Morphchecker for non-standard data: a tool for morphological error correction in learner corpora

Olga Ramzaitseva[1], Robert Zakoyan[1], Aleksandr Ivankov[1], Alina Ladygina[2]

olga.ramz.2012@gmail.com, animexones@gmail.com, asivankov_1@edu.hse.ru, aladygina@yahoo.com

[1]National Research University Higher School of Economics, School of Linguistics, Moscow, Russia

[2]University of Tübingen, Germany

Abstract. This paper describes a tool for automatic correction of morphological errors specific for texts written by second language learners of Russian and Russian heritage speakers. As a part of that task, the problem of splitting words into morphemes is also solved.

Keywords: Key words. Russian Learner Corpus , second language learner, morphological error, error correction spellchecker, word splitter, inflectional error, hunspell dictionary, erroneous word.

1 Introduction

The Morphchecker project is carried out as a part of the Russian Learner Corpus [1]. The goal of the project is to develop a module for automatic morphological error analysis for non-standard Russian. The resulting system should be able to find word forms with morphological mistakes and suggest corrections for them.

This paper describes the results of morphological segmentation and correction of inflectional errors. In Section 2, we describe the data used for development of the morphchecker tool. In Section 3, morphological error classification is presented. Section 4 describes sources and tools used in the project. Section 5 presents the algorithm for segmentation of lexical items into morphemes and correction of inflectional errors. Section 6 discusses the preliminary results of the current version of the morphchecker.

2 Data

The data was taken from The Russian Learner Corpus [1]. The corpus was created in the Linguistic Laboratory of Corpus Technologies of the -School of Linguistics at

the National Research University Higher School of Economics [2]. It contains texts written by second language learners of Russian and Russian heritage speakers. The corpus is provided with error annotation, including specific tags for morphological errors. For the purposes of our project, 220 contexts containing morphological errors were retrieved from the error-annotated part of the corpus via corpus search.

3 Error classification

First of all, it is necessary to define what is considered to be a morphological error. In order to define common affix errors, we examined morphological errors annotated in the corpus, and elaborated the following error types:

3.1 Inflection errors

Inflectional errors can occur in inflectional endings or in word stems.

Inflectional endings. Mistakes in inflectional endings are often caused by non-standard speakers' incomplete command of the Russian phonetics. The following categories of inflectional affixes tend to be frequently confused:

1. Affixes sharing the same grammemes, but belonging to different paradigms within the same part of speech.
   a. 'hard' and 'soft' stem paradigms. Russian nouns and adjectives are inflected in two types [3]. Words with stems ending in an unpalatalised ('hard') consonant follow the 'hard' inflection type, while words with stems ending in a palatalised ('soft') consonant are inflected in the 'soft' type. A mistake occurs when the speaker uses a 'soft' inflectional ending with a stem ending in a 'hard' consonant, or vice versa: *экзамении (экзамены)*. Similar cases can be found in verbs: *оцену (оценю).*
   b. confusion between gender paradigms. For instance, the noun *загрязнением* being of a neutral gender has a feminine singular instrumental case affix *–ей.* Zero affixes also fall into this subcategory, as in the case of *легендов:* the masculine plural genitive affix *–ов* was added when none is required for making the feminine plural genitive form.
2. Affixes sharing the same grammemes, but belonging to different parts of speech. For example, the adjective *руссками* (the word *русскими* seems to be intended) has a noun instrumental case affix *–ами*.
3. Pseudo-affixes similar to the existing ones. For example, the pseudo-affix *–имы* in *тёмнимы* that resembles instrumental case affixes (in this case *-ыми*).

In some cases, it is difficult to specify an appropriate correction without taking into account the context, in which an error occurred. For instance, the affix *'и'* in the word *экзамени* could denote several meanings: the masculine nominative plural (- ы)

due to confusion between hard and soft paradigms, masculine singular locative (*-е*) being a mere phonetic error or, less likely, feminine genitive (*-и)* due to confusion between genders.

Stems. Errors in stems can occur due to consonant or vowel alternations: *бежут (бегут), лёдом (льдом)*. Stem errors in verbal inflection can also result from confusion between several stem variants. For example, the stems of such verbs as *жаловаться, контролировать, сочувствовать* in infinitive and past tense forms end in *-ова- [ova]*. However, present tense forms of these verbs are formed using different stems that end in *уj- [uj]*. Non-native speakers tend to confuse the *ова- [ova]* stem and the *-уj- [uj]* stem and make incorrect forms: *жаловаются (жалуются), контролировает (контролирует), сочувствовают (сочувствуют)*.

3.2 Word formation errors.

Word formation errors occur if the speaker uses an incorrect combination of morphemes and, as a result,derives a new word. Mistakes in word formation can be divided into several subtypes depending on which morpheme is omitted, inserted or substituted with an incorrect one. Table 1 illustrates typical derivational errors:

Table 1. Types of derivational errors

Type	Examples of errors	Correct forms
suffix omission	московие	московские
suffix insertion	глупные	глупые
suffix substitution	пропагандическая	пропагандистская
prefix omission	противоставлением	противопоставлением
prefix insertion	всерьёзнее	серьёзнее
wrong prefix choice	неморально	аморально
vowel/consonant - alternations in the root	ежемесяцным	ежемесячным

4 Sources and Tools Used

Various resources were used in development of the morphchecker tool. The linguistic data was taken from the Russian Learner Corpus.

The programs for segmentation into morphemes and correction of inflectional errors were written in Python 3.5.1 with the use of pymorphy 0.5.6 [3] and NLTK [4] packages.

The programs use morphological data from A.I. Kuznetsova's dictionary of Russian morphemes [5] and the Russian Hunspell dictionary [9]. The machine-readable version of A.I. Kuznetsova's dictionary of Russian morphemes is a comma-separated values file containing the following information: words lemmata, their parts of speech, constituent morphemes, morpheme types (e.g. root, suffix, prefix etc.) and morpheme allomorphs. The Hunspell dictionary consists of dictionary and affix files. They contain the inflection rules for each given lemma. Each dictionary item is provided with a code denoting the list of all possible inflections divided into groups that can be found in the affix file. The example (1) below shows that inflection rules for the word *последний* can be found by looking up the codes CC, CD and CO in the affix dictionary.

(1) *последний/CCCDCO*

The CC code, for instance, maps to several lines in the affix dictionary, some of them are given below:

SFX CC ый ыми ый (А.мн.ч.т.п.)
SFX CC ий его [^гкх]ий (А.ед.ч.м.р.р.п.)
SFX CC ий ого [гкх]ий (А.ед.ч.м.р.р.п.)

This notation defines the structure of the corresponding word form. For example, if the word *последний* has the tag *А.ед.ч.м.р.р.п.* (i.e. adjectival singular masculine genitive form), there are two ways of generating a word form, depending on the last character of the stem. Therefore, the last letter of the stem should be additionally checked in order to find a correct ending.

In this case, it is necessary to check if the lemma has a stem ending with a consonant other than *г, к* or *х* using regular expression *[^гкх]ий,* and then *ий* affix must be substituted with *его* affix.

The algorithm for segmentation into morphemes also relies on Porter Stemmer [7] as an initial step of word stripping, see 5.2 for details.

The algorithm for correction of inflectional error uses the output of the spellchecker for the Russian Learner Corpus based on Hunspell and Aspell spell checking engines [6].

5 The Correction Algorithm

The correction algorithm includes seven steps divided into three modules: Spellchecker, Word Splitter and Morphchecker.

5.1 Getting Suggestions for the Correction. Spellchecker

The spell-checker returns four possible outcomes:

1. the word is deemed correct;
2. the word is deemed incorrect and the Spellchecker suggests some correction variants and intended word is among them;
3. the word is deemed incorrect and the Spellchecker suggests some correction variants, but they all belong to wrong lemmas;
4. the word is deemed incorrect and the Spellchecker has nothing to suggest.

The first group consists of real-word errors, i.e. correct forms of words which are incompatible with the context [7].

The other three represent non-word errors, i.e. words or word forms that do not exist in a language.

Real-word errors are not morphological, but rather syntactical, so they are of no concern here. The following algorithm is dealing only with the case b), since it is the most frequent type of error. The cases c) and d) are subject to further research.

5.2 Segmentation into Morphemes. Word Splitter

First of all, Segmentation algorithm is partly based on Porter Stemmer [8] and relies heavily on the A.I. Kuznetsova's dictionary of Russian morphemes [5], in order to check the correctness of resulting *root* morpheme if possible.

The algorithm goes with the following logic:

1. The Porter Stemmer is run to produce a *stem*. In short, it strips away a set of predefined *i-* and a very limited set of *d-* (*inflectional* and *derivational* respectively) suffixes, leaving a *stem* with at least one vowel. It is modified to collect these suffixes instead of just stripping them, so that the original analysis could be carried out later.

The logic of the next steps depends on the presence of what is presumed to be the *root* in the dictionary (this check is referred as root consistency later). These additional stripping steps are supported by the fact that Porter Stemmer tends to strip less than is needed and ignores prefixes. Note that all these steps also imply that the root has at least one vowel :

2. If the *stem* appears in the dictionary as *root*, the process stops and result is recorded.
3. A *prefix* is located and *rc-check* is done. If the root is present in the dictionary, the word is supposed to be segmented correctly and the result is recorded.
4. A *d-suffix* is located and the result is rc-checked. This step is applied to the whole *stem*, as if step 3 never happened. Again, the result is recorded if root appears in the dictionary.

5. A *prefix* and *d-suffix* are now located successively. If rc-check is passed, record results.
6. A tweak is applied to the stem - first syllable of the part that was cut by Stemmer is added back.
7. Steps 2-5 are repeated on an updated stem.

If a correct *root* is never found, the dictionary is considered to be incomplete and morphemes are stripped as is, starting from *prefix*.

The *prefixes* and *d-suffixes* are located by matching their respective lists to the beginning or the end of the *stem*.

Usually 4-th step is not trivial (e.g. more than one variant of the suffix could be found). Then the longest suffix is chosen as that strategy shows most reliable results so far.

As mentioned above, performance of this approach is heavily dependent on the representativeness and accuracy of the supported dictionary, so complementation and correction of Kuznetsova's dictionary is one of our working directions.

The idea behind word splitting is that by checking if morphemes in a target word match each other (appear in the same entry in the dictionary) a morphological error could be registred. This is not a standalone method and is supposed to cover specific cases, but it could detect those that could not be helped by other means. Consider word **добрость* that is a standard example of misuse of derivation technique. The word that was meant is supposedly *доброта*, and both of them are nouns (or, for the first case, could be labeled as noun as no such word exists). They obviously have different suffixes, both typical to nouns, but *-ость* is never put directly after добр. So, following that logic, we may conclude that this word was meant to be a noun, but got a wrong suffix, and corrections could be given. Conventional spellcheckers are usually not able to suggest any corrections at all.

Evaluation of Segmentation algorithm performance. An experiment was conducted to estimate the performance of the algorithm. A test set of 98 examples of supposedly morphological mistakes manually selected from the Russian Learner Corpus were separated by the algorithm. The result was a ~92% accuracy on the whole test set (92% of examples were separated correctly). Three examples could be considered having only spelling mistakes, so, after excluding them from the test set, the algorithm achieves ~95% accuracy.

The following is a showcase of the algorithm output. The words below are essentially easy to split because they have distinctive morphemes with no allomorphy, contain no mistakes and other difficulties:

1. Original : проповедники

   Separated: про:повед:ник:и

2. Original : персонажами

   Separated: :персон:аж:ами

3. Original : измена
Separated: из:мен:а

The following are the examples of mistakes typical for the Russian Learner Corpus and appear to be the main target of this study. They contain valid morphemes that are not supposed to be used in conjunction:

4. Original : встретаться (presumably meant встречаться)
Separated: :встрет:а:ть:ся

5. Original : хотятся (agent-passive twist of хотят)
Separated: хот:ят:ся

6. Original : уездят (уедут or уезжают)
Separated: у:езд::ят

As shown above, even though these words do not appear in dictionary, they are cut into distinctive parts and their mutual co-occurrence could be analyzed to discover the fact that the mistake happened due to a bad choice of morphemes and it is indeed morphological.

However, several cases yet pose a problem to the algorithm. Consider the following words - *падать* and *подать*. The outcome of their segmentation is:

7. Original : подать
Separated: по:да:ть

8. Original : падать
Separated: па:да:ть

In both cases the algorithm recognises an existing prefix, followed by a root also appearing in the dictionary. However, *падать* is supposed to be separated as *пад:а:ть*. Matching co-occurrences of morphemes could resolve this problem and should yet be implemented.

9. Original: физическо
Separated: физи:ческ:о

10. Original: метафорическо
Separated: метафор:ическ:о

In this example the word *физическо* is supposed to have a root *физ*, instead of *физи*, as it is derived from the word *физика (physics)*. However, because the dictionary also contains a word *физиология* with a *физи* root. So, as far as *ческ* is a valid suffix, the algorithm tends to chose a longer root, which produces a mistake in segmentation. This type of errors could be helped by dictionary standardisation - making the word *физиология* also have a *физ* root.

To summarize, we state that the algorithm could yet be enhanced. A great share of improvement is related to standardizing the input data, complementing dictionary resources and covering special cases. Nevertheless, as the target corpora of this algorithm are heritage and learners' mistakes that do not exhibit usage of overly sophisticated or domain-specific vocabulary, accuracy sufficient for the task is to be expected.

5.3 Correction of Inflectional Errors. Morphchecker

Having split an erroneous word into its constituent morphemes, we can proceed to correct affix errors. The general approach to correcting all the errors mentioned in Error Classification Section is the following: the intended form is reconstructed, taking into account all grammatical features a given morpheme can have.

The correction process can be divided into four steps:

Step 1. Grammeme Search

To know what grammatical form of the incorrect word was intended, we need the information on each grammeme that an affix can denote. All the affixes have been divided into groups given their grammatical features including the part of speech. For this purpose, the Hunspell .aff and dictionary files have been parsed to create a tool which easily maps the affixes returned by the Word Splitter to their grammemes. Pseudo-affixes have been then added to those groups that they are most similar to.

Step 2. Lemma Identification

Variants suggested by the Spellchecker are used to find all possible lemmas for a given erroneous word. All the variants suggested by the Spellchecker are lemmatized using the Python module pymorphy. Then we consider only unique lemmas in order to inflect them during step 4.

Step 3. Filtering

Two filters are used during this step.

First, part of speech of each candidate lemma is obtained and then compared to the POS-tag of the affix found during Step 1. All words with different POS-tags are ignored.

Second, the Word Splitter is used to extract the root of each lemma. If the root of the mistaken form does not match the root of a lemma nor any of its allomorphs, such candidate is not furtherly processed

Step 4. Reconstruction

The Hunspell dictionaries are used for this step. The program looks up the lemma in the .dic file. Then it gets codes mapping to the appropriate inflection rules for the lemma and looks up the code in the .aff file. Having obtained a list of the possible inflections for the word, it chooses the ones that correspond to the grammemes identified on step 1.

5.4 Pipeline

Now the whole pipeline can be illustrated, as follows:

1. The Spellchecker analyzes all the words in the text.

   input: *последного*
   output: [*подледного, последнего, последний, подледный]*

2. The Word Splitter takes each erroneous word as input and divides it into its morphemes.

   input: *последного*
   output: *последн:ого*

   The Mophchecker analyses the set of morphemes obtained and the output of the Spellchecker.

3. Each correction candidate is lemmatized by pymorphy3 [3] module and added to a set, in order to obtain only unique lexemes that the initial word could probably belong to.

   input: { *последн:ого* **: [***подледного, последнего, последний, подледный***]}**
   output: *подледный, последний*

4. The Mophchecker considers the affix of the input word. Then its grammatical information is found. It is represented as a set of tags including part of speech and the grammemes proper to that POS.

   input: *ого*
   output: *{'А.ед.ч.м.р.р.п.', 'А.ед.ч.с.р.р.п.', 'А.м.р.ед.ч.в.п.'}*

5. Some parts of speech may share one affix. Only those grammatical tags that have the same POS-tag as the lemma under consideration are kept to be processed during the next steps.

6. If there are more than one lemma, there should be a way to identify which one to choose. If the word's root is correct, we can check if it matches the root of a candidate or one of its corresponding allomorphs.

   input: *подледный, последний*
   output: *последний*

7. Now the system has a set of the most likely candidates (*последний*) to be the right correction for the word. For each lemma, there is a set of grammatical tags indicating all the possible forms it can be put into (*{'А.ед.ч.м.р.р.п.', 'А.ед.ч.с.р.р.п.', 'А.м.р.ед.ч.в.п.'}*). The correct form is identified using Hunspell dictionaries which contain the rules of the word formation.

   input: *последний*

   The corresponding line in the .dic dictionary is as follows:
   последний/CCCDCO

   In the .aff dictionary the necessary grammatical tag is found:
   SFX CC ий его [^гкх]ий (А.ед.ч.м.р.р.п.)

   output: *последнего*

6 Evaluation of the Correction algorithm performance

Performance of this algorithm is evaluated in comparison to the Spellchecker used to obtain suggestions for correction. Due to the fact that output results are qualitatively different, we can not compare Precision and Recall directly, so two tests are used to address that.

The two aspects of algorithm performance are put to test:

1) Mistake correction algorithm
2) Result filtering algorithm

The Spellchecker produces a list of candidates to choose from, while Morphchecker filters them out to give the most probable variant. From a practical point of view, this behaviour is more proficient as the users does not have to choose the correct form themselves. To measure the effectiveness of filtering technique, a test with the following set of rules has taken place:

Spellchecker gets 1 point, if a correct word is present in the results, and gets 0 otherwise.

Morphchecker algorithm gets a score equal to the number of candidates it filtered out divided by number of options it provided as a result, providing a correct option is still there, or gets 0 otherwise as well. To illustrate, consider a scenario - Spellchecker gives output of 4 candidates with a correct one, and Morphchecker narrows it down to a single correct option. In this case Spellchecker scores 1 point, and Morpchecker gets 3 points. If Morphchecker gave a result of 2 words, it would only

score 1 point, despite the fact it filtered out two wrong options. So, this test penalties poor filtering result while favors the opposite greatly.

Second test consists of a simple correction test, where both algorithms score a point if they have a correct option. This test is used to show the difference in correction accuracy.

The tests were carried out using the test set of 98 morphological mistakes manually selected from the Russian Learner Corpus.

Table 2. Evaluation results

	Mistake correction algorithm	**Result filtering algorithm**
Spellchecker	78	78
Morphchecker	88	200

The Morphchecker demonstrates much better results at filtering candidates for correction as it greatly reduces number of suggestions given by the Spellchecker. Furthermore, in 10 cases, it is shown to return the right correction even when the Spellchecker did not suggest it, which is possible as a result of lemmatization and further inflection of the lemma.

7 Conclusion

A tool for correction of various inflectional errors is presented in this paper. This project is to be implemented in the Russian Learner Corpus, a collection of texts produced by second language learners of Russian and Russian heritage speakers. The tool is a Python script including three separate modules: the Spellchecker elaborated for the Corpus by another group of researchers is used in this project to provide suggestions for correction done by the main algorithm; the Word Splitter is aimed to identify the morphemes of a given word; and the main algorithm called Morphchecker that considers grammatical characteristics of morphemes of the erroneous word, and the correction variants returned by the Spellchecker and then reconstructs the correct morphological form.

The correction algorithm shows reliable results on all types of inflectional errors described in Error Classification Section. However, we noticed some technical issues to fix, since the overall result will depend on the following.

First of all, the accuracy of further analysis heavily depends on the Spellchecker performance. It is sufficient for the Spellchecker to return at least one form (even having different grammatical features) of the intended word. If it fails to do so, the Morphchecker will not be able to find a correct lemma to put it into the appropriate grammatical form.

Secondly, the correctness of the results relies on the Word Splitter performance. Affixes returned by the Word Splitter must be easily found in the dictionary, so inconsistency is to be avoided.

Representativeness of the dictionaries is another problem to solve. Since we use two dictionaries on different steps of the analysis we should make sure that they can cover as many words as possible as new words can arise during further testing of the algorithm. A way to achieve that might be to update the dictionaries.

The next steps to proceed are the correction of derivational errors and automatic error type identification as well as elaborating the algorithm for dealing with cases when the Spellchecker is unable to suggest valid variants to be relied upon by the correction module.

8 References

1. Russian Learner Corpus http://www.web-corpora.net/RLC
2. Linguistic Laboratory of Corpus Technologies of the School of Linguistics at the National Research University Higher School of Economics https://www.hse.ru/org/hse/cfi/corpora/
3. Korobov M. (2015) Morphological Analyzer and Generator for Russian and Ukrainian Languages. In: Khachay M., Konstantinova N., Panchenko A., Ignatov D., Labunets V. (eds) Analysis of Images, Social Networks and Texts. Communications in Computer and Information Science, vol 542. Springer, Cham
4. Bird, Steven; Klein, Ewan; Loper, Edward (2009). Natural Language Processing with Python. O'Reilly Media Inc.
5. Spellchecker for Russian Learner Corpus. http://hsecompling.wikispaces.com/-/Heritage2014/Heritage2014/home
6. Jurafsky, Daniel, and James H. Martin. 2009. Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. 2nd edition. Prentice-Hall.
7. M. F. Porter. An algorithm for suffix stripping. Readings in information retrieval, pp. 313-316. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1997
8. Hunspell Official Website. http://hunspell.github.io
9. Кузнецова А.И., Ефремова Т.Ф. Словарь морфем русского языка: Ок. 52000 слов.- М.: Рус. яз., 1986.