A Fast, Compact, Accurate Model for Language Identification of Codemixed Text

Yuan Zhang, Jason Riesa, Daniel Gillick, Anton Bakalov, Jason Baldridge, David Weiss
Google AI Language
{zhangyua, riesa, dgillick, abakalov, jridge, djweiss}@google.com

Abstract

We address fine-grained multilingual language identification: providing a language code for every token in a sentence, including codemixed text containing multiple languages. Such text is prevalent online, in documents, social media, and message boards. We show that a feed-forward network with a simple globally constrained decoder can accurately and rapidly label both codemixed and monolingual text in 100 languages and 100 language pairs. This model outperforms previously published multilingual approaches in terms of both accuracy and speed, yielding an 800x speed-up and a 19.5% averaged absolute gain on three codemixed datasets. It furthermore outperforms several benchmark systems on monolingual language identification.

1 Introduction

Codemixed text is common in user-generated content, such as web articles, tweets, and message boards, but most current language ID models ignore it. Codemixing involves language switches within and across constituents, as seen in these English-Spanish and English-Hindi examples.

(1) dame [$_{NP}$ ese book that you told me about]
Give me this book that you told me about.

(2) [$_{NP}$ aapki profile photo] [$_{VP}$ pyari hai]
Your profile photo is lovely.

Codemixing is the norm in many communities, e.g. speakers of both Hindi and English. As much as 17% of Indian Facebook posts (Bali et al., 2014) and 3.5% of all tweets (Rijhwani et al., 2017) are codemixed. This paper addresses fine-grained (token-level) language ID, which is needed for many multilingual downstream tasks, including syntactic analysis (Bhat et al., 2018), machine translation and dialog systems. Consider this example, which seeks a Spanish translation for the English word *squirrel*:

(3) como se llama un squirrel en español
What do you call a squirrel in Spanish?

Per-token language labels are needed; a system cannot handle the whole input while assuming it is entirely English or Spanish.

Fine-grained language ID presents new challenges beyond sentence- or document-level language ID. Document-level labels are often available in metadata, but token-level labels are not. Obtaining token-level labels for hundreds of languages is infeasible: candidate codemixed examples must be identified and multilingual speakers are required to annotate them. Furthermore, language ID models typically use character- and word-level statistics as signals, but shorter inputs have greater ambiguity and less context for predictions. Moreover, codemixing is common in informal contexts that often have non-standard words, misspellings, transliteration, and abbreviations (Baldwin et al., 2013). Consider (4), a French-Arabic utterance that has undergone transliteration, abbreviation and diacritic removal.

(4) cv bien hmd w enti
ça va bien **al**h**a**md**ullilah** w**a** enti
ça va bien الحمد الله وانت
It's going well, thank God, and you?

Language ID models must be fine-grained and robust to surface variations to handle such cases.

We introduce **CMX**, a fast, accurate language ID model for **C**ode**M**i**X**ed text that tackles these challenges. CMX first outputs a language distribution for every token independently with efficient feed-forward classifiers. Then, a decoder chooses labels using both the token predictions and global constraints over the entire sentence. This decoder

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 328–337
Brussels, Belgium, October 31 - November 4, 2018.

produces high-quality predictions on monolingual texts as well as codemixed inputs. We furthermore show how selective, grouped dropout enables a blend of character and word-level features in a single model without the latter overwhelming the former. This dropout method is especially important for CMX's robustness on informal texts.

We also create synthetic training data to compensate for the lack of token-level annotations. Based on linguistic patterns observed in real-world codemixed texts, we generate two million codemixed examples in 100 languages. In addition, we construct and evaluate on a new codemixed corpus of token-level language ID labels for 25k codemixed sentences (330k tokens). This corpus contains examples derived from user-generated posts that contain English mixed with Spanish, Hindi or Indonesian.

Language ID of monolingual text has been extensively studied (Hughes et al., 2006; Baldwin and Lui, 2010; Lui and Baldwin, 2012; King and Abney, 2013), but language ID for codemixed text has received much less attention. Some prior work has focused on identifying larger language spans in longer documents (Lui et al., 2014; Jurgens et al., 2017) or estimating proportions of multiple languages in a text (Lui et al., 2014; Kocmi and Bojar, 2017). Others have focused on token-level language ID; some work is constrained to predicting word-level labels from a single language pair (Nguyen and Doğruöz, 2013; Solorio et al., 2014; Molina et al., 2016a; Sristy et al., 2017), while others permit a handful of languages (Das and Gambäck, 2014; Sristy et al., 2017; Rijhwani et al., 2017). In contrast, CMX supports 100 languages. Unlike most previous work–with Rijhwani et al. 2017 a notable exception–we do not assume a particular language pair at inference time. Instead, we only assume a large fixed set of language pairs as a general constraint for all inputs.

We define and evaluate CMX and show that it strongly outperforms state-of-the-art language ID models on three codemixed test sets covering ten languages, and a monolingual test set including 56 languages. It obtains a 19.5% absolute gain on codemixed data and a 1.1% absolute gain (24% error reduction) on the monolingual corpus. Our analysis reveals that the gains are even more pronounced on shorter text, where the language ID task naturally becomes more difficult. In terms of runtime speed, CMX is roughly 800x faster than existing token-level models when tested on the same machine. Finally, we demonstrate a resource-constrained but competitive variant of CMX that reduces memory usage from 30M to 0.9M.

2 Data

We create synthetic codemixed training examples to address the expense and consequent paucity of token-level language ID labels. We also annotate real-world codemixed texts to measure performance of our models, understand code-mixing patterns and measure the impact of having such examples as training data.

Synthetic data generation from monolingual text. For training models that support hundreds of languages, it is simply infeasible to obtain manual token-level annotations to cover every codemixing scenario (Rijhwani et al., 2017). However, it is often easy to obtain sentence-level language labels for monolingual texts. This allows projection of sentence-level labels to each token, but a model trained only on such examples will lack codemixed contexts and thus rarely switch within a sentence. To address this, we create synthetic training examples that mix languages within the same sequence.

To this end, we first collect a monolingual corpus of 100 languages from two public resources: the W2C corpus[1] and the Corpus Crawler project.[2] Then we generate a total of two million synthetic codemixed examples for all languages.

In generating each training example, we first sample a pair of languages uniformly.[3] We sample from a set of 100 language pairs, mainly including the combination of English and a non-English language. The full set is listed in the supplemental material. Then we choose uniformly between generating an *intra-mix* or *inter-mix* example, which are two of the most prominent types of codemixing in the real world (Barman et al., 2014; Das and Gambäck, 2014).[4] An *intra-mix* sentence like (1) starts with one language and switches to another language, while an *inter-mix* sentence like (2) has

[1] http://ufal.mff.cuni.cz/w2c

[2] https://github.com/googlei18n/corpuscrawler

[3] Both our collected codemixed data and Barman et al. (2014) indicate that more than 95% of codemixed instances are bilingual.

[4] The two types of codemixing have roughly equal proportions in our labeled corpus.

	en/es	en/hi	en/id
Number of tokens	98k	140k	94k
Number of sentences	9.5k	9.9k	5.3k

Table 1: Statistics of our YouTube and Google+ dataset, GY-Mix.

Test Set	Languages
Twitter-Mix	en, es
Web-Mix6	cs, en, eu, hu, hr, sk
GY-Mix	en, es, hi, id
KB-Mono56	56 languages

Table 2: The languages of each testing corpora in our experiments. The first three sets primarily include codemixed texts while the last one (KB-Mono56) is monolingual.

an overall single language with words from a second language in the middle. To generate an example, we uniformly draw phrases from our monolingual corpus for the chosen target languages, and then concatenate or mix phrases randomly. The shorter phrase in *inter-mix* examples contains one or two tokens, and the maximum length of each example is eight tokens.

Manual annotations on real-world codemixed text. We obtain candidates by sampling codemixed public posts from Google+[5] and video comments from YouTube,[6] limited to three language pairs with frequent code switching: English-Spanish, English-Hindi[7] and English-Indonesian. All texts are tokenized and lowercased by a simple rule-based model before annotation. Both the candidate selection and the annotation procedures are done by linguists proficient in both languages. The final annotated corpus contains 24.7k sentences with 334k tokens; 30% are monolingual, 67% are bilingual and 3% have more than two languages. Finally, we create an 80/10/10 split (based on tokens) for training, development and testing, respectively. Table 1 gives the token and sentence counts per language. In the rest of the paper, we refer to this dataset as GY-Mix.

Evaluation datasets. We evaluate on four datasets, three codemixed and one monolingual. For a fair comparison, we report accuracies on subsets of these test sets that include languages supported by all tested models. Examples with Hindi words written in Latin script are also removed because the benchmark systems we compare to do not support it.

[5] https://plus.google.com/

[6] https://www.youtube.com/

[7] Hindi texts found in both Devanagari and Latin scripts.

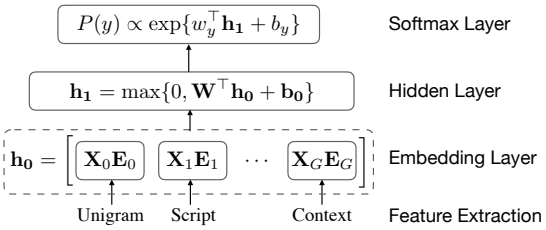

Figure 1: Basic feed-forward network unit for scoring each token in the input and predicting possible languages. Multiple features are embedded, concatenated, and fed into a hidden layer with ReLU activation.

- **Twitter-Mix**: Codemixed data from the EMNLP 2016 shared task (Molina et al., 2016b).
- **Web-Mix6**: Codemixed data crawled from multilingual web pages (King and Abney, 2013), using a subset of six languages.
- **GY-Mix**: The test set of our token-level codemixed data (en-es, en-hi, and en-id).
- **KB-Mono56:** Monolingual test set of Kocmi and Bojar (2017), using a subset of 56 languages.

Table 2 summarizes the final language setting of each test set used in our experiments.

3 Identifying Language Spans in Codemixed Text

CMX uses two stages to assign language codes to every token in a sentence. First, it predicts a distribution over labels for each token independently with a feed-forward network that uses character and token features from a local context window. Then, it finds the best assignment of token labels for an entire sentence using greedy search, subject to a set of global constraints. Compared to sequence models like CRFs or RNNs, this two-stage strategy has several major advantages for fine-grained language ID: (1) it does not require annotated codemixed text over hundreds of languages and their mixed pairings, (2) learning independent classifiers followed by greedy decoding

Features	Window	D	V
Character n-gram	+/- 1	16	1000-5000
Script	0	8	27
Lexicon	+/- 1	16	100

Table 3: Feature spaces of CMX. The window column indicates that CMX uses character n-gram and lexicon features extracted from the previous and following tokens as well as the current one.

is significantly faster than structured training (especially considering the large label set inherent in language ID), and (3) it is far easier to implement.

3.1 Token Model

Simple feed-forward networks have achieved near state-of-the-art performance in a wide range of NLP tasks (Botha et al., 2017; Weiss et al., 2015). CMX follows this strategy, with embedding, hidden, and softmax layers as shown in Figure 1. The inputs to the network are grouped feature matrices, e.g. character, script and lexicon features. Each group g's features are represented by a sparse matrix $\mathbf{X}_g \in \mathbb{R}^{F_g \times V_g}$, where F_g is the number of feature templates and V_g is the vocabulary size of the feature group. The network maps sparse features to dense embedding vectors and concatenates them to form the embedding layer:

$$\mathbf{h}_0 = vec[\mathbf{X}_g \mathbf{E}_g | \forall g] \quad (1)$$

where $\mathbf{E}_g \in \mathbb{R}^{V_g \times D_g}$ is a learned embedding matrix per group. The final size of the embedding layer $|\mathbf{h}_0| = \sum_g F_g D_g$ is the sum of all embedded feature sizes. CMX uses both discrete and continuous features. We use a single hidden layer with size 256 and apply a rectified linear unit (ReLU) over hidden layer outputs. A final softmax layer outputs probabilities for each language. The network is trained per-token with cross-entropy loss.

We explain the extraction process of each feature type below. Table 3 summarizes the three types of features and their sizes used in CMX. Character and lexicon features are extracted for the previous and following tokens as well as the current token to provide additional context.

Character n-gram features We apply character n-gram features with $n = [1, 4]$. RNNs or CNNs would provide more flexible character feature representations, but our initial experiments did not show significant gains over simpler n-gram features. We use a distinct feature group for each n. The model averages the embeddings according to the fractions of each n-gram string in the input token. For example, if the token is *banana*, then one of the extracted trigrams is *ana* and the corresponding fraction is 2/6. Note that there are six trigrams in total due to an additional boundary symbol at both ends of the token.

Following Botha et al. (2017), we use feature hashing to control the size V and avoid storing a big string-to-id map in memory during runtime. The feature id of an n-gram string x is given by $\mathcal{H}(x) \bmod V_g$ (Ganchev and Dredze, 2008), where $\mathcal{H}$ is a well-behaved hash function. We set $V = 1000, 1000, 5000, 5000$ for $n = 1, 2, 3, 4$ respectively; these values yield good performance and are far smaller than the number of n-gram types.

Script features Some text scripts are strongly correlated with specific languages. For example, Hiragana is only used in Japanese and Hangul is only used in Korean. Each character is assigned one of the 27 types of scripts based on its unicode value. The final feature vector contains the normalized counts of all character scripts observed in the input token.

Lexicon features This feature group is backed by a large lexicon table which holds a language distribution for each token observed in the monolingual training data. For example, the word *mango* occurs 48% of the time in English documents and 18% in Spanish ones. The table contains about 6.2 million entries. We also construct an additional prefix table of language distributions for 6-gram character prefixes. If the input token matches an entry in the lexicon table (or failing that, the prefix table), our model extracts the following three groups of features.

- *Language distribution*. The language distribution itself is included as the feature vector.
- *Active languages*. As above, but feature values are set to 1 for all non-zero probabilities. For example, the word *mango* has feature value 1 on both English and Spanish.
- *Singletons*. If the token is associated with only one language, return a one-hot vector whose only non-zero value is the position indicating that language.

The size of all lexicon feature vectors is equal to the number of supported languages.

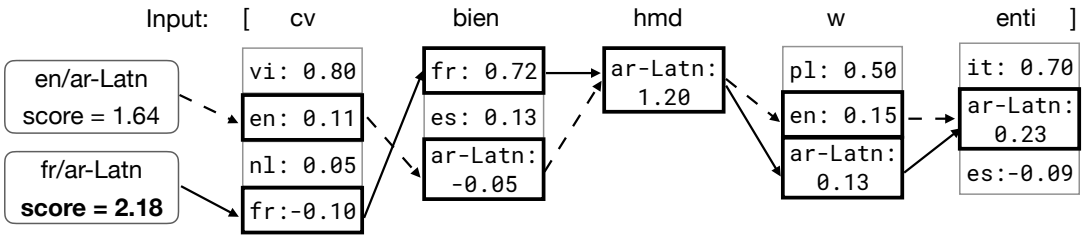

Figure 2: Example of our decoding algorithm with global constraints for example (4) for two allowed language pairs, en/ar-Latn and fr/ar-Latn.

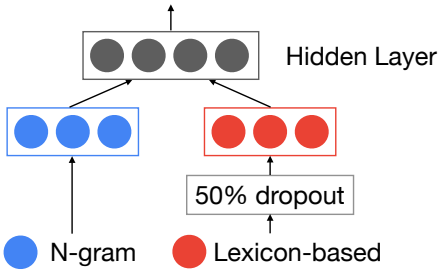

Figure 3: Our selective feature dropout method. The model randomly sets the lexicon feature vectors to zero with 50% probability while n-gram features are always used.

3.2 Selective Feature Dropout

Preliminary experiments showed poor performance, especially on informal texts, when all three types of features are simply merged. Consider the following example outputs on misspelled word *Ennnglish*, for which no lexicon features fire.

Input: *Ennnglish*

With Lexicon Features	*W/o Lexicon Features*
• $p(sv) = 0.27$	• $p(en) = 0.74$
• $p(da) = 0.24$	• $p(nl) = 0.10$
• $p(nl) = 0.18$	• $p(fy) = 0.06$
• ...	• ...

Without dropout, the model with lexicon features does not make effective use of the token's character n-grams and makes a catastrophically wrong prediction. The core problem is that lexicon features are both prevalent and highly predictive for language ID; during training, this dampens the updating of weights of n-gram features and thus diminishes their overall utility.

To address this issue and make CMX more robust to noisy inputs, we selectively apply a grouped feature dropout strategy that stochastically down-weights lexicon features during training. Figure 3 illustrates the idea: for each input, after feature extraction, the vector of lexicon features is randomly set to zero. This way, the model must rely entirely on n-gram features for this particular input. Note that our feature dropout is different from standard dropout in at least two ways: (1) dropout happens to entire feature groups rather than on individual neurons, (2) we selectively apply dropout only on a subset of features. After tuning the dropout rate on development data (Figure 5) we choose a dropout rate of 50%. Section 4.3 explains the tuning procedure.

3.3 Decoding with Global Constraints

Given a trained model, the goal of decoding is to find the sequence of per-token languages that maximizes the overall score. The simple, greedy strategy of picking the top prediction for each token over-predicts too many languages in a single sentence. For example, on average the greedy method predicts more than 1.7 languages per sentence on monolingual inputs. Because the token classier uses a window including only the previous, current, and next token, it has a quite limited view on the entire sequence.

Motivated by this observation, we add the following **global constraint** in decoding: only monolingual outputs or codemixed outputs from a fixed set of language pairs are permitted. We choose a set of 100 language pairs, primarily including the combination of English and a non-English language. The full set is listed in the supplemental material.

Finally, we introduce a straightforward variant of greedy decoding that finds the optimal language assignment in the presence of these global constraints. We independently find the best assignment under each allowed language combination (monolingual or language pair) and return the one with the highest score.

Figure 2 shows paths for example (4) with two allowed language pairs: en/ar-Latn and

DATASET	Twitter-Mix	Web-Mix6	GY-Mix				Average
LANGUAGES	es/en	6 Langs	es/en	hi/en	hi/hi-Latn/en	id/en	
LanideNN	71.3	52.1	65.7	79.6	–	22.9	58.3
EquiLID	87.9	63.5	71.0	81.9	–	64.9	73.9
CMX-small	88.8	91.0	89.9	98.2	85.0	86.7	90.9
CMX	**92.4**	**93.2**	**91.8**	**98.4**	**87.4**	**91.1**	**93.4**

Table 4: **Codemixed Texts:** Token-level accuracy (%) of different approaches on codemixed texts. "CMX-small" corresponds to our small model without lexicon features and vocabulary tables. The *hi/hi-Latn/en* column shows the accuracy on texts in English, Latin Hindi and Devanagari Hindi; the baseline models do not support identification of text in Hindi in Latin script. *Average* shows averaged accuracy on all sets except *hi/hi-Latn/en*. Boldface numbers indicate the best accuracy for each testing set.

fr/ar-Latn.[8] The two paths in dashed and solid lines indicate the best assignment for each language pair respectively. Because scoring is independent across tokens, each subtask is computed in $O(N)$ time. The total decoding time is $O(N|\mathcal{L}|)$ where $\mathcal{L}$ is the constraint set, and the global optimality of this algorithm is guaranteed because the assignment found in each subtask is optimal.

4 Experiments

4.1 Training Setup

We train CMX on the concatenation of three datasets: (a) GY-Mix's training portion, (b) synthetic codemixed data and (c) a monolingual corpus that covers 100 languages. Every token in the training set spawns a training instance. Our training set consists of 38M tokens in total, which is on the same magnitude as the sizes of training data reported in previous work (Jurgens et al., 2017; Joulin et al., 2016).

We use mini-batched averaged stochastic gradient descent (ASGD) (Bottou, 2010) with momentum (Hinton, 2012) and exponentially decaying learning rates to learn the parameters of the network. We fix the mini-batch size to 256 and the momentum rate to 0.9. We tune the initial learning rate and the decay step using development data.

4.2 Main Results

Codemixed Texts Table 4 lists our main results on the codemixed datasets. We primarily compare our approach against two benchmark systems: EquiLID (Jurgens et al., 2017) and LanideNN (Kocmi and Bojar, 2017). Both achieved state-of-the-art performance on several monolingual and codemixed language ID datasets. LanideNN makes a prediction for every character, so we convert its outputs to per-token predictions by a voting method over characters in each word. For both benchmarks, we use the public pre-trained model provided by the authors. The EquiLID model uses 53M parameters, LanideNN uses 3M, and CMX only uses 0.28M parameters.[9]

Across all datasets, CMX consistently outperforms both benchmark systems by a large margin. On average, our full model (CMX) is 19.5% more accurate than EquiLID (93.4% vs. 73.9%); the gain is even larger compared to LanideNN. Note that none of the models are trained on the Twitter-Mix or the Web-Mix6 dataset, so these two datasets provide an evaluation on the out-domain performance of each approach. In this setting CMX also yields significant improvement in accuracy, e.g. a 4.5% (absolute) gain over EquiLID on the Twitter-Mix dataset.

An Even Smaller Model We further compare between CMX and a variant we call CMX-small, which has no access to lexicon resources or lexicon features. This smaller variant has only 237k parameters and reduces the memory footprint from 30M to 0.9M during runtime, while the (average) loss on accuracy is only 2.5%. This comparison demonstrates that our approach is also an excellent fit for resource-constrained environments, such as on mobile phones.

Monolingual Texts In addition to EquiLID and LanideNN, we further compare CMX against

[8] Scores are sorted. Some languages omitted for illustration purposes.

[9] We explain how we compute the number of parameters of our model in the supplemental material.

Model	Sent Acc.	Char/Sec
Codemixing Models		
LanideNN	94.6	0.17k
EquiLID	95.1	0.25k
CMX-small	94.6	265.5k
CMX	**96.6**	206.1k
Monolingual Models		
Langid.py	92.8	183.8k
fastText-small	92.5	2,671.1k
fastText-full	94.4	2,428.3k
CLD2	95.5	4,355.0k

Table 5: **Monolingual Texts:** Sentence-level accuracy (%) on KB-Mono56. Monolingual models make per-sentence predictions only.

Langid.py (Lui and Baldwin, 2012), CLD2[10] and fastText (Joulin et al., 2016, 2017)—all are popular off-the-shelf tools for monolingual language ID. Sentence-level predictions for EquiLID and LanideNN models are obtained by simple voting. Table 5 summarizes sentence-level accuracy of different approaches on the KB-Mono56 test set. CMX achieves the best sentence-level accuracy over all monolingual and codemixing benchmark systems. The resource-constrained CMX-small also performs strongly, obtaining 94.6% accuracy on this test set.

Our approach also maintains high performance on very short texts, which is especially important for many language identification contexts such as user-generated content. This is demonstrated in Figure 4, which plots the cumulative accuracy curve on KB-Mono56 over sentence length (as measured by the number of non-whitespace characters). For example, points at x=50 show the averaged accuracies over sentences with no more than 50 characters. We compare CMX against the best performing monolingual and codemixing benchmark systems. The relative gain is more prominent on shorter sentences than on longer ones. For example, the improvement is 4.6% on short sentences ($\leq$30 characters), while the gain on segments $\leq$150 characters is 1.9%. Similar patterns are seen with respect to other systems.

Inference Speed Table 5 also shows the inference speed of each method in characters per second, tested on a machine with a 3.5GHz Intel Xeon processor and 32GB RAM. CMX (written in C++) is far faster than other fine-grained systems, e.g. it has an 800x speed-up over EquiLID. It is not surprising that monolingual models are faster than CMX, which makes a prediction for every token rather than once for the entire sequence. Of course, monolingual models do not support language ID on codemixed inputs, and furthermore CMX performs the best even on monolingual texts.

[10] https://github.com/CLD2Owners/cld2

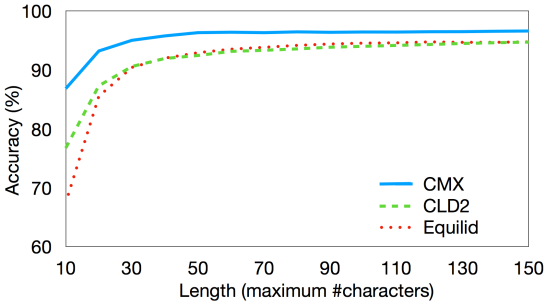

Figure 4: Sentence-level accuracy (y-axis) on KB-Mono56 as a function of the maximum number of non-whitespace characters in a sentence (x-axis). For example, the point at $x = 50$ denotes the accuracy on all the sentences with ≤ 50 characters.

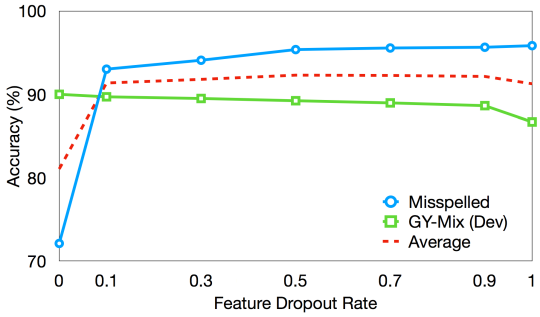

Figure 5: Accuracy on development sets with various feature dropout rate values p.

4.3 Analysis

Feature Dropout Rate To analyze how the feature dropout rate impacts the model performance, we create a set of *synthetically misspelled* tokens by random duplication or replacement of one or two characters. In addition, we ensure that every token has at least one language-unique character, so a model with character n-gram features should be able to easily identify the language of this token. Figure 5 shows the tuning results for dropout values on misspelled tokens and the GY-Mix development set. Without feature dropout (p=0.0), our model only gets 72.1% on misspelled tokens,

TRAINING DATA	Twitter-Mix	Web-Mix6	GY-Mix (Test)	KB-Mono56
All Training Corpora	92.4	93.2	93.6	95.1
w/o GY-Mix (Train)	88.5	92.9	89.3	95.0
w/o Synthetic	92.1	88.8	92.5	95.1

Table 6: Token-level accuracy of our full model with different training sets, removing either GY-Mix annotations or synthetic codemixed corpus at a time.

METHOD	GY-Mix	KB-Mono56	#Lang/Sent
Independent	87.8	91.9	1.78
Switching Penalty	89.4	93.1	1.58
Bilingually Constrained	93.6	95.1	1.27
Gold	-	-	1.15

Table 7: Token-level accuracy of different decoding methods on GY-Mix and KB-Mono56, as well as the averaged number of predicted languages in each sentence.

indicating that n-gram features are not properly trained. The proposed feature dropout method effectively addresses this issue, improving the accuracy to 95.3% with $p \geq 0.5$. We choose $p = 0.5$ (Figure 3) because it gives the best trade-off on the two tuning sets. The curves in Figure 5 also show that model performance is robust across a wide range of dropout rates between the two extremes, so the strategy is effective, but is not highly sensitive and does not require careful tuning.

Impact of Decoding Algorithm Table 7 shows a comparison over different decoding strategies, including (a) *independent* greedy prediction for each token, (b) adding a *switching penalty* and decoding with Viterbi, (c) and our *bilingually constrained* decoding. For the second method, we add a fixed transition matrix that gives a penalty score $\log p$ for every code switch in a sentence. We choose $p = 0.5$, which gives the best overall results on the development set. Our approach outperforms *switching penalty* by more than 2% on both GY-Mix and KB-Mono56. To analyze the reason behind this difference we show the average number of languages in each sentence in Table 7. Both baseline approaches on average predict more than 1.5 languages per sentence while the oracle number based on gold labels is only 1.15. Our global bilingual constraints effectively address this over-prediction issue, reducing the average number of predicted languages to 1.27. We also measure the running time of all methods. The decoding speed of our method is 206k char/sec (Table 5), while the *independent* method is 220k char/sec. Our decoding with global constraints thus only increases the running time by a factor of 1.07.

Codemixed Training Datasets Our training data consists of two codemixed corpora: manual annotations on real-world data (GY-Mix) and a synthetic corpus. To analyze their contribution, we remove each corpus in turn from the training set and report the results in Table 6. Adding the GY-Mix training set mainly improves accuracy on GY-Mix test and Twitter-Mix, while the gains from the synthetic data are greatest on Web-Mix6. This shows that synthetic data helps CMX generalize to a broader range of languages since GY-Mix has language overlap only with Twitter-Mix, not Web-Mix6. The two examples below further demonstrate the benefit of synthetic examples:

With Synthetic Data

- [Translate]$_{\text{en}}$ [**maçã**]$_{\text{pt}}$ [to English]$_{\text{en}}$
- [Translate]$_{\text{en}}$ [**Apfel**]$_{\text{de}}$ [to English]$_{\text{en}}$

Without Synthetic Data

- [Translate **maçã** to]$_{\text{pt}}$ [English]$_{\text{en}}$
- [Translate **Apfel** to English]$_{\text{en}}$

Both examples are likely potential queries "Translate *apple* to English" with *apple* replaced by its translation in German(de) or Portuguese(pt). The underlying language pairings never appear in GY-Mix. CMX with synthetic training data is able to correctly identify the single token inter-mixed in a sentence, while the model trained without synthetic data fails on both cases.

Contribution of Features CMX has three types of features: character n-gram, script, and lexicon features. n-gram features play a crucial role as back-off from lexicon features. Consider informal Latin script inputs, like *hellooooo*, for which no lexicon features fire. Foregoing n-gram features results in abysmal performance (<20%) on this type of input because script features alone are inadequate. The main impact of script features is

to avoid embarrassing mistakes on inputs that can be easily identified from their scripts. Finally, note that removing lexicon features corresponds to the CMX-small model. On monolingual inputs (Table 5), the lexicon features in CMX provide a 2.0% absolute improvement in accuracy.

5 Conclusions

CMX is a fast and compact model for fine-grained language identification. It outperforms related models on codemixed and monolingual texts, which we show on several datasets covering text in a variety of languages and gathered from diverse sources. Furthermore, it is particularly robust to the idiosyncrasies of short informal text.

Acknowledgments

We thank Emily Pitler, Slav Petrov, John Alex, Daniel Andor, Kellie Webster, Vera Axelrod, Kuzman Ganchev, Jan Botha, and Manaal Faruqui for helpful discussions during this work, and our anonymous reviewers for their thoughtful comments and suggestions. We also thank Elixabete Gomez, Héctor Alcalde, Knot Pipatsrisawat, and their stellar team of linguists who helped us to annotate and curate much of our data.

References

Timothy Baldwin, Paul Cook, Marco Lui, Andrew MacKinlay, and Li Wang. 2013. How noisy social media text, how diffrnt social media sources? In *Proceedings of the International Joint Conference on Natural Language Processing*, pages 356–364.

Timothy Baldwin and Marco Lui. 2010. Language identification: The long and the short of the matter. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the ACL*, pages 229–237.

Kalika Bali, Jatin Sharma, Monojit Choudhury, and Yogarshi Vyas. 2014. I am borrowing ya mixing, an analysis of english-hindi code mixing in facebook. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 116–126, Doha, Qatar. Association for Computational Linguistics.

Utsab Barman, Amitava Das, Joachim Wagner, and Jennifer Foster. 2014. Code mixing: A challenge for language identification in the language of social media. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 13–24.

Irshad Ahmad Bhat, Riyaz A. Bhat, and Manish Shrivastava. 2018. Universal dependency parsing for hindi-english code-switching. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*.

Jan A Botha, Emily Pitler, Ji Ma, Anton Bakalov, Alex Salcianu, David Weiss, Ryan McDonald, and Slav Petrov. 2017. Natural language processing with small feed-forward networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2879–2885.

Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Amitava Das and Björn Gambäck. 2014. Identifying languages at the word level in code-mixed indian social media text. In *Proceedings of the 11th International Conference on Natural Language Processing*, pages 378–387.

Kuzman Ganchev and Mark Dredze. 2008. Small statistical models by random feature mixing. In *Proceedings of the ACL-08: HLT Workshop on Mobile Language Processing*, pages 19–20.

Geoffrey E Hinton. 2012. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer.

Baden Hughes, Timothy Baldwin, Steven Bird, Jeremy Nicholson, and Andrew MacKinlay. 2006. Reconsidering language identification for written language resources. In *Proceedings of the International Conference on Language Resources and Evaluation*, pages 485–488.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017*, pages 427–431.

David Jurgens, Yulia Tsvetkov, and Dan Jurafsky. 2017. Incorporating dialectal variability for socially equitable language identification. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 51–57.

Ben King and Steven Abney. 2013. Labeling the languages of words in mixed-language documents using weakly supervised methods. In *Proccedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Tom Kocmi and Ondřej Bojar. 2017. Lanidenn: Multilingual language identification on character window. In *European Conference of the Association for Computational Linguistics*.

Marco Lui and Timothy Baldwin. 2012. langid.py: An off-the-shelf language identification tool. In *Proceedings of the ACL 2012: System Demonstrations*, pages 25–30. Association for Computational Linguistics.

Marco Lui, Jey Han Lau, and Timothy Baldwin. 2014. Automatic detection and language identification of multilingual documents. *Transactions of the Association for Computational Linguistics*, pages 28–40.

Giovanni Molina, Fahad AlGhamdi, Mahmoud Ghoneim, Abdelati Hawwari, Nicolas Rey-Villamizar, Mona Diab, and Thamar Solorio. 2016a. Overview for the second shared task on language identification in code-switched data. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 40–49, Austin, Texas. Association for Computational Linguistics.

Giovanni Molina, Fahad AlGhamdi, Mahmoud Ghoneim, Abdelati Hawwari, Nicolas Rey-Villamizar, Mona Diab, and Thamar Solorio. 2016b. Overview for the second shared task on language identification in code-switched data. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 40–49.

Nguyen and A. Seza Doğruöz. 2013. Word level language identification in online multilingual communication. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 857–862.

Shruti Rijhwani, Royal Sequiera, Monojit Choudhury, Kalika Bali, and Chandra Shekhar Maddila. 2017. Estimating code-switching on twitter with a novel generalized word-level language detection technique. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1971–1982, Vancouver, Canada. Association for Computational Linguistics.

Thamar Solorio, Elizabeth Blair, Suraj Maharjan, Steven Bethard, Mona Diab, Mahmoud Ghoneim, Abdelati Hawwari, Fahad AlGhamdi, Julia Hirschberg, Alison Chang, and Pascale Fung. 2014. Overview for the first shared task on language identification in code-switched data. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 62–72, Doha, Qatar. Association for Computational Linguistics.

Nagesh Bhattu Sristy, N. Satya Krishna, B. Shiva Krishna, and Vadlamani Ravi. 2017. Language identification in mixed script. In *FIRE '17: Forum for Information Retrieval Evaluation*.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, ACL 2015*.