Using G2P for Morphological Inflection

Max Szostak
Harvey Mudd College
mszostak@hmc.edu

Anirudh Satish
Harvey Mudd College
asatish@hmc.edu

Amy Tam
Harvey Mudd College
atam@hmc.edu

Abstract

We create a model for the 2018 CoNLL-SIGMORPHON Shared Task: Universal Morphological Reinflection. We reconstruct an approach that uses a grapheme to phoneme conversion tool, Phonetisaurus, to train a WFST that translates lemmas to their inflected forms. We experiment with and compare model performance across different methods for compressing the morphological tags. Our best results were obtained by compressing morphological tags into UNICODE character representations. We analyze the causes of differences in performance across compression methods, concluding that the best performing methods balance amount of compression and lost information.

1 Introduction

This project is based on solving a shared task from CoNLL: "Universal Morphological Reinflection" available on Github [1]. For this project we will focus only on the sub-task which asks participants to inflect word forms based on labeled examples. In English, an example of inflection is the conversion of a lemma *run* to its present participle, *running*. The model is provided with the source form and the morphological features of the target form, and is asked to predict the target form. Real-world applications of this task include machine translation (Minkov et al., 2007) and language learning tools (Shaalan, 2005).

We reconstruct an approach to the shared task that uses Phonetisaurus, a tool for grapheme to phoneme (g2p) conversion, to train a WFST to translate lemmas to their inflected forms (Alegria and Etxeberria, 2016). We use the dataset provided with the shared task [2]. It contains baseline, training, and test data, with files corresponding to each of 70+ languages. Each line within a file represents one inflection, which is the following three elements (tab-separated): source form, target features, and target form. Our main goal is to compare model performance across different methods to compress the target features in pre-processing, and investigate why it performs the way it does. We evaluate the performance of different models using the metrics provided for the shared task: accuracy (fraction of correctly predicted forms) and average Levenshtein distance between predicted forms and true forms.

[1] `https://sigmorphon.github.io/sharedtasks/2018/`

[2] `https://github.com/sigmorphon/conll2018/tree/master/task1`

2 Related Works

Morphology is the study of a word's subcomponents (morphemes), and has been studied as a subset of linguistics (Spencer and Zwicky, 1988). In this task, we wish to use a lemma's morphemes to reconstruct its other forms. There are many existing approaches to morphological reinflection that constitute the submissions to this shared task. We've narrowed down these approaches to two main categories: those that leverage neural networks, and those that use other tools, which include simple machine learning approaches. The best-performing submissions to this task implement deep learning, for which neural networks form the foundation (Makarov et al., 2017). However, we will be implementing a model that does not use deep learning, as it is easier to implement and analyze.

Approaches that do not use deep learning include CRF-based models (Liu and Mao, 2016) and grapheme-to-phoneme models (Alegria and Etxeberria, 2016), which is what we will be implementing. The task of grapheme-to-phoneme conversion involves converting a word's written form (grapheme) to its spoken form (phoneme). G2p models are widely studied in a variety of fields, including low-resource languages (Deri and

Knight, 2016). The task submission that used this approach applied an open-source Weighted Finite State Transducer (WFST) model developed for grapheme-to-phoneme conversion (Novak et al., 2012). This WFST tool was repurposed for morphological reinflection by replacing the phoneme labels with the desired word forms (inflections).

3 Methods

Our dataset is the one provided with the 2018 CoNLL-SIGMORPHON shared task (Cotterell et al., 2018). It contains data on approximately 100 languages, and each language contains training, test, and development files. The training files are labeled as `high`, `medium`, or `low`, based on the number of examples in the file. We filter out this data to only use languages with training data labelled `high`, which are files that have 10,000 examples each, to yield a subset of 76 languages to train on. We then train one model on **each** language, and test each model with the corresponding test set for that language. These test files contained either 1,000 or 100 examples each. We did not use the development files because we are not tuning hyperparameters.

The training and test examples each contain a lemma, inflected form, and morphological features (tab-separated), for example:

```
lyp lypkëshin V;3;PL;ADM;IPFV
```

During the testing phase, only the lemma and morphological features are provided to the model.

We explore an approach to the shared task that does not use deep learning. Specifically, we follow a 2016 submission that re-purposes a tool developed for grapheme to phoneme (g2p) conversion (Alegria and Etxeberria, 2016). We replicate one of their methods on a newer data set (2018 shared task), which has more data, and is formatted differently. This creates a challenge when it comes to incorporating the morphological features along with the lemma, which we address below.

The g2p tool that we are using is Phonetisaurus, an open-source WFST-driven phonology tool (Novak et al., 2012). Phonetisaurus, like many g2p tools, breaks the task of g2p into three steps: sequence alignment, model training, and decoding. See the source paper for more details on Phonetisaurus.

Since the Phonetisaurus model works by taking in a string (designed to be a grapheme), and outputting its corresponding phoneme, we must find a way to combine a lemma and its morphological features into a single training input. Following (Alegria and Etxeberria, 2016), we attach a condensed string containing the morphological features as both a prefix and suffix of the lemma, e.g.

```
<tags><lemma><tags>
```

A challenge we face is finding a way to condense the morphological features into a short string (ideally a few characters) so that our combined string is still mostly made up of the lemma, while still retaining the information of the morphological features. In other words, we do not want the the lemma itself to be lost in the morphological features so that our lemma and morphological features could be seperately and correctly identified. However we cannot employ the same method used by (Alegria and Etxeberria, 2016) and expect similar results because their method (for the 2016 task) depends on the training data which is organized in a different way than it is for the 2018 task.

To address this challenge, we use three different strategies (along with a baseline) and compare them to see which performs the best. This comparison also provides insight into whether the length of the string of morphological features contributes negatively to the performance of our model. In each case, we concatenate the compressed string onto the beginning and end of the lemmas to create the final training instance.

For our baseline, we removed the morphohogical tags altogether. Applied to the previous example, our training data for the baseline looks like:

```
lyp
```

For our first compression strategy, we do not use any form of compression, and use the morphological features from our data set as is. For example,

```
V;3;PL;ADM;IPFVlypV;3;PL;ADM;IPFV
```

For the second strategy, we map each morphological feature to a UNICODE character and remove the semicolons. Due to the vast number of UNICODE characters and the smaller set of morphological feature tags, we are able to compress the long string of features into a string that only uses one character per tag, without losing any information. For example:

```
01234lyp01234
```

In the above example, we replaced the UNICODE characters with numbers to better render this PDF.

	Accuracy (%)	Avg. Lev.
Baseline	4.66	7.66
Uncompressed	6.60	3.39
UNICODE	**9.01**	**3.26**
Buckets	6.13	10.51

Table 1: Model performance for baseline (no morphological features) and three feature compression strategies. The buckets method uses a bucket size of 6 for the morphological tags and 12 for the lemma.

	Predictions made (%)
Baseline	71.7
Uncompressed	73.2
UNICODE	74.1
Buckets	81.2

Table 2: Percentage of total instances for which the model was able to make a prediction, for each of the three compression strategies and the baseline. The buckets method uses a bucket size of 6 for the morphological tags and 12 for the lemma.

For our third strategy, we fix a number of characters for morphological features (after removing semicolons), and truncate any other features that come after. We believe that minimal information will be lost because we suspect that the morphological feature string has tags that happen to appear in decreasing order of importance from left to right. We also determine a fixed number of characters for the lemma to ensure that it makes up a sufficient amount of the information in the in the concatenated string. If the lemma is shorter than this fixed length, we simply repeat the lemma to fill in the required space. For example, using a bucket size of 6 for the morphological features and 12 for the lemma, this method yields

```
V3PLADlyplyplyplypV3PLAD
```

We evaluate each model on two metrics. The first is accuracy, which is the fraction of test instances that were correcty predicted (the model was unable to provide predictions for some instances, which we counted as incorrect). The second metric is the Levenshtein distance between the predicted and correct inflected forms, averaged over all instances. Levenshtein distance is the minimum number of character changes to convert one string into another (so a lower distance means better performance).

4 Results

Our results are summarized in Table 1 and Fig. 1. Of our three compression approaches to morphological tags (uncompressed, UNICODE, and buckets), we found the UNICODE compression provided the most accurate results on average (9.01%) with the smallest average Levenshtein distance (3.26).

5 Discussions

Two factors influenced the performance of each compression method: amount of compression and information lost. Each method involves a tradeoff of these two factors. If too much information is lost, the model does not have enough information to make a good prediction, or any prediction at all. If there is not enough compression, the lemma will get lost in the tags and the model will not be able to differentiate between tags and lemma.

The baseline approach maximized the amount of compression (the tags were removed entirely) but also maximized the amount of information lost (the model had no access to the tags). On the other extreme, the uncompressed tags minimized the amount of information lost (the entire set of tags was retained) but also minimized the amount of compression (the tags were not compressed at all).

The UNICODE strategy worked best because it managed to balance the two factors. No information was lost, because all the tags were still in place (they were just converted to a different syntax). But this strategy also managed to compress the morphological tags by a large margin: UNICODE compression decreased the size of the folder containing all training and test data for all languages by 10 MB, from 38.6 MB for no compression to 28.6 MB for UNICODE compression. As a result, the model is able to easier distinguish the lemma from the tags but still has access to all the information that was originally provided.

The buckets compression method did not work as well as UNICODE compression because it did not manage the tradeoff as effectively. The intent was to force a fixed information ratio of lemma to tags, but this proved to be less effective than the more flexible UNICODE compression scheme. We suspect this is due to the fact that the lemma duplication actually made it more difficult for the model to distinguish the lemma, since there are multiple copies of it next to each other.

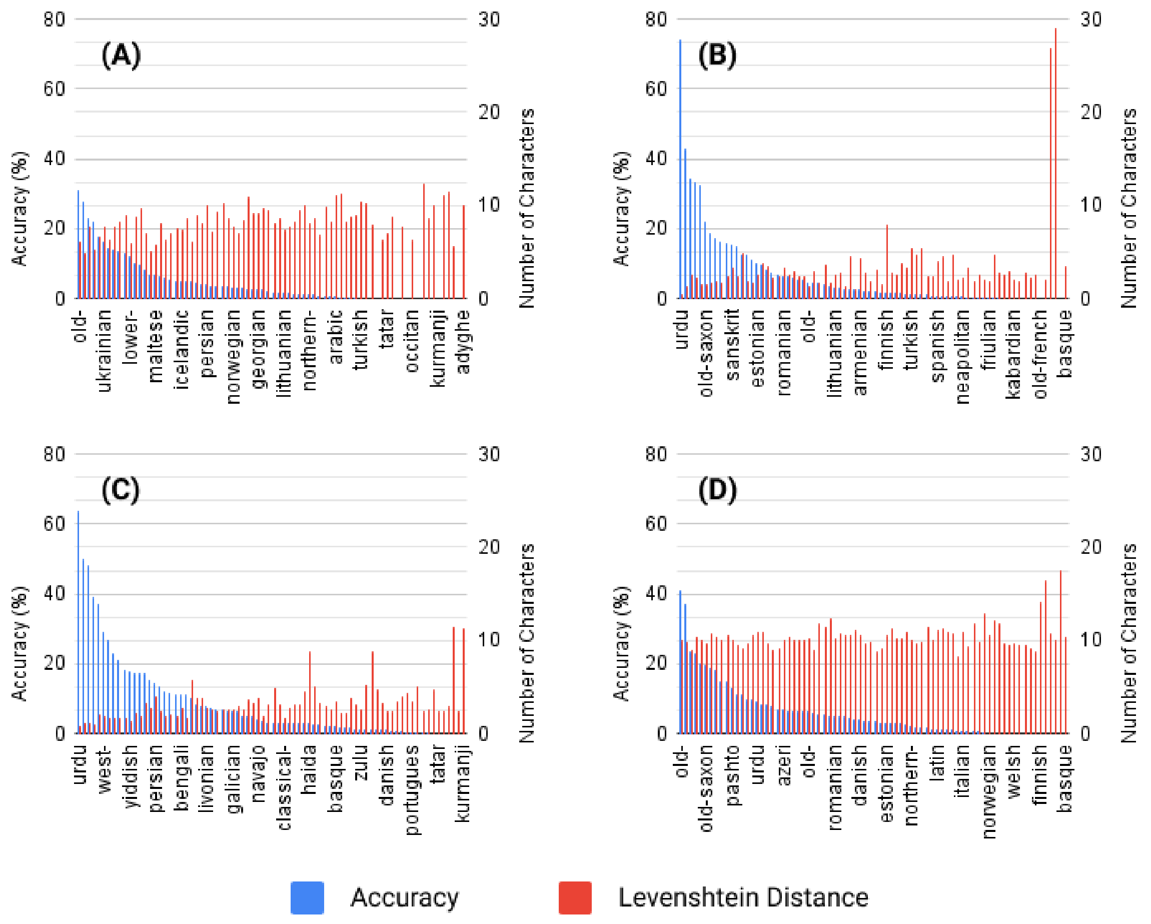
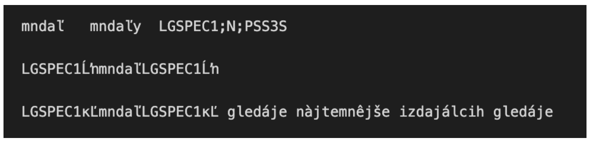


Figure 1: Graphs A, B, C, and D show our baseline, uncompressed, UNICODE, and buckets results respectively evaluated on the accuracy and Levenshtein Distance metrics. Note that the x-axis only shows a few of the languages, as we could not show all languages in a smaller graph, but that you can view more detailed information here.[3]

We also suspect that the poor Levenshtein distance of the buckets method is due to the high number of predictions made; see Table 2 (the g2p model does not predict anything for some inputs). Relative to the other models, the buckets compression model makes the highest number of predictions, even though the accuracy is second to last. As a result, it is making many predictions even though those predictions are often incorrect, leading to a high average Levenshtein distance. In comparison, the other models that make less predictions have a lower average Levenshtein distance, because if the model does not make a prediction on an instance then that instance does not contribute to the average distance. To summarize, the buckets model is more confident in incorrect predictions, while the other models have a greater tendency to withhold predictions entirely, leading to the disparity in average Levenshtein distances. We believe this overconfidence is due to the regularized nature of the bucket-compressed training instances, which likely appear to the model as more consistent than the instances from other compression methods. Further investigation is needed into this phenomenon.

We noticed that our model performed quite well on the languages Urdu, Hindi and Sanskrit. We hypothesize that the reason for this is related to the robust and straightforward inflection rules that these languages share amongst each other. These three languages are very similar in their phrases and the way they are written and spoken as well. For instance, Hindi and Urdu are common languages in both India and Pakistan, and people converse fluently despite being exposed to only one of the languages. Furthermore, Sanskrit and Hindi share the same root script, the Devanagari script, from which numerous languages are written, including the previous pair. It would be interesting to look



```
mndaľ   mndaľy  LGSPEC1;N;PSS3S

LGSPEC1Ĺŉmndaľ LGSPEC1Ĺŉ

LGSPEC1κĽmndaľLGSPEC1κĽ gledáje nàjtemnêjše izdajálcih gledáje
```

Figure 2: An example of a failure in our pre-processing for a test instance in the Sorani language. The first line is the original training instance as it appears in the shared task dataset. The second line is the compressed version of the lemma and its tags. The third line is the model's prediction.

for and understand other languages that are related to each other and share similar performance in this task. This would help us better understand how the g2p model, and others alike, could be improved for morphological inflection and other tasks related to Natural Language.

A final note we make on our results is regarding the outliers appearing in Fig. 1(b). In particular, the languages Sorani and Kurmanji have an average Levenshtein distance of above 20 characters when evaluated on the model that uses the UNICODE compression method. The extremely high Levenshtein distance is due to the model outputting very long predictions, such as Fig. 2. We believe this is caused by a failure in our pre-processing, specifically the UNICODE compression scheme. These two languages use a modified Arabic alphabet, but the test and training instances for this language in the shared task dataset use a Romanized alphabet (see Fig. 2 for an example). It is unclear why, as the training instances for Arabic use the Arabic alphabet. In any case, the use of a Romanized version of the Arabic alphabet led to unusual UNICODE characters in the lemmas and inflected forms. We believe these UNICODE characters overlapped with the characters being used for the morphological tags, leading the model to be unable (or struggling) to differentiate between the lemma and the tags. Since the tags added a considerable length to the string, and (we suspect) that the model considered both lemma and tags together, it outputted correspondingly long predictions. This resulted in the abnormally high Levenshtein distance. This problem can be resolved by using a more careful UNICODE mapping, perhaps starting our pool of mapped characters at a higher-numbered UNICODE character to sidestep the ones being used by these languages.

6 Conclusion

In this morphological inflection task, we were able to reconstruct a grapheme-to-phoneme approach to translate lemmas to their inflected forms for different languages.

To optimize, we tested different strategies for storing morphological feature tags. By mapping each morphological feature to a UNICODE character, we were able to compress the string to achieve our best accuracy of 9.01% with an average Levenshtein distance of 3.26.

The main takeaway of the analysis of our results was that different pre-processing schemes can drastically affect the outcome of the model, and if not properly implemented, can cause the model to fail completely (Fig. 2).

Future work on this project includes efforts to improve the performance of our g2p model in order to approach the results achieved by other shared task submissions. Improved compression models may help; for example, a compression scheme that uses a boundary character on either side of the lemma to help the model distinguish it from the tags. Additionally, further research includes examining the effects of a language's morphological structure on the model's performance. We believe that such research on other languages is really important to expand the use cases of Natural Language tools, that remain closed in on English and alike languages today.

References

Iñaki Alegria and Izaskun Etxeberria. 2016. Ehu at the sigmorphon 2016 shared task. a simple proposal: Grapheme-to-phoneme for inflection. pages 27–30.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Arya D. McCarthy, Katharina Kann, Sabrina J. Mielke, Garrett Nicolai, Miikka Silfverberg, David Yarowsky, Jason Eisner, and Mans Hulden. 2018. The CoNLL–SIGMORPHON 2018 shared task: Universal morphological reinflection. In *Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*, pages 1–27, Brussels. Association for Computational Linguistics.

Aliya Deri and Kevin Knight. 2016. Grapheme-to-phoneme models for (almost) any language. pages 399–408.

Ling Liu and Lingshuang Jack Mao. 2016. Morphological reinflection with conditional random fields and unsupervised features. In *Proceedings of the 14th SIGMORPHON Workshop on Computational*

Research in Phonetics, Phonology, and Morphology, pages 36–40, Berlin, Germany. Association for Computational Linguistics.

Peter Makarov, Tatiana Ruzsics, and Simon Clematide. 2017. Align and copy: UZH at SIGMORPHON 2017 shared task for morphological reinflection. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 49–57, Vancouver. Association for Computational Linguistics.

Einat Minkov, Kristina Toutanova, and Hisami Suzuki. 2007. Generating complex morphology for machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 128–135, Prague, Czech Republic. Association for Computational Linguistics.

Josef R. Novak, Nobuaki Minematsu, and Keikichi Hirose. 2012. WFST-based grapheme-to-phoneme conversion: Open source tools for alignment, model-building and decoding. In *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, pages 45–49, Donostia–San Sebastián. Association for Computational Linguistics.

Khaled F Shaalan. 2005. An intelligent computer assisted language learning system for arabic learners. *Computer Assisted Language Learning*, 18(1-2):81–109.

Andrew Spencer and Arnold M. Zwicky. 1988. *The Handbook of Morphology*. Bantam, London.