Hybrid Distance-based, CNN and Bi-LSTM System for Dictionary Expansion

Béla Benedek Szakács and Tamás Mészáros

Abstract— **Dictionaries like Wordnet can help in a variety of Natural Language Processing applications by providing additional morphological data. They can be used in Digital Humanities research, building knowledge graphs and other applications. Creating dictionaries from large corpora of texts written in a natural language is a task that has not been a primary focus of research, as other tasks have dominated the field (such as chat-bots), but it can be a very useful tool in analysing texts. Even in the case of contemporary texts, categorizing the words according to their dictionary entry is a complex task, and for less conventional texts (in old or less researched languages) it is even harder to solve this problem automatically. Our task was to create a software that helps in expanding a dictionary containing word forms and tagging unprocessed text. We used a manually created corpus for training and testing the model. We created a combination of Bidirectional Long-Short Term Memory networks, convolutional networks and a distance-based solution that outperformed other existing solutions. While manual post-processing for the tagged text is still needed, it significantly reduces the amount of it.**

Index Terms—**machine learning, convolutional neural network, bidirectional LSTM, Levenshtein-distance, dictionary.**

I. Introduction

THE task of creating a dictionary from a corpus is a complex one that requires a lot of manual labour without a sufficiently accurate automatic tool, and even with that some amount of manual post-processing is still needed, as most solutions do not provide 100% accuracy.

Automatic dictionary expansion has been a task used in various fields [1] such as biomedical data [2]. Sometimes a human-in-the-loop approach is applied [3], somewhat similar to what our research led to.

One-language dictionaries such as Wordnet [4] have been used extensively in NLP (Natural Language Processing) research. They can provide information about the text's vocabulary, can serve as a basis for knowledge graphs and other applications.

This paper was submitted on 2020.09.29.

Béla Benedek Szakács is with the Budapest University of Technology and Economics, Budapest, Hungary (e-mail: benedek.b.szakacs@gmail.com).

Tamás Mészáros is with the Budapest University of Technology and Economics, Department of Measurement and Information Systems, Budapest, Hungary (e-mail: meszaros@mit.bme.hu).

It is important that this task is analogous to stemming or lemmatizing, but that only covers part of the problem: there are headwords that have multiple meanings in a language, and a proper dictionary expansion tool should be able to decide between them in addition of finding the correct headword form.

These problems increase when we are dealing with non-conventional texts: either in languages that does not have a wide array of tools and research in terms of NLP or texts in significantly different dialects (old texts or texts of highly specific environments, such the language of online communities). In these cases, previously used algorithms and tools will provide results that will be too inaccurate for any applications.

If there is a sufficiently large corpus of text from the specific dialect we are focusing on *and* it is processed by hand, it can be enough to teach some kind of model on it. This was our approach in this case: we were trying to develop a software specifically tailored to expand an already existing dictionary in a specific format. In this case, we had two main tasks to solve: looking at a word, we had to find the corresponding form in the dictionary, or the headword if the form does not exist, and if there are multiple forms, decide which one is the most likely based on the context.

II. The Mikes Dictionary Project

This is a project [5] that was created by the Hungarian Research Center for the Humanities, and the main purpose is to create a full author's dictionary [6] based on the work of Kelemen Mikes, an 18th century Hungarian writer, who had a large body of work comprised of mostly prose and letters. The researchers will be using this dictionary for a multitude of analytical experiments in the field of Digital Humanities [7].

A. The Corpus

Kelemen Mikes was a very influential writer in the 18th century, and his work is still extensively studied. The language of Mikes is, however, very different from contemporary Hungarian: the grammar is much more inconsistent, he uses a lot of Latin words and expressions. The dialect which he uses is mostly understandable by a contemporary reader, but only because of the flexibility of the human mind.

This also means that most tools developed for processing contemporary Hungarian are significantly less accurate on these texts, so we had to create a different solution.

DOI: 10.36244/ICJ.2020.4.2

B. Manual Work

We have been involved with other experiments concerning Mikes's texts before, and it has been fully digitalized with the proper notations. The dictionary project, however, is only partially done.

The current state of the dictionary was created manually by linguists at Hungarian Research Center for the Humanities. This was an enormous task, even for one part of the whole Mikes corpus, called the "Turkish Letters", a collection of 207 letters (~106 000 words). They are only a fraction of Mikes's complete work, but a large enough body of text to use as a solid basis for training algorithms.

Because of the heavily time-consuming nature of the manual work, our task was to significantly increase its speed by developing an automatic tagger software that would predict the headwords for the words, and afterwards a researcher would correct the mistakes manually. With even a moderately high accuracy this would significantly increase the speed of work on the corpus (we are talking about years of manual labour). This human-in-the-loop approach can be compared to other machine-assisted manual works, such as Alba et. al. 2019 [8] or Ruis et. al. 2020 [9].

C. Automatic Dictionary Expansion

The manual work on the dictionary, even with the help of a simple software that allows fast tagging of words with dictionary entries and offers help based on purely by the existing dictionary, is a very time-consuming task, requiring significant expertise in linguistics. This means that to meaningfully increase effectiveness, the software should contain an automatic tagging tool.

The goal is to process the entire unprocessed corpus, and one of the most important inspirations for developing this software was to help this work, creating some kind of pre-processing application that allows linguists to quickly decide whether the tagging created by the software are correct or not, and make it accurate enough that it provides sufficient help. We focused on developing the algorithmic part of this task, the problem of automatically tagging words with predicted dictionary entries.

This is a very difficult task mostly because of the language used: even the linguists doing the manual work have difficulties quickly discerning the headword because of the archaic forms and the foreign (mostly Latin) words. This means the task of doing it automatically was expected to be a highly complex task. We should not expect as high accuracy scores as with state-of-the-art tools on contemporary texts. The goal was rather to achieve a sufficient score that provides significant help as pre-processing.

It is important to mention that there are words with multiple parts that can be separated in the text but should be recognized as one word. The recognition and categorization of these words is a very difficult task that this solution is not able to solve, so words with multiple parts are not recognized as one word, diminishing the accuracy of the tool. This is a known shortcoming of this solution.

There are also headwords that are identical in form but carry different meanings. Since in our pre-existing corpus, the example sentences are separate for each headword, we treated these similar headwords as completely different entities. This also means that tools not using the context of the word will not be able to identify it correctly.

III. EXISTING SOLUTIONS

While we expected other tools developed mainly for analysing contemporary texts to underperform, we have nevertheless inspected an array of other tools commonly used in Hungarian NLP tasks.

We were not trying to take all tools into consideration, only a select few. We were trying to use software from widely different backgrounds: state-of-the-art solutions as well as old but reliable ones.

We performed experiments with four solutions that can be divided into three groups: state-based, reliable solutions, out-of-the-box, performance-focused tools, and state-of-the-art models.

With this, the four solutions we have chosen are the EmMorph morphological analyser [10], the SpaCy NLP pipeline, the BERT [11] (Bidirectional Encoder Representations for Transformers) and Flair [12] models, using the Flair framework.

A. EmMorph

This tool was created by Attila Novák [10], and it is a finite state machine-based tool trained on contemporary Hungarian. Because of this, it is an extremely accurate and reliable tool, as it is not a probability-based model.

It uses an "Item and Arrangement" (IA)-style analysis, so the input word is analysed as a sequence of morphs, where each form is a specific realization (an allomorph) of a morpheme. This means that the EmMorph does a very detailed analysis of each word, providing a lot of morphological information.

Because this solution does not use any form of probability, it relies on its database for every word analysed. This means that new word forms will be unrecognizable to it, rendering it a lot less effective in our case: the text from Mikes contains a lot of word forms not used in contemporary Hungarian.

B. SpaCy

SpaCy[1] is an industrial-strength, out-of-the-box solution in Python for NLP problems. It is designed for production environments, not for experimentation, although it is open source, and has a fair number of contributors, as well as a very flexible architecture that allows easy integration for custom components.

The underlying technology is mostly built on convolutional neural networks, but it uses embeddings and other pre-computing strategies. They do not have a definitive paper that summarizes their methods, instead they have a summary of the technology on their webpage.

[1] https://spacy.io/

SpaCy provides a variety of functions for NLP tasks: it has a lemmatizer, a PoS (Parts of Speech) tagger, a dependency parser that builds dependency trees between tokens in a sentence, an entity recognizer for NER (Named Entity Recongition), a built-in categorizer for text classification tasks, a pattern matcher, and can incorporate custom components. This allows for a variety of configurations based on the task at hand.

SpaCy also has a dictionary-like system that stores lexemes and data about the document's vocabulary, and it is capable of full morphological analysis, including noun cases, verb tenses and others.

C. BERT

BERT [11] is an acronym for Bidirectional Encoder Representation for Transformers. It was developed by Google, mostly for NLP tasks. It is basically a multi-layer bidirectional Transformer, trained on a very large corpus, resulting in a network that can be easily adjusted to any NLP task using just an extra layer and some fine-tuning. It relies heavily on the concept of transfer-learning, the concept of using a pre-trained model with little training on a specific dataset for a specific task. It is mostly utilized in tasks where training data is scarce or absent. For NLP, this means that BERT was trained extensively on a huge multilingual corpus unsupervised, and so it learns a lot of the characteristics of the language, making fine-tuning a lot faster and less data-extensive.

The main improvement from the precursor model is that they use bidirectional unsupervised learning. This allows it to be successful at a large variety of uses, including both token level and sentence level tasks. This bidirectional training relies on a method called MLM (masked LM) as to not run into the problem of the words "seeing themselves" (the word that needs to be predicted is present for the opposite direction, making the task trivial), by randomly masking words in both directions and trying to predict them.

D. Flair

The name Flair [12] is used for multiple things: it is both an NLP library (including a data library and pre-trained models for a variety of tasks), built on PyTorch, and an embedding model.

The Flair framework is designed to make using big, complex models very simple. It is a wrapper over PyTorch, one of the most widely used machine learning libraries for Python, and it makes creating for example, a BERT model for text classification extremely simple. It also has a variety of pre-trained networks for the most common tasks, such PoS tagging.

Flair itself is a character-level recurrent network using contextual string embedding, usually fed into a Bi-LSTM-CRF (Bidirectional Long-Short Term Memory, Conditional Random Fields) model. It is currently the best solution for PoS tasks, as it outperforms every other approach, including the previously mentioned BERT. However, it was specifically designed for sequence tagging, not for more complex tasks (although it can be used in other models designed for different tasks, as it is only an embedding).

IV. THE MODELS

Both of our main tasks (finding the correct headword for unknown word forms and discerning the correct headword for unambiguous word forms) are essentially categorization tasks. We have focused on the first one, as doing it correctly technically includes the second one as well. This means that the output of the system should be one of the existing dictionary entries, whereas the input should be the word and some of its context.

Because of the strict form of example sentences (31 words, the middle one is the target for tagging), we decided for a fixed-length input, not a sentence-based one. This still allows for the procession of sentences, with the use of padding tokens, and the length of the input means that most sentences will be inside its bounds. While this special format made the system theoretically suboptimal, back-conversion of the dataset was practically impossible, but this format still allowed for keeping most of the word's context.

For the models themselves we focused on two features of the input: the words in the context (31 words) and the middle word's characters (maximum 40, 44 different possible characters). Because in the Hungarian language most words are similar to their headwords, and a character-level model is a great solution in looking for it.

The architectures we have chosen were the one-dimensional convolutional neural networks [13] and the Bi-LSTM [14]. Both of these have been extensively used in NLP tasks. In our setup, we have used two models, trained and evaluated separately: one using both character-level and word-level convolutional networks, and a CNN-Bi-LSTM solution using convolutional network for the character-level input but Bi-LSTM for the word-level input.

We have also experimented with a pure Bi-LSTM solution, but it underperformed compared to the CNN-Bi-LSTM solution and was deemed too similar to the CNN-Bi-LSTM solution to be used alongside it (more information about it can be found in the Training and Evaluation chapter).

A. Embeddings

In both cases, the input words are embedded in a simple 256-dimensional embedding, and the characters are one-hot encoded.

For most NLP applications, pre-trained embeddings are usually a staple. The problems with this approach in this case were that the unique language of Mikes' writing made it impossible to utilize any pre-trained embedding. We could have used embeddings taught on the data itself, but the size of the corpus was not sufficient for this task.

B. Pure Convolutional Model

The model is a straightforward convolutional model, with only one layer of 1D convolution (Fig. 1). Because the size of the training set was not enough for large, complicated language models, we opted for a smaller, simple model.

For optimizing the hyperparameters, we assumed that the parameters themselves can be independently optimized. This approach was necessary due to the large number of possible combinations. We selected dropout (from 0.0 to 0.5 with 0.1 increments, dense, convolutional and bi-LSTM layers optimized independently), batch size (values: 64, 128, 256, 512), the type of optimizer (Adam and Nadam), the type of

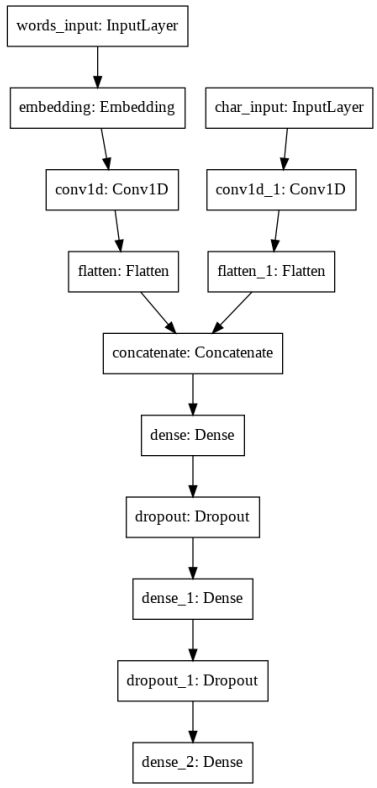

Fig. 1. The layers of the pure convolutional model

kernel initializer for all layers (values: uniform, normal, glorot_uniform, glorot_normal, lecun_uniform) and the size of the embedding layer (values: 64, 128, 256, 512) as optimizable hyperparameters. The best regularizer was Nadam, and the best batch size was 64 for both this and the CNN-Bi-LSTM model.

The optimised hyperparameters of the layers are the following (all other hyperparameters can be found in the Appendix chapter):

1. Embedding layer: input dimension = 31, output dimension = 256, embeddings initializer = ”lecun_uniform”, no regularization, no zero masking, no dropout.
2. Conv1D layers: filters = 29 for conv1d and 30 for conv1d_1 and conv1d_2 in the CNN-Bi-LSTM model, kernel size = 3, activation = relu, no dropout.
3. Dense layers: units = 512, 4096, 15829, activation = “relu”, dropout = 0.3.

The two Conv1D layers are concatenated in a 50-dimensional layer, and then a series of dense layers are responsible for increasing the dimension to the size needed for the output. Because the task is simple categorization, we used a simple softmax function at the end and sparse categorical crossentropy as the loss function. The dimension (15829) of the last layer is equal to the dictionary entries. While this means that subsequent additions to the dictionary means using a completely new model every time, the simplicity of the model means a low number of parameters (88 million), and with that a relatively fast training compared to the current, much larger models (more on that in the Training and Evaluation chapter).

Using convolutional layers in text processing is a commonly used technique ([15], [16]), although mostly used for sentiment analysis or text classification.

C. CNN-Bi-LSTM Model

The CNN-Bi-LSTM model (depicted on Fig. 2, with an example input and output) was inspired by another architecture primarily developed for Named Entity Recognition [17], although it lacks the case-embedding and uses a simple 256-dimensional embedding. It is very similar to the previously described pure convolutional model, the only difference is in the processing of the embedded words.

We used a 64-dimensional Bi-LSTM layer that was then flattened into a dense 64-dimensional layer. This meant that unlike with the pure convolutional model, here the output of the word-processing part of the model was a lot larger. We theorized that this, together with the LSTM being generally more fitted for processing word sequences, will lead to a better accuracy than the pure convolutional model.

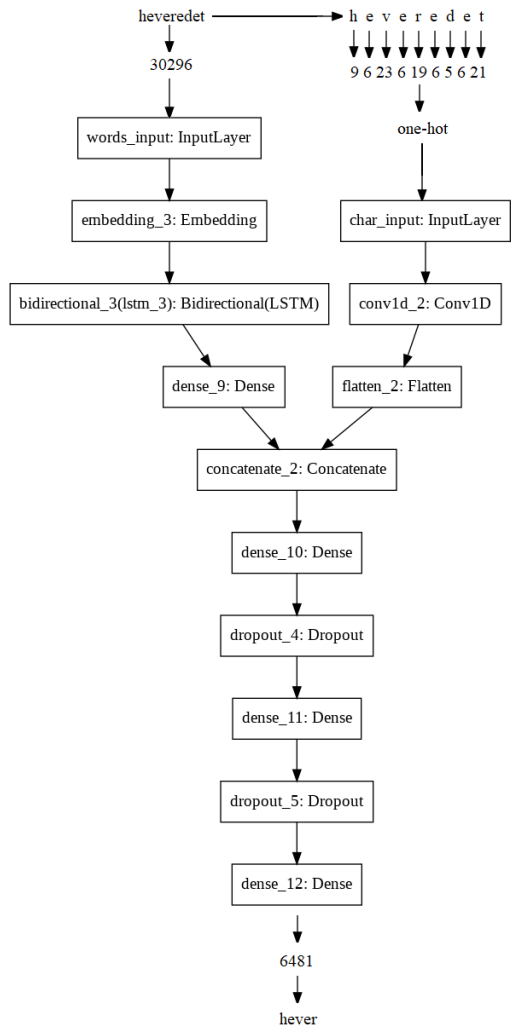

Fig. 2. The layers of the CNN-Bi-LSTM model and an example input and output

The hyperparameters of the Bi-LSTM layer were the following (the hyperparameters of the other layers are the same as the ones in the pure convolutional model, and the other hyperparameters can be found in the Appendix chapter):

Units = 64, activation = "tanh", recurrent activation = sigmoid, use bias = True, kernel initializer = "lecun_uniform", dropout = 0.1.

We expected the CNN-Bi-LSTM network to outperform the pure convolutional model, but also to learn more slowly. Also because of the difference in the word processing architecture, we theorized that the two networks would be better at identifying different words. This led to the final solution combining the result of both models and deciding between them. The problem was that we needed a three-opinion system to use majority voting, so we used a third component, a simple distance-based solution.

D. Levenshtein Distance

The Levenshtein distance is often used in approximate string matching, especially in spell checking, where one of the strings comes from a dictionary. This is somewhat similar to the task of finding a headword, although not equal. We have chosen this as an often used and simple solution in string distance measurement.

Levenshtein distance is a measurement of difference between two strings, based on the number of *edits* that are needed to transform one to another. These edits are: 1. adding a character, 2. deleting a character, 3. substituting a character with another character. This can be calculated very efficiently using a dynamic programming algorithm.

The main issue with using a distance-measurement like this in a dictionary of roughly 16 000 word is that every time we need to do the whole calculation 16 000 times. This, even with a C implementation, takes significantly longer than simply running one word and its context through the models for prediction. So, as we can see, using Levenshtein every time leads to a significant amount of time increase, which, while not necessarily one of the main considerations, is still a factor to keep in mind.

E. The Hybrid System

We have decided to use two models and the Levenshtein-distance as a three-part expert system. Because of the higher computational cost and the distribution of correct guesses (Fig. 3), the Levenshtein-distance was used as a tiebreaker.

The execution was very simple: first, both models predicted a dictionary entry, then if these were not the same, the entry predicted by the Levenshtein-distance was used, regardless of the results produced by the models.

While as we will see on Fig. 4, both the CNN-Bi-LSTM model and the convolutional model outperformed the distance-based prediction, a disparity in the results of the two models usually means they are both wrong, and in this case the distance can be a helpful third option. This is the reason why the Levenshtein distance takes priority over both of them.

It is not trivial that these three solutions complement each other well, but in this application the results show that the system together performs significantly better than the individual components, which can be explained by the varying architectures.

V. Training and Evaluation

To allow for a good evaluation, we have used the following method: we randomly chosen 5% of all known word forms as the test set and excluded them from the dataset on which we performed the training. This meant that the word forms used in testing are analogous to unknown words which the application has to predict headwords for. This set formed our testing set during the evaluation phase.

The training was always performed with 48 iterations, in every iteration a new set of 10 000 sentences were used to train the model with 10% as the validation set. We used early stopping based on validation loss with a patience of 3 and used the best weights.

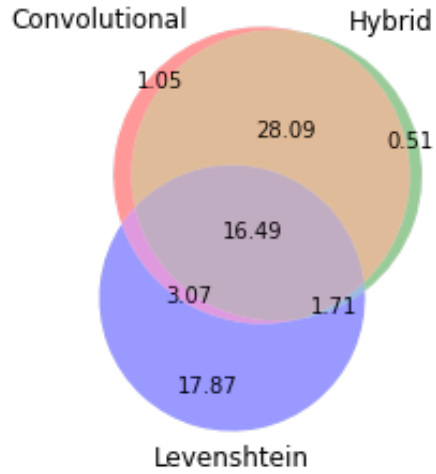

Fig. 3. Accuracy of the combination of components in percentage of accurately guessed headwords using the whole test dataset.

A. Evaluation

Pure Bi-LSTM stands for the previously mentioned model where even the character-level input was fed into a Bi-LSTM network. It did not achieve similar accuracy to the other two models, so it was discarded.

Contrast these results (Fig. 4) with the performance of the Levenshtein-distance-based solution: that achieved 37% on the same dataset. We used it as our baseline, as it is the most basic solution, and it performed remarkably well for its simplicity.

Type	Best Accuracy
CNN-Bi-LSTM	46.8%
Pure Bi-LSTM	37.5%
Pure convolution	48.7%
Levenshtein	37.0%
Hybrid System	65.9%
SpaCy	41.3%
EmMorph	21.9%
BERT/HuBERT	-
Flair	29.7%

Fig. 4. The accuracies of all solutions. BERT was not tested because training was early stopped at 19% accuracy. HuBERT achieved 20% accuracy on the training set.

Contrary to our expectations, the pure convolutional model had slightly better accuracy than the CNN-Bi-LSTM model. This can be explained with the length of the context fed into the input: only 10 other words were used as context, and Bi-LSTM networks are mostly used because of their ability to identify long-term connections. Furthermore, experiments have shown [18] that in certain sequence-labelling problems, CNNs can outperform RNNs. It is also worth mentioning that the gap in accuracy is very small (only 1.9%). Using both was, however, crucial for the system to be able to use majority voting.

The system as a whole achieved 65.9% accuracy all together on the same dataset. This means that the parts of the system do, in fact, perform significantly better together (even the convolutional model which performed the best of all the solutions achieved only 48.7%).

We also experimented with different ways of deciding ties between the components, and found out that the Levenshtein distance was, in fact, the best for this task, despite being the least accurate standalone. Fig. 3 shows that the distribution of correct guesses supports this.

Training times were typically around 3-4 hours for the whole system. All training and evaluation were performed on a PC with a GTX 1060 6GB GPU. With a TPU using more VRAM training these models would take even less time.

B. *Testing on Other Texts*

Whereas the previously mentioned experiments provide a numerical metric, in the case of a tool designed to help manual work, manual experiments were also needed. We have used it on a handful of other texts to manually test its usefulness and if any typical errors are present.

<beszélgetni U:beszélgetni> <beátrixal U:Beatrix> <egyik U:egyik> <a U:az 1> <leányi U:leány> <közül U:közül> <valoval U:ló> <akiben U:a$ki> <leg U:elég$tételi> <több U:több> <bizodalma U:bizodalom> <volt U:van> <ugyan U:ugyan> <orában U:óra> <verték U:vet> <fel U:fel$üttet> <a U:az 1> <házát U:ház> <az U:az 1> <anglusok U:ánglius> <éléonora U:Eleonora> <le U:le$ülve> <heveredet U:hever> <volt U:van>

Fig. 5. The results of testing on a letter from Mikes not from the "Turkish Letters". The red background signals the wrong predictions. The $ symbol is the separator for multi-part words (e.g. "elég$tételi"), and for multi-meaning words, a number is appended (e. g. "az 1")

<augusti U:augustus> <írott U:írott> <levelét U:levél> <kegyelmednek U:kegyelmed> <vévén U:vevés> <az U:az 1> <midőn U:a$midőn> <abból U:az 2> <s U:sok> <mind U:mind 1> <az U:az 1> <includált U:csoda> <levelekből U:levél> <értem U:ért 1> <az U:az 1> <ellenségnek U:ellenség> <kegyetlen U:kegyetlen> <actusit U:acta> <az U:az 1> <dunán U:Duna> <túl U:túl> <való U:való> <földön U:föld> <való U:való>

Fig. 6. The results of testing on a letter from Ferenc Rákóczi the II, a contemporary of Mikes. The red background signals the wrong predictions. The $ symbol is the separator for multi-part words (e.g. "a$midőn"), and for multi-meaning words, a number is appended (e. g. "mind 1")

For experimenting on these texts, we used the hybrid system, including a lookup function that uses the Solr database to look for already existing forms (the Solr query searched for an exact match amongst the word forms, and we grouped them by headword, then in the case of multiple matching headword, used the prediction function). This means that the prediction function is only used in the case of ambiguous or unknown forms.

As we can see on Fig. 5. and Fig. 6., the number of wrong predictions is low, much lower than the measured 34%. This is primarily because these texts contain a lot of words that are already known. The large size of the dictionary allows for more common words with a small number of forms to be easily identified without the uncertainty present in the predictive function. The errors do not show any certain trend (some are similar to the wrongly predicted headword, some are seemingly random, some are the result of the model's inability to correctly identify multi-part words).

C. *Comparison to Other Solutions*

We have performed the same experiment with the dataset used for training and testing on all the previously mentioned other solutions. For both the EmMorph and SpaCy we only used pre-trained models, and for the BERT and Flair we trained multilingual models on the training dataset.

EmMorph performed according to our expectations: with the score of 21.9%, it was the least accurate. It is mostly due to the large number of unknown words and a very different grammar used in Mikes's writing.

SpaCy (using the lemmatizer module, and the lemma of each word as the predicted headword) has surpassed our expectations with its score of 41.3%, as it performed closely to our individual networks. This shows the power of context-analysing (EmMorph only uses the form of the word, SpaCy uses the context as well), and the robustness of its Hungarian models.

We have used a multilingual BERT model as well as the HuBERT [19] model. We used the Flair framework for training and evaluating the BERT implementations, using Flair's own built-in categorizer architecture. We used ADAM optimizer with a learning rate of 0.1 and an annealing factor of 0.5, minibatch size was 32. The multilingual model's raining was stopped at 19% accuracy, and the training of HuBERT was stopped at 20% accuracy. We have not evaluated them on the test dataset.

We used the Flair embeddings similarly to the BERT embeddings, in the same setup. It has performed much better than BERT, reaching 57.7% accuracy during training, and when subjected to the same evaluation as our models, it reached an accuracy of 29.7%. This, although still significantly worse than our models, means that big, multilingual models can be trained for processing unusual language, but custom-built solutions will usually be better.

VI. IMPLEMENTATION

We have used Python for the implementation, mostly because most state-of-the-art machine learning tools are accessible as Python libraries, and it provides an easy and fast way to create a simple application that is capable of tagging

texts. We have uploaded our implementation to GitHub at https://github.com/szakacsb/dictionary_expander, together with links for models and for the xml file needed to initialize the Solr database.

For storing the dictionary, we used Apache Solr[2]. Solr is a powerful search platform with a multitude of functions that made it ideal for quick lookups and storing data in a dictionary-like format. We did not utilize most of its advanced functionality, but it provided a robust out-of-the-box solution for storing data.

A. The Application

The Application itself was written purely in Python, including the parts for populating the Solr server with data, the client querying the server for data, the training and evaluation of models, and the functional tagger. The machine learning parts rely on Keras, and for the distance-based part, we used python-Levenshtein.

The application does not rely strictly on the Mikes dictionary as a corpus: using the same format, any dataset can be used to teach the model. This means that reconfiguring it for different task is fairly simple, be it dictionary expansion for a different corpus or an entirely different entity recognition and tagging task.

B. The Solr Server

While Solr is a much more robust technology than required for this task, its performance is a significant upside for this application. We transformed and uploaded the half-done dictionary and used it as our database server for the experiments.

The results, after being manually checked, can very easily be fed back into the Solr server, making further training of models possible. An incremental workflow can be created, where the application tags the text, the expert manually corrects it, and then it is uploaded into the server, and used for further training for the models.

VII. Conclusion

We have created an expert system-based automatic tagger that can be used to pre-process texts for dictionary-expansion. We have demonstrated that a three-component tool performs better on Mikes Kelemen's writings that are in an archaic dialect of the Hungarian language, and we compared our results to some already existing tools on the same corpus.

Whereas the tool we created was specifically designed for this task, it can be used in many other applications, and its flexibility allows for processing other non-contemporary or otherwise drastically different dialects.

The accuracy of the predictions is not fit for unsupervised dictionary expansion; however, we have reached a 65.9% accuracy on unknown words and this makes this tool ideal for pre-processing texts before manual corrections.

We also built the system into an easy-to-use application, together with a Solr-based server that stores the dictionary itself.

For future works we will be developing the decision-making component, using the posterior probabilities of the softmax layers and trying different, more complex approaches. We will also be looking at more sophisticated distance-based methods and more complex neural networks to try to diversify the components even further.

VIII. Acknowledgments

This work was supported by the European Regional Development Fund of the European Union under the EFOP-3.6.2-16-2017-00013 Project.

References

[1] A. Toprak and M. Turan, "English Automatic Dictionary Creation with Natural Language Processing", *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*, Izmir, Turkey, 2019, pp. 1-6 DOI: 10.1109/ASYU48272.2019.8946431.

[2] X. Wang, Y. Zhang, Q. Li, X. Ren, J. Shang and J. Han, "Distantly Supervised Biomedical Named Entity Recognition with Dictionary Expansion", *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, San Diego, CA, USA, 2019, pp. 496-503 DOI: 10.1109/BIBM47256.2019.8983212.

[3] Gentile A.L., Gruhl D., Ristoski P., Welch S. "Explore and Exploit. Dictionary Expansion with Human-in-the-Loop", Hitzler P. et al. (eds) *The Semantic Web. ESWC 2019. Lecture Notes in Computer Science*, vol 11503. Springer, Cham DOI: 10.1007/978-3-030-21348-0_9

[4] George A. Miller, "WordNet: A Lexical Database for English", *Communications of the ACM Vol. 38*, No. 11, pp. 39-41, 1995 DOI: 10.1145/219717.219748

[5] Margit Kiss, "The Digital Mikes-Dictionary", In: Tüskés Gábor; Bernard Adams; Thierry Fouilleul; Klaus Haberkamm (editor), *Transmission of Literature and Intercultural Discourse in Exile [...] The Work of Kelemen Mikes in the Context of Europen Enlightment [...]*, Bern: Peter, Lang Verlag, pp 288-297, 2012

[6] Tamás Mészáros, Margit Kiss, „The DHmine Dictionary Work-flow: Creating a Knowledge-based Author's Dictionary", *Proceedings of the XVIII EURALEX International Congress: Lexicography in Global Contexts*, pp 77-86, Jul. 2018

[7] Kiss, Margit, Mészáros, Tamás, "Rethinking the Role of Digital Author's Dictionaries in Humanities Research", Feb. 2019

[8] Ruis, F., Pathak, S., Geerdink, J., Hegeman, J. H., Seifert, C., & van Keulen, M. "Human-in-the-loop Language-agnostic Extraction of Medication Data from Highly Unstructured Electronic Health Records", *20th International Conference on Data Mining Workshops 2020* IEEE EDS, 2020

[9] Alfredo Alba, Chad DeLuca, Anna Lisa Gentile, Daniel Gruhl, Linda Kato, Chris Kau, Petar Ristoski, and Steve Welch „Identifying High Value Opportunities for Human in the Loop Lexicon Expansion", *HumBL2019. The third international workshop on Augmenting Intelligence with Bias-Aware Humans-in-the–Loop. In the Web Conference 2019 Companion volume*. ACM, New York, NY, USA, 2019. DOI: 10.1145/3308560.3317305

[10] Attila Novák, "A New Form of Humor – Mapping Constraint-Based Computational Morphologies to a Finite-State Representation." in: *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC-2014)*, Reykjavík, pp. 1068–1073, 2014

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, "BERT: Pre-training of Deep Bidirecional Transformers for Language Understanding", arXiv preprint arXiv:1810.04805, Oct. 2018

[2] https://lucene.apache.org/solr/

[12] Akbik, Alan and Blythe, Duncan and Vollgraf, Roland, "Contextual String Embeddings for Sequence Labeling", in: *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 1638-1649, 2018

[13] Zhang, Wei, "Shift-invariant pattern recognition neural network and its optical architecture" in: *Proceedings of Annual Conference of the Japan Society of Applied Physics*, 1988

[14] Hochreiter, Sepp; Schmidhuber, Jürgen, "Long short-term memory" in: *Neural Computation 9 (8)*, pp 1735–1780, 1997, MIT Press
DOI: 10.1162/neco.1997.9.8.1735

[15] Alexis Conneau, Holger Schwenk, Loïc Barrault, Yann Lecun, "Very Deep Convolutional Networks for Natural Language Processing", Jun. 2016, *arXiv preprint* arXiv:1606:01781

[16] Xiang Yu, Agnieszka Falenska, Ngoc Thang Vu, "A General-Purpose Tagger with Convolutional Neural Networks", in: *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pp. 124-129, Sept. 2017, Copenhagen, Denmark, Association for Computational Linguistics
DOI: 10.18653/v1/W17-4118

[17] Jason P.C. Chiu, Eric Nichols, "Named Entity Recognition with Bidirectional LSTM-CNNs", in: *Transactions of the Association for Computational Linguistics, Volume 4,* pp. 357-370, 2016
DOI: 10.1162/tacl_a_00104

[18] Shaojie Bai; J. Zico Kolter, Vladlen Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling", *eprint arXiv:1803.01271*, March 2018

[19] Nemeskey, Dávid Márk, "Natural Language Processing methods for Language Modeling" PhD thesis. Eötvös Loránd University, 2020

Béla Benedek Szakács finished his BSc in computer engineering in 2018 and is currently doing his MSc studies in the same field at the Budapest University of Technology and Economics. He is a member of the Balatonfüred Student Research Group. His main field of study is machine learning and natural language processing.

Tamás Mészáros is an associate professor at the Budapest University of Technology and Economics. His research areas include intelligent agents, information retrieval and natural language processing.

IX. APPENDIX

The additional hyperparameters of the layers are the following:

1. Conv1D layers: padding = valid, data format = "channels_last", dilation rate = 1, groups = 1, use bias = True, bias initializer = "zeros", no kernel regularizer, no bias regularizer, no activity regularizer, no kernel constraints, no bias constraints.
2. Dense layers: use bias = True, bias initializer = "zeros", no kernel regularizer, no bias regularizer, no activity regularizer, no kernel constraints, no bias constraints.
3. Bidirectional LSTM: use bias = True, recurrent initializer = "orthogonal", bias initializer = "zeros", unit forget bias = True, no kernel regularizer, no bias regularizer, no activity regularizer, no recurrent regularizer, no kernel constraints, no recurrent constraints, no bias constraints, no recurrent dropout, return sequences = False, return state = False, go backwards = False, stateful = False, time major = False, unroll = False.