Chat Language Normalisation using Machine Learning Methods

Daiga Deksne
Tilde, Vienibas Gatve 75a, Riga, Latvia

Keywords: Non-Canonical Language, Language Normalisation, Intelligent Virtual Assistants, Intent Detection.

Abstract: This paper reports on the development of a chat language normalisation module for the Latvian language. The model is trained using a random forest classifier algorithm that learns to rate normalisation candidates for every word. Candidates are generated using pre-trained word embeddings, N-gram lists, a spelling checker module and some other modules. The use of different means in generation of the normalisation candidates allows covering a wide spectre of errors. We are planning to use this normalisation module in the development of intelligent virtual assistants. We have performed tests to detect if the results of the intent detection module improve when text is pre-processed with the normalisation module.

1 INTRODUCTION

Utterances used on social media platforms, chatrooms and internet discussion forums are a valuable source used in the creation of Artificial Intelligence applications (AI). The central task of an intelligent virtual assistant is to understand a communication partner and to detect the user's intent expressed by his/her utterance. Unfortunately, processing conversational text used on the Web with existing NLP tools is not an easy task because conversational text is significantly different from the canonical form of language. Several researchers report improvement of results when they perform normalisation of noisy user generated text prior to part-of-speech tagging (Damnati et al., 2018) or parsing (van der Goot et al., 2017). It is mentioned that two approaches can be used in such a situation: either adapt the tools or adapt the data (Eisenstein, 2013). In our work, we have focused on data normalisation.

In this paper, we describe our experience in the creation of a chat text normalisation model for the Latvian language. This model is based on a random forest classifier algorithm and uses different features for error correction, e.g. word embeddings, n-grams, spell checking suggestions and others.

We would like to fix the gross orthographic and grammar issues while preserving the style of the original text. Our goal is to make the user-created text comprehensible for other readers and valid for further processing.

2 DATA

The chat text may contain two types of errors: unconscious errors (committed either by ignorance or by accident) and deliberate errors. However, we should not refer to the second of type errors as errors; it would be more accurate to call them a derogation from the literary norm, from canonical language.

Normalisation is a subjective process. In *Guideline for English Lexical Normalisation Shared Task* (Baldwin et al., 2015a), annotators were asked to correct English Twitter messages by providing replacements for non-standard words, i.e. misspelled words, phonetic substitutions, informal abbreviations or correct words that are not appropriate for the context. Non-English words, exclamative words, proper nouns and non-alphanumeric tokens were left unchanged. While performing this task, annotators faced difficulties in reaching a consensus on drawing the border between standard and non-standard words (Baldwin et al., 2015b). In our work, we have faced similar difficulties for the Latvian language.

We have manually annotated 2,493 sentences collected from Internet discussion forums and Internet comments and 500 sentences of Twitter data. We have also created a corpus where errors are introduced artificially. This corpus contains around 2,500 entries.

Deksne, D.
Chat Language Normalisation using Machine Learning Methods.
DOI: 10.5220/0007693509650972
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, pages 965-972
ISBN: 978-989-758-350-6

2.1 Corpus Analysis

To define what should be normalised in Latvian chat texts, we started with analysis of the Latvian Tweet corpus (Pinnis, 2018). We tagged unique Tweet sentences (399,010 sentences) with the part-of-speech (POS) tagger and extracted the words not recognised by the POS tagger. We assumed that these words can reveal peculiarities of chat language. We created a list of these words (1,252 items) and included the words occurring at least twice. The word groups found in the list are the following:

- Contractions with vowels dropped (some have even spelling variations), e.g. '*mnpr*', '*mnprt*' and '*mprt*' for the canonical '*manuprāt*' (in English, 'in my opinion');
- Acronyms;
- Hyphenated abbreviations, e.g. '*D-tilts*' for the canonical '*Dienvidu tilts*' (in English, 'the South bridge');
- Neologisms, mostly new compounds, e.g. there are 14 Latvian compounds starting with '*tviter*', English equivalents would be 'twitterfellow', 'twitternation', 'twittername', 'twittertalks', 'twitterists', 'twittertrolls';
- Slang;
- Words without diacritic marks (there are 11 letters with diacritics in the Latvian alphabet, and the use of diacritics is mandatory, e.g. '*lapa*' ('leaf') and '*lāpa*' ('torch') are different words);
- Words in some other language, mostly in English, e.g. 'fakenews', 'megasale';
- Orthographic errors;
- Non-words, e.g. hashtags, url-s, e-mails, etc.

2.2 Creation of Data Sets for Training, Testing and Development

For development, we have created a set of 500 manually corrected Twitter sentences. This set contains 4,611 tokens, of which 767 tokens (~16%) have been corrected.

A small part of the training corpus is also manually annotated. Originally, we had 6,522 sentences collected from Internet discussion forums and Internet comments. After an annotator has corrected the texts, data is tokenized and the words from the original and the corrected file are aligned. Only sentences with the same number of tokens are kept as our normalisation system does not allow N-to-1 changes and only allows limited cases of 1-to-N changes. For example, we do not consider inserting missing commas, although the use of commas is mandatory in canonical language. As a result, we have 2,493 sentences with 29,277 aligned tokens, and 4,416 tokens (15%) have corrections.

Table 1: Examples of regular expressions used for error creation.

Search	Replace	Type
\b(([^\s]*)ēž([^\s]+))\b	$2ez$3	Diacritics
\b(manuprāt)\b	mnprt	Missing vowels
\b(kaut kur)\b	kkur	Words together
\b(ak mans dievs)\b	omg	Slang abbrev.
\b(jā)\b	yup	Colloquial
\b(vēl([^\s]*))\b	vel$2	Long vowel

There is no ready-made annotated chat language corpus for Latvian that would be large enough for automatic normalisation model training. Creation of such a corpus takes a lot of time and requires significant human resources. Besides, it is problematic to develop strict guidelines for normalisation. This depends on the purpose of further use - whether we want to create a literary text or make it more comprehensive for further processing with the existing NLP tools. For this reason, we decided to create a corpus where errors are introduced artificially. Based on error types learned in Twitter corpus n-gram analysis, we defined a set of 688 regular expressions (see Table 1) that introduces different types of errors - dropped vowels, doubled letters, switched letters, missing diacritics, common spelling errors, words written together, etc. From the POS tagged Tweet corpus, we take only correct sentences, e.g. sentences in which every word is recognised by the POS tagger. We apply the regular expressions to these sentences. In every sentence, a random number of errors is introduced.

Table 2: Summary of the data.

Name	Description	Size
train1	Manually annotated Internet discussion forums and Internet comments	2,493 sentences
train2	Twitter data with artificially introduced errors	199,097 sentences
dev1	Manually annotated Twitter data	500 sentences
test1	Twitter data and utterances from live chat with artificially introduced errors	509 sentences

The test corpus is prepared in a similar way; the errors are created artificially. Part of the test corpus is formed from the Twitter utterances. As we are planning to use the normalisation module for the text pre-procession prior to intent detection, the other part of the test set is composed from the utterances found in logs of some company's customer service live chat. See Table 2 for summary of the data.

3 METHOD

For the normalisation of the Latvian chat language, we adapt the modular normalisation system *MoNoise* proposed by van der Goot and van Noord (2017). This system uses the random forest classifier (Breiman, 2001) to choose the best normalisation candidate for every word considering different features. Candidates are generated by different modules. As different types of anomalies require different normalisation actions, the use of spell checker, word embeddings, N-gram module and some other features allows beating the state-of-the-art results on the normalisation task for the English LexNorm corpus and for the Dutch GhentNorm corpus.

Table 3: The data sources used for the generation of the normalisation candidates and the ranking of the candidates for Latvian.

Type	Quantitative parameters
Word embeddings, built on general text corpus	dimensions: 100 tokens: 1,163,817
N-grams, built on general text corpus	unigrams: 352,503 bigrams: 1,463,503
N-grams, built on Twitter corpus	unigrams: 67,953 bigrams: 181,381
Spelling checker supplemented with slang lexicon	stems: 105,235 affix classes: 304
A list of regular expressions	items: 799
A list of correct and popular words	Words from the Latvian Balanced Corpus (available from *http://www.korpuss.lv/id/LVK2018*) occurring at least 100 times: 11,000 Words used as corrections in train1: 21,904

We start with the features similar to the *MoNoise*. The normalisation task has two steps: 1) the generation of the normalisation candidates for every word (token) in a sentence (token string) and 2) the ranking of the candidates in order to distinguish the correct normalisation candidate (the one with the highest score).

3.1 Generation of the Normalisation Candidates

Generation of the candidates is performed with the help of several modules employing diverse data sets (see Table 3) and some hardcoded rules. As the first candidate with the top most score is added to the list of candidates the original word itself (as most words in an utterance does not need to be normalised).

3.1.1 Word Embeddings

Training models for different NLP tasks using machine learning methods requires a large, appropriately prepared (labelled) text corpus, which is not always available. It has been verified that better results could be achieved when using pre-trained word vectors. For example, syntactical analysis (Socher et al., 2013) or a module for error detection in learners writing (Rei et al., 2016) show better results when using pre-trained word embeddings. Such vectors are trained using a larger corpus without specific marking. In this process, the multidimensional real-number vectors are assigned to the names or phrases in the corpus. Words that are in the vicinity of the certain word or that are used in a similar context acquire similar vectors in the vector space.

There are several toolkits available for word embedding training. We use the *Word2Vec* toolkit (Mikolov et al., 2013). This toolkit offers two methods. With the *Continuous Bag-of-Words* architecture, the neural feed-forward language model is employed. The current word is predicted by its context and by surrounding words; the order of surrounding words is ignored. With the *skip-gram* architecture, prediction of surrounding words depends on the current word; the word order is considered, and the nearest words have a greater weight compared to more distant words.

We train the 100-dimensional word embeddings model using *Continuous Bag-of-Words* architecture and the context window of 10 surrounding words.

3.1.2 Spelling Checker with Slang Lexicon

As our goal was to preserve the style of user generated text, we replenished the lexicon of our spelling checker with the words used in the

conversational language that were not included in the general canonical lexicon.

The number of stems in the lexicon of the general spelling checker is 106,019, but it is 108,109 in the new jargon spelling checker.

The jargon spelling checker's noun list is complemented with new compounds and abbreviated words with pejorative tone or colloquial nature. There are also person names found in the Tweet corpus. Those words should probably be added to the lexicon of the general spelling checker as well.

The new verbs in the jargon spelling checker's verb list come from different sources. Some are found in printed jargon dictionaries. The largest part is collected from the transcripts of the SAEIMA (Latvian Parliament) sessions, from the corpora of IT domains and from the corpora of websites in the Latvian language.

While generating spelling suggestions, the engine of the general spelling checker uses a set of rules like phonetically wrong letters, missing diacritics, one or two letters dropped or inserted, transposed adjacent letters, etc. These rules don't always cover the typical mistakes made by users. A very common habit of writing in chat language is to write without diacritic marks. Sometimes, double vowels are used to signal a long vowel ('aa'→'ā', 'ee'→'ē', 'ii'→'ī', 'uu'→'ū'), the letter 'j' is used after some consonants to signal the softness of the consonant ('gh'→'ģ', 'kj'→'ķ', 'lj'→'ļ', 'nj'→'ņ'), and the letter 'h' is used to signal some other phonetical peculiarities ('ch'→'č', 'sh'→'š', 'zh'→'ž'). Yet, this system is not consistently respected even in the boundaries of a single word. The habit not to use diacritics is probably related to the situation in the past when electronic devices supported only a limited set of symbols, i.e. the Latin alphabet.

We are using the open-source *Hunspell* engine for the spelling checker (available from *https://github.com/hunspell/hunspell*). In the jargon spelling checker engine, additional rules for suggestion generation are defined; there are 95 expressions in total. The rules describe replacements of more than one character or in consideration of the context. The rules fix errors concerning phonetic writing (as in (1)) and the use of diacritics (as in (2)).

REP zdam sdam (1)

REP nsh ņš (2)

3.1.3 Regular Expressions

We used 688 regular expressions for artificial error creation. For generation of normalisation candidates, we reversed this list and added more regular expressions (there are 799 regular expressions in total). We use this module together with the spelling checker. After substituting some characters with the help of the regular expression module, we add the changed word to the normalisation candidate list and also pass this new candidate to the spelling checker engine. If the spelling checker generates some suggestions, they are also added to the candidate list. If a word has several mistakes, there is a greater chance to get the correct candidates by a joint effort of the regular expression module and the spelling checker module. For example, for the erroneous word *filmeshana* ('the filming') the regular expression module generates the less erroneous word *filmešana*; in turn, the spelling checker suggests two candidates for this word - *filmēšana* and *filmēšanā* ('the filming' in nominative and in locative).

3.1.4 N-grams

We use two sets of N-grams: build on data from the general corpus and build on data from the Tweets corpus. In the candidate generation process, the current word is looked up in the unigram list. Unigrams give more weight to the popular words. Bi-grams, with the left and the right adjoining word, encode information about the current word's context; the pairs that are found in a corpus with higher frequency are normalisation candidates with a higher degree of credibility.

3.1.5 Dictionary of Corrections

Prior to generation of the normalisation candidates, the system reads through the training data and builds a dictionary of corrections with information about every corrected word in the training corpus and how many times it is corrected to another word. This data is used in the next step when the system generates the candidates for every word in a sentence. The current word is looked up in the dictionary, and, if it contains some corrections, it is added to the correction list along with the occurrence frequency number.

3.1.6 Word Splitting

One of the errors found in user created text is a missing space between some words. If a word is

more than two letters long, we try to split it in two parts. We accept candidates consisting of two parts if such a word combination is found in the bigram dictionary built on data from the general corpus.

Our first approach was to check if the two separated words are accepted by the spelling checker. Unfortunately, this approach led to the production of never occurring candidates. The typical faults for such action are a prefix separated from a prefixed verb or both parts of a compound written separately (in Latvian, compound parts must always be written together). For example, the verb *aplikt* ('to put on') is separated as *ap* ('around') and *likt* ('to put'), or the compound *asinsvadi* ('blood-vessels') is separated as *asins* ('blood') and *vadi* ('wires' or 'cords'). Such results prompted to change the algorithm and perform lookup in the bigram dictionary.

3.1.7 Words with Same Root

The Latvian language is an inflectional language; most word-forms are formed by combining the root and the ending. We search for the candidates in a list of correct and popular words. We accept the candidates that have one or two extra letters at the end compared to the current word. We cut the typical endings from the end of the current word (the single letter 'a', 'e', 'i' or 'u' and two letters if the last one is 's' or 'm') and search for the candidates that differ in length from the current word by no more than two symbols. We also add the same root base-form of a word supplied by the spelling checker module. For example, for the word *komanda* ('a command' in singular nominative), the following candidates are chosen: *komandai* ('a command' in singular dative), *komandas* ('a command' in singular genitive or plural nominative), *komandu* ('a command' in singular accusative or plural genitive), *komandām* ('a command' in plural dative), *komandē* ('commands' a verb in the 3rd pers.), *komandēt* ('to command').

3.1.8 Diacritic Marks

The diacritic restauration module tries to add diacritic marks to every character in the current word. Correctness of the newly constructed word is checked with the spelling checker module. The words with correct diacritics are added to the candidate list. With this method, for the incorrect word *speletajs*, the correct candidate *spēlētājs* ('a player') is generated. Also, for the nominative of the correct word *attiecības* ('relationship'), a locative form *attiecībās* is generated.

3.2 Ranking of the Candidates

For the ranking of the candidates, the features related to the candidates are employed. The list of features is similar to the ones used in the normalisation system *MoNoise* (van der Goot et al., 2017). For every candidate, a feature vector is constructed containing the following values:

- a binary value (the number '0' or '1') signalling if the candidate is the original word;
- the candidate's and the original word's cosine similarity in the vector space and the rank of the candidate in a list of top 20 most similar words if the candidate is supplied by the word embeddings module;
- a binary value signalling if the candidate is generated by the spelling checker module and the candidate's rank among other correction candidates that spelling checker generates for the misspelled original word;
- a number of times the original word is changed to particular candidate in the dictionary of corrections built on the basis of the training data;
- a binary value signalling if the candidate is created by changing some final characters of the original word, i.e. has the same root, or by adding diacritic marks to some letters of the original word;
- a binary value signalling if the candidate is created by splitting the original word;
- the candidate's unigram probability in the general corpus and in the Twitter corpus;
- the candidate's bigram probabilities with the left and the right adjoining word in the general corpus and in the Twitter corpus;
- a binary value signalling if the candidate is in the good and popular word list;
- a binary value signalling if the original word and the candidate have a matching symbol order;
- the length of the original word and the candidate;
- a binary value signalling if the original word and the candidate are constructed of valid utf-8 symbols, are not e-mail addresses or Web links.

Random forest classifier algorithm creates an ensemble of decision trees taking into account different features. Different trees are responsible for different normalisation actions. Classifier ranks every candidate at every position in a given text string (see Table 4). Candidates with a top score form the normalised text string.

Table 4: Results of normalisation for the sentence ‘I want to pay the bill.’.

Input string	*Gribu apmaksat rekinu.*
Normalised string	*Gribu apmaksāt rēķinu.*
Normalised string with top 5 candidates	0 Gribu 1 0.999686 0 griba 1 0.000157 0 gribam 1 0.000115 0 gribētu 1 0.000030 0 es 1 0.000012 1 apmaksāt 2 0.998305 1 maksāsimies 2 0.000678 1 apmaksāti 2 0.000431 1 apmaksāta 2 0.000425 1 apmaksātas 2 0.000160 2 rēķinu 3 0.998016 2 rēķina 3 0.001495 2 rēķinās 3 0.000250 2 rēķins 3 0.000172 2 rēķinos 3 0.000066 3 . 4 1.000000

4 RESULTS AND DISCUSSION

While trying to improve the results, we have trained several models. Recall, precision and F1 score (Rijsbergen, 1979) for different experiments are reported in Table 5.

For Model 1, we use a reduced set of training data (13,378 examples out of 201,589) since we encountered memory problems while training the model using a full set of training data.

For Model 2, we use the reduced training data for lookup dictionary creation, but we generate normalisation candidates and build the feature matrix for random forest classifier training only for the first 5000 examples. As the results for Model 2 are even better than for Model 1, we conclude that it is not necessary to use more examples for feature matrix building. This new approach allows us to reduce the memory amount required for training. For Model 3, we return to the full set of training data for lookup dictionary creation and use the first 5000 sentences for feature matrix building.

For Model 4, we introduce two changes. While generating normalisation candidates, we try to split every word only if the two new words are found in the bigram dictionary. The second change is that we process the text twice while performing normalisation. If there are several errors in a text, the normalisation model cannot deal with all errors in a single run. The results improve after the second run. For the sample (3) (‘Just going with the flow as to say’), the first four words are erroneous. After the first run (4), three words are fixed, but the first word still remains incorrect. After the second run (5), all errors have been corrected.

Vienkarsi laujos pluusmai kaa saka (3)
Vienkarsi ļaujos plūsmai kā saka (4)
Vienkārši ļaujos plūsmai kā saka (5)

There are cases when the correct word is not among the generated normalisation candidates. It is difficult to predict in advance the whole range of errors that a potential user could make and generate corrections according to it. For Model 5, we improve our function of diacritics restoration and add some suggestion generation patterns for the spelling checker module.

Table 5: Results of normalisation performed by different models (for the development set).

No	Model	Recall	Precision	F1
1	13,378 examples for dictionary building and training	0.7817	0.8092	0.7952
2	As Model 1, but only first 5000 for training	0.7843	0.8186	0.8011
3	201,589 examples for dictionary building, only first 5000 for training	0.8105	0.8367	0.8234
4	As Model 3, but split word if both parts in the bigrams and correct text twice	0.8288	0.8431	0.8359
5	As Model 4, but use improved rules for spelling candidate generation and restoration of diacritics	0.8353	0.8397	0.8375

The results of normalisation for the test set are better than for the development data (see Table 6). The test set contains sentences with artificially introduced errors. The high results attest that the model has learned to normalise such errors quite well. Most utterances in the test set are from some company’s customer service live chat. Besides, these utterances are short, and the quality is better than for the Twitter data included in the development corpus.

This explains an increase in accuracy and precision.

Table 6: Results of normalisation performed by Model 5 (for the development and the test sets).

Data	Recall	Precision	F1
Development set	0.8353	0.8397	0.8375
Test set	0.9633	0.949	0.9561

5 EVALUATION

We test the normalisation module in practice by normalising user created text prior to intent detection.

While developing a virtual assistant for customer service, we have created a module for intent detection in users' utterances (Balodis et al., 2018). This intent detection module is based on convolutional neural network architecture. From the company's live chat log, we have compiled a test set of 236 utterances for testing of the intent detection module.

Table 7: Results of intent detection for different data sets.

Data set	Accuracy (mean)	Accuracy (median)
Original	43.13%	43.43%
Normalised	44.84%	45.30%
Manually corrected	45.41%	46.02%

The results acquired after running the intent detection module 10 times (see Table 6) testify to the positive impact of prior text normalisation on the quality of intent detection. After normalisation, the median value of accuracy increases by 1.87% compared to the original text and stays behind the manually corrected text by only 0.72%.

6 CONCLUSION AND FUTURE WORK

We have achieved our goal of creating a normalisation module that is able to normalise most typical errors found in user created text. By experimenting with different properties and training several models we have gradually found the optimal set of features that allowed us to increase precision of the model from 0.8092 to 0.8397 and recall from 0.7817 to 0.8353. We have overcome the lack of marked training data by creating a corpus with artificially introduced errors. The types of errors were determined by analysing the most common errors in Twitter data. We have also examined whether the use of such a model in pre-processing could improve the accuracy of the intent detection module. The results are good, text normalisation helps to detect the user's intent more precisely.

Still, there remain some unresolved questions about what to consider a mistake. In informal conversation, users often change the language of the text by introducing some common English words or phrases ('priceless', 'free shipping', 'like', 'my life goals') or using loanwords formed from some English word stem with a Latvian ending (*followeriem*, *settingi*, *friendslisti*, *storiji*). We should try to detect them and not normalise or normalise using some special rules. Otherwise, we can get a word with a completely different meaning (the current normalisation model changes *friendslisti* 'friendslist' to *orientālisti* 'orientalist').

ACKNOWLEDGEMENTS

The research has been supported by the European Regional Development Fund within the project "Neural Network Modelling for Inflected Natural Languages" No. 1.1.1.1/16/A/215.

REFERENCES

Baldwin, T., de Marneffe, M., Han, B., Kim, Y., Ritter, A. and Xu, W., 2015a. Guidelines for English lexical normalisation. https://github.com/noisy-text/noisytext.github.io/blob/master/2015/files/annotation_guideline_v1.1.pdf.

Baldwin, T., de Marneffe, M. C., Han, B., Kim, Y. B., Ritter, A., and Xu, W., 2015b. Shared tasks of the 2015 workshop on noisy user-generated text: Twitter lexical normalization and named entity recognition. In *Proceedings of the Workshop on Noisy User-generated Text* (pp. 126-135).

Balodis, K., and Deksne, D., 2018. Intent Detection System Based on Word Embeddings. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications* (pp. 25-35). Springer, Cham.

Breiman, L., 2001. Random forests. *Machine learning*, 45(1), 5-32.

Damnati, G., Auguste, J., Nasr, A., Charlet, D., Heinecke, J., and Béchet, F., 2018. Handling Normalization Issues for Part-of-Speech Tagging of Online Conversational Text. In *Eleventh International Conference on Language Resources and Evaluation (LREC 2018).*

Eisenstein, J., 2013. What to do about bad language on the internet. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, ACL, Atlanta, Georgia, pp. 359.

van der Goot, R., and van Noord, G., 2017. MoNoise: Modeling Noise Using a Modular Normalization System. arXiv preprint arXiv:1710.03476.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).

Pinnis, M. 2018. Latvian Tweet Corpus and Investigation of Sentiment Analysis for Latvian. *Frontiers in Artificial Intelligence and Applications. Volume 307: Human Language Technologies – The Baltic Perspective* (pp 112-119).

Rei, M. and Yannakoudakis. H. 2016. Compositional Sequence Labeling Models for Error Detection in Learner Writing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Rijsbergen, C.J., 1979. *Information Retrieval*, Vol. 2, University of Glasgow.

Socher, R., Bauer, J. and Manning, C.D., 2013. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics* (Vol. 1, pp. 455-465).