A dual-encoding system for dialect classification

Petru Rebeja
Faculty of Computer Science,
"Alexandru Ioan Cuza" University
16 Berthelot St., Iași
petru.rebeja@gmail.com

Dan Cristea
Faculty of Computer Science,
"Alexandru Ioan Cuza" University
16 Berthelot St., Iași
Institute for Computer Science,
Iași branch of the Romanian Academy
2 Codrescu St., Iași
danu.cristea@gmail.com

Abstract

In this paper we present the architecture, processing pipeline and results of the ensemble model developed for Romanian Dialect Identification task. The ensemble model consists of two TF-IDF encoders and a deep learning model aimed together at classifying input samples based on the writing patterns which are specific to each of the two dialects. Although the model performs well on the training set, its performance degrades heavily on the evaluation set. The drop in performance is due to the design decision which makes the model put too much weight on presence/lack of textual marks when determining the sample label.

1 Introduction

The VarDial evaluation campaign (Zampieri et al., 2017; Zampieri et al., 2018; Zampieri et al., 2019) proposes several tasks related to the study of language variations across geographical regions.

One of its oldest challenges is the is the identification of some specific dialects which, throughout the years, was tackled using several tasks. The challenge started initially with Arabic Dialect Identification in the third edition of VarDial (Malmasi et al., 2016). The fourth edition (Zampieri et al., 2017) expands the family of dialect identification tasks with German Dialect Identification. The sixth edition (Zampieri et al., 2019) comes with yet another task named Moldavian vs. Romanian Cross-dialect Topic identification.

The current edition of the evaluation campaign (Găman et al., 2020) introduces the task of Romanian Dialect Identification (RDI). This is a closed task which is aimed at classifying a text sentence as whether it pertains to the Moldavian idiom[1] or one of the other idioms of Romanian language.

For this task, our team proposes a simple[2] ensemble model which is built on the idea of exploiting the known differences in writing between the two dialects and on emphasizing such differences for classification.

This paper describes the architecture of the proposed model, the setup for training experiments, the processing pipeline, and the results obtained for the RDI task. Towards the end of the paper we also provide some insights into how we can easily tear-down and recreate the whole environment used for our experiments.

The paper is structured as follows: Section 2 presents some of the previous approaches in dialect identification tasks which bear a resemblance to our approach, Section 3 briefly introduces the dataset used for the model, Section 4 describes the architecture of the model, Section 5 describes the experiments and the results, and in the end, Section 6 presents the conclusions and future work.

2 Related Work

The task of discriminating between dialects of the same language has been the subject of various approaches that employ feature engineering, shallow or deep learning, and/or a combination of the afore-

[1]Less than a dialect, but in the following we will refer to them as being dialects.
[2]The model architecture (see Section 4) contains relatively few layers.

mentioned.

One such example of using deep learning models for dialect classification is the Word-Based Convolutional Neural Network (Elaraby and Abdul-Mageed, 2018), developed for Arabic Dialect Identification. As authors note, their model "is conceptually similar" to the model described in (Kim, 2014). This model uses pretrained and fine-tuned word embeddings as the input to of a convolutional layer with max-pooling, followed by a fully-connected (dense) layer with softmax activation.

Another approach for dialect classification is to use ensemble models. Examples of such models include a Meta-classifier (Malmasi and Zampieri, 2017) for Arabic Dialect Identification, which leverages the collective knowledge of a cluster of individual classifiers by predicting the dialect based on their individual decisions, and an equally-weighted voting schema between a character-level convolutional network, a Long Short-Term Memory network and a string-kernel model for German Dialect Identification (Butnaru, 2019).

At the previous edition of VarDial (Zampieri et al., 2019), the highest-scoring model for the Moldavian vs Romanian Cross-dialect Topic Identification task, was also an ensemble model (Tudoreanu, 2019). The model consists of a triplet loss network, a skip-gram network and a SVM classifier. The SVM classifier receives the concatenated representations learned by each network and based on these makes the prediction of the dialect.

Unlike the model presented in (Tudoreanu, 2019) where the architecture emphasizes the learning of representation for taking a decision, the model presented herein uses shallow representations and puts more weight learning the distinction between them when deciding which label to predict.

3 Dataset

The data source for Romanian Dialect Identification task is the MOROCO dataset (Butnaru and Ionescu, 2019) which consists of short text sentences (one per line) from both Romanian and Moldavian dialects of the Romanian language. The sentences were extracted from various news articles and besides the already mentioned dialect labels, the dataset also provides additional labels for the topic within each sentence falls: culture, finance, politics, science, sports, and tech.

The novelty of the current evaluation campaign is that a supplementary corpus was used for evaluation: MOROCO Tweets (Găman and Ionescu, 2020), containing tweets as data samples instead of sentences from news articles.

Loading Data and Preprocessing. Our approach to working with the data from MOROCO dataset is to construct an in-memory table, using Pandas library[3] with four columns: sample id, dialect label, topic label, and the sample itself.

After loading the dataset in memory, we perform on the fly, simple preprocessing on it. First, the sample text is converted to lowercase, then the tokenization mechanism splits it into its component words.

4 Model Architecture

The model we developed for the RDI task is an ensemble of three parts: two encoders and a deep-learning model consisting of five layers excluding the output layer. The entire architecture of the ensemble model is shown in Figure 1.

As stated in Section 1, at the core of our model lies the intuition that the peculiarities of each of the two Romanian dialects involved in the RDI task can be detected by spotting the differences in writing.

One of the most obvious examples of this intuition lies in how each dialect deals with writing words such as *atât*. The official rules stated by Romanian Academy say that in the middle of the words the character *â* should be used whereas the Moldavian dialect uses the form *î* for the same sound, thus the same word — *atât* — is written in Moldavian dialect as *atît*. Our model aims to capture such differences and exploit them in the classification.

Each input sentence is transformed into two different encodings where each encoding is obtained from the Term Frequency-Inverse Document Frequency (Salton and Buckley, 1988) (TF-IDF) representation

[3]https://pandas.pydata.org/

of the sentence in each of the two dialects. In other words, each sentence gets two TF-IDF representations — one for each dialect.

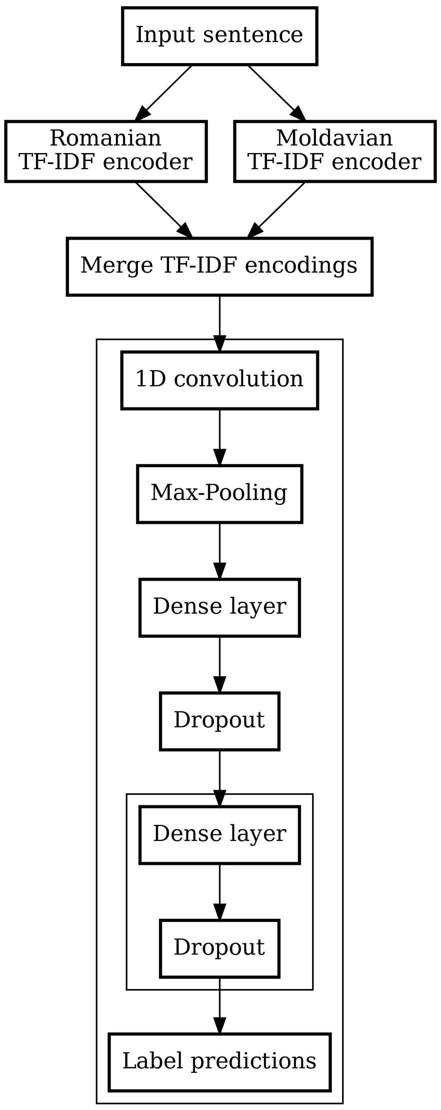


Figure 1: The architecture of our ensemble model for Romanian Dialects Classification task.

After obtaining the TF-IDF encodings for an input sentence, the encodings are merged into a single matrix by concatenating columns. Although this operation (represented by the *Merge TF-IDF encodings* node in the model diagram) is represented by a separate node it is a *logical* operation in the sense that it is not part of the model architecture.

The tensor containing concatenated encodings is passed through a one-dimensional convolutional layer aimed at identifying the prominent features from each of the two encodings of the dialects. The Global Max-Pooling layer that follows the convolutional layer picks the aforementioned features and send them to the following layers.

The results of the Global Max-Pooling layer are then passed through a Dense layer followed by a Dropout (Srivastava et al., 2014) layer with Rectified Linear Unit (Nair and Hinton, 2010) (ReLU) activation. Then the results pass through yet another group consisting of a Dense and Dropout with ReLU activation layers after which, the final Dense layer outputs the predictions for each of the RDI labels. A single dropout rate was used for both Dropout layers whereas the Dense layers are shrinking in size towards the final layer (see Table 1).

Lastly, the predictions of the output layer, together with the expected output are fed to a categorical crossentropy loss function and an Adam (Kingma and Ba, 2014) optimizer is used to find the global minimum.

4.1 Hyper-Parameters

Since the hardware we used to train our ensemble model has relatively low resources the training process was affected by high memory usage, which led to longer training sessions. Under such hardware and time constraints it was prohibitive for us to perform a full-blown hyper-parameter tuning session of the model.

As such, we were forced to settle for the best-performing set of hyper-parameters from a small set of variations.

We present the parameters and their associated values for the best results on evaluation set in Table 1.

5 Experiments and Results

We approach training of the full ensemble model in three separate stages: one training stage per each TF-IDF encoder (for Romanian and Moldavian dialects) and a third and final stage for training the deep-learning classification model.

All of these stages are performed in a single run, as part of a training pipeline that follows closely the architecture of the model presented in Section 4 (Figure 1) with the exception that TF-IDF encoders were trained one after the other rather than in parallel. We describe each of these stages in the following subsections.

In training the whole ensemble model we followed the classical recipe: the model was trained on the train data and validated against the validation set. We report the accuracy on training and validation sets and the score obtained on evaluation set in Section 5.4.

Parameter name	Value
1D convolution	
Number of filters	512
Kernel size	7
Dropout rate	
All layers	0.3
Output dimensions	
Dense layer 1	128
Dense layer 2	32
Adam optimizer	
Learning rate	0.001
Decay rate β_1	0.9
Decay rate β_2	0.999

Table 1: The hyper-parameters used for training the deep-learning model which obtained the best results on the evaluation set.

5.1 Training TF-IDF Encoders

We start training the TF-IDF encoders for each dialect by building a global vocabulary from the train data. Having a unified vocabulary ensures that both of the encoders properly encode the same features of the input sample and thus both encodings can be compared against writing regularities such as the one mentioned in the beginning of Section 4.

In other words, when working on the same vocabulary, the Romanian TF-IDF encoder would give a lower score to the word *atît* and high score to the word *atât* whereas the Moldavian TF-IDF encoder would assign scores inversely.

To build the common vocabulary, we make a full pass over the training set. Once the vocabulary is built, we split the training data into two sets according to its dialect label: a set consisting solely of samples for Romanian dialect and another one consisting of samples for Moldavian dialect. Finally, a TF-IDF encoder is trained on each of these two sets, resulting in the encoders presented in Figure 1.

5.2 Generating Batches On-the-Fly

Due to hardware constraints mentioned in Section 4.1, we were unable to load the whole training set into memory after encoding it using the aforementioned TF-IDF dialect encoders.

In order to be able to perform the training we decided to create a custom batch generator that would build each batch of training/validation samples on the fly.

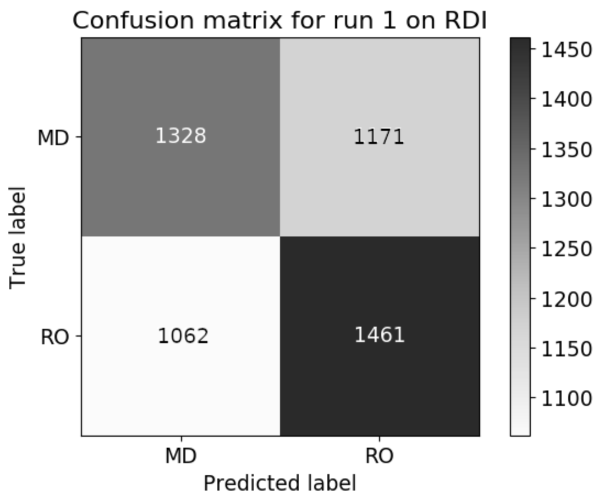
The batch generator has references to the collection of training samples and their associated labels alongside the two TF-IDF encoders for Romanian and Moldavian dialects. It then determines the total number of batches according to the batch size parameter specified when starting the training. After determining the total number of batches to be generated, the generator then encodes each batch of samples on request.

The encoding process itself is performed as follows: for each sample indexed between the start and end indices of the batch, a TF-IDF encoding is computed for both Romanian and Moldavian dialects. We will denote those encodings with $\mathbf{RO_{TF-IDF}}$ and $\mathbf{MD_{TF-IDF}}$ respectively. Afterwards, the encodings are concatenated column-wise to form the tensor $\mathbf{x} = [\mathbf{RO_{TF-IDF}} \quad \mathbf{MD_{TF-IDF}}]$ which will be sent to the deep-learning model. Practically, the batch generator encapsulates the *Merge TF-IDF encodings* node from Figure 1 in Section 4.

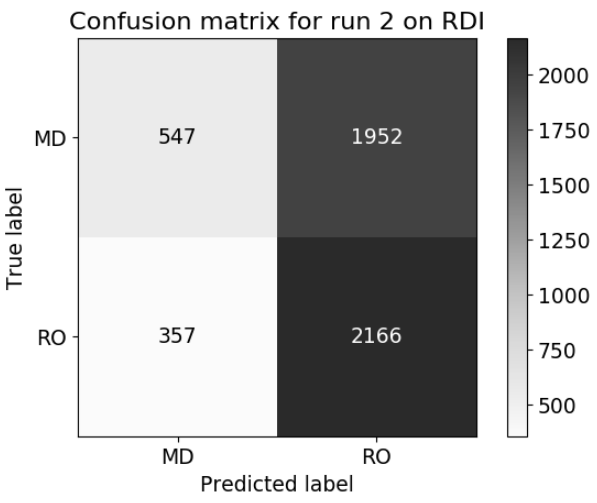
At the end of each training epoch, the batch generator shuffles the samples to prevent overfitting.

5.3 Dialect Classification Model

For each of the submitted runs of the RDI task, we created a variation of the dialect classification model, where the changes applied for migrating a model from a prior to posterior versions consist of adding more layers.



(a) Confusion matrix for first submission.



(b) Confusion matrix for second submission.

Figure 2: Confusion matrices for our ensemble model on the RDI task for first run and second run. The images were edited to improve readability.

For the first submission of results we started with a model containing a smaller number of layers, namely: a *1-dimensional convolution* layer, a *Global Max-Pooling* layer, a *Dense* layer, a *Dropout* layer, and an output layer.

In the second version of our model we introduced another group of layers consisting of a second *Dense* layer followed by a *Dropout* layer.

The architecture presented in Figure 1 depicts the changes applied when upgrading from first version of the model to second version by encapsulating them into the innermost rectangle.

5.4 Results and Interpretation

For each of the two submitted runs, the ensemble model was trained for 25 epochs with a batch size of 32 samples. We programmed the model to save its weights each time there was an improvement in the accuracy score thus the results presented in this section are obtained using the highest performing weights for each run.

For the first submission, the ensemble model obtained a 0.8091 accuracy on the training set, 0.7988 accuracy on the validation set and 0.5553 accuracy on the evaluation set. For the second submission, the scores were 0.8220 on training set, 0.8106 on validation set, and 0.5402 for evaluation set respectively. The accuracy scores for each run are also presented in Table 2.

	Model accuracy		
Run	Train	Validation	Evaluation
1	0.8091	0.7988	0.5553
2	0.8220	0.8106	0.5402

Table 2: Accuracy scores of our ensemble model for both of the submitted runs.

Looking at both the results presented in Table 2 and the confusion matrices presented in Figure 2, it seems that our ensemble model suffers from overfitting.

The overfitting phenomenon is also suggested by the inverse correlation between model parameters, accuracy scores on train and validation data and accuracy scores on evaluation set obtained in each submission. The total number of parameters of the model has increased from first submission to second submission due to adding more layers and also widening the existing *Dense* layers. This change, as can be seen from Table 2 led to an increase in accuracy on training and evaluation sets but in the same time we can see the performance on the evaluation set degrading from first to second submission, which is a clear indication of overfitting.

However, we consider the poor performance of the model to be based on its designed intention. As described in Section 4, the core intention of our model is to identify text patterns which are specific for each of the two dialects and discriminate the samples based on identified patterns.

If we are to compare the confusion matrix from first submission (Figure 2a) with the confusion matrix from the second submission (Figure 2b) we can see that for the second submission the model fails to detect the text peculiarities of the Moldavian dialect. Another observation that can be made from comparing the confusion matrices from Figure 2 is that in the second submission the model heavily favors the Romanian dialect.

Since the training data is relatively balanced (there are slightly more samples from Romanian dialect than from Moldavian), we cannot attribute such heavy favoring of the Romanian dialect to a skewed dataset thus, the conclusion is that the model fails to detect discriminatory patterns in the evaluation samples which leads to improper classification. Although we did not evaluate the TF-IDF encodings computed by both encoders in any way, we consider the idea of improper TF-IDF encodings as being highly unlikely.

Rather, we consider the poor performance in identifying the discriminatory text patterns to be caused by the lack of thereof. We were not able to find in the evaluation set a sufficiently large amount of samples that exhibit markings which would allow for immediate discrimination. Furthermore, we observed multiple samples labeled as pertaining to Moldavian dialect that rather than obeying the rule of using the character *î* inside the words were using the *â* character which is a characteristic of the Romanian dialect.

As such, it becomes clear that our design decision on discriminating predominantly based on text markings is not sufficient alone to properly discriminate between Romanian and Moldavian dialects and the model needs additional signals to perform proper classification.

5.5 A Note on Experiment Reproducibility

Since our initial runs of the experiments required switching from a development machine to a more powerful machine used for training we asked ourselves the question on how to reproduce the same environment in training as used for development.

Although `Python` provides a way to recreate a virtual environment by using a file from which to install the required packages, we needed a bit more choreography in order to synchronize the environments on the machines used for developing and training the model such as: having a predefined directory structure from where to load the data and where to save the output, the need to activate the virtual environment for `Python` etc.

In order to fulfill the need for homogeneous environments, we turned ourselves to the Docker platform[4] which allows for full replication of an environment onto different machines.

Thus, for our environments we created a Docker image which contains not only the packages required by `Python` but also the system-wide packages required for loading and reading the data and training the model. This way, the entire experiment can be reproduced at any time by following three simple steps: building the Docker image from the provided file, starting a container with mounted volumes for directories containing data, and run the training.

The source-code of our ensemble, including the `Dockerfile` used for building the image is available on GitHub[5].

6 Conclusions and Future Work

In this paper we describe the architecture of an ensemble model designed to classify an input sentence as pertaining to Romanian or Moldavian dialects based primarily on the text markings which make the distinction between those two dialects. Alongside the model architecture and performance we describe the processing pipeline used for training the model and the setup of the training environment which allowed us to easily replicate the training environment on the machine where training was performed.

To our great disappointment, the results of the evaluation set shown both in Table 2 and in Figure 2 indicate that the design decision of identifying textual marks and discriminating primarily on those marks

[4]https://www.docker.com/. We used the freely available Community Edition.

[5]https://github.com/RePierre/vardial2020

leads to poor performance. Our main conclusion in regards to the model performance is that relying only on such marks and differences in text is not enough to discriminate real-world examples because — as data from the evaluation set shows — only some input samples display such marks, not all of them. Furthermore, as our subsequent analysis revealed, some of the marks which the model sought for are lacking (e.g. *Sfântul NE NE este a doua finalistă a ediţiei inaugurale a NE NE NE* — pertaining to Moldavian dialect but uses *â* instead of *î*), which blurs even more the lines between Romanian and Moldavian dialects and points to a tendency of the Moldavian dialect to adhere to the writing norms of its Romanian counterpart. In such case, the model doesn't learn to properly discriminate between the two dialects.

Even though our model makes use of techniques such as *Dropout* (Srivastava et al., 2014) which are meant to prevent overfitting and boost generalization, the model generalizes poorly and its confidence drops when the input sample lacks the strong textual marks it was trained to look for.

In order to improve results — as one might expect — the future work should heavily concentrate on the classification model part of the ensemble presented herein. First and foremost, we need to investigate what is the root cause of long training time and improve the part responsible for it. This will allow us to test multiple variations of the model and perform a proper hyper-parameter tuning. However, if our investigation will conclude that there is no single part responsible for the most of the incurred delay in training, it make sense to consider rewriting the whole model architecture.

References

Andrei Butnaru and Radu Tudor Ionescu. 2019. Moroco: The moldavian and romanian dialectal corpus. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 688–698.

Andrei Butnaru. 2019. Bam: A combination of deep and shallow models for german dialect identification. In *Proceedings of the Sixth Workshop on NLP for Similar Languages, Varieties and Dialects*, pages 128–137.

Mohamed Elaraby and Muhammad Abdul-Mageed. 2018. Deep models for arabic dialect identification on benchmarked data. In *Proceedings of the Fifth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial 2018)*, pages 263–274.

Mihaela Găman and Radu Tudor Ionescu. 2020. The unreasonable effectiveness of machine learning in moldavian versus romanian dialect identification. *arXiv preprint arXiv:2007.15700.*

Mihaela Găman, Dirk Hovy, Radu Tudor Ionescu, Heidi Jauhiainen, Tommi Jauhiainen, Krister Lindén, Nikola Ljubešić, Niko Partanen, Christoph Purschke, Yves Scherrer, and Marcos Zampieri. 2020. A Report on the VarDial Evaluation Campaign 2020. In *Proceedings of the Seventh Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial).*

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882.*

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Shervin Malmasi and Marcos Zampieri. 2017. Arabic dialect identification using ivectors and asr transcripts. In *Proceedings of the Fourth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial)*, pages 178–183.

Shervin Malmasi, Marcos Zampieri, Nikola Ljubešić, Preslav Nakov, Ahmed Ali, and Jörg Tiedemann. 2016. Discriminating between similar languages and arabic dialect identification: A report on the third dsl shared task. In *Proceedings of the Third Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial3)*, pages 1–14, Osaka, Japan, December.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA. Omnipress.

Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Diana Tudoreanu. 2019. Dteam@ vardial 2019: Ensemble based on skip-gram and triplet loss neural networks for moldavian vs. romanian cross-dialect topic identification. In *Proceedings of the Sixth Workshop on NLP for Similar Languages, Varieties and Dialects*, pages 202–208.

Marcos Zampieri, Shervin Malmasi, Nikola Ljubešić, Preslav Nakov, Ahmed Ali, Jörg Tiedemann, Yves Scherrer, and Noëmi Aepli. 2017. Findings of the VarDial Evaluation Campaign 2017. In *Proceedings of the Fourth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial)*, Valencia, Spain.

Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Ahmed Ali, Suwon Shuon, James Glass, Yves Scherrer, Tanja Samardžić, Nikola Ljubešić, Jörg Tiedemann, Chris van der Lee, Stefan Grondelaers, Nelleke Oostdijk, Antal van den Bosch, Ritesh Kumar, Bornini Lahiri, and Mayank Jain. 2018. Language Identification and Morphosyntactic Tagging: The Second VarDial Evaluation Campaign. In *Proceedings of the Fifth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial)*, Santa Fe, USA.

Marcos Zampieri, Shervin Malmasi, Yves Scherrer, Tanja Samardžić, Francis Tyers, Miikka Silfverberg, Natalia Klyueva, Tung-Le Pan, Chu-Ren Huang, Radu Tudor Ionescu, Andrei Butnaru, and Tommi Jauhiainen. 2019. A Report on the Third VarDial Evaluation Campaign. In *Proceedings of the Sixth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial)*. Association for Computational Linguistics.