Gonna take ~1500 words to get to a 10 minute video

Hello and welcome to my presentation on Weakly Supervised Person Name Transliteration using Twitter Data. My name is Greenland Yu and this project was done in collaboration with Fivecast. My supervisors are Matt Lowry and Jason Signolet, my academic supervisor is Lingqiao Liu.

**Introduction**

Firstly, let me introduce what exactly is transliteration. Transliteration is the act of converting a word from one writing system to another. Transliterations express the word directly through sounds. Note that this is different from translations, translations instead convert the meaning of a word from one language to another. <NEXT>

As an example, going from Arabic to English, the Arabic word Allah transliterated would be Allah, we express the sounds of the Arabic word using English sounds. Allah translated would be God in English, here we express the meaning of the Arabic word in English.

**Motivation**

You might be wondering, why keep looking into transliterating? Surely the act of converting sounds of one language into another has already been implemented already. And you are correct, there exists standard transliteration systems for pretty much all language pairs.

<NEXT>

However, when choosing a name, not everyone is going to use a standard transliteration system, many will self-transliterate their name. In this project we are interested in capturing this informal transliteration process. The produced system will provide an alternative transliteration to the standard transliteration systems.

<probably remove>In this project we are interested in Person Name transliterations, this is the act of transliterating a person’s name from one language to another. <probably remove>

A real-life application of transliterations is the security vetting process of foreign nationals. To accurately determine the profile of someone given a single name, we would want to investigate documents in multiple languages. To identify the person of interest we need to search using a transliterated name.

**Project Goal**

Now that we are properly motivated, we will have a look at the project goals. The overarching goal is to create a name transliteration pipeline that takes in raw Twitter data and produces a language model capable of transliterating between languages. This pipeline has to be robust and easy to utilise.

After the completion of this pipeline, the focus of this project is on experimentation on the Twitter data using this pipeline. Experiments include ways to increase model accuracy and decrease model loss by different methods of cleansing.

**Data source**

<Maybe remove or make shorter>

Twitter provides the data that is driving this whole operation. Twitter is chosen specifically because: it has a global user space, users have to enter an English user name along with a screen name that can be in any language. Therefore, a user signing up to Twitter is compelled to transliterate their screen name when entering their username. However, this is very noisy data as users do not have to provide a transliteration of their screen name. The solution to this problem will be explained in the cleansing section. In total this project used 2,000 Twitter files, where each file contained 100,000 Tweets. But note that these tweets are collected from the public API stream so the number of tweets in a particular language is only a small fraction.

[show graph of language distribution in sample file]

As you can see Japanese was the most prevalent language other than English, for this reason the initial build of the pipeline used Japanese.

**Methodology**

The bulk of this project’s work was creating the name transliteration pipeline. This pipeline can be separated into three distinct stages: Filtering, Cleansing and Training. I will now take you on a journey through the pipeline starting with the raw Twitter data.

The raw Twitter data is in the form of numerous JSON Tweets, the fields we are interested in are the Tweet author user\_name, the Tweet author screen\_name and Tweet language. We perform an initial filter using the Tweet language to only obtain Tweets in the language we are interested in, say Japanese. <The data is in the form of user\_name and screen\_name pairs.>

Then we extract out the user name and screen name fields to create a name pair. As you can see the name pair is not in the format we expect it to be, especially the screen name where characters that are not part of the Japanese language is present. Thus, we perform a second filter on the screen name using regex to remove all characters that are not part of the target language. As a side effect, emojis are also removed in the process.

We now move onto the cleansing stage where name pairs are evaluated to see if they are legitimate transliterations of each other. This is because when users enter their user name and screen name, it can be anything they want. And we are only interested in the name pairs where the user has self-transliterated the screen name into user name.

The cleansing stage also performs a number of pre-processing steps to further clean the names coming in from the filtering stage. Pre-processing applied to the user name include: removing numbers, changing underscores to spaces, adding a space between lower case and upper case characters and finally case folding.

Before getting into the nitty gritty of how we deduce legitimate name pairs, I will first introduce edit-distance. Edit-distance is a measure of how similar two words are. And it is defined as how many edits need to be applied to go from one word to another. Edits can be in the form of additions, deletions and substitutions. For example the edit distance for the words “kitten” and “sitting” is three. As the edits are: k substitute for s, e substitute for i and addition of g has to occur to transform kitten to sitting. This project uses a modified edit distance to measure the similarities of two words. The average length of the two words to be compared is calculated and this is used to normalise the edit-distance. Doing so, we get fairer comparisons for names with lots of characters. <note that the scale of comparison between edit-distance and modified edit-distance are different>

Name pairs are found by applying a standard transliteration on the screen name, this name is then compared to the user name using the modified edit-distance. If the modified edit-distance between these two names is low enough, we accept this name pair as a legitimate transliteration. The name pairs that survive cleansing go to form the data set that is used to train our name transliteration model.

We are using a character-level recurrent sequence to sequence model to train data.

From a very high level overview, this basically means that there are two layers, the first layer accepts a sequence of user names and produces hidden states. The second sequence takes these hidden states as input and outputs screen names. We can also think of this as the first layer encoding the user names into the target language and the second layer predicting the next target character given the user name and previous target characters. Character level means that the sequence is made up of characters instead of words.

**Results**

<assessed on presentation quality of results?>

**Future Work**