Weakly-supervised person name transliteration from Latin script to other writing systems trained on Twitter data.

Greenland Yu
a1740184@student.adelaide.edu.au
The University of Adelaide
Australia

Lingqiao Liu
lingqiao.liu@adelaide.edu.au
The University of Adelaide
Australia

Matt Lowry
Fivecast
Kent Town, Australia
matt.lowry@fivecast.com

Jason Signolet
Fivecast
Kent Town, Australia
jason.signolet@fivecast.com

Abstract

Person name transliteration is the process of converting a person's name from one writing system to another by systematically changing characters. It plays an important part in the wider field of Machine Translation where transliterations are necessary to convert a person's name into the target language without translating its meaning. While standard transliteration systems exist between almost all language pairs, in this project we are interested in informal transliterations; i.e. those done without reference to the formal systems. The goal of this project is to build a pipeline for producing informal transliteration systems. This project is heavily influenced by Mubarak et. al. [9] and can be seen as a continuation of it. A pipeline with distinct stages was constructed to filter and cleanse raw Twitter data into a usable data set. This data set was then used to train a sequence-to-sequence RNN model. The main goal of this project to build a name transliteration pipeline was achieved. Additionally, it was found that models trained with slightly noisy data performed well over a testing sets with standard and non-standard transliteration name pairs, while models produced with high standard data performed well over standard transliterations but poorly over data with non-standard transliterations.

1 Introduction

Transliteration in general is the process of swapping characters from one writing system to another in order to express the sounds in the target writing system. Standard transliteration systems for many script pairs exist. However, when individuals transliterate, results may differ from these official systems. For this project we are interested in person name transliteration, which is the transliteration process applied to a person's name. In this project, I have focused on the process of back-transliteration, which is the process of working out the original, native-script characters given a transliteration. The process of back-transliteration poses more problems than forward transliteration as there is usually only a single correct back-transliteration of a word [3]. Additionally, information from the source language is lost in the original process of transliterating since in the act of transliterating, the language you are transliterating to might not contain the means of expressing the sounds of the source language.

An example transliteration from Chinese characters (logogram writing system) to English (alphabetical writing system) for the name "余地" would be "yudi". The back transliteration of "yudi" to Chinese characters could have multiple different results such as "余底" or "芋地" or "芋底", this is due to the loss of tonal sounds in the initial transliteration from English to Chinese.

Due to the advent of social media, there is now an abundance of international social media users that are at times forced to use Latin characters to express their names; Twitter is a prime example. To create a Twitter account, a user must enter a screen name that can be in any writing system under UTF-8. However, the user is forced to input a username that must be in Latin script, often times the user self-transliterate their screen name. During this project we took advantage of this by scraping Twitter user data to create a weakly labelled data set. This data set is a representation of how people would informally transliterate their own names. This dataset can be described as weakly labelled (there are bound to be errors when self transliterating), and we used this dataset to train models to learn how informal transliterations are done.

2 Problem Motivation

2.1 Importance of Problem

With the continuation of globalisation, there is a great need for efficient and accurate machine translations so that people of different cultures can effectively communicate and

express themselves to people of other different cultures. Person name transliteration is an important part of the wider machine translation field of study. Representation of a person's name in different languages is needed so that accurate and sensible machine translations between text documents can be achieved. An example of applied machine translation is Google Translate, a widely used service that applies neural machine translation techniques to translate between 109 different languages.

A concrete example of person name transliteration in use is the vetting procedure for foreign nationals. For anyone with access to sensitive data, that person must undergo security vetting to ensure their interests are aligned with the government agency. For foreign nationals their English name is usually a transliteration of their name from their country of origin. To perform accurate security vetting using these names, a back transliteration of their English name must be performed to obtain the person name in the writing system of their country of origin.

2.2 Novelty

This project is an extension of a previous paper Mubarak et al. [9] that presents Arabic Twitter data as a novel data set to build a language model for transliteration between Arabic and English. This project will extend beyond Arabic Twitter data and aim to capture multi-lingual Twitter data to build language models for each language. Additionally, this project aims to evaluate the effectiveness of these language models built using Twitter data. The advantage of using Twitter data to train language models is that this data set represents transliterations done in an informal style and would more accurately capture how the average non-english speaking netizen would transliterate their own name. Additionally by conducting this project, we can evaluate Mubarak et al. [9] on how valuable Twitter is as a data source.

3 Related Work

In this section we are going to survey work done in the field of machine translation and in particular person name transliteration. The main paper of focus is Mubarak et al. in which this project is based upon.

The basis of this project is Mubarak et al. [9] in which the authors present Twitter as a novel data source in which a parallel corpora can be extracted. The authors point out that in addition to having a sizeable amount of data, Twitter data is also easily obtainable. In comparison to traditional person name parallel corpora, such as multilingual Wikipedia, the Twitter sourced corpora reflects how real users would transliterate their own names to Latin script without a standard transliteration system. The authors demonstrated this by collecting 936,000 unique Arabic users from Twitter. A significant amount of effort in this study was the pre-processing of the Twitter data, this was to ensure that the dataset was valid and would be good candidates for statistical machine learning. Pre-processing of the data involved removal of titles from names and mapping of informal Arabic decoration characters to formal Arabic characters. Additionally, a similarity score was calculated between the Arabic and transliterated name. This similarity score was used to determine which dataset a name pair should go. The end result was three labelled datasets, one with name pairs of 100% similarity, one with 80% to 100% similarity and one with 70% to 80% similarity. The contents of these datasets were user name (Latin script), screen name (Arabic script) and geographical region. The reasoning for including geographical region was that name transliterations would vary region to region, including geographical region would mean that the researchers could infer a user's geographical region given their username. The datasets was evaluated by comparing against multilingual Wikipedia data, in which the combination of the Twitter dataset and Wikipedia dataset was used to build a character based translation model with Moses [6]. The model that used combined Twitter data showed a higher BLEU score [10] compared to the model built using solely Wikipedia data.

The second paper we are looking at is Prabhakar et al. [11] in which the researchers conducted a survey into machine transliteration and text retrieval methods. Their motivating factor in looking into machine transliterations is due to the advent of "Web 2.0" in which multi-lingual user generated data is increasing at a rapid rate, due to this there needs to be an effort to process this incoming data. The researchers come to the conclusion that "mining" approaches (where parallel corpora are mined from the internet and used to extract rules for transliteration) generally outperformed "generative approaches" (in which transliterations are governed by formal, bespoke rules for particular language pairs). The paper also discusses "fusion" approaches (a combination of the "generative" and "mining" approaches) to transliterate, in which researchers only show moderate success.

Bilac et al. [3] explores the specific problem of back transliteration. They state the main challenges in back transliteration is that information is lost in the process of transliterating the person name. They survey previous work done under person name back transliterations and summarise approaches into two categories, "grapheme based" and "phoneme based"; the researchers propose a new approach which combines these two approaches. They achieve this by assigning weights to both models such that the weights maximise the accuracy when training the data. They evaluate their combined model in comparison to single model system and find that the combined approach outperforms the single model systems.

An interesting problem when transliterating is the case when the origin of the name is neither from the source language nor the target language; this particular case is addressed by Cohen et al. [4] in which they propose a method

to handle this. Their method relies on a monolingual resource list (such as a name list) and phonetic knowledge of the target language. The proposed method adds upon existing methods in the form of incorporating an origin specific transliteration table and origin specific n-grams model. These additions improved transliteration accuracy by 44% for names of Arabic origin from English to Hebrew. This specialised approach to transliteration could be incorporated into this project given there is enough time.

An approach to person name transliteration is through statistical translation models, Lee et al. [7] describe this process. Compared to previous models, statistical translation models do not require a pronunciation dictionary or manually assigned phonetic similarity scores between source and target words. Instead a statistical translation model takes a noisy channel approach to word transliterations. The noisy channel approach utilises conditional probabilities and large amounts of parallel corpora data to generate a model. The authors describe this approach and show results on experimental data with 97.8% precision for person name transliterations from Chinese (logogram writing system) to English (alphabetical writing system). An implementation of a statistical machine translation engine is Moses [6], which was used in the Mubarak et al. [9] study.

Benties et al. [2] presents the TRANSLIT dataset, a large scale name transliteration resource. This dataset was formed by combining four different multilingual name corpora together which included JRC[5], Geonames[1], SubWikiLang[8] and En-Ar[12]. Additionally, Wiki-lang-all was compiled and added into this dataset by the researchers. Much of the work done during this study was the fusion of the different data sources, this involved removing duplicate names and processing all of the data to a standard format. This dataset would be interesting to compare to the dataset we plan to create in this project. Additionally, the background knowledge presented in this paper was a useful resource.

4 Methodology

The bulk of the work done in this project was the construction of the language model pipeline. The purpose of this pipeline is to provide a robust and straight-forward system to turn raw Twitter data into a language model. This pipeline can be separated into three distinct sections: the filtering section, the cleansing section and the modelling section.

4.1 Data Source

The data used in this project was all sourced from the public Twitter stream. This data was collected over a period of 2 years. In total 2,000,000 individual Tweets were collected. Due to Twitter being a global social media platform, Tweets of many languages were present. The components of a Tweet that we are interested in are: the language tag of the Tweet, the user name of the Tweet author and screen name of the Tweet author.

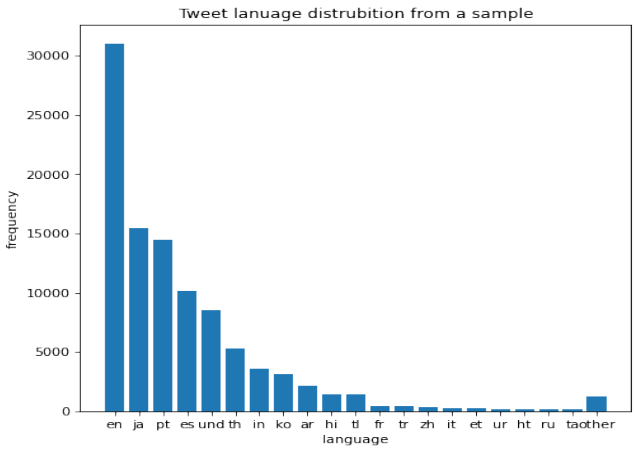

Figure 1. Tweet language distribution taken from a sample

Basic data exploration was conducted at the start to determine which language would be best to use. A random sample was taken and the histogram of languages can be seen in Figure 1. From this figure, it can be seen that after English, Japanese is the most popular language. The Japanese language also presents interesting challenges in transliterating as it has a very different writing system to English. Therefore, for the initial build of the pipeline, the Japanese language was used to test each component. Although, any language with enough representation in the data can be used also.

4.2 Technology Stack

The language of choice to construct this pipeline was Python. Python allowed for quick and iterative development, appropriate for this project as the timeline for development is short. Additionally, Python supports libraries that can perform standard transliterations (such as pykakasi) and machine learning (such as tensorflow) which were utilised in major sections of the pipeline. The sections of the pipeline were implemented as Python modules. Creating an instance of a pipeline and performing operations would be equivalent to instantiating a Python class from the module and calling class methods.

4.3 Filtering

The filtering module performs the following actions:

- Read in Twitter stream files
- Filter using the Tweet language tag
- Extract user name and screen name of Tweet
- Remove characters from screen name that are not part of the target language's character set
- Save filtered data

As the first section of the pipeline, the filtering module serves as a way to filter out irrelevant data from the initial data set, greatly reducing the amount of the data that goes into the other sections of the pipeline. This module performs two filter operations on the data, a filter by Tweet language tag and filter by screen name character language. Tweets are accompanied with a language tag in the meta-data of the Tweet, this is used in the initial filtering of the data. These language tags are in the ISO 639-2 Language Code format, for example the language tag for the Japanese language is: "ja". Twitter screen names can contain any UTF-8 character, thus many Twitter users decide to decorate their screen name with numerous emojis and characters not part of the target language's character set. Due to this, a filtering operation was applied to the screen name. This filtering operation utilises the python regex library to remove all characters in the screen name not in the target language's alphabet. For example, the character sets to keep for the Japanese language are "Katakana", "Hiragana" and "Han". The filtered data set was stored in a pandas DataFrame and can be accessed via a get function associated with the filtering module. The design decision to store the filtered set as a pandas DataFrame was made because the cleansing module (the next section of the pipeline) needs to perform operations on the entire data set and the pandas DataFrame format allows for these types of operations.

4.4 Cleansing

The cleansing module performs the following actions:

- Accept data from the filtering module
- Perform pre-processing on user name and screen name
- Filter data using modified edit-distance measure
- Split data into training and testing sets
- Save cleansed data

The cleansing module is the second major section of the pipeline. The data from the filtering module is passed directly into this section of the pipeline. The cleanser performs pre-processing upon the user name and screen name. This pre-processing performed on the user name involves: removing numbers, changing underscore characters to spaces, adding a space between lower case and upper case characters and finally case folding. For example, the user name "Kanye_WestFan1991" will be turned into "kanye west fan". The decision to perform these specific pre-processing steps was made after observing a sample of user names. Only minor pre-processing was done on the screen name as generally it was clean from the filtering stage where characters not belonging to the target language was removed.

The major function of the cleansing module is to remove name pairs that are determined to not be transliterations of each other. To determine if a name pair is a transliteration of each other, a standard transliteration is performed on the screen name, then the transliteration is compared to the user name. The comparison used is a modified edit-distance measure (see the below section). If this modified edit-distance measure is smaller than a certain threshold, we can assume that the name pair is a legitimate transliteration.

4.4.1 The Modified Edit-Distance.

Levenshtein edit-distance, is measured by how many operations it takes to transform one string into another. The set of operations are: removing a character, adding a character and changing a character to a different character. For example, the edit-distance between the string "kitten" and "sitting" is three as there is a minimum of three operations needed to transform between the two strings. These operations are:

1. **k**itten -> **s**itten (change "k" to "s")
2. sitt**e**n -> sitt**i**n (change "e" to "i")
3. sittin -> sittin**g** (adding "g")

In the context of name transliterations, this edit-distance measure favours name pairs that are short in length. For example, the edit-distance for the transliteration between "o" and "a" is one, but the pronounciation of the entire string is changed; while the edit-distance between "apple" and "applle" is also one, but there is significantly less change in the pronunciation of the two strings. For this reason, a modified edit-distance measure was used instead used to compare name pairs. The modified edit-distance makes comparisons between long strings and short strings relative. It does so by incorporating the average length of the two names in its calculation.

$$\text{modified edit distance} = \frac{\text{edit distance}}{\text{average word length}} \qquad (1)$$

Using the previous example, the average length of the two strings "o" and "a" is $(1 + 1)/2 = 1$, and the edit-distance is 1; therefore, the modified edit-distance measure is $\mathbf{1/1 = 1}$.

And for comparison, the average length of the two strings "apple" and "applle" is $(5+6)/2 = 5.5$, and the edit-distance is 1; therefore, the modified edit-distance measure is $\mathbf{1/5.5 = 0.\overline{18}}$.

4.5 Training and Testing Data

The data that is produced from the filtering and cleansing stages of the pipeline are used to form the dataset used by the language model. In total three training and three testing sets were created in this project. These training and testing sets were produced using three different modified edit-distance thresholds. The three modified edit-distance levels are: 0, 0.1 and 0.25. The sets with modified edit-distance threshold of 0 represent name pairs that are of standard transliterations of each other. The sets with modified edit-distance threshold of 0.1 represent name pairs that have a little variation from the standard transliteration. Finally, the sets with modified edit-distance threshold of 0.25 represent

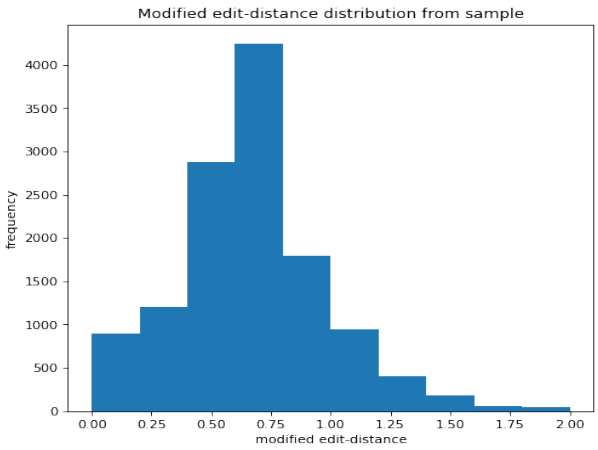

Figure 2. Modified edit-distance distribution taken from a sample

name pairs that have some variation from the standard transliteration. The purpose of using these variations of edit-distance thresholds is to capture the name pairs that may not be of standard transliteration, but are in fact informal transliterations.

Figure 2 displays the distribution of modified edit-distances taken from a sample of collected name pairs. As can be seen, the bulk of the data is unusable as the modified edit-distance is too large for the name pair to be considered some sort of transliteration. Using an modified edit-distance threshold of 0.25, approximately 90% of the data produced from the filtering stage would be removed as part of the cleansing stage.

4.6 Model Training

With the datasets created from the filtering and cleansing stages of the pipeline a language model was trained. The trained language model accepts English names as input and produces a transliteration of that name in the target language.

The model used to achieve this is a character-based recurrent neural network employing the encoder-decoder architecture. LSTM gates were used in the recurrent neural network to solve the vanishing gradient problem.

A recurrent neural network is a variation of the traditional neural network, that is able to handle time-sequence data. This is because in a RNN with each time-sequence, the input is passed back into the same network.

In the context of name transliteration, progressing through a time-sequence would be progressing though the characters in a string. For example, the first time step for the string “tom” is “t” and the second time step is “o”. This also works in other languages such as Japanese, the first time step for the string “トム” is “ト”.

	time taken	min. validation loss	max validation accuracy
Model 1	1h:34min:53sec	0.208	0.964
Model 2	1h:50min:29sec	0.264	0.956
Model 3	2h:16min:21sec	0.436	0.927

Table 1. Run statistics when training

An encoder-decoder architecture is one that uses two models, an encoder and decoder, to perform predictions. The input of the encoder is a time sequence (in our case an English name) and at each time sequence, a hidden state is produced by the encoder. The last hidden state of the encoder is fed into the decoder as input. The output of the decoder is a sequence (in our case the transliterated English name). This encoder-decoder type model allows for sequence inputs and sequence outputs, exactly in the format of what name transliteration is.

Traditionally, in Machine Translations, word level recurrent neural networks are used due to better performance over long pieces of text. In the context of this project, name transliterations are typically done upon English strings no longer than 30 characters. The advantages of word level recurrent neural networks are lost when the input sequence size is this small. For this exact reason, the dataset only containing short sequences, the attention model variant of the encoder-decoder model was not implemented.

All models were trained using tensorflow-gpu 2.5.0 on a GTX-1080ti. All three training sets contained 268,034 name transliterations.

4.7 Model Evaluation

As mentioned in the “Training and Testing Data” section of this report, three sets of testing data was produced to provide different evaluations on the produced models. Model evaluation was done using the built in evaluation method provided by the keras model class, metrics included model loss and model accuracy.

5 Experimental Setup

With the pipeline built, we have a convenient and robust method to convert raw Twitter data into language models that can transliterate English names. Using this pipeline, experiments were performed to investigate the effects of changing the modified edit-distance threshold in the cleansing stage had on the accuracy of the model. Additionally, another experiment that was done was investigating an alternate method of cleansing the data, as opposed to using the modified edit-distance measure.

5.1 Varying the modified edit-distance threshold

A variable in the pipeline that greatly effects the produced language model is the modified edit-distance threshold. The

modified edit-distance threshold dictates the amount of variability name pairs can have from the standard transliteration. A high modified edit-distance threshold means that names which are very different from the standard transliteration are accepted as legitimate name pairs and go on to form the dataset in which the model is trained on.

This experiment investigates the affect modified edit-distance threshold has on the model evaluation accuracy and evaluation loss. As explained in the "Training and Testing Data" section of this report, three training and three testing datasets was created. By creating three different models with the three training sets and evaluating each model performance against the three testing sets, we are carrying out this experiment.

5.2 An alternate cleansing method

The next experiment was one that investigated an alternate form of cleansing data, cleansing in the sense of removing name pairs that are not any form of transliterations. This experiment utilised an already trained model to give confidence predictions on name pairs, and if the model is confident that the name pair were transliterations, that name pair was accepted as a legitimate transliteration. The exact steps of this experiment are as follows:

1. Filter the data using the filter module
2. Split the filtered data into two sets, set A and set B
3. Pass set A through the cleansing module, with modified edit-distance threshold set to 0.1, to produce cleansed set A
4. Train using the training module on cleansed set A to create model A
5. Using model A, predict on set B, if the confidence of predictions are below a certain threshold, we remove those name pairs, remaining names become alternatively cleansed set B
6. Train using the training module on cleansed set B to create model B
7. Using model B, predict on (un-cleansed) set A, if the confidence of predictions are below a certain threshold, we remove those name pairs, remaining names become alternatively cleansed set A
8. The steps 5-7 can be repeated multiple times
9. Combine alternatively cleansed set A and alternatively cleansed set B to form set C
10. Train using the training module on this alternatively cleansed set C to create model C

6 Results

6.1 Project results

The primary result of this project was the construction of a name transliteration pipeline that is able to accept as input raw Twitter data and through a series of functions, output a language model capable of performing transliterations on English person names. This pipeline is separated into three modular sections: filtering, cleansing and training.

With this built pipeline, the experiments described in the "Experimental Setup" section of this report were carried out and the results are presented below.

	test set 1	test set 2	test set 3
Model 1	0.966	0.931	0.884
Model 2	0.965	0.955	0.908
Model 3	0.957	0.943	0.924

Table 2. Model accuracy against different test sets

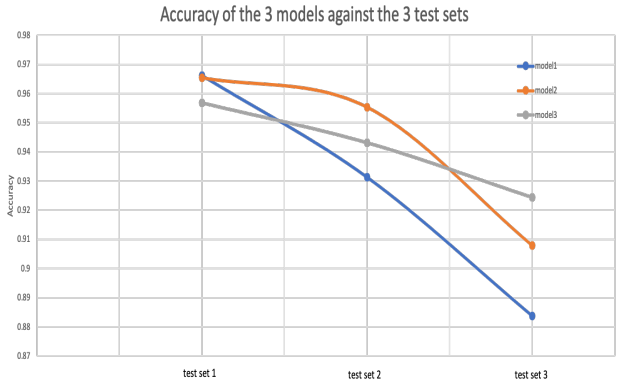

Figure 3. Plotted model accuracy against different test sets

	naruto	yuzu	lily	joshua
Model 1	なると	ゆず	ひいや	ジョシュ一
Model 2	なると	ゆず	りり	ジョシュア
Model 3	なると	ゆず	りり	じょしあ

Table 3. Transliteration results of the different models

6.2 Varying the modified edit-distance threshold

Described in section "5.1", these results are from the experiment where the modified edit-distance threshold was varied. Table 2 describes the model accuracy performance against each test set. For a more graphical view Figure 3 plots the accuracy of each model against each test set on a scatter plot.

Additionally, transliterations of select names produced by each model is displayed in Table 3. The names "naruto" and "yuzu" were chosen as they are very typical Japanese names. For comparison, the names "lily" and "joshua" were chosen as they are more western names.

6.3 An alternate cleansing method

Described in section "5.2", these results are from the experiment where a new type of cleansing is used. As described

	loss	accuracy
Test set 1	0.187	0.962
Test set 2	0.233	0.955
Test set 3	0.293	0.945

Table 4. Model A performance against different test sets

	loss	accuracy
Test set 1	0.410	0.940
Test set 2	0.472	0.935
Test set 3	0.539	0.930

Table 5. Model B performance against different test sets

	loss	accuracy
Test set 1	0.613	0.942
Test set 2	0.712	0.936
Test set 3	0.824	0.931

Table 6. Model C performance against different test sets

in that section, three models were created. The performance measures of model A is displayed in Table 4, the performance of model B is displayed in Table 5 and the performance of model C is displayed in Table 6. Note that these models are different from the models in the previous experiment where modified edit-distance was varied.

7 Discussion

7.1 On experiment 5.1

This experiment tested the effect the modified edit-distance threshold had on the model accuracy. As can be seen from Figure 3 models had lower accuracy as the data in the test set got "noisier". I am going to refer to noise in the data as name pairs that are dissimilar to standard transliterations. However, looking closer interesting patterns between the models and test sets arise. For test set 1, where name pairs were all perfect transliterations of each other, model 1 had the highest accuracy. However, model 2 also had high accuracy, only a 0.1% drop from model 1. On test set 2, where name pairs were allowed to have a little variation from the perfect transliteration, model 2 out-performed model 1 by 2.5%. While on test set 3, the performance of both model 1 and model 2 dropped and model 3 had the highest accuracy. Looking at the general shape of the plots created by the models, model 1's and model 3's performance looks linear, with model 1 having a higher negative slope.

These results unsurprisingly show that models that have been trained on noisy data perform well over noisy testing data. However, what is interesting is that models trained on noisy data did not perform significantly worse on non-noisy data when compared to models trained on non-noisy data. This can be seen in the case of model 1 and model 2 performance over test set 1 data. These results also show that for increasingly noisy data, the model that has only been trained on not-noisy data had a significant drop in performance. This can be seen in the case of model 1 performance over test set 3 data.

Looking at the transliterations produced by the three models in Table 3. For very typical Japanese names such as "naruto" and "yuzu", all three models were able to correctly produce a transliteration. For less typical Japanese names such as "lily" and "joshua" the three models produced varied transliterations. In the case of the name "lily", the Japanese language in fact does not have the sound for the letter "l", instead the nearest sound used is "r". When "lily" is transliterated by the models, model 2 and model 3 were able to correctly produce the Japanese sounds " リ リ " which mimic the substitution of "l" and "r". However, when "lily" is transliterated by model 1, the transliteration result is completely different from what one would expect, with sound being closer to "hiya". In the last case of "joshua", all three models produced different transliterations with model 1 and model 2 producing similar transliterations. When pronounced, all three transliterations can be described as legitimate transliterations, with model 2 producing the closest sounding transliteration to "joshua".

We can conclude that for typical Japanese names, all three models are able to produce legitimate transliterations. For less typical Japanese names, only models that have been trained on "noisier" data are able to produce legitimate transliterations.

7.2 On experiment 5.2

This experiment investigated the effectiveness of an alternate method of cleansing. This alternate cleansing method used a trained model to determine legitimate name pairs, as opposed to using modified edit-distance to determine legitimate name pairs.

The hypothesis of this experiment was that for each subsequent model produced, the model would have higher evaluation metrics. Looking at Table 4 to Table 6, the performance in fact decreases slightly for each subsequent model. The model accuracy stays roughly the same for each subsequent model. While, the model loss notably increased for each subsequent model.

A reason for why this might be occurring is that the initial model, model A, was not able to learn all of the relevant patterns in the data. Therefore, each subsequent model that initially spawned from model A, suffered from the same problem. A possible solution would be to train model A using a larger dataset with low modified edit-distance threshold. Doing so model A will have learnt more patterns with higher certainty and subsequent models produced by model A will also reflect this.

An interesting observation from this experiment was that during the training phase of each model, the model training metrics improved on each subsequent model. This showed that the patterns found in the training stage of each model was applied really well on the training data, but did not apply nearly as well on the testing data.

7.3 Future work

If given more time to work on this project, a number of improvements can be implemented and further investigations can be conducted.

An improvement in the general pre-processing steps of the pipeline would be to implement more robust rules to clean usernames and screen names. For example, the current rules for pre-processing usernames involve removal of numbers. This will cause problems for usernames in the form of leet-speak, as leet-speak names use numbers in place of letters. Creating robust and custom rules to cover the varied forms of how users may express their names will in turn generate a larger and higher quality dataset.

Due to hardware constraints, the entire 2,000,000 Tweet dataset was not able to be used when creating the models for experiment "5.1". A simple solution to this would be to upgrade existing hard drive space to allow for more virtual memory. Another solution would be to batch runs for each section of the pipeline. These solutions were not implemented due to time constraints. Another improvement to the current model is changing how it makes inferences. Currently the model makes inferences in a very greedy manner, by simply choosing the token with the highest probability at each time step as the prediction. This greedy method may not yield the sequence with highest probability. An improvement would be to use the beam search algorithm instead.

An interesting investigation would be a deep dive into how noise effects the final model evaluation. Because as previously observed, the introduction of some noise in the training data of models did not have a significant negative impact on the final model evaluation. Gaining a better understanding of this phenomenon may benefit other weakly-supervised machine learning tasks.

8 Conclusion

Informal self-transliterations are an alternative form of the standard transliteration systems that exist. The intricacies of how individuals transliterate their own name is captured by a global audience of Twitter users. Twitter user names are used as the English transliteration of Twitter screen names. In this project, a pipeline that takes in Tweets and produces a language model mapping English names to a transliteration in other languages was produced. Experiments using this pipeline was conducted to further explore the effectiveness of these created language models. The pipeline consists of three main stages, the filtering stage where the languages other than the select language is filtered out, the cleansing stage where user names are compared to a standard transliteration of the screen name and ones that are not similar enough are removed and finally the training stage where the remaining data is used to build a language model. Three language models were produced, each trained using different edit-thresholds. Three testing sets were also produced using different edit-thresholds. Out of all the language models produced, the lowest edit-threshold one performed best over the testing set which used the lowest edit-threshold. However, language models produced using higher edit-thresholds also performed well over all of the test sets. This signifies that noise in the training data does not have significant impact on the quality of transliterations. This project was a worthwhile adventure, hopefully the results will be used in the future. The link to where the code developed for this project is:

https://github.com/yu-greenland/name_transliteration

References

[1] URL: http://geonames.org/.

[2] Fernando Benites et al. "TRANSLIT: A Large-scale Name Transliteration Resource". English. In: *Proceedings of the 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 3265–3271. ISBN: 979-10-95546-34-4. URL: https://www.aclweb.org/anthology/2020.lrec-1.399.

[3] Slaven Bilac and H. Tanaka. "Improving Back-Transliteration by Combining Information Sources". In: *IJCNLP*. 2004.

[4] R. Cohen and Michael Elhadad. "Sideways Transliteration: How to Transliterate Multicultural Person Names?" In: *ArXiv* abs/1911.12022 (2019).

[5] Maud Ehrmann, Guillaume Jacquet, and Ralf Steinberger. "JRC-Names: Multilingual entity name variants and titles as Linked Data". In: *Semantic Web* 8 (Apr. 2016), pp. 1–13. DOI: 10.3233/SW-160228.

[6] Philipp Koehn et al. "Moses: Open Source Toolkit for Statistical Machine Translation." In: June 2007.

[7] C. Lee, J. Chang, and J. Jang. "Extraction of transliteration pairs from parallel corpora using a statistical transliteration model". In: *Inf. Sci.* 176 (2006), pp. 67–90.

[8] Yuval Merhav and Stephen Ash. "Design Challenges in Named Entity Transliteration". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 630–640. URL: https://www.aclweb.org/anthology/C18-1053.

[9] H. Mubarak and Ahmed Abdelali. "Arabic to English Person Name Transliteration using Twitter". In: *LREC*. 2016.

[10] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://www.aclweb.org/anthology/P02-1040.

[11] D. Prabhakar and Sukomal Pal. "Machine transliteration and transliterated text retrieval: a survey". In: *Sādhanā* 43 (2018), pp. 1–25.

[12] Mihaela Rosca and Thomas Breuel. *Sequence-to-sequence neural network models for transliteration*. 2016. arXiv: 1610.09565 [cs.CL].