Distinguishing Romanized Hindi from Romanized Urdu

Elizabeth Nielsen[†] **Christo Kirov**[°] **Brian Roark**[°]
[†]School of Informatics, University of Edinburgh, UK [°]Google
e.nielsen@ed.ac.uk {ckirov,roark}@google.com

Abstract

We examine the task of distinguishing between Hindi and Urdu when those languages are romanized, i.e., written in the Latin script. Both languages are widely informally romanized, and to the extent that they are identified in the Latin script by language identification systems, they are typically conflated. In the absence of large labeled collections of such text, we consider methods for generating training data. Beginning with a small set of seed words, each of which are strongly indicative of one of the languages versus the other, we prompt a pretrained large language model (LLM) to generate romanized text. Treating text generated from an Urdu prompt as one class and text generated from a Hindi prompt as the other class, we build a binary language identification (LangID) classifier. We demonstrate that the resulting classifier distinguishes manually romanized Urdu Wikipedia text from manually romanized Hindi Wikipedia text far better than chance. We use this classifier to estimate the prevalence of Urdu in a large collection of text labeled as romanized Hindi that has been used to train large language models. These techniques can be applied to bootstrap classifiers in other cases where a dataset is known to contain multiple distinct but related classes, such as different dialects of the same language, but for which labels cannot easily be obtained.

1 Introduction

Hindi and Urdu are considered the two standardized registers of the pluricentric Hindustani language. In informal speech, they are highly mutually intelligible, to the point where it can be difficult to immediately assess which one is being spoken (Masica, 1993). Written (and more formal) Hindi and Urdu, however, have more noticeable differences. First, outside of the colloquial vocabulary commonly used in speech, they do differ in broad historical influence on the lexicon – Hindi making use of more Sanskrit-derived words and Urdu using more Arabic- or Persian-derived words. Most notably, however, the languages differ in their native scripts: Hindi is written in Devanagari, a Brahmic script, while Urdu is written in a Perso-Arabic script. Despite these stark differences, efforts have been made to unify linguistic resources for the languages (e.g., Bhatt et al., 2009; Visweswariah et al., 2010; Bhat et al., 2016, 2017).

Additionally, however, both languages are frequently written informally in the Latin script, which is known as romanization (Wellisch, 1978). Informal romanization makes text in these languages far more difficult to distinguish than when they are written in their distinct native scripts. Despite their overall linguistic similarity, Hindi and Urdu do represent different cultural contexts, and may have different patterns of expression that are useful to capture correctly. Predictive models in service of, for example, romanized text entry – perhaps providing next word prediction and other utilities that should match the user's desired language – will be expected to provide culturally appropriate predictions, which may be difficult if all Urdu and Hindi data is conflated in the training data. In general, given the larger number of speakers, romanized Hindi text may be more prevalent and thus yield degraded performance for Urdu speakers in a range of applications that process romanized text if the two languages are conflated.

In this preliminary study, we look at leveraging multilingual large language models (LLMs) that have been pretrained on data that includes (conflated) romanized Hindi and Urdu text, along with a small seed word list, to generate training data that can capture characteristic differences between romanized Urdu and Hindi. LLMs have recently become the backbone of many state-of-the-art NLP systems performing a wide range of tasks, often after some amount of fine-tuning (see, e.g., Ruder et al. (2021) for multilingual task benchmarks). In a recent paper, Nielsen et al. (2023) demonstrated that large language models learn some degree of long-distance sensitivity to spelling convention differences in English — i.e., the T5 LLM (Raffel et al., 2020) is more likely to produce the British spelling of a word following an earlier instance of British spelling than otherwise, even though the English language pretraining data is not labeled with the particular spelling convention. Unlike the well-understood and conventional set of spelling differences distinguishing US and UK English, romanized Hindi and Urdu represent a case where (a) there is no fixed orthography, i.e., spelling varies heavily; and (b) as far as we know, there are no documented widely attested differing romanization conventions between the two languages to rely upon. We thus try to exploit any implicit knowledge about such differences that a pretrained LLM may contain, as the means to build systems that can distinguish between the two languages.

We demonstrate that a simple decision tree classifier using character n-gram features can be profitably trained on LLM generated text to distinguish romanized Hindi from romanized Urdu, even in the face of domain-mismatch, at nearly the same accuracy as that classifier's topline (i.e., when trained on domain- and annotator-matched data). Interestingly, a more powerful neural classifier, which yields a substantially higher topline accuracy, overfits on the LLM generated training data to the point of performing essentially at chance on the validation set, suggesting that the neural classifier relies too heavily on reliable yet spurious differences between the classes in the generated text. We use the resulting decision tree classifiers to estimate the prevalence of Urdu in the mC4 corpus, and also examine their most important features, yielding some potentially useful generalizations about the romanization tendencies in the two languages.

2 Background

2.1 Romanized Hindi and Urdu

As stated earlier, distinguishing between Hindi and Urdu when written in their native scripts, Devanagari and Perso-Arabic respectively, is straightforward. Informal romanization removes this key distinction between the languages, and methods for automatic identification of romanized Hindi/Urdu text often conflate the two, sometimes deliberately (Ansari et al., 2021). This is particularly true since the romanization in Hindi and Urdu is typically less transliteration (i.e., driven by writing system correspondences) than rough phonetic transcription, hence the written differences between the two languages are lessened. For example, Urdu romanizations tend to include vowels even when the vowel is omitted in the Perso-Arabic orthography.

To see examples of this, we can examine the Dakshina dataset[1] (Roark et al., 2020), which includes both single word and full sentence romanizations of Wikipedia data in 12 South Asian languages, including Hindi and Urdu. The word "گزر" (pass) is romanized in the Urdu portion of the collection as either "guzar" or (less frequently) "gujar", despite having no vowels specified in the native script. The same word in Hindi (गुज़र) is romanized in the Hindi portion of the collection once as "guzar" and once as "gujar".

Such conventions obviously make it far more difficult to distinguish romanized Hindi from Urdu than when they are written in different native scripts. Despite the lack of a widely used standardized orthography in the Latin script in the languages, there may be some romanization conventions associated with each community that would help tease them apart. As we are not aware of any previous studies describing such distinguishing features in the linguistics literature, we turn to automated, data-driven methods to find them.

Romanized Hindi is frequent enough that it is commonly included in multilingual text collections scraped from the internet, such as mC4 (Xue et al., 2021), the multilingual corpus derived from Common Crawl[2] that is used

[1]https://github.com/google-research-datasets/dakshina

[2]http://commoncrawl.org/

to train mT5 (Xue et al., 2021), the multilingual version of the T5 language model (Raffel et al., 2020). Six languages are included in that corpus in both their native script and the Latin script – Chinese, Japanese, Hindi, Greek, Russian and Bulgarian – presumably because the language identification system used to identify the languages for the collection, CLD3,[3] only includes Latin script class labels for those six languages. Given the similarity of romanized Hindi and Urdu, and the lack of romanized Urdu as an option within the system, one might expect that some fraction of the Latin script Hindi data in mC4 is in fact romanized Urdu instead.

2.2 Related work

Transliteration of informally romanized text into the native script of the language has been explored for languages making use of Perso-Arabic scripts, including Arabic (Al-Badrashiny et al., 2014), and South Asian languages Urdu and Sindhi (Roark et al., 2020), as well as languages using Brahmic scripts such as Hindi, Bengali and Tamil (Roark et al., 2020). Work has also examined directly applying NLP models to informally romanized text in Arabic (Chalabi and Gerges, 2012), Persian (Maleki and Ahrenberg, 2008) and Urdu (Bögel, 2012; Irvine et al., 2012; Rafae et al., 2015). Language identification has been shown to be a particularly tricky problem for a variety of informally romanized languages (Bögel, 2012; Banerjee et al., 2014; Das and Gambäck, 2014; Eskander et al., 2014; Adouane et al., 2016; Zhang et al., 2018; Kreutzer et al., 2022). We direct the interested reader to Roark et al. (2020) for a more extensive background on these and related topics.

The problem of distinguishing romanized Hindi and Urdu, given a small set of seed words believed to be indicative of each language – the approach we pursue in this paper – can be thought of as an instance of weak supervision, or semi-supervised learning. We have a large, unlabeled dataset assumed to contain both Hindi and Urdu, and can use the seed set to "label" a small subset of the data depending on which seed words it contains.

From here, a typical semi-supervised approach would be to try to use the distribution of unlabeled sentences around the "labeled" points to build a decision surface that separates the two classes. Various general methods exist, including transductive support vector machines (TSVM) (Vapnik, 1998), or graph-based methods within the framework of manifold regularization (Belkin et al., 2004). These classic approaches, however, may require making additional assumptions, such as defining a distance metric between data points.

In this paper, we attempt a different approach. We exploit the implicit knowledge contained in a pre-trained LLM, as well its ability to maintain context over longer spans of generated text. In particular, we prompt the model with a frame containing one of our seed words, and allow it to generate an arbitrary amount of text based on that template. Then, we simply use this generated text as labeled data and train a standard supervised classifier to decide whether new text is either Hindi or Urdu. Before presenting these methods in detail, we first present data resources (two existing and one new) used for validation.

3 Datasets

Our work makes use of three independent data sources, including a training/validation set derived from Wikipedia, a general web-scraped text collection labeled in part as being romanized Hindi, and a set of Hindi and Urdu language-indicating seed words.

Dakshina While most relevant datasets do not distinguish between romanized Hindi and romanized Urdu, the Dakshina corpus[4] (Roark et al., 2020) does distinguish between the two. It contains hand-romanized sentences (10k per language) taken from Hindi and Urdu Wikipedia articles. This makes it ideal for evaluating our language ID system. We split the Hindi and Urdu portions of the corpus into training, development, and test sets,[5] and we use the development set (965 sentences from each language) to evaluate all versions of our language ID system. We also use the training

[3]https://github.com/google/cld3

[4]https://github.com/google-research-datasets/dakshina

[5]This data split and the seed words are available at https://github.com/google-research/google-research/tree/master/distinguishing_romanized_hindi_urdu.

Hindi		Urdu		English
urja	ऊर्जा	tawanai	توانائی	energy
chhati	छाती	seena	سینا	chest
shunya	शून्य	sifar	صفر	zero
ang	अंग	aazoo	عاضو	organ
prakar	प्रकार	qisam	قسم	type

Table 1: Examples from the seed list, including romanization and native script for both Hindi and Urdu alternatives along with an English gloss.

set in one of our baselines (see Section 4.1). All of the training and validation sets are balanced between the two languages.

mC4 The mC4 corpus, described above, consists of web-scraped data divided into 101 language partitions, of which "hi-Latn," nominally corresponding to romanized Hindi, is one. However, as previously mentioned, we believe this parition is likely to contain romanized Urdu as well, which has been conflated with Hindi due to the coverage of the CLD3 LangId system used to build the corpus.

Seed words Consulting with professional linguists who are familiar with both Hindi and Urdu, we collected 147 Hindi/Urdu pairs of words that differ between the two languages, but otherwise share the same meaning and are used in mostly the same semantic contexts. These were elicited by asking for words that, if seen in romanized text, would be strongly indicative of either Urdu or Hindi. The seed words were provided in the native scripts of Hindi and Urdu along with common romanizations for those words. Table 1 presents five example pairs from the set in both Latin and native script, along with an English gloss.

4 Methods

In this paper, we focus on comparing different sources of training data for distinguishing romanized Hindi and Urdu, rather than developing new classification architectures.

As such, initial comparisons are done using a straightforward off-the-shelf decision tree classifier (Breiman et al., 1984) from Scikit-Learn[6]. This model has the advantage of being highly interpretable, which makes it simple to determine which features are most important for distinguishing Hindi from Urdu. We train the decision tree with a maximum depth of 5 nodes, and use character 1- through 4-gram features.

To see how a more complex neural model behaves, we also finetune[7] the same mT5 checkpoint we use for data generation (see Section 4.2) to act as a classifier, where the input is a romanized sequence with the added task prefix 'Classify_HIUR:', and the output is either the string 'hi' or 'ur'.

For each of the three sources of classifier training data – Dakshina, mC4 and mT5 generated text – we provide the size of the resource and example strings in Table 2.

4.1 Baselines

Dakshina Topline. We train the classifiers on the training portion of the romanized Dakshina fullstring data. The specific articles in the Hindi and Urdu portions of the data differ, but otherwise span the entire range of Wikipedia topics, so there is unlikely to a be confound due to mismatched domains. The romanizations were produced by specific sets of annotators – disjoint between the two languages – hence the text may contain individual romanization styles that can make detection easier if present in the training data. Since this kind of labeled training data — along with a strong domain-match between the training and development data — is unlikely to occur in a realistic scenario, we consider this a topline condition. Our Dakshina training corpus has a total of 15.8k sentences, with a total of 1.6M characters.

mC4 Sentences. We train our classifiers on a balanced sample of sentences taken directly from the hi-Latn portion of mC4 (which we believe contains both Hindi and Urdu). In order to distinguish Hindi-aligned and Urdu-aligned sentences from the corpus, we use a simple heuristic: We give a sentence the Hindi label if it contains at least one of our Hindi seed words, and none of our Urdu seed words. The same applies in reverse to select potential Urdu sentences. From these candidates, we select 45.9k sentences for each language. This

[6]https://scikit-learn.org/stable/modules/tree.html

[7]The number of finetuning steps varied according to training data size — 5k for the Dakshina topline, 50k for the mc4 sentences baseline, and 100k for the data generated by mT5.

Dataset	Lines	Lang	Example
Dakshina train set	15.8k	HI	kul milakar yah 800 kilometer ki unchai tak pahunchegi.
		UR	jo bulandi mein duniya mein doosre number par hai.
mC4 sentences	91.9k	HI	mainyual prakriyaon ko svachaalit kyon karen vyaapaar prakriya prabandhan sophtaveyar
		UR	aik din mujhay asad bata raha tha blue flim banay mein bht paisa hai aik flim banao tu 1lac ruppe
mT5 generated data	4.5m	HI	dva ka title oot bhi vechain wala tadap dono.
		UR	mulaqat ka abjad majamiyat duniya dono se se ghazal.

Table 2: Size of each dataset in number of lines, along with one line labeled with each language from each set. All datasets are balanced, so half the data is labeled Hindi, and half Urdu.

results in a training corpus of 91.9k sentences, with a total of 97.7M characters.

4.2 Improving language ID with generated text

In this section, we present a method for using an LLM to generate training data for identifying romanized Hindi and Urdu. As we describe in Section 2, romanized Hindi and Urdu are not easily distinguishable, and so a corpus like the romanized Hindi section of mC4 likely contains both romanized Hindi and romanized Urdu.

We perform our experiments using mT5 (Xue et al., 2021), a multilingual offshoot of the original T5 model (Raffel et al., 2020), trained on the entire mC4 dataset. mT5 is an encoder-decoder transformer architecture pre-trained on a span corruption task, a form of masked language modeling. Spans of text in the input string are replaced with a sentinel token, whose contents are recovered during decoding (e.g., "The cat in the <extra_id_0>." maps to "<extra_id_0> hat <extra_id_1>"). The model uses a 250k sentencepiece (Kudo and Richardson, 2018) vocabulary, combined with 100 additional vocabulary items to represent the text spans.

We start with a publicly available mT5 checkpoint, using the "large" configuration on the t5x (Roberts et al., 2022) codebase[8]. We fine-tune specifically on the romanized "Hindi" (hi-Latn) portion of the mC4 dataset, using the original span corruption task, for an additional 100k steps. This imparts a bias to output Hindi and Urdu content specifically, while the original checkpoint tends to generate output from a wider language distribution which is not relevant to our task.

It seems reasonable that most sentences in the hi-Latn portion of mC4 come entirely from either a Hindi or Urdu source. We hypothesize that this will allow mT5 to learn that Urdu-aligned features are more likely to co-occur with other Urdu-aligned features, rather than Hindi-aligned features, and vice versa – much as such models have been shown to learn that words written using British spelling conventions tend to co-occur with words that also follow British spelling conventions (Nielsen et al., 2023).

In order to extract the information that mT5 has learned about the features that distinguish Hindi and Urdu, we first use mT5 to generate strings. We construct prompts for generation that contain words from the list of Hindi- and Urdu-specific seed words described in Section 3. We do this by inserting these seed words into short frame sentences, with a blank span elsewhere for the model to fill in. See Table 3 for the set of frame templates — outside the seed words, the sentences are designed to be language-neutral, and to be semantically generic so as not to strongly constrain possible generated continuations. For each combination of seed word and template, we generate 1000 different continuations via random sampling — given a prompt, each subsequent symbol in a generated string is sampled from the multinomial vocabulary distribution produced by the decoder at every timestep, with decoding stopping when an end-of-string token is produced. This resulted in a total of 4.5M strings, with 54.4M total characters,

[8]https://github.com/google-research/t5x/blob/main/docs/models.md#mt5-checkpoints

Frames with Glosses
mainne[I] **SEED** aur[and] BLANK likha[wrote].
maine[I] likha[wrote] **SEED** BLANK
ye[this] kaho[say]: **SEED** BLANK
usane[he] yah[this] likha[wrote]: **SEED** BLANK
ye[these] hamaaree[our] pasandeeda[favorite] cheejen[things] hain[are]: **SEED** BLANK
maine[I] likha[wrote] **SEED** aur[and] BLANK
ye[this] kaho[say]: **SEED** aur[and] BLANK
usane[he] yah[this] likha[wrote]: **SEED** aur[and] BLANK
ye[these] hamaaree[our] pasandeeda[favorite] cheejen[things] hain[are]: **SEED** aur[and] BLANK
maine[I] likha[wrote] **SEED**, BLANK
ye[this] kaho[say]: **SEED** BLANK
usane[he] yah[this] likha[wrote]: **SEED**, BLANK
ye[these] hamaaree[our] pasandeeda[favorite] cheejen[things] hain[are]: **SEED**, BLANK

Table 3: Frames for text generation, with approximate glosses.

half of which are generated from Hindi-aligned prompts, and half from Urdu-aligned prompts. We label each generated string with the language of the seed word in the prompt, strip away any <extra_id> sentinel symbols, and then train classifiers on these labeled strings.

For example, we can make use of the second template in Table 3 and the first Hindi seed word in Table 1, to construct the specific prompt: "maine likha urja <extra_id_0>". The model would then map this input to an output that effectively fills in the blank (<extra_id_0>) at the end of the string with some amount of output text that is prompt-appropriate according to the training data.

5 Results and discussion

In this section, we first determine the language identification classification accuracy of our two classifiers when trained on three different sourcs, before attempting to estimate the amount of Urdu in the romanized Hindi section of mC4. We additionally examine the most important features of the decision tree model.

5.1 Language ID performance

Table 4 shows the accuracy of each model on the Dakshina development set, under three training conditions: training on (a) the Dakshina training set, which is the classifier's topline performance for the validation set, since the training data is matched to the validation set for annotators and domain; (b) the mC4 extracted sentences, which is a baseline method for making use of the provided seed words; and (c) the mT4 generated data.

	Accuracy	
Training data	**DT**	**mT5**
Dakshina training set (topline)	85.6	96.7
mC4 sentences (baseline)	49.0	50.8
mT5 generated data	83.4	49.2

Table 4: Accuracy on the Dakshina development set, for both Decision Tree (DT) and finetuned mT5 (mT5) classifiers.

Examining the topline result for each classifier, i.e., training on the well-matched Dakshina training set, we can see clearly that the mT5 classifier achieves much higher accuracy (96.7%) than the decision tree classifier (85.6%). The neural classifier is simply more powerful, having access to more than just the local character n-gram features used by the decision tree model, and is able to leverage pretraining effectively. This is exactly why the Dakshina training is labeled as a topline evaluation, because strong classifiers can make use of well-matched annotator and/or domain characteristics that permit more effective discrimination between examples in the collection. The decision tree classifier fails to exploit such dependencies, hence its topline performance suffers relative to the neural model.

The mC4 trained baseline classifiers, however, perform essentially at chance (near 50% accuracy) for both classificiation methods. Interestingly, the decision tree model trained on the mT5-generated data performs quite close to the topline model for that classifier at 83.4% accuracy. The classifier manages this in spite of being trained on mT5-generated data, which, unlike the Dakshina topline, is neither domain- nor annotator-matched to the development set.

Surprisingly, the decision tree model trained on the generated data approaches the classifier's topline even though the generated data itself is not very separable — the *training* accuracy of the model is only 55%. Even though the generated data must be very noisy, there is a very large amount of it, which allows for the detection of a few signal-rich features while the remaining noise averages out.

Note that the useful features used by the decision tree model cannot just be character n-grams found in our seed words. Otherwise, the balanced mC4 baseline would have performed better than chance. The large amount of generated data must thus contain additional information, effectively extracting knowledge from the LLM.

In contrast, the neural classifier fails to rise above chance performance on the validation set in this condition. It has the capacity to memorize the training data with nearly 100% accuracy, but hovers around chance when tested on the development set. This is likely due to a domain mismatch. The dev set (as well as the mC4 data that mT5 was pre-trained on) largely consists of proper sentences, while the generated data often appears to be random word sequences, since it was produced by having mT5 fill in blanks in a generic template. Such a global mismatch is not a problem for the decision tree, since it sees all text as a bag of unordered ngram features. In this instance, performance actually seems to benefit from that simplification.

The ability of the decision tree classifier trained on mT5 generated training data to perform with relatively high accuracy on the dev set also suggests that, indeed, a substantial amount of the text labeled as romanized Hindi in mC4 is romanized Urdu. Otherwise the independently created set of prompts would not have yielded data sufficient to perform better than chance on the task. While we now have validation that this text exists, we can go further and try to estimate how much of the data is actually romanized Urdu rather than Hindi.

	Est. Urdu %
Generated data	61.1
Topline	35.0

Table 5: The percentage of mC4's romanized Hindi data that our models estimate to be Urdu.

5.2 Reconsidering mC4's Hindi section

Given our decision tree model's relatively high performance on out-of-domain language ID, we can use it to offer a tentative estimate of how prevalent Urdu text is in the "Hindi" section of mC4. This isn't easily verified, since mC4 doesn't distinguish between the two languages, but we offer these estimates in Table 5 as a suggestion of what percentage of mC4's Hindi data is actually Urdu. The higher number seems likely to be an overestimate, given differences in speaker population between the two languages, hence this is an indication that our model is somewhat biased towards Urdu (when in fact a prior bias towards Hindi is likely warranted). Further validation of this is needed.

5.3 Interpreting top features

In addition to scoring the overall performance of the language ID model, we investigate which features are most important to the decision tree model's performance. We use Gini importance to rank the features of each version of the decision tree. We then use Pearson's correlation coefficient to determine which of the two languages each feature is correlated with.

When we examine the top features for the topline model and the model trained on generated data (see Table 6), we can see some patterns emerge. Most obviously, the character *v* is more associated with Hindi, while *q* and *z* are more associated with Urdu. One reason for this is that the phonemes /z/ and /q/ are more frequent in words with Arabic or Persian origins. Hindi speakers are much more likely to pronounce these phonemes as [$\widehat{d\text{ʒ}}$] and [k] respectively (Kachru, 2006). In addition, although the phoneme /$\widehat{d\text{ʒ}}$/ exists in both Hindi and Urdu, as noted in Section 2, when we look at Dakshina data, we find that Urdu speak-

Generated data		Topline	
ngram	Pearson's r	ngram	Pearson's r
v	-0.46	v	-0.46
z	0.42	**q**	0.37
q	0.37	**z**	0.42
pr	-0.37	men_	-0.28
f	0.24	pra	-0.36
rez	0.01	va	-0.39
ve	-0.14	_men	-0.27
ove	-0.01	**kee**	0.24
pra	-0.36	**ovel**	0.03
a	0.01	_me_	-0.23
ohol	0.00	dh	-0.34
qu	0.09	_pra	-0.35
e	0.05	equ	-0.03
tra_	-0.17	**ee**	0.34
que	-0.04	d	-0.04

Table 6: Top 20 features for the model trained on generated data and the topline model. Note that the space character is represented here with an underscore. The features shown in **bold magenta** are correlated with the Urdu label (shown by the positive Pearson's r), and the features shown in cyan are Hindi-correlated (negative Pearson's r).

ers are more likely to transliterate it with the character *z* than *j*.

These patterns in the top features suggest that we may be able to use these features to uncover previously undocumented language variation between two related language varieties. Of course, insofar as these features have not been documented, it is difficult to evaluate how successfully they reflect meaningful variations. One direction for future work would be to verify this method on language varieties with well documented variations, such as US and UK English.

6 Conclusion

We demonstrate that it is possible to make use of pretrained large language models to generate useful training data for language identification, even if the distinction between the languages was only implicit in the pretraining data. Our method only requires a corpus of unlabelled, mixed data from the two language varieties in question and a short list of seed words from each language. It can therefore be applied in cases where only unlabeled textual data exists, including lower-resource language scenarios.

Interestingly, the combination of a powerful neural LLM for generating training data and a relatively simple decision tree classifier making use of local word-level features, yielded the best results. By focusing on local word-form features, the decision tree classifier avoided exploiting more global (but less relevant) cues in the generated strings, and thus was able to learn interesting word-level dependencies that the more powerful model simply ignored.

Future work will include manual validation and error analysis of classifier performance on a range of texts. Further, we intend to examine this method, as suggested earlier, on more clearly documented written language varieties, such as those found with US and UK English spelling differences. We also plan to investigate similar language variety confounds, such as that found between Bosnian, Croatian and Montenegrin in the Latin script.

Acknowledgements

The authors wish to thank Raymond Doctor, Cibu Johny, Anna Katanova and Alexander Gutkin for help with this project.

Ethics Statement

This work does not propose a new model or dataset, but rather probes the behavior of existing models. Thus novel ethical considerations about model behavior and dataset contents are not directly raised by this work. While not explicitly focused on ethical considerations, this paper's methods hopefully contribute to better understanding model behavior, and could be used to understand the ways in which large language models treat underrepresented and marginalized language varieties.

Limitations

Our work is focused on just a single case study of language identification of romanized text. As detailed in Section 2, distinguishing romanized Hindi and Urdu is a good candidate for a case study for several reasons, but it would be beneficial to extend this work to other language situations.

Another limitation was our choice to focus on already existing pre-trained models, rather than directly controlling the training data that is input to each model. This means some of the conclusions about the connection between training data and outcome are tentative, pending experimental confirmation.

References

Wafia Adouane, Nasredine Semmar, and Richard Johansson. 2016. Romanized Berber and romanized Arabic automatic language identification using machine learning. In *Proceedings of the Third Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial3)*, pages 53–61.

Mohamed Al-Badrashiny, Ramy Eskander, Nizar Habash, and Owen Rambow. 2014. Automatic transliteration of Romanized dialectal Arabic. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 30–38, Ann Arbor, Michigan. Association for Computational Linguistics.

Mohd Zeeshan Ansari, MM Sufyan Beg, Tanvir Ahmad, Mohd Jazib Khan, and Ghazali Wasim. 2021. Language identification of Hindi-English tweets using code-mixed BERT. In *2021 IEEE 20th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*, pages 248–252. IEEE.

Somnath Banerjee, Alapan Kuila, Aniruddha Roy, Sudip Kumar Naskar, Paolo Rosso, and Sivaji Bandyopadhyay. 2014. A hybrid approach for transliterated word-level language identification: CRF with post-processing heuristics. In *Proceedings of the Forum for Information Retrieval Evaluation*, pages 54–59.

M Belkin, P Niyogi, and V Sindhwani. 2004. Manifold regularization: A geometric framework for learning from examples. Technical report, Department of Computer Science, University of Chicago TR-2004-06.

Riyaz A. Bhat, Irshad A. Bhat, Naman Jain, and Dipti Misra Sharma. 2016. A house united: Bridging the script and lexical barrier between Hindi and Urdu. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 397–408, Osaka, Japan. The COLING 2016 Organizing Committee.

Riyaz Ahmad Bhat, Rajesh Bhatt, Annahita Farudi, Prescott Klassen, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, Ashwini Vaidya, Sri Ramagurumurthy Vishnu, and Fei Xia. 2017. The Hindi/Urdu treebank project. In Nancy Ide and James Pustejovsky, editors, *Handbook of Linguistic Annotation*, pages 659–697. Springer Netherlands, Dordrecht.

Rajesh Bhatt, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Sharma, and Fei Xia. 2009. A multi-representational and multi-layered treebank for Hindi/Urdu. In *Proceedings of the Third Linguistic Annotation Workshop (LAW III)*, pages 186–189, Suntec, Singapore. Association for Computational Linguistics.

Tina Bögel. 2012. Urdu-Roman transliteration via finite state transducers. In *FSMNLP 2012, 10th International Workshop on Finite State Methods and Natural Language Processing*, pages 25–29.

L. Breiman, Jerome H. Friedman, Richard A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. The Wadsworth Statistics/Probability Series. Chapman and Hall, New York.

Achraf Chalabi and Hany Gerges. 2012. Romanized Arabic transliteration. In *Proceedings of the Second Workshop on Advances in Text Input Methods*, pages 89–96, Mumbai, India. The COLING 2012 Organizing Committee.

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.

Ramy Eskander, Mohamed Al-Badrashiny, Nizar Habash, and Owen Rambow. 2014. Foreign words and the automatic processing of Arabic social media text written in Roman script. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 1–12, Doha, Qatar. Association for Computational Linguistics.

Ann Irvine, Jonathan Weese, and Chris Callison-Burch. 2012. Processing informal, Romanized pakistani text messages. In *Proceedings of the Second Workshop on Language in Social Media*, pages 75–78, Montréal, Canada. Association for Computational Linguistics.

Yamuna Kachru. 2006. *Hindi*. John Benjamins, Philadelphia.

Julia Kreutzer, Isaac Caswell, Lisa Wang, Ahsan Wahab, Daan van Esch, Nasanbayar Ulzii-Orshikh, Allahsera Tapo, Nishant Subramani, Artem Sokolov, Claytone Sikasote, Monang Setyawan, Supheakmungkol Sarin, Sokhar Samb, Benoît Sagot, Clara Rivera, Annette Rios, Isabel Papadimitriou, Salomey Osei, Pedro Ortiz Suarez, Iroro Orife, Kelechi Ogueji, Andre Niyongabo Rubungo, Toan Q. Nguyen, Mathias Müller, André Müller, Shamsuddeen Hassan Muhammad, Nanda Muhammad, Ayanda Mnyakeni, Jamshidbek Mirzakhalov, Tapiwanashe Matangira, Colin Leong, Nze Lawson, Sneha Kudugunta, Yacine Jernite, Mathias Jenny, Orhan Firat, Bonaventure F. P. Dossou, Sakhile Dlamini, Nisansa de Silva, Sakine

Çabuk Ballı, Stella Biderman, Alessia Battisti, Ahmed Baruwa, Ankur Bapna, Pallavi Baljekar, Israel Abebe Azime, Ayodele Awokoya, Duygu Ataman, Orevaoghene Ahia, Oghenefego Ahia, Sweta Agrawal, and Mofetoluwa Adeyemi. 2022. Quality at a glance: An audit of web-crawled multilingual datasets. *Transactions of the Association for Computational Linguistics*, 10:50–72.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Jalal Maleki and Lars Ahrenberg. 2008. Converting Romanized Persian to the Arabic writing systems. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

Colin P Masica. 1993. *The Indo-Aryan languages*. Cambridge University Press.

Elizabeth Nielsen, Christo Kirov, and Brian Roark. 2023. Spelling convention sensitivity in neural language models. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 1334–1346.

Abdul Rafae, Abdul Qayyum, Muhammad Moeenuddin, Asim Karim, Hassan Sajjad, and Faisal Kamiran. 2015. An unsupervised method for discovering lexical variations in Roman Urdu informal text. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 823–828, Lisbon, Portugal. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Brian Roark, Lawrence Wolf-Sonkin, Christo Kirov, Sabrina J. Mielke, Cibu Johny, Isin Demirsahin, and Keith Hall. 2020. Processing South Asian languages written in the Latin script: the Dakshina dataset. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 2413–2423, Marseille, France. European Language Resources Association.

Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, Curtis Hawthorne, Aitor Lewkowycz, Alex Salcianu, Marc van Zee, Jacob Austin, Sebastian Goodman, Livio Baldini Soares, Haitang Hu, Sasha Tsvyashchenko, Aakanksha Chowdhery, Jasmijn Bastings, Jannis Bulian, Xavier Garcia, Jianmo Ni, Andrew Chen, Kathleen Kenealy, Jonathan H. Clark, Stephan Lee, Dan Garrette, James Lee-Thorp, Colin Raffel, Noam Shazeer, Marvin Ritter, Maarten Bosma, Alexandre Passos, Jeremy Maitin-Shepard, Noah Fiedel, Mark Omernick, Brennan Saeta, Ryan Sepassi, Alexander Spiridonov, Joshua Newlan, and Andrea Gesmundo. 2022. Scaling up models and data with `t5x` and `seqio`. *arXiv preprint arXiv:2203.17189*.

Sebastian Ruder, Noah Constant, Jan Botha, Aditya Siddhant, Orhan Firat, Jinlan Fu, Pengfei Liu, Junjie Hu, Dan Garrette, Graham Neubig, and Melvin Johnson. 2021. XTREME-R: Towards more challenging and nuanced multilingual evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10215–10245, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Vladimir N. Vapnik. 1998. *Statistical learning theory*. Wiley, New York.

Karthik Visweswariah, Vijil Chenthamarakshan, and Nandakishore Kambhatla. 2010. Urdu and Hindi: Translation and sharing of linguistic resources. In *Coling 2010: Posters*, pages 1283–1291, Beijing, China. Coling 2010 Organizing Committee.

Hans H. Wellisch. 1978. *The Conversion of Scripts: Its Nature, History, and Utilization*. Information sciences series. John Wiley & Sons, New York.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Yuan Zhang, Jason Riesa, Daniel Gillick, Anton Bakalov, Jason Baldridge, and David Weiss. 2018. A fast, compact, accurate model for language identification of codemixed text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 328–337, Brussels, Belgium. Association for Computational Linguistics.