Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 6228–6236
Marseille, 11–16 May 2020

Minority Positive Sampling for Switching Points - an Anecdote for the Code-Mixing Language Modeling

Arindam Chatterjee[1], Bodla Vineeth Guptha[2], Parul Chopra[3], Amitava Das
Wipro AI Labs
Bangalore, India
{arindam.chatterjee4, bodla.guptha, parul.chopra1, amitava.das2}@wipro.com

Abstract

Code-Mixing (CM) or language mixing is a social norm in multilingual societies. CM is quite prevalent in social media conversations in multilingual regions like - India, Europe, Canada and Mexico. In this paper, we explore the problem of Language Modeling (LM) for code-mixed *Hinglish* (Hindi-English language pair) text. In recent times, there have been several success stories with neural language modeling like *Generative Pre-trained Transformer* (GPT) (Radford et al., 2019), *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin et al., 2018) *etc.*. Hence, neural language models have become the new *holy grail* of modern NLP, although LM for CM is an unexplored area altogether. To better understand the problem of LM for CM, we initially experimented with several statistical language modeling techniques and consequently experimented with contemporary neural language models. Analysis shows that *switching points* (junctions in the text where the language switches) are the main challenge for the CM language model and the reason for the performance drop, as compared to monolingual LMs. To handle this impediment, in this paper we introduce the concept of *minority positive sampling*, to selectively induce more samples, to achieve better performance. The neural language models for CM demand a huge corpus, still they exhibit improvement in performance, after the samples are induced. Finally, we report a perplexity of 139 for *Hinglish* LM for CM using statistical bi-directional technique.
Keywords: code-mix, language modelling, Hinglish, switching point

1. Introduction

Mixing languages, also known as code-mixing, is a norm in multilingual societies. Multilingual people, who are non-native English speakers, tend to code-mix using English-based phonetic typing and insertion of anglicisms in their native language. In addition to mixing languages at the sentence level, it is also fairly common to find code-mixing behavior at the word level. This linguistic phenomenon poses a great challenge to conventional NLP systems. The following phrase is an example of code-mixing in Hinglish *i.e.*, mixed between Hindi and English. In the example only one English word *enjoy* has been used, but more noticeably for the Hindi words - instead of using the native Devanagari script, English phonetic typing is a popular choice.

Aye$_{\texttt{HI}}$ aur$_{\texttt{HI}}$ enjoy$_{\texttt{EN}}$ kare$_{\texttt{HI}}$
Eng. Trans.: come and enjoy

Naturally, code-mixing is more common in geographical regions with a high percentage of bi or multilingual speakers, such as in Texas and California in the US, Hong Kong and Macao in China, many European and African countries, and the countries in South-East Asia. Multi-linguality and code-mixing are also very common in India.

Code-mixing has received a significant attention very recently, mainly due to the availability of large-scale code-mixed data in blogs, micro-blogs (*e.g.*, Twitter), WhatsApp, and chats (*e.g.*, Facebook messages) *etc.* On the other hand, in recent years, the application of LM in natural language processing became an interesting field which has attracted many researchers' attention. There have been several success stories with neural language modeling techniques like *Generative Pre-trained Transformer - 2* (GPT-2) (Radford et al., 2019), *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin et al., 2018) *etc.* The goal of language modelling is to estimate the probability distribution of various linguistic units, *e.g.*, words, sentences *etc.* There are two different paradigms of language modeling - count-based and continuous-space language models.

LM for CM (LMCM) is an unexplored area altogether. In this paper, we present a detailed study on language modeling techniques for Hinglish text using both statistical and neural techniques. Training a language model for Code-mixed (CM) language is known to be a difficult problem for many reasons - words from two languages mixed together, even grammar of two languages mixed together. There are only two recent significant contributions in Hinglish LM, by (Pratapa et al., 2018), and (Samanta et al., 2019). The reported perplexities of LM for Hinglish code-mixing text in the previous two papers are 772 and 2090.78 respectively. Both the works relied on neural language models trained on synthetic data. In contrast, we have collected sizeable naturally mixed data. Our experiments yielded better performance than both the previously existing techniques (Pratapa et al., 2018), (Samanta et al., 2019) - achieved a perplexity of 655.32, 1701.49, and 139.05 respectively using statistical, neural models, and bi-directional statistical models. Our analysis shows that switching points are the main cause for the LM performance drop, and hence, in this paper we introduce the idea of *minority positive sampling* to selectively collect more data to achieve *better performance* and *faster convergence*. Neural models require a huge

1, 2, and 3 all the authors contributed equally.

dataset. However, the proposed sampling techniques also improve the performance for NLMs.
The rest of the paper is organized as follows. Section 2. covers the benchmarks for code-mixing corpora. Section 3. describes statistical language modelling techniques we have explored, followed by section 4. where we present our work on code-mixed language modelling using state-of-the-art neural network models. Section 5. discusses a bottle neck for code-mixed language modelling *viz., Switching Points (SPs)*. Section 6. discusses novel sampling techniques called *minority positive sampling*, that we have used for generating better language models. In section 7. we discuss how bi-directionality in statistical models can enhance the LM performance and in section 8. we compare language models generated from synthetically created data, against naturally code-mixed data. Correspondingly, we discuss our observations and analysis, and conclude our paper in section 9.

2. Qualitative and Quantitative Benchmarks for Code-Mixing Corpus

Performance of language models depends on the size of the training corpus and ultimately on the size of the vocabulary. The current *State-of-the-art* (SOTA) neural language models, learn from at least a billion word corpus, for monolingual language modeling. Apart from the data scarcity, in case of code-mixing, there are additional challenges: i) Words of both languages are present, ii) *Hindi* is written in *English* phonetic typing, therefore no standardization of spelling - which in effect creates more surface word forms, iii) Code-mixing is a syntactic fusion, and netizens being creative about their mixing, create several new word conjunction patterns. Therefore, we started with several fundamental questions before we set our data acquisition pipeline:

1. How much of social media content is actually mixed?
   Empirical conclusions: This is an essential question. When we claim CM is prevalent in social media, it is also necessary to ask how prevalent it is. We found at least 50%+ tweets are code-mixed on Indian twitter. Please see the Section 2..2 for more discussions.
2. The performance of any NLP method is dependant on the complexity of the data *i.e.*, how much mixed the data is.
   Empirical measurement: Lets say we are comparing two 4-word tweets T_i and T_j with 2 words each from the languages $L1$ and $L2$. Thus the mixing ratio of both the tweets T_i and T_j is $(4-2)/4 = 0.50$. But if T_i only contains 1 code alternation point (*e.g.*, if the words are $w_{L1}w_{L1}w_{L2}w_{L2}$), while T_j contains 3 switches (*e.g.*, $w_{L1}w_{L2}w_{L1}w_{L2}$), then T_j will most likely be more difficult to process. Therefore, it is desirable to have a measurement of the level of mixing between languages when reporting LM performances for CM. To measure such complexity we use *Code-Mixing-Index* (CMI). We have discussed CMI in details in Section 2..1.
3. Can we use any sampling method to collect more data in a controlled way?
   Empirical conclusions: We found that the performance of LM is dropping at the junctions of language switching points i.e. Hindi to English or English to Hindi switching points. For example situations like -

   aur_{HI} $enjoy_{EN}$
   Eng Trans. *and enjoy*
   $request_{EN}$ $hain_{HI}$
   Eng Trans. *have a request*
   $holiday_{EN}$ $hain_{HI}$
   Eng Trans. is holiday.

   Please see the Section 5. for the detailed discussion. Indeed, the straight forward solution to the problem is to collect more data or generate synthetic data, but we propose minority positive sampling technique to collect additional curated sample sets that can boost the overall performance of LMCM. Please refer to the Section 6..

For our experiments, we have used twitter *Hinglish* data *i.e.*, the tweets that have English and the Hindi language words and are written in English. For positive sampling based data augmentation, we have used actual switching points as keywords to collect more tweets, unlike the previous works, we have not used any synthetic data.

2..1 Code-Mixing Index (CMI)

When comparing different language modeling techniques on CM corpus it is essential to have a measure to quantify how much mixed the data is, in particular, since error rates for various language processing applications would be expected to increase, as the level of code-mixing increases. To measure such complexity in CM corpus an index called Code-mixing Index has been introduced by (Gambäck and Das, 2016) as the following:

$$C_u(x) = w_m f_m(x) + w_p f_p(x)$$

$$= \mathrm{w}_m \frac{N(x) - \max_{L_i \epsilon L}(tLi)(x)}{N(x)} * 100 + w_p \frac{P(x)}{N(x)} * 100$$

$$= 100 * \frac{w_m((N(x) - \max_{L_i \epsilon L}(tLi)(x)) + w_p P(x)}{N(x)} (1)$$

As proposed by (Gambäck and Das, 2016) - there are two main sources of information are utilized to fully account for the code alternation at utterance level: the ratio of tokens belonging to the matrix language $(w_m((N(x) - \max_{L_i \epsilon L}(tLi)(x))/N(x)$ as in Equation 1) and the number of code alternation points per token $(w_p P(x)/N(x)$, where $P(x)$ is the number of code alternation points; $0 \leqslant P(x) \leqslant N(x))$.
(Gambäck and Das, 2016) did also note that the overall CMI is relatively higher for language pairs like English-Hindi, and English-Nepali compared to language pairs like English-Spanish, Dutch-Turkish, and English-German.

2..2 Data Acquisition Pipeline

We choose Twitter as the source for the CM data collection. Twitter supports search functionality through API, which makes easier to collect sizeable data. We start with the ICON 2017 Hinglish sentiment analysis dataset (Patra et al., 2018) . The corpus is marked with word level language. A unique Hindi word-list is needed in order to search in twitter, so that the tweets we receive should be in Hinglish, but there when Hindi is written in English phonetics, there are several lexical overlaps between English and Hindi. For example - *to, do* could be confusing. The meaning of *to* is *so*, and the meaning of *do* is *two* in Hindi. To avoid such confusions we followed the following steps -

1. **Step 1:** Two vocabulary has been created - V_{HI} and V_{EN}.
2. **Step 2:** As there are lexical overlap we made an intersection among these two vocabularies $I = V_{HI} \bigcap V_{EN}$
3. **Step 3:** The common words now being removed from the Hindi vocabulary $V_{HI-UNIQ} = V_{HI} - I$. The $V_{HI-UNIQ}$ set is then sorted in descending order based on the word frequency.
4. **Step 4:** $V_{HI-UNIQ}$ set is then used to search in Twitter using the search API. Twitter language marking is not very accurate. Typically, if the tweet has any Devnagri (unicode Hindi script), then Twitter typically assumes it is Hindi, but if it is a code-mixed tweet with English phonetics written Hindi and English then Twitter marks them as *IN* - stands for Indic. Although such *IN* marking is not very accurate, but still we decide to stick to that for our data collection - as we have used yet another word-level language identifier later.
5. **Step 5:** A word level language identifier (Barman et al., 2014) then being used on the collected tweets and then CMI at the tweet level is being calculated. The language identifier is 90%+ accurate. Tweets with CMI = 0 are discarded. We notice among collected tweets 50%+ are actually mixed. We observed CMI > 50 is very very rare, rather unnatural.

CMI	No. of Tweets	Percentage
0-10	96,842	39.37%
11-20	1,03,741	42.18%
21-30	35,904	14.60%
31-40	8503	3.45%
41-50	984	0.4%
avg. CMI = 14	total tweets 2,45,974	

Table 1: Details of the collected corpus. The CMI range distribution of the corpus is natural. The average CMI of the corpus is 14.

One can argue why didnt we collect a corpus where all the CMI ranges were equally distributed. Our stand to that question would be - we want to keep the normal distribution intact in the collected corpus, so that the resultant LMCM trained on this corpus would be able to handle the real data.

We start our LM experiments with statistical language models. The motive is to vividly understand where exactly LMCM fails. Further analysis reveals switching-points are the main bottlenecks for the LMCM. Indeed, the straightforward solution to the problem is to collect more data or generate synthetic data, but we propose minority positive sampling technique to collect an additional curated sample set that can boost the overall performance of statistical language models and neural language models. All the experiments of statistical and neural models are reported from the following sections.

3. Statistical Language Modelling

Statistical language modeling technique tries to estimate probability of a given word in a given sequence. Lets say we are trying to predict what is the probability of w_4 in a given a sequence of words w_1, w_2, w_3. The problem is formalized in the following way -

$$P(w_1, ..., w_n) = \prod_{i=1}^{m} P(w_i | w_1, ..., w_{i-1}) \quad (2)$$

The common practice is then to apply Markov assumption - that the current sate is only dependant on the previous state. The simplified version of the previous formulation could be re-written then in the following way -

$$P_R(w_i | w_{i-(n-1)}, ..., w_{i-1}) = \frac{count(w_{i-(n-1)}, ..., w_{i-1}, w_1)}{count(w_{i-(n-1)}, ..., w_{i-1})} \quad (3)$$

Statistical language models (SLM) are also know as *n*-gram based model, or count based model. Size of n is typically being decided empirically based on language domain and various other factors. A simple n-gram model would give zero probability to all of the combination that were not encountered in the training corpus, therefore smoothing techniques are being used to handle such data sparsity.

3..1 Smoothing, Backoff and Interpolation

Smoothing: Smoothing techniques are used to distribute the probability mass among the seen and unseen events. These techniques prevent the model from assigning zero probability to the unseen data by shaving off some probability mass from the seen data and giving to the unseen data. If we take 2 consequential words, *San Francisco* and *San Andreas*. Since *San Francisco* is a common name, it might be in the training set and so model assigns probability for the word. But whereas the words *San Andreas*, which is not so common word may not be present in the train but the words individually may be present. The model removes some probability from San Francisco and gives some probability to *San Andreas*.

Backoff and Interpolation: There are situations when the exact n-gram has not been seen in the training corpus, then it is wise to come down to lower-order models with non-zero counts for the same sequence. For example we are trying to predict the probability of word *be* give a sequence of *to be or not to*, and lets say exact hexagram never occurred in the training corpus, then the practice is to come down to pentagram,

quadgram, trigram, or bigram eventually whenever we get the match. The idea of interpolation is a weighted sum of unigram, bigram, trigram,..., n-gram probabilities to have better confidence of the predictions. It could be described using the following equation -

$$P(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n) \quad (4)$$

Weights of λ_n are being learned either empirically or experimentally.

However smoothing, backoff, and interpolation techniques for LM is well studied subject, but we could not find any such endeavor for CM text. We have used Kyoto Language Model toolkit[1] for our experiments.

3..1.1 Good Turing Smoothing

Good Turing Smoothing technique uses the frequencies of the count of occurrence of N-Grams for calculating the maximum likelihood estimate. For example, consider calculating the probability of a bigram (*chatter/cats*) from the corpus given above. Note that this bigram has never occurred in the corpus and thus, probability without smoothing would turn out to be zero. As per the Good-turing Smoothing, the probability will depend upon the following:

For the unknown N-grams, the following formula is used to calculate the probability:

$$P_{unknown}(\frac{w_i}{w_{i-1}}) = \frac{N_1}{N} \quad (5)$$

In above formula, N_1 is count of N-grams which appeared one time and N is count of total number of N-grams. For the known N-grams, the following formula is used to calculate the probability:

$$P(\frac{w_i}{w_{i-1}}) = \frac{c*}{N} \; where \; c* = (c+1) \times \frac{N_{i+1}}{N_c} \quad (6)$$

In the above formula, c represents the count of occurrence of n-gram, N_{c+1} represents count of n-grams which occured for $c+1$ times, N_c represents count of n-grams which occured for c times and N represents total count of all n-grams.

3..1.2 Kneser-Ney Smoothing

In Good Turing smoothing, it is observed that the count of n-grams is discounted by a constant/absolute value. The same intuition is applied for Kneser-Ney Smoothing where absolute discounting is applied to the count of n-grams in addition to adding the product of interpolation weight and probability of word to appear as novel continuation.

$$P_{Kneser-Ney}(\frac{w_i}{w_{i-1}}) = \frac{max(c(w_{i-1}, w_i - d, 0))}{c(w_{i-1})} + \lambda(w_{i-1}) * P_{continuation}(w_i) \quad (7)$$

where λ is a normalizing constant which represents probability mass that have been discounted for higher order. The following represents how λ is calculated:

$$\lambda(w_{i-1}) = \frac{d \times |c(w_{i-1}, w_i)|}{c(w_{i-1})} \quad (8)$$

[1] http://www.phontron.com/kylm/

3..1.3 Modified Kneser-Ney Smoothing

Initially, Kneser-Ney smoothing uses backoff technique. Chen and Goodman modify it to use interpolation technique and further modify it to have multiple discounts. This is called modified Kneser-Ney smoothing technique.

$$P_{MKN}(w_i|w_{i-n+1}^{i-1}) = \frac{C(w_{i-n+1}^{i}) - D(C(w_{i-n+1}^{i-1}))}{\sum_{w_i} C(w_{i-n+1}^{i})}$$
$$\gamma_n(w_{i-n+1}^{i-1}) = \frac{\sum_{w_{j=1}}^{3} D_j N_j(w_{i-n+1}^{i-1})}{\sum_{w_i} C(w_{i-n+1}^{i})} \quad (9)$$

D is discounting values which is applied to sentences with nonzero probabilities. $N_j(w_{i-n+1}^{i-1}) = |\{C(w_{i-n+1}^{i-1}) = j\}|$ is a number of words that appear after the context w_{i-n+1}^{i-1} exactly j times. Modified Kneser-Ney used 3 different discounting values D_1, D_2, and D_{3+} which are discounting value for n-grams with one, two, and three of more counts, respectively.

3..1.4 Written bell Smoothing

To get the γ value, Witten-Bell technique considers the number of unique words following the history w_{i-n+1}^{i-1}. This number is formally defined as:

$$N_{1+}(w_{i-n+1}^{i-1}) = |\{w_i : C(w_{i-n+1}^{i-1} w_i) > 0\}| \quad (10)$$

3..1.5 Absolute Smoothing

It is also known as absolute discounting smoothing. The idea of the absolute discounting method is to lower the probability of seen words by subtracting a constant from their counts.

$$P_{absolute}(w|d) = \frac{max(c(w;d) - \delta, 0)}{\sum_w c(w;d)} + \sigma P(w|C) \quad (11)$$

where $\sigma \epsilon [0,1]$ is a discount constant and $\sigma = \delta |d|_u / |d|$, so that all probabilities sum to one. Here $|d|_u$ is the number of unique terms in document d, and $|d|$ is the total count of words in the document, so that $|d| = \sum_w c(w;d)$.

3..2 SLMs for CM - Performance

The collected corpus is divided in a 70:30 ratio for the training and testing purpose. Here in the the Table 2 we report perplexities of different smoothing techniques on our data. However, we also tried right to left and bi-directional SLM. For right to left set of experiments we got similar results, so we are not reporting them here. For the bi-directional SLM please refer to Section 7.. It is observed that GT is performing best on the overall, but ABS is performing better on switching points.

4. Neural Language Modelling

The journey of language modelling using neural networks was pioneered by (Bengio et al., 2003). NLP research has come a long way from there to word embeddings to recent large transformer based pre-trained language models like *GPT-2* and *BERT*. Neural network based approaches are achieving better results than classical statistical models, both on standard language modelling tasks as well as other challenging NLP

applications such as machine translation, speech recognition *etc.* While exploring neural language modelling for code-mixing, we have investigated the following approaches.

4..1 Transformer based models

When transformer was not introduced, RNN and LSTM based models were the popular choices for LMs. Despite advancements like LSTMs *etc.*, *long term dependencies* still remain a challenge for such models. Also, since these models are sequential, there is no scope for *parallelization* of these models, which results in increased time for training such models.

The transformer architecture (Vaswani et al., 2017), which is a very recent development by *Google Brain*, takes care of the above impediments faced by RNN based models, primarily by using a strategy called *multi head attention*. We have explored the transformer based LM as well as a few transformer based architectures. The transformer based architectures we have used for our code-mixed language modelling problem is described below.

4..1.1 GPT-2 based model

In 2019 OpenAI came out with a novel transformer based language model called GPT-2 or *Generative Pre-trained Transformer - 2*, (Radford et al., 2019). It is a large scale unsupervised transformer based language model, which provides SOTA accuracy on language model benchmarks as well as for various NLP tasks like machine translation, text summarization *etc.* GPT-2 is an upgradation over the initial GPT (Radford, 2018) model, with over 10 times the data and over 1.5 billion training parameters

We have deployed the GPT-2 based transformer model on our code-mixed data. The results and inferences for the same have been discussed in section 4..2.

4..1.2 BERT

BERT or *Bidirectional Encoder Representations in a transformers* based multi-layer bi-directional transformer based model, proposed by Google in 2018 (Devlin et al., 2018). We have trained RoBERTa model (Liu et al., 2019),later on released by Facebook on our code-mixed data. The training perplexity came out to be 981, performance on the test data has been reported in the section 4..2

4..2 NLMs - Performance

Among all the neural LMs, GPT-2 (Radford et al., 2019) stands out to be the best performing one. Results of all the models are reported in Table 3. It can be observed that NLMs are significantly performing better on SPs in comparison to SLMs, but the overall performances are not significant. The main reason could be that we need more data to learn CM patterns for the NLMs. For BERT, we are still working on intermediate results on SPs.

5. Switching Points - The Bottleneck for the Language Modeling

It is obvious that LMs fail in switching points. If we consider SPs as normal bigrams then it is easier to infer particular kinds of SP bigrams are relatively rare in a given corpus. Therefore such minor occurrences made any LMs difficult to learn their probabilities. Detailed report on how SLMs and NLMs are performing on SPs is reported in Table 2, and Table 3 respectively. NLMs are better performing than SLMs in terms of SPs, but to perform better overall NLMs need more data to train on.

6. Minority Positive Sampling

In order to obtain better perplexity for our language models, we explored several sampling techniques. Simple sampling methods like *probabilistic sampling* or *cluster sampling* (Thompson, 2012) did not aid to our cause. Instead we followed a more guided and careful class of sampling philosophy called *purposeful sampling* (Webb and Wang, 2013). We take our motivation from yet another sampling strategy called *stratified sampling* (Thompson, 2012), where the dataset is partitioned into groups or *strata* and samples are chosen intelligently from each strata. In our case the switching points, constitutes the strata, and we sample our data based on the frequency of these switching points. The presence of switching points in the corpus plays a critical role in pushing the perplexity scores towards low. This is because, the training algorithm has not seen a lot of switching points earlier in the corpus. Hence, we came up with better sampling strategies based on switching points. The paradigm of sampling technique we use here is called *minority positive sampling*, wherein, we selectively increase the counts of minority switching points *i.e.*, more samples of minority switching points are sampled and added to the training data. This strategy takes care of the bias-variance trade off as it increases the amount of unseen or less observed switching points for the training algorithm (variance), and consequently improves the perplexity of the model (bias). The sampling though, is done in a controlled setting so as to not disrupt the existing performance of the model by introducing further noise in the corpus. For example, lets say, the following is an example of minority HI-EN switching point. Then we have to selectively (till a threshold, decided by sampling strategies) increase more samples of this particular switching points in the corpus. For the same, we collected more tweets having this particular SP present.

$$\text{se}_{\text{HI}}\ \text{request}_{\text{EN}}$$

Eng.Trans. request to you

6..1 Sampling Approaches

While sampling minority switching points it was important to understand, to what extent we can oversample minority data points. We asked a few very basic questions to start with:

1. Does the word distribution in our CM corpus adhere to Zipf's law?
2. Whether even n-grams (where $n > 1$) in the CM corpus also follow the same power law?

CMI Range	KN			WB			ABS			GT			MKN		
	Overall	HI-EN	EN-HI	Overall	HI-EN	EN-HI	Overall	HI-EN	EN-HI	Overall	HI-EN	EN-HI	Overall	HI-EN	EN-HI
0-10	722.16	24471.27	20885.86	714.36	19804.33	18483.72	695.21	19594.61	18199.90	616.04	26812.25	22060.71	710.71	23422.82	20585.26
11-20	801.31	24327.14	21619.41	778.51	19659.54	19182.33	764.06	19445.16	18963.57	658.39	26782.54	22978.06	783.64	23243.65	21242.87
21-30	825.50	23075.08	21002.10	789.86	18575.33	18609.05	782.30	18416.33	18446.20	655.74	25255.42	22436.76	803.50	22095.94	20585.61
31-40	697.04	19274.39	17336.59	631.35	15462.28	15286.73	636.55	15511.13	15216.28	506.72	21050.12	18343.27	672.00	18526.18	17053.62
41-50	1381.35	17744.37	14052.32	1135.18	15075.44	12686.40	1182.93	15075.56	12617.55	839.72	17653.04	14054.70	1291.49	17379.48	13815.22
avg.	885.47	21778.45	18979.25	809.85	17715.38	16849.64	812.21	**17608.56**	**16688.70**	**655.32**	23510.67	19974.70	852.27	20933.61	18656.52

Table 2: Performances of various smoothing techniques on the CM data. Overall refers to the perplexity of all the bi-grams in the tweet, whereas HI-EN and EN-HI refers to the perplexity of bi-grams changing language from hindi to english and english to hindi respectively. These experiments are for left-right sequence prediction. However, we experimented with several n-size. Although PPL for CMI > 40 should be comparatively higher, but in the corpus such samples are less prevalent.

CMI Range	Transformer			GPT2			BERT
	Overall	HI-EN	EN-HI	Overall	HI-EN	EN-HI	Overall
0-10	1018.54	3712.01	3461.43	823.71	3535.08	3172.42	666.48
11-20	1210.11	3801.11	3588.16	967.01	3404.73	3208.72	782.19
21-30	1401.37	3921.19	3689.15	1334.72	3621.11	3416.21	1007.34
31-40	2688.00	4011.10	3855.90	2334.73	3918.94	3671.70	1714.42
41-50	4421.22	5833.00	5100.22	3905.87	5648.26	4723.80	4337.02
avg.	2147.848	4255.68	3938.97	1873.20	**4025.62**	**3638.57**	**1701.49**

Table 3: Performance of Neural LMs based on CMI range. Among all the neural LMs GPT2 stand out to be the best performing. Overall, NLMs are performing better on SPs, but the overall performances are not significant. For BERT we are still working on to assess its performances on SPs.

3. Finally, do the switching points' frequency distribution too follow Zipf's law?

We devised the following sampling strategies.

6..1.1 Zipf's Law Fit based Sampling (ZLFS)

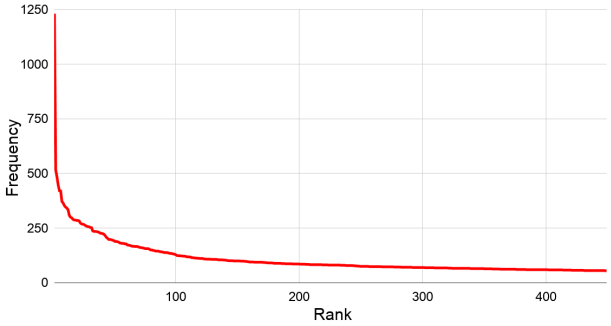

Figure 1: Switching Points frequency distribution in the corpus. It could be seen that the distribution does not follow Zipf's pattern.

Zipf's law states that the frequency of words in a corpus, decays linearly as their rank increases, on a double logarithmic scale. If $f(r)$ denotes the frequency of a word, r represents the rank of the word, and α is a constant, the so-called exponent of the law (typically), Zipf's can be depicted by equation $f(r) = r^{-\alpha}$. Zipf's law was initially conceived to unravel how languages function (Zipf, 1949). Several researchers have discovered other implications of Zipf's law as well (Li, 1992) (Jayaram and Vidya, 2008). The X axis of the curve is no.of SPs and the Y axis is frequency. It is observable that only a tiny portion of SPs repeat and the overall curve does not fit the Zipf's law, but majority of them occur only a few times. Clearly its sparse distribution - therefore it is difficult learn any pattern out of them. We observed that the distribution of switching points by rank, in our code-mixed corpus was not following the Zipf's law as can be seen in Figure 1. In order to get a Zipf's distribution of switching points, which are also bi-grams in other terms (which typically follow Zipf's law in any natural language corpus), we decided to fit the switching points distributions to emulate Zipf's law. This formed the basis for our first sampling strategy, where we constructed an ideal Zipf's law curve based on the lowest ranked switching point frequency and moved upwards. We thus sampled each switching point to fit its Zipf's law frequency, based on the rank of the switching point. We sampled $4,69,630$ English to Hindi switching points and $4,06,060$ Hindi to English switching points. The results for SLM on our code-mix data on the sampled datasets are exhibited in Figure 2.

6..1.2 Non-switching Point based Sampling (NPS)

Sampling based on Zipf's law curve fitting samples a higher number of frequently occurring switching points, rather than the minority switching points, which actually disrupt the perplexity of our statistical model. Alternatively, a switching point can also be considered as a *bi-gram*. In an ideal code-mix scenario, the distribution of switching points should be commensurate with non switching point bi-grams. This formed the strategy for our second sampling method. In this case, we sampled the less frequent switching points based on the average frequency of top 20% non switching point bi-grams in the Hinglish corpus, following the *Pareto Principle* (Grosfeld-Nir et al., 2007). The *Pareto principle* is a *80:20* ratio of cause-to-effect. The principle predicts that 80% of effects come from 20% of causes. In a supervised environment, we sampled around $4,50,440$ English to Hindi switching points and around $4,06,060$ Hindi to English switching points based on the bi-gram average frequency of top 20% data. This can be observed in figure 2.

6..1.3 Switching Point based Sampling (SPS)

We have already observed in the Sections 5. and 3..2 that switching points are one of the key deciding parameters for the perplexity of LMCM. Keeping this in mind, we devised our third and final sampling strategy. In the NPS technique discussed in section 6..1.2, the sampling of switching points was based on the average frequency of bi-grams. Hence, this was not relative to the most frequent switching points. In this sampling strategy, we decided to sample based on the top 20% switching points and not bi-grams using the Pareto Principle as discussed in section 6..1.2. We calculated the average frequency of switching points in the top 20% of our corpus and boosted the frequency of less frequent (minority) switching points to this average frequency value, through controlled tweet collection around such switching points and further added them in the training corpus. We sampled around $1,68,940$ English to Hindi switching points and around $1,42,562$ Hindi to English switching points. This technique captures more number of effective switching points *i.e.*, it boosts the frequencies of minority samples much better than the other sampling strategies, hence even for less number of samples it provides better results. The results are illustrated in figure 2.

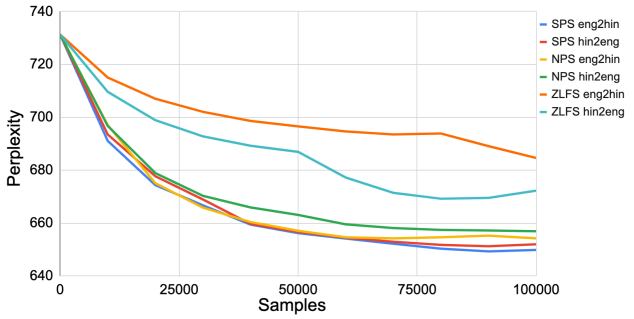

Figure 2: Effectiveness of sampling strategies on SLM performance. We can observe that the SPS strategy works best, even for less number of samples

7. What happens if We Empower SLMs with Bi-Directionality?

Neural language models like *Generative Pre-trained Transformer 2* (GPT-2) (Radford et al., 2019), *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin et al., 2018) get trained from bi-directional contexts, whereas statistical models we discussed so far are uni-directional; and it's not fair to compare uni-directional LMs with bi-directional models. We thus decided to add bi-directionality to the statistical models. Essentially we consider both the left and right sequences for a word, for the statistical language models as well.

Now, say P_L is the probability of a word occurring in the the left sequence and P_R is the probability of a word occurring in the right sequence. In order to have a bi-directional language model we have to combine the probability values P_L and P_R. (Genest and McConway, 1990) show that there are several ways to merge two (or several) information sources, in particular if they are independent. In our case, we use *linear opinion poll* - giving equal weights (0.5) to both the contexts. Equation 14 shows the mathematical representation of the linear opinion poll. With a window size of n, there are $n-1$ tokens in both the left and right sequences. Since, we are using a *bigram* model, we take a window of size $n = 2$ around our target word. So, essentially there are $1(= n-1)$ words in the left sequence as well as the right sequence. To explain, if the target word is w_i, the left bigram sequence is (w_j, w_i), the right bigram sequence is (w_i, w_k). If the left and right sequences have been seen in the training dataset, then we use the probabilities of the observed sequences. However, if (w_j, w_i) or (w_i, w_k) have not occurred, we then replace the probabilities of the words with $(< unk >, w_i)$ or $(w_i, < unk >)$. Also, if w_i itself is not observed in the training data, we assign the probability as 0.000000001. The heatmap for the next word generation task for all models i.e SLMs, NLMs and bi-directional are illustrated in the Figure 3. The results for bi-directional statistical model are mentioned in Table 4.

$$P_L(w_i|w_{i-(n-1)}, ..., w_{i-1}) = \frac{count(w_{i-(n-1)}, ..., w_{i-1}, w_i)}{count(w_{i-(n-1)}, ..., w_{i-1})} \quad (12)$$

$$P_R(w_i|w_{i+1}, ..., w_{i+(n-1)}) = \frac{count(w_i, w_{i+1}, ..., w_{i+(n-1)})}{count(w_{i+1}, ..., w_{i+(n-1)})} \quad (13)$$

$$P_{bi} = 0.5 * P_L + 0.5 * P_R \quad (14)$$

CMI	PPL.
0-10	137.03
11-20	142.63
21-30	139.12
31-40	134.52
41-50	330.23
avg.	139.60

Table 4: Results of Bi-Directional SLM. Due to bidirectionally, there is a significant improvement in performance.

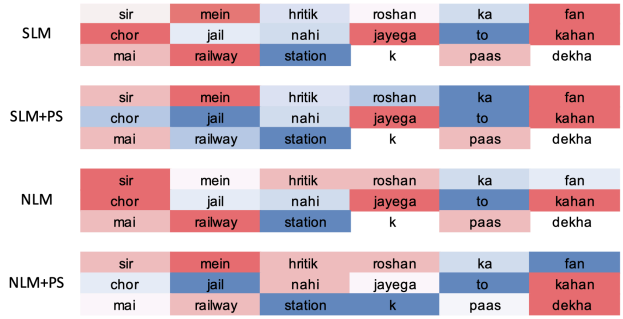

Figure 3: Attention Heat Map - How different Language Models learn to emphasize on switching points after and before positive sampling. Here darker blue shade depicts higher probability and more reddish shade depicts lower probability. It is observable that switching points like *ka fan*, *chor jail*, *station k etc.* got higher learnt probabilities after positive sampling.

8. Synthetic vs. Naturally Mixed Data

There are only two research efforts (Pratapa et al., 2018), (Samanta et al., 2019) could be found in the literature on Hinglish language modeling. Both the

papers relied on synthetic data creation and reported perplexities of 772 and 2090.78 respectively.

(Pratapa et al., 2018) proposed an Equivalence Constraint based (Poplack, 1980); (Poplack, 2001) synthetic data generation method. The basic assumptions were 1) syntactic structure of a given monolingual sentence and the generated CM sentence is identical, 2) CM sentence does not at any point, deviate from both monolingual grammars, 3) Following the Equivalence Constraint rules authors presumed only those word units are replaceable that follow the same grammatical constraints.

Here, we argue (Poplack, 2001) constraint based model has limitations. For example (Joshi, 1982) claimed functions words could not be mixed or switched. This assumption also supports (Poplack, 2001) equivalence constrained model. But, for many cases when code mixing happens between two very distinct language families - for example English and Hindi function word mixing is a common phenomena. We have seen many such examples in our corpus.

$\text{leader}_{\text{EN}}$ ki_{HI} $\text{search}_{\text{EN}}$

Eng. Trans.: search for leader

Indeed, above cited constraint free word distribution makes LMCM a very hard problem. Moreover, authors (Pratapa et al., 2018) also mentioned that the equivalence constrained based synthetic generation techniques worked well for the language pairs with good structural correspondence like English-Spanish, but the performance degrades with weaker correspondence like English-Hindi.

(Samanta et al., 2019) introduced the idea of variational autoencoders for code-switching (VACS) based synthetic data generation technique. VACS generates code-mixed sentences from a noise distribution instead of any learned latent embedding space. However, word embedding for code-mixed text is yet another unsolved problem and the authors did not clearly mention what kind of word embeddings they did use for their experiments. According to their report, they got only 1K data to train their VACS model, but we wonder how much unique switching points and their repetitions would be available in such a tiny dataset. However, authors shared their synthetically generated data with us and we found their data is abnormally mixed. By quoting abnormal we mean tweets in (Samanta et al., 2019) corpus have English words, Hindi words written in English alphabets, and a significant portion (40%+) of the corpus has Devnagri Hindi like the following example. Here we argue that different forms of Hindi word mixing i.e. Hindi words written in English alphabets, and Devnagri Hindi is not a common practice. Therefore, we are not sure how realistic the synthetically generated data is, whereas in comparison our collected data is mixed between English words and Hindi words written in English alphabets - the common practice.

pak_{NE} $\text{और}_{\text{DEV-HI}}$ $\text{न}_{\text{DEV-HI}}$ wrong_{EN} $\text{excited}_{\text{EN}}$
india_{NE} ko_{HI} ipl_{NE}

Eng. Trans.: pak and nine wrong about India IPL

Now to further test whether such abnormal modalities of mixing makes it difficult for language models to learn word sequences - we applied statistical models on their dataset and surprisingly we achieved a perplexity of 354 using Good-Turing smoothing technique. To compare, authors (Samanta et al., 2019) reported a perplexity of 2090.78 using VACS. Moreover, we have tried their VACS system on our data and it achieved a perplexity of 1144.

CMI Range	KN	WB	ABS	GT	MKN	Transf.	GPT-2
0-10	538.64	608.50	534.07	459.41	539.45	503.84	383.02
11-20	461.55	489.62	434.50	373.17	455.79	350.33	275.42
21-30	439.79	455.32	405.60	344.26	435.80	333.68	267.39
31-40	421.75	433.48	384.36	321.60	413.92	311.00	248.43
41-50	359.71	357.01	321.02	272.20	342.64	265.34	210.05
avg.	444.29	468.79	415.91	354.13	437.52	328.71	**260.52**

Table 5: Performance of SLMs on VACS (Samanta et al., 2019) generated synthetic data. GT achieves the best performance.

9. Conclusion & Takeaways

In this paper we report a detailed language modeling experiments for code-mixed *Hinglish* text. Take away points from this research could be summarized as following:

1. CMI distribution in the CM corpus is an important point to give thrust on.
2. Switching points are the main bottlenecks for the language modeling of code-mixed data. Our analysis shows performances of various LMs are better for HI-EN cases in comparison to EN-HI cases. Analysis reveals, in most of the cases, Hindi is the matrix language (Myers-Scotton, 1993) and English gets mixed, whereas the reverse is not largely true. This leads to the next question, whether we can make better models with this understanding.
3. We introduce the idea of minority positive sampling for switching points. Empirically, sampling switching points based on Pareto Principle turns out to be the best sampling technique. Minority positive sampling effectively improved the performance on the unidirectional statistical model. But, there was not much significant improvement in the bi-directional statistical model.
4. It is evident from the results that neural language models in the current form are not effective to capture contextuality for code-mixed data - at least for the given corpus size. It is indeed a well known fact that deep learning models are data hungry.
5. As there are only two research efforts on Hinglish language modeling - we purposefully avoided writing a dedicated section on related works, rather we discussed on the same in the Section 8..
6. Classic n-gram models of language cope with rare and unseen sequences by using smoothing methods. Neural network models however, have no notion of discrete counts, and instead use distributed representations to combat the curse of dimensionality (Bengio et al., 2003). We are working towards introducing controlled smoothing in neural language modeling.

References

Barman, U., Das, A., Wagner, J., and Foster, J. (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 13–23, Doha, Qatar, October. Association for Computational Linguistics.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Gambäck, B. and Das, A. (2016). Comparing the level of code-switching in corpora. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1850–1855, Portorož, Slovenia, May. European Language Resources Association (ELRA).

Genest, C. and McConway, K. (1990). Allocating the weights in the linear opinion pool. *Journal of Forecasting*, 9(1):53–73, January. Article first published online: 23 SEP 2006.

Grosfeld-Nir, A., Ronen, B., and Kozlovsky, N. (2007). The pareto managerial principle: when does it apply? *International Journal of Production Research*, 45(10):2317–2325.

Jayaram, B. D. and Vidya, M. N. (2008). Zipf's law for indian languages. *Journal of Quantitative Linguistics*, 15(4):293–317.

Joshi, A. K. (1982). Processing of sentences with intra-sentential code-switching. In *Coling 1982: Proceedings of the Ninth International Conference on Computational Linguistics*.

Li, W. (1992). Random texts exhibit zipf's-law-like word frequency distribution. *IEEE Transactions on Information Theory*, pages 1842–1845.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Myers-Scotton, C., (1993). *Duelling Languages: Grammatical Structure in Code-switching*. Oxford : Clarendon Press.

Patra, B. G., Das, D., and Das, A. (2018). Sentiment analysis of code-mixed indian languages: An overview of sail_code-mixed shared task @icon-2017. *CoRR*, abs/1803.06745.

Poplack, S. (1980). Sometimes i' ll start a sentence in spanish y termino en espaÑol: toward a typology of code-switching 1. *Linguistics*, 18:581–618, 01.

Poplack, S., (2001). *Code Switching: Linguistic*, pages 2062–2065. 12.

Pratapa, A., Bhat, G., Choudhury, M., Sitaram, S., Dandapat, S., and Bali, K. (2018). Language modeling for code-mixing: The role of linguistic theory based synthetic data. In *Proceedings of ACL 2018*. ACL, July.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Radford, A. (2018). Improving language understanding by generative pre-training.

Samanta, B., Reddy, S., Jagirdar, H., Ganguly, N., and Chakrabarti, S. (2019). A deep generative model for code-switched text. *CoRR*, abs/1906.08972.

Thompson, S. K., (2012). *Sampling (3rd ed.)*, pages 141–157. Hoboken : John Wiley Sons.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Webb, L. and Wang, Y., (2013). *Techniques for sampling online text-based data sets*, pages 95–114. 09.

Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Reading MA (USA).