Saarland University
Summer Semester 2018
Course Statistical Natural Language Processing
Lecturer: Prof. Dr. Dietrich Klakow

SNLP Project Report

# Morphological Inflection

August 15,2018

Student 1 — Ayan Majumdar
2571656
s8aymaju@stud.uni-saarland.de

Student 2 — Chirag Bhuvaneshwara
2571703
s8chbhuv@stud.uni-saarland.de

Student 3 — Hui-Syuan Yeh
2571618
s8huyehh@stud.uni-saarland.de

# Contents

# 1. Introduction

We are given the task of analysing how morphological inflections occur in different languages and recreating the same type of inflections on some unseen data. To do this, we have the lemma of the word, it's inflected form and inflection features in our training data.

The Baseline system we have implemented aligns the lemma and inflected form and uses Levenshtein distance to obtain the prefix and suffix changes. And if the language is mostly prefixing, we work with reversed strings. This allows us to capture all the prefix and suffix changes given the inflection features. We then use the learned prefix and suffix changes to inflect the words in the test data in a similar way using the given inflection features.

In Task 2, we are required to improve the performance of the baseline system on the language assigned to our team: Faroese. We have come up with some grammatical conditions using a Faroese grammar book to improve the accuracy score. This is explained in more detail in Part 2.

In Task 3, we are required to output the inflection features using the lemma and inflected forms. This is done by using our Task 2 code and is explained in more detail in Part 3.

# 2. Task 1

We first discuss the system that is designed for the first task. The problem is as follows: given a particular English word, and the inflection rule, the system should be able to predict the correct inflected form. For this the system is provided a training file that has the original word, the inflected form, and the inflection rule, each of these elements separated by whitespaces. The inflection rule is given by separation by ';'. The system that we have designed works as per the required specifications of the project. The code is provided in the file: task1.py. We also refer to the basic code as given in [1], from which we took some basic functions such as the computation of the hamming distance, and the alignment using recursive Levenshtein.

The function **halign(…)** returns the alignment of two strings that are passed as arguments that minimizes the hamming distance between the two. The function **levenshtein(…)** uses the memorization function **memolrec** and returns the best alignment of the two strings that minimizes the cost. This alignment is basically of three parts → a prefix, a stem and a suffix. These are organized by **alignprs(…)** and returned. Then, the function **get_prefix_suffix_changes(…)** is used, a brief sketch of this function is provided. The suffix and the prefix change rules are stored in dictionaries. The keys are the the inflection rules, and each key holds another dict. This sub-dict holds the key as the suffix/prefix change in form of a tuple, and the value is the total count this change has been seen in the training file. This count is used later for tie-breaking. These have been used and developed with the reference of [1].

```
def get_prefix_suffix_changes(lemma, inflection_form):

   1. Get the prefix, stem and suffix of the lemma and inflection_form
   2. Append symbol '$' to the strings that constitute the stem and suffix terms
      appended. They are represented as lemma_string and inflection_string.
   3. Loop i from 0 to the minimum of the length of lemma_string and
      inflection_string
         a. Add suffix rule lemma_string[i:end] > inflection_string[i:end]
   4. Remove any '_' symbols that remain as they were used by alignprs(…)
   5. Store the suffix rules as a set.
   6. Now append symbol '@' to the lemma and inflection_form prefix parts.
   7. Repeat steps 3, 4 and 5 for the prefixes.
```

We also check for if a particular language is actually prefix based or suffix based. This is computed from the training file of the language by aligning each lemma-inflection by Hamming distance, and computing the changes that are done on the prefix side and the suffix side. Whichever is greater, the language is judged to be corresponding to that. This heuristic is used throughout the system. If the language is prefix based, we simply reverse it, use the same system so that the system looks at it as suffix based, and finally while giving output, reverse it again.

Finally, during testing, we read each line, and then call the function **generate_inflection(…)** which looks up the suffix rules dictionary and finds which suffix rule transformation has the longest length satisfaction with the given test term. The pseudo-code is described below.

```
def generate_inflection(lemma, infl_rule, prefix_rules_dict,
suffix_rules_dict):

   1. Prepend '@' and append '$' to lemma, call it base.
   2. If the infl_rule is not in either prefix or suffix rules dicts, return
      original lemma
   3. Get all the rules for this infl_rule from suffix_rules_dict, such that
      the lemma_suffix is there in the key of the suffix dict.
   4. Sort the rules – first by length of the match of the lemma suffix, then
      by the count of this rule change, and finally by the length of the
      changed suffix. <This is tie-breaking strategy>.
   5. Find best suffix rule.
   6. Do same for prefix. For prefix, only see the most frequent prefix rule
      for the given infl_rule.
```

These functions have been implemented using [1] as reference. The performance is given in the Table 1.

Table 1: Task 1 (Baseline)

| Lang ID | Language | Accuracy on Low | Accuracy on Medium |
|---|---|---|---|
| L00 | English | 73.2% | 90.3% |

This system will work as a basic backbone for the next tasks as well. In Task 2, this system would be improved upon to work better specifically for the assigned language: Faroese. In Task 3, a similar system would be used to predict the inflection rules given the lemma and a target inflection.

# 3. Task 2

The following task was to build on the current system on the language assigned to our group: Faroese. First, we tried running the baseline code from Task 1 to see what sort of performance that gives on the language. This is seen in Table 2.

Table 2: Task 2 (Baseline)

| Lang ID | Language | Accuracy on Low | Accuracy on Medium |
|---|---|---|---|
| L02 | Faroese | 33.90% | 60.10% |

Next the goal was to try and improve the performance of this system. For this we tried to implement some additional grammar rules as seen from [2], [3], [4].

## 3.1. Grammatical Features

As seen in [3] and [4], Faroese grammar has certain distinct classes for each of Nouns, Adjectives and Nouns. E.g. Nouns have strong and weak classes. Each of them certain sub classes as Class 1, 2 and so on. Each of these classes have certain rules for certain suffix ends. E.g. If the inflection rule is N;DEF;ACC;SG, and the noun ends in 'ur$', that is if the lemma is a noun in definite accusative case, and singular – then the following rule is always applied: 'ur$' > 'in$'. This is for Strong Masculine Nouns. Similarly, for Weak Class 4 of Verbs, inflection rule being V;IND;PRS;2;SG, that is Verb Indicative sense in Present tense (2nd Person Singular), then apply 'gva$' > 'rt$'. Also, in the stem, 'ó' > 'ø' or 'ú' > 'ý' change is done. The characteristic of these verbs is that the verbs would always end with 'gva$'. Simillarly, we also did for adjectives. E.g. for adjective 4th class, for the inflection rule ADJ;NOM;MASC;PL that is, adjectives used in nominative case, masculine and plural, the adjectives (these always end in "in$") always have this transformation -- "in$" > "nir$". These are applied in the function **generate_inflection(…)** function.

```
def generate_inflection(lemma, infl_rule, prefix_rules_dict,
suffix_rules_dict):

   1. If infl_rule is for V (or, N or ADJ):
         a. If it belongs to a certain class (depending on ending, or
            for special set of terms):
         b. Apply rule according to the Faroese grammar rules. Return.
   2. Otherwise, go through the dictionary of rules and try to find a
      match.
```

We have created additional feature function in form of strict conditions for:

- Strong Masculine Nouns
- 2nd Class Masculine Nouns
- 4th Class Adjectives
- 1st Class Adjectives
- Weak Verb Classes 1, 2, 3, 4

We provide a certain example for Weak verb class 3 on how we transform an Inflection Table into a set of if-else conditions.

```
1. If inflection rule is for Verb:
2. If the term ends in 'gva$': #Weak Class 4
      a. <Set of if-else conditions for every possible inflection
         rule possible for a verb of this class according to the
         grammar table>
3. Else if term ends in 'ggja$': #Also Weak Class 4
      a. <…same…>
4. Return if a change was applied
```

A verb term ending in 'gva$' or 'ggja$' indicates a Weak Class 4 verb in Faroese grammar, and depending on that ending they not only have certain changes in the suffix, but also **vowel changes** in the stem, which can often be not learnt properly if we don't have enough data. Adding such additional conditions and features helps the system in being able to handle them well.

The Table 3 shows the performance with the changes. As we see, we see an improvement of 8.90% for low training data, and 2.80% for medium training data.

Table 3: Task 2 (Improvements)

| Lang ID | Language | Accuracy on Low | Accuracy on Medium |
|---|---|---|---|
| L02 | Faroese | 42.8% | 62.9% |

# 4. Task 3

This task is the reverse of the previous tasks, so, given the original lemma and a target inflection, the system is to predict the correct inflection rule. The basic system is as follows.

```
1. Go through training file, maintain a list of all the inflection
   rules seen. Also generate the suffix and prefix rules
   dictionaries.
2. Set correct_guess = 0, num_test_cases = 0
3. For each line in test file:
      a. For each rule in rules_list:
            i.  If lemma + rule -> target (using the Task2 system)
                equals the correct_target and rule == correct_rule,
                then correct_guess += 1
      b. Set num_test_cases += 1
4. Print accuracy
```

To try and improve the performance, we also increase the list of seen rules with whatever feature function rules we added in Task 2, and used the improved Task 2 functionality. Table 4 shows the performance with the baseline system, and Table 5 shows the same with the improved Task 2 system. As we see, we get a 9.6% increase in performance for low data and 3.5% for medium data.

Table 4: Task 3 (Baseline)

| Lang ID | Language | Accuracy on Low | Accuracy on Medium |
|---|---|---|---|
| L02 | Faroese | 31.40% | 59.40% |

Table 5: Task 3 (With Improved Task 2)

| Lang ID | Language | Accuracy on Low | Accuracy on Medium |
|---|---|---|---|
| L02 | Faroese | 41.00% | 62.90% |

# 5. Conclusions

We conclude that the system presented here is able to predict the inflections given a lemma and inflection rules. It performs particularly well for English (Task 1). We try to add certain features and conditions to help improve the performance for the specific language of Faroese, and we see slight improvements when we apply those changes on the validation data. Finally, we also try to do the reverse task of trying to predict the inflection rules given inflection and lemma (Task 3). We see that using the added features of Task 2 helps in performance for this task for Faroese language.

# 6. References

[1] https://github.com/sigmorphon/conll2018/

[2] https://en.wikipedia.org/wiki/Faroese_language/

[3] https://en.wikipedia.org/wiki/Faroese_grammar/

[4] "Faroese: A Language Course for Beginners, by Hjalmar P. Petersen and Jonathan Adams"