as one of the features that makes machine translation problematic. Therefore, this is one of the issues that could make machine recognition harder: agglutinative morphology. In terms in of syntax, however, Kazakh language follows the structure common to all Turkic languages²⁶, with the same SOV structure, which should not prove too challenging for machine processing, as the authors note.

²⁶ WALS.info; Muhamedowa, 2016

One of the more practical considerations in terms of morphological analysis is the expanded inventory of “parts of speech”. The “traditional” inventory, which is described commonly in various textbooks, does provide a solid foundation, however, it is precisely that – the foundation to build upon. In this sense, the lexical file of Apertium, available on GitHub, offers a peek into some of the practices commonly used for creating a parser. Aside from the commonly established parts of speech (nouns, verbs, adjectives, adverbs etc), the morphological parser should be able to distinguish between proper and regular nouns. To be more precise, the nouns marked as “proper”, can potentially present a challenge, as they should be treated separately. At the moment, Apertium-kaz distinguishes 6 types of proper nouns in Kazakh: toponym (which is a name of a place), anthroponym (name of a person), cognomen (last name of a person), patronym (middle/father’s name), organization and other. This kind of information should be presented in the metadata about the text, so that the morphologizer is able to process those correctly.

It’s necessary to note, though, that the topic of morphological features of the Kazakh language with regards to machine translation has been addressed before, but the articles on the topic are scarce. One of the articles on the Kazakh language and its challenges for machine interpretation is called “A Study of Certain Morphological Structures of Kazakh and Their Impact on the Machine Translation Quality”, published in 2014 by Bekbulatov et al. The authors acknowledge there is a body of research regarding machine translation and language processing, but point out there is one paper that deals with Turkish-English language pair — while Turkish is a language closely resembling Kazakh, some of its aspects are still different. The article provides a very brief overview of Kazakh language from a morphological standpoint, specifically highlighting agglutinative morphology. The agglutinative

morphology of the Kazakh language created some problems for the segmentation of the raw corpus. In order to correct the segmentation, the researchers used a mix of several algorithms, including Naive Bayes classification, and leaving the unsegmented portions to finite state machines. Their selection of morpheme separation methodologies sheds light on the possible problems regarding the corpus approach towards Kazakh — as was noted before, it is essential that a machine can process the corpus, and the problems with the processing can be battled using their methodologies. This project presents a hands-on approach towards the quantitative study of Kazakh language and offers a set of helpful strategies for those who wish to improve upon the existing knowledge. This, combined with the previously mentioned methodologies of studying interlanguage/code-switching, will guide my own creation of methodologies for corpus analysis.

Chapter 3: Methodology

Research methods background

Before any discussion about corpus analysis may take place, it's imperative to establish how one can go about the creation of a corpus. Some of the general guidelines for corpus assembly methodologies can be found in "Research Methods in Linguistics"²⁷, by Litosseliti et al (2010), and most of those are echoed by Norbert (2010), in "An Introduction to Applied Linguistics". The authors establish three basic criteria of a good corpus: machine readability, authenticity, representativeness. Those criteria set a wide set of possibilities for exploring the language quantitatively. The authors provide practical recommendations on how to build a good corpus.

²⁷ Litosseliti et al, 2010

While the book does not extensively cover the ways to make the text readable by machine, it goes into details on making the corpus authentic and representative. Gries (2017), in the book ““Ordinary Meaning and Corpus Linguistics” highlights the steps needed to make a good corpus: making a corpus first requires determining the language variation to be studied: whether it is web, natural spoken language, official papers — this needs to be clarified, otherwise the corpus risks becoming an unsystematized assembly of texts. Litosseliti et al. do note, though, that there are corpora that attempt to encompass as much as possible — English has several such corpora. While this could make for an interesting project, it is beyond the scope of this work; as such, I will focus on one field, which is going to be online medium, or Internet. One of its advantages is its diversity —the Internet has a wide variety of texts, and virtually any kind of corpus can be assembled through it. However, when doing so, it’s important to keep representativeness in mind — the texts must be balanced. In other words, there should be equal amounts of all the types of text. In the case of Kazakh language, one could argue that texts using Cyrillic and Latinized alphabet should be included in equal measures, among other things. This is justifiable, as corpus has to be representative of a particular language or language variety, and excluding an alphabet from a corpus may violate that principle. However, given the current uncertainties with the Latinized alphabet, it is not feasible to include that alphabet just yet; instead, this task shall be relegated to future research.

I chose the news articles on the Internet as the research medium because it is one of the most convenient and diverse resources available. However, working with the Web requires its own special set of assembly methodologies that go beyond what was outlined before. The ways of creating a corpus based on the Web are

described in the book “The Web as Corpus: Theory and Practice” by Gatto (2014). As the Internet developed, it created new mediums for interaction, which brought about new language variations. Therefore, it is necessary to start considering online interactions as a source of corpus which can be analyzed. The author covers the early arrival of the web as a corpus tool, and possible issues arising from its use. They address the problems of using the web as a corpus, including its authenticity, representativeness, size, composition, and copyright. The authors also list some of the tools and the engines available to systematize the web and compile a list of possible tools that would help make the web corpus more systematic in nature. The authors argue its size is its advantage — because it’s constantly growing, it presents a unique opportunity to study an ever-increasing authentic body. The issue of multilingualism is also addressed in the book via the case of Web 2.0 Wikipedia, which is, essentially, a perfect example of a multilingual corpus that is uniform, large and fairly representative. Overall, the book explores the potential of the web as a linguistic medium for quantitative analysis. An issue the book does not address, however, is the issue of a language that has more than 1 writing system — which is the case of the Kazakh language, especially online. Given the recent push for Romanization of the Kazakh language, the Kazakh Internet saw a divide, with some of the materials written in Latinized alphabet and its variations, while others maintained Cyrillic variation. This raises the question — how should those variations be treated as a single language, or 2 different language variations?

Corpus analysis software

At the moment, there are many corpus analysis programs available. However, they are often very specialized in terms of the data they can work with. It’s often the case that software requires some special processing of raw texts to make sense of

them, but the conventions for each program are different, which makes it tougher to use the same annotated corpora with a different program if the need arises. Some corpora use their own tools, such as British National Corpus, or the corpora of Kazakh language. Those built-in tools are often difficult or impossible to access, so the researcher only has access to the output data, but not the source ones. In addition, it is often the case they can't be used with a different corpus, because it was not made for it. There are a lot more versatile programs²⁸ available nowadays, yet they all fall to the same issue of demanding some unity of data. As discussed before, it would be more challenging for Kazakh language, given the variation in writing system, although this can be overcome with proper annotation, as mentioned before.

This highlights the need for a unified system of annotation. However, as noted before, Kazakh language has certain structures that could be unique to it, or only occur within the language family/branch of Kazakh language, which is Turkic language family. Therefore, the new system of annotation could be implemented in a new software that would be ready to deal with the peculiarities of Kazakh language, both in terms of its grammatical and morphological structure.

Annotating a corpus by hand is a very time-consuming task which is not feasible. Therefore, there needs to be a software that would be able to recognize the words and annotate them according to morphological and syntactic features. For this task, it's going to be useful to turn to artificial language processing tools that are commonly included in almost every single development environment — lexers/tokenizers, which are conventionally part of compilers. The book “Compilers, Principles, Techniques, and Tools” by Aho, Alfred V., Ravi Sethi, and Jeffrey D.

²⁸ <https://corpus-analysis.com/>

Ullman (1986) is one of the most well-known books that covers the basic principles and practices of creating compilers. The authors define a lexer/tokenizer as a program that breaks down the input stream into units, and then assigns a category to them based on a set of rules. There can be different ways to break down an input stream (text, in our case), and assign categories to the individual units, but one of the most widespread methods of doing it is via pattern matching. Pattern matching, as the authors say, is when the current token (in our case, word) is matched against the patterns of characters, which are generally written as regular expressions, it gets assigned that category. While this approach works very well for highly formal languages, such as programming languages, languages with less formal rules and more flexibility, which natural languages are, can be more problematic to process. Still, the main idea of using preset patterns to define what some pieces of text are is going to be useful. Another point worth considering is that the analysis will have to be done in 2 parts: for individual words on syntactic level, and for groups of symbols on morphological level, because the morphology of the word has to be kept in mind when parsing; otherwise, we may get inaccurate parsing results. A similar approach, albeit more morphemic in nature, was employed by Munday et al. in the "Corpus-Based Translation Studies: Research and Applications", which highlighted how the input stream can be broken down into morphemes for accurate translation, and then merged back into one word. This kind of approach runs slightly contrary to what Aho et al. did with the artificial languages, but it's important to keep in mind that natural languages are a lot more flexible, as the authors say, and the principles that are used for the artificial language might not work in the same way for the natural languages. In particular, one of the big differences is likely to be expressed in the methodology of reading the words due to the properties of Kazakh as an

agglutinative language – while artificial languages can be read as a human would (that is, left to right), the words from natural agglutinative language, while still being “read” from left to right, will have to be processed from right to left, symbol by symbol, via a string, which is essentially a char array. Since the affixes in Kazakh language are conventionally attached to the end of the word, and given there is no way to certainly tell where the root ends and affixes begin, it is more expedient to start reading from the end. Each character that has been read will be added to a token, which is a smaller array, and once that smaller array corresponds to a particular suffix (plural -lar, or past tense -di, for example), that suffix will be considered parsed, and the information about it ready to be displayed. The parser will proceed until a token grows to a size of 5, which is when the rest of the string will be considered a root. The number 5 is taken because according to the Apertum-kaz lexicon and Muhamedowa, there are no individual suffixes in Kazakh language that exceed 5 symbols. Therefore, at token length 5 the remaining characters can be considered a root.

Corpus analysis methodology

Before further discussion may take place, some ground rules and principles of corpus analysis need to be established. Corpus is understood as a collection of natural texts representative of a language or a language variety that can be processed by a machine²⁹. A machine, in this case, a computer, cannot understand the text the same way humans do in terms of semantics. A text is not much more than a collection of characters to the computer, there is no “meaning”. It is up to humans to make it readable and meaningful for the machines. In part, we could

²⁹ McEnergy, 2012

borrow some ideas from the artificial language processing tools -- compilers³⁰. While there are definitely differences between natural and formal, artificial languages, such as programming languages, one of the biggest ones being ambiguities, some principles would still apply. One of the principles that is generally considered a good practice for natural and artificial languages alike would be using a token system and reading the text character-by-character, instead of trying to read the whole word as a string of characters.

There are several approaches to text processing for part-of-speech tagging via morphological analysis and syntactic parsing using computers. One of the widely used ways is rule-based analysis³¹, which is one of the ways to represent a language through a context-free grammar, which shall be discussed later. Rule-based analysis, as follows from the name, uses an extensive set of rules to analyze the text. The “rules” in this context may refer to a variety of skeleton structure of a language. Say, if we know the sentence structure in Kazakh language is SOV³², then the rule for the parsing would be “For any sentence we encounter, the subject will be before the object, and the object shall come before the verb”. Of course, this is not how one would actually write that rule in their code, but this is the general idea behind it. So, when the machine encounters the sentence “*Men qalamdy syndyrdym*” (“I broke a pen”), it will know according to the rule, the first sequence of letters before the space is the subject (and hence, a noun or a pronoun), the second sequence is the object (likely a noun or a pronoun as well), and the final sequence is the verb. Like this, through the set of rules, the machine can make sense of the text.

³⁰ Aho, 1986

³¹ Kennedy, 2014

³² Muhamedowa, 2016

Text analysis should start with a parsing stage³³. At this stage, the text will be read and processed by the machine according to the rules set by the researcher. Depending on the goals of the project, we may parse the text symbol by symbol, word by word, or sentence by sentence even. Parsing allows the machine to organize the text for more convenient processing, as was described above. After parsing, the text should be annotated -- additional linguistic information that will provide the machine and the researcher with a set of identifiers within the text, making the work easier. There are different types of the annotation, depending on the needs of the researcher.

One of the approaches to natural language processing is using neural networks. The idea of using machine learning and neural networks to perform part-of-speech tagging, among other tasks, is not foreign to the field of corpus linguistics. The approach has already seen its use in some projects, such as part of speech annotation of social media texts³⁴, done by Meftah et al (2020). However, the use of this approach is mostly done in the domain of the languages that have already been studied extensively. This is a rather logical development, as it stems from the fact that a neural network has to be trained before it can be used; and in order to train the network for part-of-speech tagging, the researcher has to have a corpus that has been already tagged. As mentioned before, while there are corpora for the Kazakh language, not all of them offer the researchers a possibility to download the corpus; it is often only accessible via the search tools on the website of the corresponding corpus. Therefore, neural networks have not seen application in Kazakh language processing thus far.

³³ McEnery, 2012

³⁴ Meftah et al, 2020.

Corpus analysis in practice

One of the methods of corpus analysis is pattern recognition; these patterns are a lot wider than the patterns described in the section on software. Here, patterns refer to the patterns of word usage in text. “Applications of Pattern-driven Methods in Corpus Linguistics”, edited by Joanna Kopaczyk and Jukka Tyrkko, describes one of the ways, which is, likely, the most appropriate for this type of research. Similar approaches are described in other works. The authors review corpus methodologies and analysis techniques, and draw an important distinction about corpus-based and corpus-driven studies. Overall, the studies utilizing corpus-based approach use it as a tool to deny or confirm their assumptions; in a sense, this is a traditional way to use the data when the theory has already started developing. On the other hand, a corpus-driven approach makes use of the corpus as a starting point, approaching it with as few preliminary theories and assumptions as possible, and working on creating theories based on the analysis results. Such an approach would be beneficial for the study of Kazakh language, given it enjoyed little to no attention in terms of quantitative studies to this point. Having made the difference clear, the authors advance to present some case studies that illustrate possible pitfalls of corpus-driven approach, but also showcase its advantages.

While patterns can provide a great way to systematize the language, those patterns need to be first discovered. This is where the morphological parsing comes into play. There are several ways and practices to analyze natural languages: simple morphological parsing, word vectors, and others. For this research project, the focus is on the methodologies of natural language processing using morphological parsers. Some of the practices for the study could be adapted from artificial language processing. As mentioned before, one of the most comprehensive guides on the

topic is a book by Aho et al. "Compilers, Principles, Techniques, and Tools". The basic technique offered by the authors is using finite-state machines, that come from the regular expressions, in order to break down the text into smaller pieces that are used in the later stage in order to construct a token. That token later can be processed more in-depth. However, Kazakh as an agglutinative language demands a different type of approach, as was noted by Bekbulatov et al.; given that, it appears that a sensible approach would be to still break down natural text into segments according to punctuation, but then parse the segments from end to start of the word. That way, morphological constructions such as plurals, cases etc can be defined via feeding characters into tokens, and once a token exceeds a certain length, or the word ends, but it doesn't match any of the defined morphological constructions, that could be called a root of the word. The root can then be analyzed further according to various needs, but that goes outside the scope of the current project. On average, according to Apertium lexicon files and Muhamedowa, a typical suffix does not exceed 5 characters, so once the token stream exceeds that, it can be called a root, saving some time. Granted, this approach works well in most cases, but it would still be unable to deal with the "qap-qara". For this case, a separate morphologizer instance can be used, which reads the words from the start to check for this specific type of prefix and define it.

Corpus methodologies can be useful across multiple layers of the language, not just one. In particular, semantics, that studies word meaning and usage patterns, can benefit from the corpus linguistics approach. The book "Corpus Methods for Semantics: Quantitative studies in polysemy and synonymy" by Glynn and Robertson (2014) explores the ways in which the meaning of words can be studied using the corpus approach. The book compiles several major works on semantics

and corpus linguistics. The authors of the studies survey a wide variety of languages, not just English, explaining how semantics can benefit from corpus studies. In this sense, the book is beneficial to this research because it doesn't limit itself to purely English. While there are no languages from Turkic language family, the same methods could be applied to semantic analysis, only with some minor adjustments. In addition, some studies even go in-depth on how one should approach mixed languages: it is not an uncommon phenomenon to encounter 2-3 languages within the same sentence, and it's necessary to create a way of dealing with those.

There is a body of scholarly literature showcasing the ways in which corpus linguistics can be utilized for the benefit of the society. In particular, some of the widest prospects are open in the judicial expertise area. The article "Advancing Law and Corpus Linguistics: Importing Principles and Practices from Survey and Content Analysis Methodologies to Improve Corpus Design and Analysis" (Phillips and Egbert, 2017) sets out to explore the ways in which corpus linguistics as a judicial methodology can be improved and used in legal cases. The main argument is that the introduction of social science methodologies will allow for more rigorous case examinations and increased transparency. The main value of the article lies in 3 points: methodological application, content analysis and coding, and finally, the methodologies themselves. The authors first showcase the shortcomings of the current judicial system, and offer the ways in which introduction of other methodologies could help improve it. While the judicial systems of Kazakhstan and the United States are rather different, the local system could use the same benefits, without a doubt.

Next, the authors study the appropriate ways to analyze and encode the content, demonstrating how it can be used in the judicial system using corpus.

Finally, they discuss 4 coding methodologies and how they are used. They highlight minimalist approach, dictionary-driven approach, grounded theory and register selectivity, discussing the ways in which those methods are different, and how they can be used together. Some of the methods focus on a particular word form, while other methods analyze corpora as a whole and try to uncover patterns, which goes in line with previously mentioned book on pattern-driven research.

The actual body of judicial cases presented so far is not particularly impressive, though, but the specific cases highlighted illustrate that corpus linguistics can benefit society. One of the best examples is an article "Ordinary Meaning and Corpus Linguistics" by Gries and Slocum (2014), talked about the ways in which semantics of common words can be redefined as time goes by, and what implications that has for linguistic expertise. The number of cases so far is not very sizable, but nonetheless, the article provides a glance into various uses of corpus linguistics.

Empirical considerations

One of the first issues that needs addressing is the issue of derivational morphology. As mentioned before, at this point the neural network will not consider the derivational suffixes as part of the morphology. This is a rather salient problem, however, at this stage, it does not appear to be particularly relevant, since the goal of this stage is to break down the words to their parts of speech. The etymology of a given word, like "satuwi" (seller/cashier), is not relevant as of yet. What matters is the fact that in a particular sentence, the words like "oqywi" (student) are a noun, and the origin from the verb "oqu" (to learn) does not add anything to the corpus. Granted, it does become relevant later in the project, once it is used for further breakdown or even text synthesis. However, at the current stage, the derivational morphology will

be treated as part of the word. The flexible structure of the corpus once it is completed will allow the words with derivational suffixes that are present in the database to be analyzed further into the derivational morphemes and the roots. Alternatively, the derivational morphemes may not be considered at all, and the words like the ones mentioned above will be treated as just the roots without morphemes. While it will certainly affect the data for academic purposes, it will not have a drastic impact on the development of the corpus for non-scholarly pursuits, like speech synthesis and recognition.

Chapter 4: Practical approach

Practical challenges

One of the very first challenges I encountered during my project was storing the texts and preparing them for the subsequent analysis. There are a number of approaches one could use to digitize and store text. One of the more obvious ways to store text is by storing it as regular plain text. This approach requires the least amount of processing, as all it requires to do is to copy and paste the text into a .txt file.

However, one of the more significant drawbacks is the encoding problem. Various machines can have various encodings installed on them, and that means that not all machines will be able to interpret the symbols and reproduce them, if needed, in a form that would be readable by humans. Moreover, if there are several files with different unmarked encodings, that would make parsing a lot more challenging, as it would have to adapt to every individual set of characters, which would unnecessarily inflate the code.

The second storage method - and the one I ultimately opted for - was the number system. Granted, the encoding problem exists in this domain as well, but it is solved a lot earlier in the project — in the second stage, text conversion. This system is not my own creation — systems like Unicode, Windows-125x, US-ASCII have existed for a while, and they cover a wide array of characters across different writing systems. However, using a standardized system like ASCII poses its own challenges. In the early stages of the research, when developing the tools to read the text, it was often the case that the Kazakh Cyrillic symbols would appear to be broken down into two separate symbols that bore no resemblance to the original characters anymore. This issue would have significantly impeded the text processing, as it would be entirely possible for the same program to produce different outputs depending on the encoding in use, rendering the efforts virtually unusable. Thus, I took it upon myself to come up with an encoding system that would work well, producing consistent, readable, and clear results.

Cyrillic and Latinized alphabets

One of the most important questions in this study concerns the question of shifting writing systems in Kazakhstan. For quite some time in modern history, Cyrillic alphabet was the dominant writing system in the Kazakh language. However, in the more recent times, there have been suggestions and even official meetings regarding transition to Latinized script. No doubt, there are pros, cons, and implications of such a change in terms of political and social life, however, those are not relevant to the project.

The main challenge that arises from this change is the multitude of options for Kazakh Latinized alphabet. At the moment, it appears that there is no single

alphabet that has been widely accepted; several orthographies can be currently seen both online and outside. One of the options that seems to gain popularity is Qazaq Grammar³⁵, with some options presented by Inform.kz³⁶ and other less popular options.

This multitude of options poses a serious challenge to the research. A single letter can be written differently in all those writing systems, with the word “шай” (tea) being a very prominent example.

Qazaq Grammar	cay
Inform.kz	shai
Inform.kz	shay

I would like to direct particular attention towards the Qazaq Grammar project. The Facebook page was established on June 10, 2016, and they published their version of the alphabet on July 31, 2020. Their orthography appears to be widely used across Kazakhstani social media, however, it is still not official. While it is rather convenient, it has its own problematic areas related to specific letters. On their Facebook page, they give the following spelling rules:

“У — W/Uw/lw/İw: у — uw, may — taw, елу — eliw, мамы — tatıw

** У кірме сөздерде → U: Самурн → Saturn, Уран → Uran, Нептун →*

Neptun

** И — Іу/İу: и — iy, му — mıу, қи — qıу, би — bıу, ки — kıу*

³⁵ QazaqGrammar on Facebook. July 31, 2020.

³⁶ Informburo.kz, 2019

* И кірме сөздерде → *li: испан → ispan, чип → chip, фин → fin*

* Я — *Ya: ұя — uya, сия — siya*

* Я кірме сөздерде → *Ä/Ja: заряд → zarät, Ява → Jawa*

* Ю — *Yiw/Yiw: қюю — quuiw, кюю — küyiw*

* Ю кірме сөздерде → *U/Ü/Yu/Ju: Юта → Uta, дзюдо → judo, блюз → blüz, Юпитер → Jupiter*

* Э → *E: электр → elektr, эстон → eston*

* Щ → *CC: ащы → acci, тұщы → tucsi, борщ → borc*

* Ч → *CH: чек → chek, чех → cheh, чарт → chart*

* Ц → *S/TS: цент → sent, цех → seh, пицца → pitsa, полиция → politsiya”*

The problem here is the multiplicity of options for some letters, and lack of distinction between others. Say, letter Ю has 6 spelling options, with 2 of those containing 3 symbols; letter У, similarly, has 5 spelling options, with 3 being 2-letter ones. In addition, there is no differentiation between Э and Е, Ц and С, CC and Щ, to name a few. This all makes working with text very challenging, because the text will have to be parsed not 1, but 2-3 symbols at a time depending on the current input.

The final point regarding the Latinized orthography is that at the current stage, the text parsing programs will need to have a separate script identifier module, that will be able to tell the program which alphabet the text is written in, so that the parsing will use the correct one. This can be done by searching through text and finding some letter combinations that correspond to a particular alphabet. Such an approach will make the program consume more time in the long run.

Generally, there is no consensus on what shall be used for Kazakh language at the moment. In addition, many major news websites (Tengrinews, Informburo,

Zakon.kz, to name a few) do not have any versions with Latinized orthography.

Therefore, it has been decided that the research project will only focus on the Cyrillic alphabet, as introducing the Latinized alphabet into the research will complicate the processing of text significantly without adding a significant benefit. However, due to the flexible nature of the code, when the need to add Latinized alphabet arises, it can be done without complications.

Some practical considerations regarding the coding stage

One of the first choices I had to make was whether I wanted to hardcode as much as possible or to leave more room in the code for future changes. In this context, hardcoding refers to initializing variables in the program's body with a particular value and then using it throughout the program. Say, if we know there are 41 letters in the alphabet, we could initialize a variable `alphabet_number` to 41, and use it in the code throughout the program. Undoubtedly, this is a very convenient solution, but only short-term: if something needs to be changed in the program, the code would have to be redone. This is especially troublesome with the hardcoded values not tied to a particular variable.

For example, let us take a fairly simple “while” loop that would compare the character currently being analyzed with the character from the predefined alphabet. The loop would look like this:

```
while (index<41){  
    char ch = read_char_from_file(file_name);
```

```
compare(ch, alphabet[index]);  
  
if (ch == alphabet[index]){  
    print_in_file (index);  
}  
  
i = i+1;  
}
```

The pseudocode above shows a general way of assigning an index to a particular character that's been read from the file; this is not the accurate representation of the actual running code. The loop is supposed to go until there are no more letters left to compare to (that is when \mathcal{A} has been checked) and stop after that. The array "alphabet" would also contain only 41 characters, so going above 40 would be pointless and may produce an error when trying to access the unit that is not actually defined — trying to compare `ch` to `alphabet[43]` may yield unexpected results, as that value is undefined.

Thus, arises the problem. The code above uses 41 as one of the key values — it's the number of characters in the alphabet, it's the value the loop iterations should not exceed, etc. Whenever the code has to be changed (say, add more characters, or remove some temporarily), those numbers will have to be changed in multiple places as well. Refactoring (looking for a particular value/variable through the code and changing it) may solve the problem in some cases, but not necessarily in all of them, creating a lot of hassle and generally poor practice of coding.

Therefore, I chose to keep my code as flexible as possible through global variables.

Global variables are defined in the header area, before any function, using the

#define tag. This allowed me to tie several variables to a particular key number, and that number can be changed effortlessly at any given moment.

Another part of hardcoding came from the alphabet itself. The alphabet is a relatively rigid notion that is not very easy to change in a short period of time. Therefore, it would make sense to just put it into the program manually, code all the values in, and allow the program to run like that. However, this leads to another potentially troublesome hardcode problem.

The Kazakh alphabet has 43 letters (including those that are only encountered in loanwords from the Russian language); multiplying that by 2, because there are small and capital letters, gives us 86. In addition to that, there are also punctuation marks - a standard keyboard allows us to type in 39 characters, plus 10 digits, for 49 in total. Adding that up yields 135 characters that would need to be put into the program. It does not seem to be a lot for a one-time operation. However, during the development process, I have been changing the character set more than once (the reasons for those changes will be described below), and doing so each time I test an approach does not appear feasible.

Therefore, I chose to go a more flexible way yet again and created a text file that contains all the characters. The program would read the text file, load the characters from it into the array, and then proceed with the rest. The text file can be edited much easier than the program code; it also allows greater flexibility in the future, if something in the alphabet changes. To a certain extent, the software could even accommodate a different alphabet.

Wildcard approach to morphology

Upon the examination of Kazakh morphology, it becomes clear that generally, Kazakh morphology is known to follow certain patterns³⁷. One of the clearest examples of that is the case suffixes of the Kazakh language. The genitive case of Kazakh language has the following orthographic allomorphs: -ның, -нің, -дың, -дің, -тың, -тің. The pattern is “-н/д/т+ы/і+ң”. Therefore, there is a possibility of representing the set of suffixes with a set of symbols, one symbol per specific set of letters. Let us divide the pattern into three distinct groups: 2 variables and 1 constant. 2 variable groups are “н/д/т” and “ы/і”, while the constant group is “ң”. The variable groups change depending on the word the suffix is attached to, following the vowel harmony, while the constant group is invariant. The first group can be also noticed across other cases, but not in all of them – for example, the locative case does not have “н” in the first group. Following that, the first group, “н/д/т” can be declared CC1 (consonant variant 1), the second group – CV1 (vowel variant 1); the group without “н” can be named CC2, the “а/е” group is CV2, and last, “м/б/п” sequence can be named CC3. Thus, the entire case may be represented as “%CC1+%CV1+ң”. This approach is convenient, because instead of storing the entire variety of suffixes, it is enough to store the rules according to which those suffixes are formed, articulating the rules via small finite-state machines.

Pronominal hardcoding

Another consideration related to the morphology and coding certain language elements in the program is the pronoun class. Kazakh language has a limited

³⁷ Muhamedowa, 2016

number of personal pronouns, seven, to be precise. In addition, there is a set of suffixes that looks identical to the personal pronouns. Therefore, one of the decisions I have made in relation to text post-processing is to encode the personal pronouns separately first, and then generate the list of possible combinations of those pronouns with the suffixes (plural, case endings etc) automatically, and then verify the data manually and fix if necessary. This approach helps eliminate the possible confusion in the classification algorithm. The confusion may arise from the fact that, for example, “мен” can be a personal pronoun (I/me) or a case ending (-with). Since cases are generally a property of either pronouns or nouns, the presence of case ending contributes to the likelihood of a particular word being a noun/pronoun. So, when the algorithm starts searching for the potential words that fit the definition, it will detect “мен” and attempt to classify it; the behavior is unpredictable in this scenario, which may lead to inaccuracies. Therefore, to avoid this scenario, I have coded the pronouns and their possible morphological forms into the programs, thus they may be intercepted and tagged before being fed into the neural network. So, as the program is attempting to determine what a particular word is, for example, “онымен” (“with him/her”), it will first determine that the word end with “-и” and “-мен”; since there are still letters before the “-мен” suffix, the program will not automatically assign it as a pronoun, but instead will treat it as a morphological marker of case. When the program finds the root “мен-”, it detects that there is nothing preceding this part, which allows it to conclude, through the comparison with the list of pronouns, that this is a pronoun. Alternatively, it is possible to generate the list of most possible pronoun+suffix combinations via the use of regular expressions, and then picking those that fit the vowel harmony, and use this dictionary to completely avoid parsing of pronouns through the morphological features, and instead use a pre-coded

dictionary, which has all the possible variants of pronouns annotated. Granted, this approach is more-time consuming during the training, as for each word, the network will have to reference the pronoun dictionary. The neural network shall still search for the words with case endings in order to try and tag them with a part of speech based on their morphological features, however, it will not notice the words that have already been tagged.

Advertising and suggested content

Another issue that has arisen is the issue of advertisement. Given the online nature of the texts that are involved in this project, it becomes a rather salient problem that has to be addressed. Advertisement may come in various forms — it may be a JavaScript built into the page, or a `<div>` container. Generally, it is quite rare to see an advertisement in the middle of the news article, however, it definitely does take place, albeit not on all websites. This poses a problem for the research, because those texts do not fit the definition of “news article”, and might not be in the target language (Kazakh). Therefore, there needs to be a way to remove those.

This task proved to be more challenging than initially expected. The main problem with the “ad remover” is the individual nature of the ad placement and style across different websites. There is no one specific format of advertisement that makes it easy to detect and remove those. While it is definitely possible to remove all `<script>` tags from the page and be reasonably sure there are no more JavaScript advertisements on the page, removing all `<div>` tags will erase the entire article. Thus, this approach needs more careful consideration.

Given that, I have decided to gather the articles from a limited number of online newspapers, and tailor the scraper for the possibilities of each of those newspapers. In addition, the chosen websites generally display little to no advertisement in the body of the news, making them good candidates for the research.

Another possible problem is the “Suggested news” section. Those may appear at any point in the HTML as well, sometimes in the middle of the article, as shown in Appendix A. The code, obtained via Show Element, is demonstrated in Appendix B.

Those pieces are generally the titles of the articles and potentially some images. While they do constitute a “valid” piece of a news article, their location disrupts the overall text. Hence, those pieces have to be removed as well. The ways to do that may also vary across the different websites, which, therefore, reinforces the previously made decision to adhere to a specific set of websites. Filters can be set up within the scraper, or the post-processing, to ensure those code pieces are removed from the text. The example above can be improved by attempting to remove the `<div>` portion of the code. Granted, there is a degree of risk to this approach, as there may be more tags with similar names and classes, but with different contents. Still, as noted before, an individual approach to every single news source must be developed. Below is an explanation of approaches for certain websites

- Tengrinews does not require specific filtering that needs to be added in addition to standard scraping techniques used in the research

- Inform.kz contains the “Related news” within a `<div class = “frame_news_article”>`; while counterintuitive, this tag can be removed from the HTML code.
- Azattyq contains the “Related news” within a `<div class = “media-block also-read”>`; while counterintuitive, this tag can be removed from the HTML code. This is a highly specific tag class, which ensures that it is unlikely that something important gets removed from the body

Classification according to the word order

One of the premises of the project is the fixed word order of the Kazakh language. As it has been outlined above, the Kazakh language does possess a fixed word order — SOV, with subject being first, followed by an object, and concluded by a verb. This is generally true of the Kazakh language — for example, the sentence “Ол үй салды” (*Ol ui saldy/He built [a] house*) is a perfect representation of that word order. Granted, informal speech may not adhere to this rule as strictly, however, given that the texts gathered are from the online newspapers, the general expectation of adhering to the standard grammar appears to be justified.

Even if that is true, there are still some issues arising when working with the Kazakh language strictly in terms of word order in a sentence. While some of the issues are not critical for parsing, others may be more challenging, demanding a different approach.

Non-fixed order of adverbs and adjectives

One of the initial approaches I was attempting to develop for determining whether a particular word is a noun, verb, or another grammatical category, was to

correlate the position of the word in the sentence with the grammatical category. While it could definitely be more convenient to just use a dictionary approach, comparing the words with the body of known words, the dictionary has to exist in the first place, and the options are limited at the moment. To give an example, if the word “үкімет” (government) is generally placed first, then it is likely to be a noun; on the contrary, if the word “жету”(to reach) and other its forms are generally located last, then it is highly likely to be a verb. For the strict SOV sentences this kind of distribution would be valid. However, there are some problems that I will be discussing below.

The first issue is that the adverbial modifiers, among other elements, may be placed in multiple places within a sentence — the SOV rule does not dictate that those should be placed in a certain spot. Thus, the sentences,

1. Кешке қарай дәрігер жұмыстан босатты
2. Дәрігер кешке қарай жұмыстан босатты
3. Дәрігер жұмыстан кешке қарай босатты

meaning “The doctor left the work in the evening”, are all equally valid grammatically, however, if we take a look at the indices of words, we shall see that it disrupts the system outlined above. The index of the word “дәрігер” in the first sentence is (3), but (1) in the other two sentences. This difference makes it difficult for a machine to say with a degree of certainty that the word is, in fact, a noun.

One of the better approaches to this problem is not to use the direct numerical indexation (1, 2, 3, etc), but rather resort to a percentile system, with a particular emphasis on verbs. The percentiles may be adjusted, however, the system employing 25-50-75 distribution may be the most effective for an average length of a sentence in the Kazakh language. However, for longer sentences, especially the

complex ones, they are unfit, as that percentile may include nouns that can be mistaken for verbs. This is especially salient in the early stages, when there is no preliminary corpus yet — as the vocabulary expands, the program will be able to reference it and draw conclusions based on the existing data.

One other way to make the percentiles more precise is to dynamically adjust them based on the number of words within a sentence which is ultimately a hybrid between the direct index and percentage approach. At a certain threshold, past 8 words, the system should adjust to a 20-40-60-80 system, with each percentile containing about 2 more words than the previous one. While this approach may still be faulty, it is more likely to include the verbs that have 2 words in them, like “келіп кетті” — the verbs with auxiliary verbs.

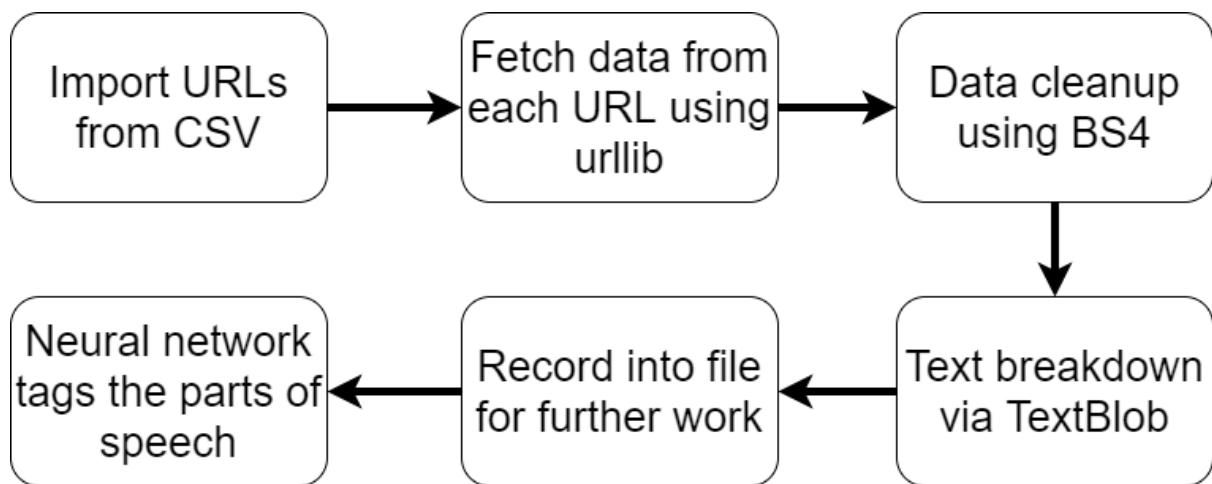
Kazakh as a pro-drop language

As it is the case with many SOV languages, Kazakh is a pro-drop language — the subject is likely to be dropped, as the speakers can generally infer the subject of the sentence, based on the context. However, for the machine to do the same, it would have to keep track of the previous sentences, and it would also have to have correctly identified the subject among the nouns/pronouns of that sentence, which might not necessarily be the case. However, the extent to which this problem is important might not be completely significant. Even though there is no subject within the sentence “Кітап оқыдық”([We] were reading a book), the personal suffix at the end of the verb indicates the missing element of the sentence. Moreover, we have to consider what information would be lost for the language processing machine. If the percentile approach is followed, the 50-percentile would be the word “кітап” (book), which, according to the model, is not likely to be a verb (which is true), and since it is at the beginning, it is likely to be a noun or a pronoun (also true). In general, what is

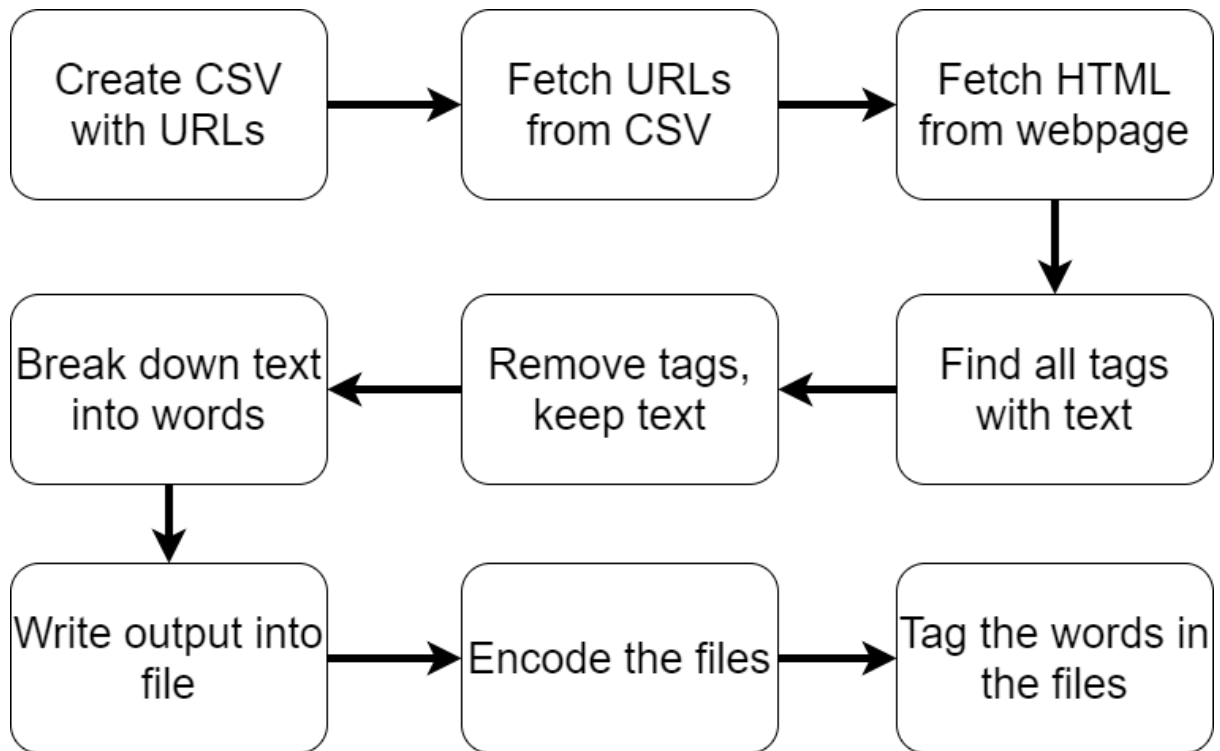
important for the percentile model is not the exact syntactic role of a particular word in a given sentence, but rather the part of speech that word belongs to. Therefore, the fact that Kazakh is a pro-drop language does not necessarily impede the process.

The development process

The following diagram demonstrates the general data flow between the processes described in the following section.



This diagram shows a more detail-oriented version of the previous diagram, demonstrating the processes of “data cleanup” etc.



Converter v.0.1.0-C — Alpha stage

When writing the stage 1 converter, I have decided to first test my ideas on a more limited character set. Thus, my very first converter was only able to process capital letters of the Kazakh Cyrillic alphabet. The data for the array would come from a text file that I have created in advance.

For this iteration, I decided to use an array that would contain only the capital letters, and the index would be used to assign the code. The example is presented below.

index	0	1	2	...	39	40
letter	А	Ә	Б	...	Ю	Я

The scheme above is a rather simplified representation of the way I intended to store the data in the machine. Essentially, this array is a simple data structure that ties keys (letters) to a numeric value. From this point onwards, for a convenience sake, I shall refer to this system as Pseudo ASCII, or PASCII.

The rationale behind this implementation is that conventionally, the characters are not stored as characters per se, despite the “char” type; instead, they are generally stored as numbers. The numbers that signify the characters come from a unified system known as ASCII table. The ASCII table is a table that contains many characters commonly used on the machines - from letters and numbers to punctuation and the new line character. Initially, I tried to approach the current task with that table in mind - by reading the characters and outputting their numeric values. However, I have discovered that this attempt fails.

The first reason for this was that the symbols are not stored consistently across the machines. Through trial and error I have found out that specifically Kazakh characters are stored as 2 bytes instead of 1: when trying to read symbol by symbol, the programs consistently output the garbage values, but when reading 2 symbols, the output was correct. Another test I ran showed that when I calculated how many array “cells” are used in total, and the program indicated that 2 “cells” are used per letter. This created certain difficulties in implementation, such as properly reading the input, and displaying it correctly. Thus, after trying to make the array work, I decided to take a different approach.

Converter v.0.2.0-C — Second stage

For the second stage, I have decided to adopt a different strategy, utilizing the data structures that are present in C. I created a structure called “alphabet” and

created 2 members in it — *pascii* integer and *letter* string. A string is essentially a set of characters — given what I discussed previously regarding the storage of characters, this was the only viable option.

```
#define letter_num 42

struct alphabet{

    int pascii[letter_num];

    char letter[letter_num][3];

}

int main(){

    struct alphabet cyrillic;

}
```

Index in arrays	0	1	2	...	40	41
pascii	1	2	3	...	41	42
Letter bit 1						
Letter bit 2						
End character	\0	\0	\0	\0	\0	\0

The code piece above shows the declaration of the structure “alphabet” and

then its initialization in the body of the program (int main). The first line is a preprocessor definition, *letter_num*, which defines how many elements there will be in the arrays within the structure right under it, and *letter_num* can be used anywhere in the code later on. In addition, despite *letter* being declared as a character, the [3] at the end makes it a two-dimensional array of characters, or in other words, an array of strings that are 2 characters long. Notable, strings in C have to end with the “\0” symbol, which is the symbol that shows the string is over, and it also needs its own place. Thus I cannot declare an array of size 2 for a 2-character string, it has to be 3.

This attempt appeared to work a lot better than the previous one. The problem with the character storage worked for now, and the code became a lot more flexible. By including the *letter_num* I was able to expand the arrays as needed, so the inclusion of small letters was easy in this iteration. However, this iteration still lacked the ability to work with the punctuation, and I shall expand on the problem in the next section.

Performance-wise, though, there was still room for improvement. While the program was able to work with the letters only, longer sequences led to progressively longer execution times. The table below shows the times it took for the program to work with 50, 75, 100, 500, 1000, and 3000 characters — I do not foresee texts going above 3000 characters in a newspaper. A similar test will be conducted for future programs. An important note here is that n=0 indicates pure initialization of the program - filling the structure described above with values, without processing any text. The tests were conducted using the worst-case scenario — the very last element of the array, “я”, iterated n amount of times. Calculation was done by running the program 10 times with the same values, then finding the average. The

time (in C) was calculated using the `clock()` function imported from `<time.h>`; the timer was started at the beginning of the code, and finished 1 line before the termination of the program.

```
#include <time.h>
int main(){
    clock_t start = clock();
    //program body
    clock_t end = clock();
    double time = (double)(end-start)
    printf("%lf", time);
    return 0;
}
```

n	0	50	75	100	500	1000	3000	10000
T(C)	11	12	12	13	14	14	17	23

Algorithm execution time, in C (in ms)

However, another issue arose. Like most texts, newspapers contain punctuation and numeric character, with some coming from English (website name, for example). Therefore, those symbols have to be processed as well. The rather obvious solution for the problem would be to have them in a separate *alphabet* structure, called it *non-alphabetic*, and compare the symbols to them as well; alternatively, it is also possible to just add the punctuation and numbers into the existing *alphabet* structure.

This problem was the one that made me finally reconsider the choice of the programming language. As it was mentioned before, the Kazakh symbols in C using UTF-8 are treated as 2 characters, however, punctuation and numbers are still one. Therefore, the question arose: how can a machine read

a character and determine correctly whether it is a punctuation/number or a Kazakh letter?

I have tried several approaches to the problem. The first solution was based on the assumption that punctuation marks are often followed by a space. Given that, it would be possible to read off 2 characters from the file and compare it to “punctuation mark+space” in the array. This would allow me to make minimal changes to the code, only changing the way the punctuation is introduced into the arrays.

I	M	Ы	С	А	Л	Ы	,							
II	ch1	ch2	ch1	ch2	ch1	ch2	ch1	ch2	ch1	ch2	ch1	ch2	ch1	ch2

The table above demonstrates my approach towards reading the text with the assumption “punctuation+space”. Row I in the table represents the letters in the text - in the table above, it is the word “мысалы” (example/for example), followed by a comma with space. The second line shows how the characters are represented in the array. As mentioned before, each letter is represented by 2 characters (ch1 and ch2), and the punctuation is grouped together with space as a single letter. I have colored the cells green to represent that the frame of reading is functioning correctly, and the text will be processed without difficulties.

While this method was convenient, I have realized the problem rather quickly. Upon another examination of the texts, I have noticed that there are, in fact, multiple cases of punctuation marks not being followed by space.

I	“	M	Ы	С	А	Л	Ы						
II	ch1	ch2	ch1	ch2	ch1	ch2	ch1	ch2	ch1	ch2	ch1	ch2	ch1

This table shows what happens when there is a quotation mark at the beginning of the sentence. The first character is read correctly as a quotation, but not identified yet, since the identification only runs when there are 2 characters read off. The second character that will be fetched and counted as the second part of the first symbol (supposedly space) is the first character of M. This is the part when the problem starts occurring. In fact, there will be 2 issues resulting from this.

First, there will be no character that will be recognized and processed correctly. The start of the phrase will see an unexpected character after “, which will lead to that symbol not being recognized - that is, the punctuation will not be processed correctly at all. The second problem is that the frame of reading is shifted by 1 character to the right, not being in the correct position. In other words, instead of having ch1 and ch2 of a particular letter, what the program will get is the ch2 of one letter and ch1 of the next one. Predictably, there will be no entries that will correspond to that sequence, therefore, no symbols will be recognized.

One of the possible ways to solve the problem is to run the characters through 2 separate functions. The first function will only use one character (not parsing the second just yet), and run the comparison only for the punctuation marks and numbers (which are only 1 character in storage). If the first function returns no result for the symbol, then the second symbol will be read off and added to the first one, initializing the second function. The second function will go through the alphabetic

characters and find the match. If no match is found even then, the program could output the symbol that was undefined. In this case, though, the processing time grew slightly. Therefore, the algorithm became even less optimal. At this point it became obvious that the project needs to be done in a higher-level language to be optimal.

Converter v1.0.0-Py — First stage

The language of choice became Python. Python offers a wide selection of tools for text processing and getting the source data from a variety of file formats. In addition, Python is a high-level language, which means memory management and other processes are already automated to a certain degree.

As with C, the first step in the development in Python was to test the ideas carried over from C. As before, I used a file to create an alphabet within my code by reading from it. I used a dictionary data structure present in Python; the dictionary is an unordered collection of items that cannot be changed, and may not have duplicates; the dictionary contains key:value pairs. It was a perfect fit for the project: the alphabet doesn't contain duplicates, and each key (letter) would have exactly 1 value (PASCII code).

Python treated each letter as a separate object, therefore, I did not have to calculate how much memory would a particular letter take, which is a significant improvement over C. Therefore, it was enough to have the program read symbols one by one from file and put them in the dictionary, while PASCII code was assigned according to the order of the letters in the source files. The source file contained capital letters first, and small letters after them at this stage.

Another advantage of using Python came from the fact that finding a particular key in the dictionary and returning its value is realized through the `get(key)`

method (key being the letter) that returns the value corresponding to this key, eliminating the need to code the search algorithm. Note that since the dictionary does not allow duplicates, there will be no case when one key will correspond to 2 values in this program.

The tests on a variety of strings were successful: the program was able to read the letters from the input file and correctly output its number into a corresponding output file.

Converter v1.1.0-Py — Second stage

After the successful completion of the first stage, I have introduced the other elements of a regular text into the program: punctuation marks and numbers. The signs like “;” and “!” were not included in the list, as it is unlikely they will be used in Kazakh language. The approach described for Stage 1 worked for the punctuation as well, without any modifications to the code. Whenever the program encounters a foreign character (the one that is not found in the dictionary), it will omit it, going to the next one. This is particularly useful to remove the URL of the website from within the text (kaz.tengrinews.kz often has Tengrinews.kz in its news).

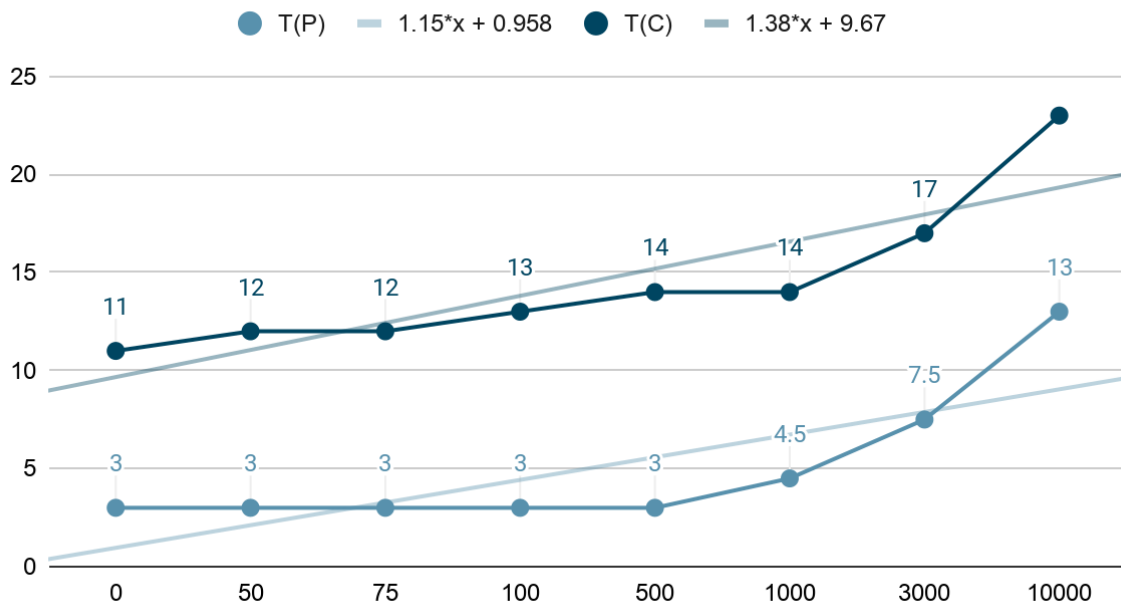
The table below shows the performance tests ran on the same amount of the same symbol as for C. The tests in Python were run using `default_timer` imported from the `timeit` package, as it is the better option for measuring performance³⁸.

n	0	50	75	100	500	1000	3000	10000
T(P)	3	3	3	3	3	4.5	7.5	13
T(C)	11	12	12	13	14	14	17	23

Algorithm execution time, in Python, comparing with C (in ms)

³⁸ <https://docs.python.org/3/library/timeit.html>

Performance



Performance comparison, visualized; algorithm complexity is shown to be identical, being $O(n)$ for both.

The table above shows that Python offers a significant improvement in performance compared to C, with both algorithms being of $O(n)$ complexity, meaning that the time it took the program to process the input was directly proportional to the input. The difference at lower values of n is negligible for Python.

Scraper

The next step is building a web page scraper. A scraper is a program that takes a web document (in my case, HTML) and extracts the needed text from it, along with any other information that may be relevant for the research. Python offers some ways to do that. I have decided to first test the tools with the fixed URL address hard-coded into the program, and if successful, implement the requests to

the database that contains the URLs of the articles I am planning to use in my research.

The first Python package I used was the `urllib`. `Urllib` is a package that collects several modules that can work with URL addresses³⁹. This was required to access the web-page online and fetch its contents. I used the `urllib.request.urlopen(URL)` function to gain access to the webpage. The function returns the encoded webpage content in a document; the content is usually encoded, so I decode it using the `read().decode("utf-8")`, which is a widely used encoding. The end document is the HTML content of the page.

However, not all content that is on the page is needed or even relevant. A lot of the source code, as I found, contains links to social media sharing options, advertisement blocks, various widgets and other content. The relevant text is generally located in tags like `<title>`, `<tag>` (this tag may vary depending on the source website), and `<p>`. Title generally contains the title of the news article, tags contain the information about the topics in the article, and `p` contain the text of the article. All those have to be extracted from the document and processed in the text converter.

Beautiful Soup 4 is a Python package specifically for working with HTML and XML documents. It uses Python's internal HTML parser to work with them; note that BS4 is unable to open the links directly, hence I had to use the `urllib` package.

The difference between a method and a function in Python is that a method is defined within a class, so the methods are called following the `object.method` syntax. On the other hand, functions are independent of the objects, and are instead called on the object like `function(object)`. In the examples above,

³⁹ <https://docs.python.org/3/library/urllib.html>

`urllib.request.urlopen()` is a function, and `read().decode()` is a method of class.

BS4 has a lot of methods and functions that are useful for working with the HTML files. At this stage in my project, I have made use of several of those:

`BeautifulSoup()`, `prettify()` and `find_all()`, among others.

`BeautifulSoup()` function takes the HTML document (received earlier via `urllib.request.urlopen()`) and transforms it into a bs4 object, which is essentially an object of type `soup`, with the methods defined by BS4. A `soup` object is, according to BS4 documentation⁴⁰, a nested data structure, a parse tree, with the enhanced navigation and netter search functionality than a regular HTML page.

`Prettify()` is a method of `soup` object that makes the `soup` object more readable for humans, converting it into a Unicode string with a separate line for each tag⁴¹.

Finally, `find_all()` method looks for all the tags of a certain type in the `soup` (which is just a parse tree), and then returns them. I have used this method to extract the entirety of the news body, which is generally contained in `<p>` tags, into a separate document, which will be then ready for parsing using the converter. The tags `<title>` and `<tag>` will be used later on, when building the actual database for the corpus; at this stage, they are not relevant yet.

However, the output of `find_all()` may be rather cluttered at this stage — tags like `img`, `strong`, `a href` (images, bold font, and hyperlinks) will still be present, and those have to be removed. There are several ways to clean up the text, either at this or the later stages; since `BeautifulSoup` offers this functionality, using it is the better course of action.

⁴⁰ <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

⁴¹ *ibid*

Having collected all the text of the web page via `find_all('p')` and placing it in a separate soup object, which I shall reference as `soup_p`, I have then run another set of `find_all()` within `soup_p()`, this time with a set of other tags, such as the ones mentioned above. This time, the tags will have to be either removed completely with all of their content (like an image URL) or extracted (keeping the text and removing the tag, like bold, italics, or any other formatting option that is irrelevant).

BeautifulSoup provides two methods for doing that. `decompose()` method destroys the tag specified along with its contents — anything within the tag specified will get completely removed from the soup. This method is useful for removing images and advertisements.

Final stage of scraping is creating a storage for the output of the BeautifulSoup. Note that by default, the “soup”, which is what the output of BeautifulSoup4 is, is not specifically a string of text; rather, it is a binary tree, which contains all the tags, as described above. This entails that there has to be a conversion from soup to a piece of text. In addition, the output of the conversion should not be a plain string of text per se, it has to be a more “processed” version that can be searched and analyzed. One of the ways to do so is via another package, TextBlob. TextBlob is another Python package that offers extensive functionality for natural text processing. It has been built on the basis of other existing packages, namely NLTK (Natural Language Toolkit) and “pattern”. TextBlob creates a `textblob` object, which can be later broken down into “sentence”, “words” and “word” (the last two are two distinct objects), and all of those have their own separate methods. However, they will be postponed to a later stage of post-processing. Within the scraper, TextBlob is used to scan through the text and ensure there is only Kazakh text, and no extraneous tags left. However, it is impossible to

directly convert soup into a textblob; this can be achieved by typecasting — forcing the program to treat a type of object as something different. In this case, the typecasting shall look like this:

```
output = textblob(str(soup_p))
```

with `str` being the typecast. What this achieves is it forces the binary tree (soup) to be a string, which is an acceptable type for a textblob container.

Following this, the text is checked for inconsistencies like tags, extraneous text that could have been generated by BS4, and then written to an output file. This stage completes the scraper.

Neural network approach

General information

There are several ways to break down a sentence into words and analyze them. The first way involves the use of an extensive dictionary that will serve as a reference to the program. Such a dictionary will have a list of words with their corresponding parts of speech, and the program will reference the dictionary for each word.

This method has a series of drawbacks. First, the dictionary itself will have to be exhaustive, with as many words as possible included for reference. Online dictionaries for Kazakh language exist, however, they are not well-suited for use in this research, as they don't offer a way to access the data bypassing the search boxes on their websites. In case a word is not found in the dictionary, it will remain unidentified, and therefore not analyzable. There is also a certain degree of ambiguity present in the natural language. Some words will have more than one meaning, thus representing two or more entries belonging to the same or different

parts of speech, so the program will not be able to analyze that correctly. For example, the phrases “Look!” (imperative verb) and “Black” (color adjective) are both “kapa”.

In order to correctly break down sentences for the parser to work with individual words, I have decided to use a neural network. The main idea of the neural network will be to recognize the patterns present in the language in terms of syntax, analyze the input (the sentence that has to be broken down), and output separate words with their classifier (noun, verb, etc).

In a broad sense, this is a corpus-driven approach. At this stage, “corpus” is the collection of the natural text from online sources, which does fit the definition partially. However, at this point (pre-neural network) it is a collection of raw data which is not understandable for the machines. The task of the neural network in this project is to break the text down into the smallest possible units of meaning and assign the meaning to them. The unit of analysis in this stage is the words, without their relation to each other, and the goal of analysis is to narrow down their possible classes as much as possible. The second stage will employ sentences as its unit of analysis as well, but this time, the expected outcome of the analysis will be a more precise definition of the class a particular word belongs to based on its syntactic role within the given sentence.

As the project goes on, though, those neural networks will be collapsed into one single neural network that will be able to work with the texts from start to finish, enabling a seamless addition of new words to the corpus continuously.

The preprocessing is one of the most intensive steps of the process. One of the most optimal approaches is to implement the Recurrent Neural Network working structure. It has been widely discussed that this type of neural network is optimal for

natural language processing because this neural network also takes into account the time of appearance as one of the factors to use for the weights. The input has to have a good representation so that the machine will learn the maximum number of patterns present in the language.

The machine will not be able to automatically know which word is which without the training data set. Therefore, the first set of data will include the parts of speech, their syntactic roles and the morphological markers that are able to be attached to them. At this point, the derivational morphemes are not entered in the database, only the inflectional morphemes. This leads to certain restrictions in the text processing: a word like “satuwi” (seller/cashier) is treated as a single root without any suffixes, despite the fact it carries “-wi” derivational suffix that transforms the verb “satu-” (to sell) into a noun “the person who sells”. For example, this is what the entry for nouns would look like using a table representation.

Part of speech	Syntactic roles	Morphology
Noun	Subject	Case
	Object	Number
		Possessive

Note that there is no intrinsic connection between “Syntactic roles” and “Morphology” other than the fact they both belong to the same category of “noun features” – that is, it is not imperative that an object carries a plural marker. In a sense, it is useful to think about this table as two tables merged into one.

According to Muhamedowa (2016), there is another marker for nouns – definitiveness, however, that marker is generally a separate word like “bir” (one) or “sol” (that), to give a brief example. Therefore, this is not a morphological marker of definitiveness, which excludes it from the list of morphological features to be included in the table.

Recurrent neural networks, as mentioned before, take into account the time of appearance, which makes a difference for the recognition. In terms of the natural language processing framework of this project, “the time of appearance” is most closely aligned with the position of a word in the sentence. A great example of that is the word “мен” – in the Kazakh language, it can be either a first-person singular pronoun or a conjunction (and). The position of the word in a sentence is one of the defining factors in distinguishing between the two – if the word occurs at the beginning of the sentence, it is highly unlikely that it is going to be a conjunction, therefore “мен” in this case will be classified as the first-person singular pronoun. This is confirmed by a search through a third-party corpus, Apertium in this case — there was no case of “мен” classified as a conjunction at the beginning of a sentence⁴². In turn, if the word occurs mid-sentence between two nouns or two objects, there is a higher likelihood of the word being conjunction, for example, “шай мен нан”. For this to work, the neural network will have to loop around on the preceding words to adjust the weight for the particular word.

In terms of analysis, the Kazakh language is very convenient, as it has a fixed word order (SOV), therefore, most transitive sentences will follow that structure. Granted, there will also be other elements, such as numerals, proper nouns,

⁴² Apertium

temporal and spatial indicators, etc. The task of the neural network will also be to determine what those are and classify them accordingly.

The list of features for the neural network will contain as many of them as possible. Location in the sentence, neighboring syntactic structures, capitalization will all be included. However, as a starting point, I am planning to use morphological markers. Some of the markers are unique to a certain part of speech, which carries over to a group of syntactic structures. For example, case markers are commonly an attribute of a noun or a pronoun. Therefore, a character sequence that has those will be marked as a noun or pronoun, and then, depending on its position in the sentence and other possible patterns the machine will establish, it will determine its role in the sentence and the part of speech.

Another important aspect of the work with natural language is its ambiguity. Depending on the position of the word within a sentence and the context, the word in question may change the meaning dramatically. This is yet another argument supporting the neural network approach. A particular position and the concordance analysis results may reveal the patterns of meaning that can only be discovered through the corpus approach. At the moment, providing an example of such is impossible, as the corpus is still in development, however, it is plausible those patterns will be revealed later on.

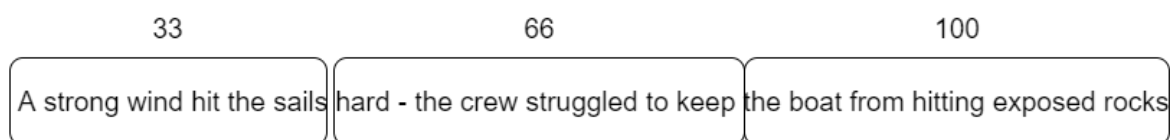
Neural network with word percentile as a weight-influencing factor

As mentioned above, one of the approaches that was taken when developing and testing the neural network was to use the percentile as one of the factors that will ultimately influence the assignment of the part of speech to the analyzed words.

The idea behind this approach was based on the idea that the Kazakh language has a fixed word order – subject, followed by an object, concluded with a verb. This reflects the general structure of a simple sentence in the Kazakh language: “Dosym qalamdy syndyrdym “ ([My] friend broke [a] pen) is an example of such a sentence.

The percentile is not a precise measure of the position of a given word in a sentence, but rather an approximation – since the number of words in a sentence is not constant, some percentiles will produce a result with a fraction. For example, 25% (twenty-fifth percentile) of a sentence with four words is going to include one single word, while the same percentile will produce 1.25 words in a sentence with six words. Obviously, 0.25 of a word means nothing in this case, therefore the measures are approximate.

Despite that, a framework that would be usable in the scope of this project ought to have specific boundaries; based on those, it would then assign the part of speech to a word. An important note here is that the percentile has to end at a hundred, marking the end of a sentence. Initially, it may appear that based on just three elements of a simple sentence, the best candidate for the percentile set would be one based on step 33 – 33%, 66% and 100%; values after the decimal point have been truncated, as the accuracy did not add anything to the degree of the model’s precision.



The figure above demonstrates the principle behind the percentile model

An important consideration here also involves the average number of words in a sentence in the news article genre in the Kazakh language. Establishing an

average amount of words per sentence in the Kazakh language overall would be too vague – it is highly likely that in literary texts the number would fluctuate beyond usable margins. For example, a sentence “Kel!” (Come!), while being a perfectly valid sentence, is only a single word, giving the number of 1. On the other hand, a sentence from “Abay Zholy” (Abay’s Path), a novel by Mukhtar Auezov “Екеуі жарыса жөнелгенде, амалсыз егеске түсіп, "мен озам, мен озаммен" тепкілесіп, созыла берді”⁴³ (“When the two of them were racing, they kept shouting “I’m ahead, I’m ahead””) contains 13 words; it is entirely possible that there are longer sentences in the book. What this indicates is that the average of words per sentence may vary according to the genre of the text – some genres, like literary, are likely to have to longer average length, while others, for example, online messaging, may be very short, as omissions are acceptable. After surveying the news articles gathered over the course of the research, I have determined the average number of words per sentence is 8.36.

Given the data above, I set out to determine which step would offer the most accuracy while still maintaining a reasonably large gap. After testing several sets of values, with steps ranging from 10 to 50, I have determined that the optimal step for this approach would be 25. This produces a set of 25-50-75-100, with the hundredth percentile being the end of the sentence. Essentially, those numbers represent the portion of the sentence where the word occurs. 25, in this case, would mean that the word is encountered in the first 25% of the sentence, 50 — word encountered after the 25th percentile, but before the half of the sentence, and so forth. This set of percentiles allows for maximum flexibility in terms of number of words in a sentence for which it is applied.

⁴³ Kitap.kz “Абай жолы”

The expected behavior of this model is as follows. Given the SOV structure of Kazakh sentences, it can be safely assumed that most sentences will follow that pattern, or a variation of it. However, it is highly likely there will also be other words representing other syntactic categories. It is also possible that the verb will consist of multiple words — there is plenty of examples in the Kazakh language, such as “келе бер” (“come”), “жазылып кет” (“get better”), with the second verb being an auxiliary one (“келу, кету, жату, тұру”). There are nuances in meaning when those auxiliary verbs are changed to one another, however, they do not carry their original meaning anymore. Still, they are a part of the verb. Thus, according to the percentiles laid out above, the sentence “” will be broken down into 4 parts: 25th percentile - “”, 50th percentile - “”, 75th percentile - “”, and the 100th percentile - “”. The word in the first percentile is likely, according to the model prediction, to be a subject of the sentence. Given that it is a subject, there are 2 main possibilities: the word in the 25th percentile is either a pronoun or a noun. The disambiguation between the two is rather simple: I opted for having the pronouns hardcoded, as there is a small number of those, and to avoid confusion with some case suffixes. Therefore, it is expected that the program will assign either noun or pronoun category to the word based on its syntactic properties.

The words in the 50th percentile, according to the model, are likely to be related to the object; an object in the Kazakh language is often represented by a noun, or a pronoun (granted, there are more complicated cases, but those shall be covered below). In order to avoid falsely identifying an adjective as a noun, the program first checks for typically adjective suffixes within the words of that percentile; if the search returns nothing, then the program will follow the same steps as for the previous percentile – check for pronouns, and then for nouns.

Finally, the concluding percentiles are mainly allocated for verbs. 100th percentile is generally either the auxiliary verb, or a regular verb with a lexical meaning; 75th, on the other hand, is an ambiguous percentile, which may contain a verb or a different part of speech. There is a high confidence in the prediction in the final percentile, as the sentences would have to end with a verb.

Unfortunately, while in theory the system would have made the part of speech assignment based on syntactic features easy to implement, the practical tests have shown a rather different picture. While the neural network with the percentile setup was able to work with some sentences, it was falling short on many occasions. The only percentile that achieved a reasonable degree of accuracy ($\pm 84.63\%$) was the last, 100th percentile. Other percentiles demonstrated a significantly worse performance, rendering the model unusable in its current state.

Some of the failures can be attributed to the fact that while the skeletal structure of a Kazakh sentence is SOV, there is no guarantee that a subject will come first in the sentence. This became more apparent in the longer sentences. To illustrate, let's consider the following sentence: "Әдемі қыз дауыстап ән айтады." (A beautiful girl sings loudly). While this sentence is unlikely to appear in the press, which is the main focus of this corpus, it nonetheless demonstrates the shortcomings of the percentile approach clearly. By approximating the 20 step-based percentile ($100/5 \text{ words} = 20$) to the working 25 model, the percentile breakdown is as follows:

25 - Әдемі (beginning of the sentence)

50 - қыз дауыстап (40 and 60 are closest to 50)

75 - ән (80 is closest to 75)

100 - айтады (final word, 100).

According to the model outlined above, the first percentile (25) is for subject; however, in this case we have a noun phrase (adjective+noun). As the adjective is first, it is allocated to the first percentile, and thus becomes a noun instead of an adjective, which is erroneous. Percentile 50 and 75 had consistently correct assignments, and so did 100, which contains a verb. Therefore, the noun phrase has a potential of disrupting the work of the neural network, which negatively impacts further progress - since the network learned that there is a possible ambiguity in terms of the first word being either a noun or an adjective, further results were becoming less predictable.

It became apparent that this model would require manual supervision and correction of errors after each epoch, which delays effective learning significantly. In addition, it did not work well with a set of words when the structure of the sentence was not preserved: dictionary-style text file (a word per line with no indication of syntax produced very inaccurate results, rendering the program unusable. In addition, Kazakh is a pro-drop language, which can drop the subject (noun/pronoun) if the object of reference is clear from the preceding context. In order to account for that, the neural network had to be expanded significantly to allow for context memorization from the previous sentences, and it also had to learn to detect the missing features. While the neural networks are great at detecting present hidden patterns and features, it is unable to deal particularly well with the absence of those features. In addition, the attempts to determine when the previous context should be wiped to allow replacement with the new context were futile – the references to a subject could be throughout the entire article under analysis, and there were more references to different subjects within 2 sentences. The multi-layered nature of the context made it difficult to maintain the network, and it started to run out of memory

past various points, with the run time becoming increasingly large to the point of it being unfeasible, given large enough samples of test data.

The network could potentially be adjusted to work with more complex sentence structures that have multiple clauses, however, at this stage of the project this effort is not feasible.

Neural network based on morphological features with cross-references to training dataset as a dictionary

The second approach employed in the research was based on purely morphological features with cross-references to the training dataset, dictionary style. This approach uses the morphological features present in the word in order to assign the part of speech to it. If the word has no distinguishable morphological features (a noun in nominative case, for example), the network would then use the previous word to try and determine the part of speech (concordance recognition). If this fails, the neural network will switch to dictionary lookup, scanning the training data to find the unknown word. Finally, if even that yielded no results, the neural network will mark the word for review and move ahead. During the subsequent runs, it may be able to tag some words correctly without the manual correction; however, I was unable to register what is there in common between the tagged words.

The training dataset for this network is a mix of publicly available data from Apertium's GitHub repository and sentences, annotated semi-automatically by me. This provides the program with a large volume of training data, improving the accuracy, and a sizable dictionary to reference for later.

As the program runs through multiple epochs, it expands its vocabulary and discovers more features that would allow it to determine the part of speech. Context-

free grammar provides the basis for the lookup of those features. For example, assignment of adjectives has not been an issue for this iteration of the neural network, as generally the adjectives have predictable morphological patterns, such as suffixes, initial reduplication, that distinguish them unambiguously in most cases. For example, a context-free grammar rule indicates that if there is an adjective, there has to be a noun adjacent to it ($N \rightarrow NP \rightarrow Adj+N$). While it may not hold 100% correct all the time (a case of pronoun+adjective is also possible), it provides part-of-speech assignment with a high degree of accuracy, higher than the previous model.

This model was able to achieve an admissible accuracy threshold of 88.32% over a sample of 40 articles, which is 9624 words, with the initial training dataset of 5000 words. The test sample can be easily expanded by providing more URLs to the scraper and can be done on demand.

Chapter 5: Conclusion

Conclusion

Corpus linguistics is a very powerful modern tool for both the academics and general public. It is a rapidly growing branch of linguistics that demands attention. However, not all languages enjoy equal coverage in the digital era. The study highlights that the Kazakh language has still relatively poor coverage from the computational side. The descriptive literature on the Kazakh language is plentiful, but only a part of it is English. One of the most recent books, by Muhamedowa (2016), while offering insights into the grammatical and morphological aspects of the Kazakh language, has no guidance on the ways to approach the Kazakh language computationally. As identified in Muhamedowa (2016) and Bekbulatov et al. (2014),

an agglutinative language, Kazakh poses some challenges for the natural language processing engines, since it uses both prefixation and suffixation, which makes it hard for automated processors to distinguish between the root and the derivational/inflectional morphemes. These problematic issues in Kazakh language are well-discussed by Bekbulatov et al. (2014), but the ways to solve them were not offered. The status of National Corpus of Kazakh Language remains largely uncertain, with little to no updates over the past years, which indicates the corpus is likely becoming outdated, given its initial source was mostly literary texts.

The software that can be expected to work with the Kazakh language should be able to deal with complex morphology of the Kazakh language. Therefore, programs/engines that work well for a language like English will probably not work well for the Kazakh language. This means that when developing or testing software for the Kazakh language, anything ranging from morphological transducer to speech synthesis engine, it is imperative to ensure that the application can work with the morphology. On the other hand, Kazakh features a fixed word order, unlike Russian, for example, which makes it easier to create rule-based processing tools. However, Kazakh language is also a pro-drop language, which means it may drop the subject or pronoun if the contextual clues give an indication of what the subject of the sentence is. This issue has the potential of disrupting the workflow of some programs that expect there to always be at least a subject or verb – the software may either consider the object to be the subject, mark sentence as ungrammatical, or stop working altogether if this case was not considered.

Generally, the lack of a unified annotation system for the Kazakh corpus, coupled with the challenging nature of Kazakh language in terms of morphology, contribute towards its poor representation computationally. Those challenges can be

overcome, by ensuring the standardization of the Kazakh language description system. The steps are being taken, as there are multiple projects currently being developed; however, more is yet to be done.

Overall, this project sheds light on studying the Kazakh language through the quantitative methodology via corpus linguistics, its challenges and approaches. By surveying the literature available, I have identified the key problematic and explored possible solutions. The project developed a method for creating a sample corpus and possibly a standalone software for corpus analysis that would fit Kazakh language with its complexities. The method implies merging the sample corpus with the lexc file used by Apertium, a free open-source software, to contribute to the Kazakh language digitization effort.

In the process of accomplishing this, I have first identified the areas that could be potentially challenging for machine recognition, such as complex Kazakh morphology and two competing writing systems. I have also identified a particular utility of a uniform, publicly available system of notations that can correctly reflect the nuances of the Kazakh language. To address the issue, I have used the available descriptions of Turkic language family in general, and Kazakh language specifically. In the process, I have also consulted the practically applied solutions of Apertium, identifying its shortcomings and advantages.

Having accomplished the above, the next issue is gathering the data for the corpus. The literature suggests the Internet can be a representative corpus for written Kazakh language; at this stage, this is the best data collection environment. The data collection involved several stages: first, I created a list of URLs from news websites that would serve as the sources of the data. Then, I made a series of programs that worked with the text from start (fetching the raw HTML data from the

web) to finish (encoding/decoding the data and processing it via a neural network). I have used several available Python packages for this purpose – urllib for request to the web servers, BeautifulSoup4 for cleaning up the raw HTML data and finding relevant tags, and TextBlob, for breaking the text down into smaller pieces and recording the data and some basic info about the text.

Next, specialized software is created utilizing the annotation system devised at the earlier stage. This software is designed to perform several tasks: first, to break down the text into one-word-long chunks with identified part-of-speech affiliation. The next task applies morphological transducer to parse the words into affixes and roots. This concludes the current scope of the project.

As the neural network offers a potent tool for discovering the principles behind the language, I have decided to utilize it within this project as part-of-speech tagger. By first manually annotating a set of words, as well as including a part of previously tagged corpus, tagging them, and feeding them to the network, it developed an understanding of what constitutes a particular word class in the Kaza kh language. This builds a vocabulary of known and tagged words that can be addressed later, making the neural network involvement smaller as more words are tagged.

The project also made use of the neural network for part-of-speech tagging. This method of tagging parts of speech can achieve a good accuracy using some human input during the training process. The approach eliminates the need for constant human supervision the longer the model trains. Essentially, the neural network built for this project resembles a perceptron – it had to classify the inputs (words) as belonging to a particular part of speech based on a set of features such as previous word, the presence of prefixes/suffixes, the kind of suffixes/prefixes present, and others. The network was trained on a data set consisting of data taken

from Apertium-kaz repository and data manually tagged by me. The final version of the neural network was able to classify the words with 88.32% accuracy – while the state-of-the-art software can produce more accurate results, I would consider this neural network to be a successfully implemented model based on the limited amount of the training data.

This research project in its current design has achieved several goals: identifying the key parameters of the annotation system for the Kazakh language; creation of applications able to work with the Kazakh language utilizing the said annotation system; creation of a sample corpus to be annotated and analyzed using the above applications to prove the project accomplished its goals; subsequent integration of the above solutions with the Apertium-kaz system to address the larger aim of contributing to digitization of the Kazakh language through the creation of more diverse corpora, which includes not only literary Kazakh texts, but also more contemporary versions through use of newspapers. This method enables the researchers to gain access to the version of language that is used on the regular basis, but at the same time does not have the same set of challenges associated with processing spoken data or informal conversations.

Project outlook

Still, the scope of work for the future research is grand. The present project does not take the multilingual nature of the region into account, as that requires a separate research in and of itself. The source of the data for the corpus has been chosen in an attempt to stay as close to monolingual as possible – speech or Internet communication will likely be multilingual. Therefore, this is one of the

potential research topics for the future - adapting the corpus technologies to more than one language.

References

- Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. *Compilers, Principles, Techniques, and Tools*. Reading, Mass: Addison-Wesley Pub. Co, 1986.
- Almaty Corpus of Kazakh. http://web-corpora.net/KazakhCorpus/search/?interface_language=ru Accessed October 30, 2020.
- Apertium Wiki. http://wiki.apertium.org/wiki/Main_Page. Accessed on March 4, 2020.
- Apertium GitHub repository. <https://github.com/apertium/> Accessed on March 6, 2020.
- Bekbulatov, Eldar, and Amandyk Kartbayev. "A Study of Certain Morphological Structures of Kazakh and Their Impact on the Machine Translation Quality." In *2014 IEEE 8th International Conference on Application of Information and Communication Technologies (AICT)*, 1–5. Astana, Kazakhstan: IEEE, 2014. <https://doi.org/10.1109/ICAICT.2014.7036013>.
- Culpeper, Jonathan, Paul Kerswill, Ruth Wodak, Tony McEnery, and Francis Katamba. *English Language: Description, Variation and Context*. Second edition. London: Palgrave, 2018.
- Flowerdew, Lynne, and Václav Březina, eds. *Learner Corpus Research: New Perspectives and Applications*. Research in Corpus and Discourse. London: Bloomsbury Academic, 2018.
- Gatto, Maristella. *The Web as Corpus: Theory and Practice*. Corpus and Discourse. London ; New York: Bloomsbury, 2014.

- Glynn, Dylan, and Justyna A. Robinson, eds. *Corpus Methods for Semantics: Quantitative Studies in Polysemy and Synonymy*. Human Cognitive Processing, volume 43. Amsterdam ; Philadelphia: John Benjamins Publishing Company, 2014.
- Gries, Stefan Th, and Brian G Slocum. "Ordinary Meaning and Corpus Linguistics" 2017, no. 6 (n.d.): 1417–72.
- GNU. Free Software Philosophy. <https://www.gnu.org/philosophy/free-sw.en.html> Accessed on April 10, 2021.
- ISSAI. Kazakh speech corpus. <https://issai.nu.edu.kz/kazakh-speech-corpus/> Accessed on April 1, 2021
- InformBuro. "Латын әліпбиінің "заңды нұсқасы" тағы да өзгеруі мүмкін. Өзбек, түрік, әзербайжан және қазақ әліпбиі қаншалықты ұқсас?" <https://informburo.kz/kaz/latyn-lpbin-zady-nsasy-tay-da-zgeru-mmkn-zbek-trk-zerbayzhan-zhne-aza-lpbi-anshalyty-sas.html> Accessed on February 21, 2021.
- James, Joshua, Pavel Gladyshev, Mohd Taufik Abdullah, Yuandong Zhu. "Analysis of Evidence using Formal Event Reconstruction" <https://arxiv.org/ftp/arxiv/papers/1302/1302.2308.pdf> Accessed April 12, 2021.
- KazCorpus. <http://kazcorpus.kz/> Accessed July 3, 2020.
- Kitap.kz "Абай жолы. I том. Мұхтар Әуезов." [https://kitap.kz/book/4738/read#epubcfi\(/6/2\[id19\]!/4/2/4/1:0\)](https://kitap.kz/book/4738/read#epubcfi(/6/2[id19]!/4/2/4/1:0)) Accessed on January 12, 2021.
- Galiev, Alexander. Profit.kz "Rynek Smartfonov v Kazakhstane: svetloe pyatno", Smartphone Market in Kazakhstan: a white spot. June 25, 2013.

<https://profit.kz/articles/1980/Rinok-smartfonov-v-Kazahstane-svetloepyatno/> Accessed August 26, 2019.

Horgan, John. Scientific American. "Is Chomsky's Theory of Language Wrong? Pinker Weighs in on Debate". November 28, 2016.

<https://blogs.scientificamerican.com/cross-check/is-chomskys-theory-of-language-wrong-pinker-weighs-in-on-debate/> Accessed December 13, 2020

Kopaczyk, Joanna, Jukka Tyrkkö, and European Society for the Study of English, eds. *Applications of Pattern-Driven Methods in Corpus Linguistics*. Studies in Corpus Linguistics, 1388-0373 82. Amsterdam ; Philadelphia: John Benjamins Publishing Company, 2018.

Litosseliti, Lia, ed. *Research Methods in Linguistics*. Research Methods in Linguistics. London ; New York: Continuum, 2010.

Loria, S., 2018. textblob Documentation. *Release 0.15, 2*.

McEnery, Tony, and Andrew Hardie. *Corpus Linguistics: Method, Theory and Practice*. Cambridge Textbooks in Linguistics. Cambridge ; New York: Cambridge University Press, 2012.

Meftah, Sarah, Nasredine Semmar, Fatiha Sadat. "A Neural Network Model for Part-Of-Speech Tagging of Social Media Texts".

<https://www.aclweb.org/anthology/L18-1446.pdf> Accessed on January 23, 2021.

Mozilla CommonVoice. <https://commonvoice.mozilla.org/kk/> Accessed March 29, 2021.

Muhamedowa, Raihan. *Kazakh: A Comprehensive Grammar*. Routledge Comprehensive Grammars. London ; New York: Routledge, 2016.

Munday, Jeremy, Kim Wallmach, and Alet Kruger. *Corpus-Based Translation*

Studies: Research and Applications. London: Bloomsbury Academic, 2013.

National Corpus of Kazakh Language. <http://web->

[corpora.net/KazakhCorpus/search/](http://web-corpora.net/KazakhCorpus/search/) . Accessed on March 5, 2020.

Phillips, James C., and Jesse Egbert. “Advancing Law and Corpus Linguistics:

Importing Principles and Practices from Survey and Content-Analysis

Methodologies to Improve Corpus Design and Analysis.” 2017, no. 6 (n.d.):

1589–1620.

Python Docs, URLLIB library. <https://docs.python.org/3/library/urllib.html>

Accessed September 16, 2020.

Python Docs, Timelibrary. <https://docs.python.org/3/library/timeit.html> Accessed

October 10, 2020.

Qazaq Grammar on Facebook.

<https://www.facebook.com/qazaqgrammar/photos/a.269068856817479/1372>

[192576505096](https://www.facebook.com/qazaqgrammar/photos/a.269068856817479/1372) Accessed on August 12, 2020.

Richardson, L. Beautiful soup documentation. *April*.

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/> 2007.

Schmitt, Norbert, ed. *An Introduction to Applied Linguistics*. 2nd ed. London:

Hodder Education, 2010.

Sjur Nørstebø Moshagen, Trond Trosterud. “Rich Morphology, No Corpus – And

We Still Made It. The Sámi Experience”. Proceedings of the Language

Technologies for All (LT4All) , pages 379–383 Paris, UNESCO

Headquarters, 5-6 December, 2019.

<https://lt4all.elra.info/proceedings/lt4all2019/pdf/2019.lt4all-1.95.pdf>

Stanford University. Slides. web.stanford.edu/ Accessed April 5, 2021

Tools for Corpus Linguistics. corpus-analysis.com/ Accessed May 2, 2020.

Universal Dependencies, Kazakh language corpus.


https://github.com/UniversalDependencies/UD_Kazakh-KTB Accessed on April 10, 2021.

Washington, Jonathan N, Ilnar Salimzianov, Francis M. Tyers, Memduh Gökırmak, Sardana Ivanova, Oğuzhan Kuyrukçu. “Free/Open-Source technologies for Turkic languages developed in the Apertium project”. International Conference on Turkic Language Processing (TURKLANG 2019)

Appendix A

сілтеме жасап.

ҰҚСАС ЖАҢАЛЫҚТАР



Атырау тұрғыны жұмысқа орналастырамын деп 14 адамды алдап кеткен

Астанада 90 адам қаржы сауаттылығы бойынша компанияға алданған

Өскемен тұрғыны алаяққа алданып, жарты млн теңгесінен айырылды

Телефонмен жасалатын алаяқтықтан қалай сақтануға болады

Вашингтон сотының шешімі молдавандық кәсіпкерлер Анатол және Габриэль Стати жасаған кең ауқымды алаяқтық схемадағы рөлі үшін Чэпменге қарсы Қазақстан Республикасының шағымын қарауды тоқтатуға қатысты нью-йорктік қаржыгер Д.Чэпменнің әрекетін жоққа шығарады.

Алаяқтық 2013 жылғы Қазақстанға қарсы төрелік шешімді алу үшін пайдаланылды.

2021 жылғы 19 наурыздағы шешімімен АҚШ-тың Колумбия округінің аудандық соты Дэниэл Чэпменнің және оның төрт компаниясының – Argentem Creek Holdings LLC, Argentem Creek Partners LP, Pathfinder Argentem Creek GP LLC және ACP I Trading LLC (бірлесіп «Чэпмен» деп аталады) Қазақстанның Чэпменге қатысты талаптарын қарауды тоқтата алатын алдын ала тыйым салу туралы өтініш беруге рұқсат алу үшін берген өтінішін қабылдаған жоқ.

2020 жылдың маусым айында Қазақстан Чэпменге қарсы Нью-Йорктің Жоғарғы сотына шағымын берді.

18:15

Алма бойж
Аймаг

18:01

Таны озды
Оқиға

17:50

Қорғ әскер
Аймаг

17:48

Атыр Науры



17:11

Коро жағд

Example of “Suggested news” (https://www.inform.kz/kz/stati-isi-aksh-soty-kazakstannyn-bergen-talap-aryzyn-toktatudan-bas-tartty_a3767338)

Appendix B

```

<p></p>
...
<div class="frame_news_article"> == $0
  <div class="frame_title_art"> Ұқсас жаңалықтар </div>
  <div class="main_frame_news">
    <a rel="nofollow" href="/kz/atyrau-turgyny-zhumyska-ornalastyramyn-dep-14-adamdy-aldap-ketken_a3766196">...</a>
    <a rel="nofollow" href="/kz/atyrau-turgyny-zhumyska-ornalastyramyn-dep-14-adamdy-aldap-ketken_a3766196">Атырау тұрғыны жұмысқа о алдап кеткен</a>
  </div>
  <a rel="nofollow" href="/kz/astanada-90-adam-karzhy-sauattylygy-boyyynsha-kompaniyaga-aldangan_a3766001">Астанада 90 адам қаржы сау алданған</a>
  <a rel="nofollow" href="/kz/oskemen-turgyny-alayakka-aldanyo-zharty-mln-tengesinde-ayyryldy_a3765244">Өскемен тұрғыны алаяққа алда айырылды</a>
  <a rel="nofollow" href="/kz/telefonmen-zhasalatyn-alayaktyktan-kalay-saktanuga-bolady_a3764218">Телефонмен жасалатын алаяқтықтан к
</div>
<p>...</p>
<p>Алаяқтық 2013 жылғы Қазақстанға қарсы төрелік шешімді алу үшін пайдаланылды.</p>
<p>...</p>
<p>...</p>

```

Code representation of the Appendix A