Classifying ASR Transcriptions According to Arabic Dialect

Abualsoud Hanani, Aziz Qaroush
Electrical & Computer Engineering Dept
Birzeit University
West Bank, Palestine
{ahanani,qaroush}@birzeit.edu

Stephen Taylor
Computer Science Dept
Fitchburg State University
Fitchburg, MA, USA
staylor@fitchburgstate.edu

Abstract

We describe several systems for identifying short samples of Arabic dialects, which were prepared for the shared task of the 2016 DSL Workshop (Malmasi et al., 2016). Our best system, an SVM using character tri-gram features, achieved an accuracy on the test data for the task of 0.4279, compared to a baseline of 0.20 for chance guesses or 0.2279 if we had always chosen the same most frequent class in the test set. This compares with the results of the team with the best weighted F1 score, which was an accuracy of 0.5117. The team entries seem to fall into cohorts, with the all the teams in a cohort within a standard-deviation of each other, and our three entries are in the third cohort, which is about seven standard deviations from the top.

1 Introduction

In 2016 the Distinguishing Similar Languages workshop (Malmasi et al., 2016) added a shared task to classify short segments of text as one of five Arabic dialects. The workshop organizers provided a training file and a schedule. After allowing the participants development time, they distributed a test file, and evaluated the success of participating systems.

We built several systems for dialect classification, and submitted runs from three of them. Interestingly, our results on the workshop test data were not consistent with our tests on reserved training data.

Our accuracy rates cluster around 40%; the rates of the best systems were a little better than 50%. If we take the raw scores as drawn from a binomial distribution, the standard deviation is $\sqrt{p(1-p)n}$. With $n = 1540$, and $p = 0.5$ or $p = 0.4$ the standard deviation is 19.2 or 19.6 correct answers respectively, corresponding to a difference in accuracy of about 1.25%. Since the best overall accuracy score is 51.33%, our best score is 6.9 standard deviations below it. (The scores of the top three teams don't seem to be significantly different from each other.)

On reserved training data, our systems all scored much better than they did on the test data, with our best system achieving an accuracy rate of 57%. No doubt the best systems in the trial also scored better in training. In addition to describing our systems, we speculate what factors might account for the difference in training and test results.

2 Related Work

The Arabic dialects have a common written form and unified literary tradition, so it seems most logical to distinguish dialects on the basis of acoustics, and there is a fair amount of work there, including (Hanani et al., 2013; Hanani et al., 2015; Ali et al., 2016). Determining the contents of a transcript, i.e. what word that sound sequence is most likely to be, is easier if you know what language model and what dictionary to apply. (Najafian et al., 2014)

Language modeling of Arabic dialects has been held back by an absence of appropriate corpora. Work has been done by Al-Haj et al. (2009; Ali et al. (2008; Elmahdy et al. (2012; Elmahdy et al. (2010; Novotney et al. (2011; Elmahdy et al. (2013; Vergyri et al. (2005; Zaidan and Callison-Burch (2011) and

Proceedings of the Third Workshop on NLP for Similar Languages, Varieties and Dialects,
pages 126–134, Osaka, Japan, December 12 2016.

EGY	Egyptian	The dialect most often called Egyptian is an urban dialect used in Cairo and Alexandria. The next largest Egyptian dialect, Sa'idi, with 20 million speakers, is said to be incomprehensible to Cairene speakers.
GLF	Gulf	The dialects from the Arabic Gulf countries of Bahrain, Kuwait, Oman, Saudi Arabia, United Arab Emirates, and sometimes Iraq are often grouped together.
LAV	Levantine	This group may include dialects from Jordan, Palestine, Syria. (The label *LAV* is used consistently in this corpus.)
MSA	Modern Standard Arabic	This includes most Arabic literature and most formal speech, including television news.
NOR	North African	Dialects from north Africa including the countries of Algeria, Libya, Morocco, Tunisia.

Table 1: Dialect Labels

Ali et al. (2016), most of whom developed corpora for the purpose, several of which are now publicly available.

Ali et al. (2016) developed the corpus on which the DSL Arabic shared task is based. Their own dialect detection efforts depended largely on acoustical cues. Malmasi et al. (2015) do Arabic dialect identification from text corpora, including the Multi-Dialect Parallel Corpus of Arabic (Bouamor et al., 2014) and the Arabic Online Commentary database (Zaidan and Callison-Burch, 2011). Zaidan and Callison-Burch (2014) builds on their own corpus to do dialect identification. ElFardy and Diab (2013) also build a classifier based on the Zaidan and Callison-Burch (2011) corpus.

Most of the work identifying Arabic dialects from text uses character features; many also use word features. Many use Support Vector Machines (SVM.) We investigated building SVM models using character n-gram features.

In the 2014 DSL workshop shared task, the second place entry (Porta and Sancho, 2014) used a *whitelisted words* feature, the 10,000 most frequent words in each language, which is slightly similar to the idea we implement in Section 4.3. However, given the substantial overlap in vocabulary between Arabic dialects, our approach is to look for unusual frequencies, both excesses and scarcities.

3 Methodology and Data

3.1 Training and Test Data

The training data is drawn from the speech corpus collected by Ali et al. (2016). The text is provided in the Buckwalter transcription (Buckwalter, 2002). There are no vowels, and no punctuation, except for the space character.

The training data comprises 7619 segments, ranging in size from one to 18017 characters each. Each segment is labeled with one of five Arabic dialects as shown in table 3.1. The labels are broad and imprecise. For example rural and urban dialects in a single country are likely to differ a great deal, and both accents and vocabulary might have big differences between countries. For another example, urban Palestinian and urban Syrian are both Levantine, but are easily distinguished by pronunciation, vocabulary, and the grammar of negation.

Many of the very short segments appear more than once, with different dialect labels. For example, ﻻ lA "no" appears three times, labelled Gulf, Levantine, and North African. This reflects vocabulary overlap between dialects, as well as a small sample bias (less than 300 of the segments are only a single word) since this word could also appear in Egyptian and MSA.

The number of segments of various sizes is shown in table 2. Notice that almost 20% of the segments are less than 40 characters long, but less than 2% of the data is in these segments. Similarly only 10% of the segments are greater than 520 characters, but 38% of the training data is in these segments, with 18017 characters, 1% of the data, in a single segment.

The most common segment size is seven characters, with 61 occurences, slightly less than 1% of the segments. Three and five character segments are in second and third place, with 3.4% of all segments less than eight characters in length. One might expect that segmental structure would be an unreliable feature for small segments.

In contrast, the test data for the shared task, also shown in Table 2, has less than 8% of the segments less than 40 characters in size. The largest segment is 2454 characters, the mean is 239 characters. There are only 21 segments, or 1.4% less than 10 characters in length. The twenty commonest sizes are all larger than 90 characters. This is much more suitable for discovering features in segments, but doesn't perfectly match the training data.

segment size	number of segments		total characters in all in this range	
	training	test	training	test
1 - 40	1510	119	30027	2576
41 - 80	1063	169	64032	10593
81 - 120	898	224	89242	23002
121 - 160	702	227	98197	31975
161 - 200	556	219	99443	39220
201 - 240	512	145	112377	31748
241 - 280	387	110	100655	28418
181 - 320	327	60	98136	18192
321 - 360	251	51	85298	17276
360 - 400	215	23	81505	8759
401 - 440	167	21	70047	8724
441 - 480	155	15	71291	6904
481 - 520	114	14	57125	7012
521 - ∞	762	143	651156	132073
sums	7619	1540	1708531	366472

Table 2: Training and Test data by segment size

3.2 Character N-gram feature vectors

The N-gram components of the sequence of characters generated from a sentence S can be represented as a D-dimensional vector p where, D is the number of all N-grams, C_j is the jth N-gram and the probability p_j of Cj is estimated using counts of N-grams,

$$p_j = \frac{count(C_j)}{\sum_i count(C_i)} \tag{1}$$

Where the sum in (1) is performed over all N-grams and $Count(C_j)$ is the number of times the N-gram component C_j occurs in the produced sequence of tokens.

Assuming p^{tar} and p^{bkg} are probability vectors of the target and background Arabic dialects respectively, the SVM can be applied to find a separating hyperplane h by using different kinds of kernel functions. The most commonly used SVM kernels are the Gaussian and the polynomial. The simple linear dot-product kernel is used in this system because other kernels gave no improvement.

3.3 Weighting

Before applying SVM, the generated probabilities vectors, p_j, are weighted to emphasize the most discriminative components (i.e. those which occur frequently in one dialect and infrequently in others). The

N-gram components which are common in most dialects, such as common characters or words, contain little discriminative information and are de-emphasized. Numerous weighting techniques are available for this purpose, such as the Inverse Document Frequency (IDF) from Information Retrieval (IR), Usefulness from Topic Spotting and Identification, and the Log-Likelihood Ratio (LLR) weighting technique proposed in (Campbell et al., 2007).

The LLR weighting w_j for component C_j is given by:

$$w_j = g\left(\frac{1}{P(C_j/all)}\right) \tag{2}$$

Here $g()$ is a function used to smooth and compress the dynamic range (for example, $g(x) = \sqrt{x}$, or $g(x) = \log(x) + 1$). $p(C_j/all)$ is the probability of N-gram component C_j across all dialects.

The components which have zero occupancy in all dialects are removed since they do not carry any useful information. A benefit of discarding these non-visited components is that it reduces the feature dimension dramatically, particularly for the high order N-gram system as the dimension of the N-gram increases exponentially (M^n), where M is the number of distinct Buckwalter Arabic transcription characters in the data set ($M = 51$ for the training data.)

Those N-gram components which have a very small probability have a very high weighting, allowing a minority of components to dominate the scores. To prevent this, a minimum threshold T1 on the weighting w_j was applied. According to Zipfs law, the rank-frequency distribution of words in a typical document follows a decaying exponential. The high ranking words with high probability are not useful for discrimination because they appear in most of the documents. Conversely, the low-rank words are too rare to gather useful statistical information. The area of interest is somewhere in the middle. To address this we apply a second, maximum, threshold T2 on the weighting vector to deemphasize the common components. The values of T1 and T2 were determined empirically on the training data set.

3.4 Feature Selection

In addition to the weighting and thresholds described in the above sub-section, a feature selection technique is needed to minimize the number of N-gram components by keeping only those which are most discriminative. This is particularly necessary in high-order N-gram systems because the dimension is increased exponentially. Consequently, reducing the number of N-gram components decreases the computational cost and the required amount of memory.

A powerful feature selection technique based on the information entropy is applied to all n-gram feature vectors.

4 The systems

Six different systems were investigated during development; test runs were submitted for three of these. In addition, after the testing was completed, we ran some experiments on three additional SVM variants.

- Two of the systems, discussed in section 4.1, are based on the same set of extracted character n-grams. One run was submitted from these two systems, an SVM based on a subset of 3-gram character sequences.

- Our second run is the output of a system based on word frequencies (section 4.3.)

- Two neural network models were built (section 4.2.) Neither appears as a stand-alone run, but their output is incorporated into into the input for the system used for our final run.

- A neural network system (section 4.4) was built to combine the word and neural-network models, and this system was used for our third run.

4.1 Characteristics of the SVM systems

In developing these system, the provided training data was split, with 70% going into a training set, and 30% retained for validation and testing.

For the submitted system, the model was built using the WEKA tool (Hall et al., 2009).

The model was trained on all training tri-gram data set using 10-fold cross validation on SVM classifier after doing feature selection using information gain.

This system gave our best run on the test data, with an accuracy of 0.4279220779 and a weighted F1 score of 0.4264282701. It performed much better against the reserved test data, achieving an accuracy rate on that set of 57% with character trigram features.

4.2 LSTM systems

For the LSTM and word-feature systems, we chose a different split of the training data, into 90% training, 5% validation, and 5% test.

Our LSTM system is based on the char_rnn software described in (Karpathy et al., 2016). This is a character-based language model for text, built with a neural net.

The char_rnn software implements a Long Short Term Memory recurrent neural network (LSTM RNN) which reads the test or training data one character at a time. In training, the error function is effectively the entropy of the next character; in testing, the same network is repeatedly used to predict the most likely next character, resulting in excerpts in the style of the training data.

We modified the program[1], which is written in the Torch language (Collobert et al., 2011) so that it classifies text samples, instead of predicting the next character.

We developed two LSTM RNN models. Both have one-of-48 character inputs, and the hidden (LSTM) layers have forget gates, etc. at each node. As in other RNN models, the state of each node in a hidden layer is an input to the next input step, so that at each character input, not only the current input character, but the past input history affects the output.

We specify a maximum sequence length n in training, and at each training step the past states of the neural net are *unrolled*, so that internal states of the hidden layers for past states are considered while adjusting the parameters being trained.

Our better-performing LSTM has two hidden layers of 128 nodes each. This amounts to 223877 parameters to train. It was trained with a maximum sequence length of 520 characters, so during training up to 520 complete past states of the neural net need to be retained. (520 was chosen as the 90th percentile of sizes from the training data.) In addition, Karpathy's code was built to take advantage of parallel processing in a graphics controller, and handles batches of sequences at a time. Typically, at each training step a batch of sequences, all the same length, would be processed through a single character of input, and the single set of parameters used by all batches would be adjusted to optimize the loss function for the current character, given that changes in the parameters would have affected previous states.

Although we did not use a graphics controller, we kept the batch structure, which averages out the parameter changes, and reduces the training time per segment, since all the segments in a batch contribute to the training step.

We trained with a maximum batch size of 50, but given the training data, such large batches occurred only for small sequence sizes.

Our training technique was to check the loss function for the validation set every thousand training steps, roughly seven times an epoch. When the validation loss began to diverge, we'd reduce the learning rate and continue to train from the last good point. Our best loss function is 1.3176 (compare below.) This gave us an accuracy rate on the reserved test data of 0.4368.

We also experimented with a three hidden layer LSTM, again with 128 nodes in each hidden layer. The number of parameters in this LSTM is 355973, and this proved to be an issue. For longer sequences, there was not enough memory available to keep copies of the state for each character in the sequence for modest batch sizes. It proved necessary to train with a smaller batch size (25) and a smaller sequence

[1] Our changes are available at `https://github.com/StephenETaylor/varDialNN`

length (420.) For whatever reason, this network did not converge as well. We achieved a best loss function of 1.4369.

4.3 Outlier unigram system

This system uses word features, in the hope that they would be independent of the character features other models were using. It goes through the training data, and builds frequencies for uni-grams, bi-grams, and tri-grams for the whole set and for each dialect.

For n-grams which occur five or more times in the training set, it estimates by how many standard deviations the count in each dialect diverges from the expected mean, assuming a binomial distribution.

An advantage feature of this model is that it gives an intermediate result which is easily interpretable. The list of common and uncommon words is interesting, and probably forms a big part of how a human would make the distinction between dialect samples. Of course, many of the most-divergent words are named entities. The commonness of place names in language samples supposedly tied to geographic areas isn't surprising, but there's no reason that a discussion of Casablanca shouldn't be carried out in Gulf Arabic, so the fact that it happens not to have been doesn't convey deep information about the dialect.

Tallying standard deviations of words in a sample as positive and negative votes for whether the sample is in a dialect turns out to be very effective.

Using unigrams alone, it gave an accuracy rate on the 5% reserved test data of 0.5103.

Before considering how to merge in bigram and trigram data, we turned to attempting to combine the results from this program with the output of our two LSTM models.

4.4 System Combiners

As soon as it became clear that the two-layer LSTM was nearing a local if not global maximum performance, we looked for models with independent errors which could be combined with it (or replace it.)

The three-layer LSTM and two-layer LSTM use a soft-max function to determine the dialect; we arranged to normalize and output the assigned probabilities for each dialect from the two LSTMs and the unigram frequency model. A Python script was written to combine the probabilities from each model with addition. This is the plurality model of Malmasi and Dras (2015).

However, the normalized output of the section 4.3 model was too extreme. The rankings were not probabilities and after normalization, "probabilities" came out to either almost zero or almost one. We tried various rescaling functions to get the "probabilities" better distributed, without hurting the accuracy rate. This helped the voting work better, but not significantly. And it seemed logical that the two LSTMs, which have lower accuracy, should be down-weighted somehow.

Therefore we trained a simple neural network on the validation data. This combiner network has 15 inputs – the dialect weights from each system, one hidden layer of 10 nodes, and 5 softmax outputs. This system accepted the outputs from the two LSTM systems and the outlier system, and gave us an accuracy rate of 0.57 on our reserved training data.

4.5 Post-test experiments

After the results of the test runs came back, we conducted some experiments to see whether using all 7619 segments of the training data would have made any difference to our SVM models.

Three different orders of n-grams, 2, 3 and 4-gram, are used to model the sentences of training and testing data set. The n-gram feature vectors produced from training data are used to train Multi-class SVM model. This results in three SVMs; 2, 3 and 4-gram based models.

After removing the n-gram components with zero counts over all training data, the dimensions of resulted feature vectors are 2601, 13656 and 82790 for 2-gram, 3-gram and 4-gram, respectively.

Three SVM systems were trained on bigram features, trigram features and 4-gram features respectively, and evaluated on testing data (1540 sentences.) The bigram system achieved a score of 515 out of 1540 correct, or 0.3344. The trigram system's score was 597 out of 1540, or 0.3877. The 4-gram system's score was 649 out of 1540 or 0.4214, essentially the same as our best submitted system.

5 Results

The results are rather modest – substantially above the baseline, but significantly below the best systems.

In the discussion below, we attempt to address why our test results in the shared task were so far below our test results during development.

Herewith the judgement on our runs:

Run	Accuracy	F1 (micro)	F1 (macro)	F1 (weighted)
SVM	0.4279	0.4279	0.4257	0.4264
unigram outliers	0.3948	0.3948	0.3462	0.3409
combined unigram and LSTMs	0.4091	0.4091	0.4112	0.4117

Table 3: Results for test set C (closed training).

6 Discussion and ideas for future work

There are a few possible reasons our results on the training data were better than our results on the test data. Dealing with some of them might have made our systems not only more robust, but perform better in both contexts.

For the two SVM systems, we split the training data into 70% training data, 30% evaluation data. For the LSTM and word-feature systems, we chose a different split of the training data, into 90% training, 5% validation, and 5% test. A consequence of the small amount of validation and testing data is a fairly large standard deviation in the test results, not observed during our testing, but perhaps apparent in the difference between results with the reserved training data (380 samples) and the workshop test data (1540 samples.) With accuracy/error rates about 0.50 we'd expect standard deviations of 8.5 out of 380 or 2.2% and 20 out of 1540 or 1.2%, respectively. While a jack-knife approach to cross-validation might have given us a better judgement for the mean accuracy achieved by this technique, the time involved in re-training the LSTM would have been substantial.

There are numerous experiments that we did not carry out, which might also have improved our development.

- Experimenting with shorter sequence lengths *should* have sped up training our LSTM systems. This would have let us experiment more with different configurations.

- Seeking out a system or cloud system with a graphics coprocessor could have also sped up neural network development.

- In general, we did not experiment with many constants, but chose them based on plausibility.

- Our word and neural net systems are both attentive to segment size, and the distribution of segment sizes in the test data is different than the training data.

- In spite of its surprising effectiveness on the reserved training data, our unigram word system isn't well-thought-out. A better mathematical foundation for combining standard deviations of words is the cumulative probability distribution of the normal curve. Adding logs of the cpdf is equivalent to multiplying probabilities, and seems mathematically justified, whereas adding positive and negative deviations is very ad-hoc.

  Even using the cpdf will over-emphasize variability in common words, however. Reviewing the

standard deviations of words, we see that, for example, يعني yEny "that is", which occurs in MSA in the training data 16 standard deviations less frequently than might be expected from its occurrence in the whole corpus, still has 190 occurrences in 917 MSA training segments, and a frequency of $190/44932 = 0.004$ in MSA data. So the presence of *yEny* in a test segment, while interesting, is nowhere near as exciting for ruling out MSA as its standard deviation indicates.

In fact, the second and fourth 'most unusually frequent' words in the MSA training data, respectively 18.2 and 16.3 standard deviations more common than expected, are الفلسطينية AlflsTynyp "female Palestinian" and الإسرائيلية Al<srA}ylyp "female Israeli". These words are probably topical, rather than typical of MSA. The frequency of *AlflsTynyp* in the MSA training data is about 0.002 (87 occurences) – and once in EGY, twice in GLF, eleven times in LAV. Judging from the metadata labels in the original dataset, it is used in six or more different stories in MSA. *AlflsTynyp* occurs twice in the training data we reserved for testing. In the test data, it occurs twice in EGY, seven times in LAV, and seven times in MSA.

Similarly, *Al<srA}ylyp* occurs 45 times in the MSA training data and five times in all other dialects, and occurs three times in the reserved testing data. In the test data it occurs once in EGY, 8 times in LAV, 6 times in MSA.

It seems plausible that this story-topic effect may apply to other words in the training data, and that this alone might be sufficient to account for a fall-off in the performance of our software on the test task.

- Our procedure for splitting the training data was non-random, so that evaluation, verification and test data may have shared common prefixes, since the training sentences were sorted.

- We should have used a common split of the training data for all systems, so that the SVM systems could be combined with the others. As things stand, testing data reserved for one set of systems overlaps training data for others.

In the months to come, we hope to use the training and test data from the workshop to carry out some of the experiments we did not do in time to present. We owe a big 'thank you' to the organizers for giving us this opportunity!

References

Hassan Al-Haj, Roger Hsiao, Ian Lane, Alan W. Black, and Alex Waibel. 2009. Pronunciation modeling for dialectal Arabic speech recognition. In *Automatic Speech Recognition and Understanding Conference*.

Mohammed Ali, Moustafa Elshafei, Mansour Al-Ghamdi, Husni Al-Muhtaseb, and Atef Al-Najjar. 2008. Generation of Arabic phonetic dictionaries for speech recognition. In *International Conference on Innovations in Information Technology, 2008.*, pages 59–63. IEEE.

Ahmed Ali, Najim Dehak, Patrick Cardinal, Sameer Khurana, Sree Harsha Yella, James Glass, Peter Bell, and Steve Renals. 2016. Automatic dialect detection in Arabic broadcast speech. In *Proceedings of Interspeech 2016*, pages 2934–2938.

Houda Bouamor, Nizar Habash, and Kemal Oflazer. 2014. A multidialectal parallel corpus of Arabic. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC14)*. European Language Resources Association (ELRA), May.

Tim Buckwalter. 2002. Aramorph 1.0 program.

William M. Campbell, Joseph P. Campbell, Terry P. Gleason, Douglas A. Reynolds, and Wade Shen. 2007. Speaker verification using support vector machines and high-level features. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(7):2085–2094.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. 2011. Torch7: A MATLAB-like environment for machine learning. In *BigLearn, Neural Information Processing Systems Workshop*.

Heba ElFardy and Mona Diab. 2013. Sentence level dialect identification in Arabic. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 456–461.

Mohamed Elmahdy, Rainer Gruhn, Wolfgang Minker, and S. Abdennadher. 2010. Cross-Lingual Acoustic Modeling for Dialectal Arabic Speech Recognition. In *International Conference on Speech and Language Processing (Interspeech)*, September.

Mohamed Elmahdy, Mark Hasegawa-Johnson, and Eiman Mustafawi. 2012. A baseline speech recognition system for Levantine colloquial Arabic. In *12th ESOLEC conference on Language Engineering*.

Mohamed Elmahdy, Mark Hasegawa-Johnson, and Eiman Mustafawi. 2013. A transfer learning approach for under-resourced Arabic dialects speech recognition. In *The 6th Language and Technology Conference*.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1).

Abualsoud Hanani, Martin J. Russell, and Michael J. Carey. 2013. Human and computer recognition of regional accents and ethnic groups from British english speech. *Computer Speech and Language*, 27(1):5974.

Abualsoud Hanani, Hanna Basha, Yasmeen Sharaf, and Stephen Taylor. 2015. Palestinian Arabic regional accent recognition. In *The 8th International Conference on Speech Technology and Human-Computer Dialogue*.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2016. Visualizing and understanding recurrent networks. In *5th International Conference on Learning Representations*.

Shervin Malmasi and Mark Dras. 2015. Language identification using classifier ensembles. In *Proceedings of the Joint Workshop on Language Technology for Closely Related Languages, Varieties and Dialects (LT4VarDial)*, pages 35–43, Hissar, Bulgaria.

Shervin Malmasi, Eshrag Refaee, and Mark Dras. 2015. Arabic dialect identification using a parallel multidialectal corpus. In *Proceedings of the 14th Conference of the Pacific Association for Computational Linguistics (PACLING 2015)*, pages 209–217, Bali, Indonesia, May.

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 3rd Workshop on Language Technology for Closely Related Languages, Varieties and Dialects (VarDial)*, Osaka, Japan.

Maryam Najafian, Andrea DeMarco, Stephen Cox, and Martin Russell. 2014. Unsupervised model selection for recognition of regional accented speech. In *Proceedings of Interspeech 2014*.

Scott Novotney, Rich Schwartz, and Sanjeev Khudanpur. 2011. Unsupervised Arabic dialect adaptation with self-training. In *Proceedings of Interspeech 2011*, pages 1–4.

Jordi Porta and José-Luis Sancho. 2014. Using maximum entropy models to discriminate between similar languages and varieties. In *Proceedings of the First Workshop on Applying NLP Tools to Similar Languages, Varieties and Dialects (VarDial)*, pages 120–128, Dublin, Ireland.

Dimitra Vergyri, Katrin Kirchhoff, Venkata Raman Rao Gadde, Andreas Stolcke, and Jing Zheng. 2005. Development of a conversational telephone speech recognizer for Levantine Arabic. In *Proceedings of Interspeech 2005*, pages 1613–1616.

Omar F. Zaidan and Chris Callison-Burch. 2011. The Arabic Online Commentary dataset: An annotated dataset of informal Arabic with high dialectal content. In *Proceedings of ACL*, pages 37–41.

Omar F. Zaidan and Chris Callison-Burch. 2014. Arabic dialect identification. *Computational Linguistics*, 40(1):171–202.