Geolocating Tweets in any Language at any Location

Mike Izbicki
Claremont McKenna College
mike@izbicki.me

Vagelis Papalexakis
UC Riverside
epapalexcs@ucr.edu

Vassilis Tsotras
UC Riverside
tsotras@cs.ucr.edu

ABSTRACT

Most social media messages are written in languages other than English, but commonly used text mining tools were designed only for English. This paper introduces the *Unicode Convolutional Neural Network* (UnicodeCNN) for analyzing text written in any language. The UnicodeCNN does not require the language to be known in advance, allows the language to change arbitrarily mid-sentence, and is robust to the misspellings and grammatical mistakes commonly found in social media. We demonstrate the UnicodeCNN's effectiveness on the challenging task of content-based tweet geolocation using a dataset with 900 million tweets written in more than 100 languages. Whereas previous work restricted itself to predicting a tweet's country or city of origin (and only worked on tweets written in certain languages from highly populated cities), we predict the exact GPS locations of tweets (and our method works on tweets written in any language sent from anywhere in the world). We predict GPS coordinates using the *mixture of von Mises-Fisher* (MvMF) distribution. The MvMF exploits the Earth's spherical geometry to improve predictions, a task that previous work considered too computationally difficult. On English tweets, our model's predictions average more than 300km closer to the true location than previous work, and in other languages our model's predictions are up to 1500km more accurate. Remarkably, the UnicodeCNN can learn geographic knowledge in one language and automatically transfer that knowledge to other languages.

KEYWORDS

Multilingual; Geotagging; Twitter; Convolutional Neural Networks

ACM Reference Format:
Mike Izbicki, Vagelis Papalexakis, and Vassilis Tsotras. 2019. Geolocating Tweets in any Language at any Location. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3357926

1 INTRODUCTION

Social media platforms are becoming increasingly multilingual. More than 100 languages are known to be actively used on Twitter [25], and while English is still the most popular, its popularity is shrinking. An analysis of tweets sent in 2012 found that 53% of tweets were written in English [21], but our analysis of tweets sent in 2018 shows that only 42% of tweets were written in English. Most prior work only studies these English messages, and unfortunately this English-only approach can create unfair bias against minority communities [e.g. 5, 12, 23, 55].

This paper introduces the *Unicode Convolutional Neural Network* (UnicodeCNN) for analyzing multilingual social media text. The UnicodeCNN generates features directly from the Unicode characters in the input text and requires no tokenization, stemming, or other preprocessing. Character-level convolutional neural networks have previously been successfully applied to text corpora containing only English [10, 62] or a limited number of European languages [33, 58], but our UnicodeCNN can take any Unicode string as input and therefore works with all languages. We use the same model to analyze English, Arabic, Japanese, and the more than 100 languages present in our Twitter dataset. The UnicodeCNN is particularly well-suited to analyzing social media text because it is naturally robust to the spelling mistakes and non-standard grammar commonly found in this domain.

We apply the UnicodeCNN to the challenging problem of *content-based tweet geolocation*, i.e. predicting where a tweet was sent from based only on its text. Geolocation is a well-studied problem, and we improve the state-of-the-art in three ways.

1. Our method works with all languages. Most prior work considers only English-language tweets [6, 9, 13, 14, 20, 34, 37, 45, 46, 61], or tweets in a single non-English language like Spanish [17, 38, 54] or Italian [42]. Han et al. [21] propose the only previous multilingual system, but they explicitly "do not consider underrepresented languages" due to the difficulty of modeling them. A common feature of all these methods is that they generate features from words in the tweet's text. Their learning pipeline therefore requires special tools for tokenizing the input text into a list of words. Existing tokenization tools work only for some languages, and extending the word-level approach to the highly multilingual setting requires developing tokenizaiton tools for all possible languages. The UnicodeCNN, in contrast, works at the character level and processes all languages in a unified method. Remarkably, we show that certain geographic information learned in one language is automatically learned in other languages as well.
2. Our method works on tweets sent from anywhere in the world. All previous work considered only a subset of tweets that were particularly easy to geolocate. In particular, all works cited above filter their datasets to contain only tweets sent from certain countries or major cities. We perform no such filtering. A side effect of this lack of filtering is that our dataset is orders of magnitude larger than previously considered tweet geolocation datasets. Our dataset contains more than 900 million tweets, whereas the largest previously considered dataset WORLD+ML contains only 23 million tweets [21].

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357926

3. We predict exact GPS locations and exploit the Earth's spherical geometry. Most previous work solves a classification problem, where each tweet is associated with either its country or a nearby city. Duong-Trung et al. [14] propose the only prior method for predicting exact GPS coordinates, but they do not consider the non-Euclidean nature of the Earth's surface and so have only limited success. We instead use a *mixture of von Mises-Fisher* (MvMF) distributions to predict the exact GPS location of tweets. (See Figure 1 for an example output.) The MvMF is flexible enough to use input features from either the existing word-level methods or from the UnicodeCNN, and we show that the UnicodeCNN geolocates tweets more than 300km closer to their true locations on average.

Improved tweet geolocation has many important applications. Geotagged tweets have been used to map the spread of influenza [43, 50], for measuring the impact of earthquakes [49], for coordinating emergency services [28, 48],for measuring the spread of political opinions [2, 11], for comparing dietary habits in different locations [60], for measuring neighborhood happiness levels [40], for measuring unemployment rates [1, 35], and for understanding differences between African American English and Standard American English [4, 16, 27]. Unfortunately for these applications, only about 1% of tweets are geotagged by their users. Our system can predict the location of the other 99% of tweets and therefore improve the accuracy and applicability of all of these methods.

Paper Outline. The next section qualitatively illustrates the advantage of the UnicodeCNN over other models. We show specific linguistic properties that the UnicodeCNN is sensitive to that no word-level model can detect. Section 3 describes the UnicodeCNN and the MvMF output layer for predicting GPS coordinates. Section 4 describes our dataset of 900 million tweets, baseline comparison models, and our experimental results.

2 SIMPLE EXAMPLES

This section presents two simple examples that illustrate the complexity of the content-based geolocation problem. We use these examples to demonstrate that the UnicodeCNN generates more powerful features than previous state-of-the-art methods.

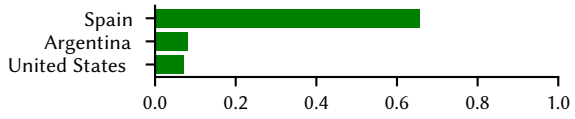
Example 1. Our first example illustrates how the UnicodeCNN is able to learn verb conjugation rules and use them for geolocation. Consider the following short Spanish tweet:[1]

`No me habléis`

which translates to

`Don't speak to me`

Figure 1 shows the GPS coordinates that our model predicts this tweet was sent from, and the plot below shows the model's top three predicted countries and associated probabilities.



[1]Tweets in both examples are real tweets. Out of respect for the sender's privacy, we do not disclose the tweet IDs.

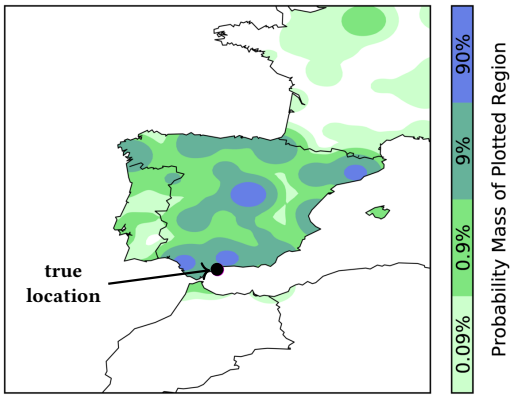
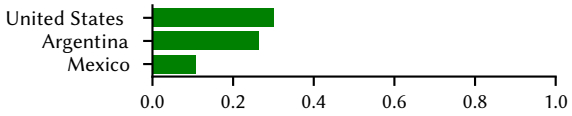


Figure 1: We model a tweet's location as a mixture of von Mises-Fisher distributions over the Earth's surface. This map shows the distribution for the tweet `No me hablen` from Example 1 below. Darker, bluer regions have higher probability, and lighter, greener regions have lower probability. Notice that the tweet's true location is assigned relatively high probability.

This tweet was in fact sent from Spain as our model predicts. Our model made this prediction by learning that the word `habléis` is a conjugation of the verb `hablar` in the vosotros form, which is only used in the Castilian Spanish dialect of Spain. American Spanish dialects use the ustedes form instead, which conjugates the verb as `hablen`. If we modify the tweet to use the American Spanish dialect:

`No me hablen`

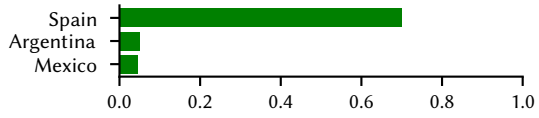
then the model's predictions change accordingly:



The UnicodeCNN's ability to understand verb conjugations is quite robust. For example, if we modify the original tweet by removing spaces (to get `Nomehabléis`), by removing the accent (`No me hableis`) or by dropping letters (`No me ableis`), then our model still predicts the tweet was sent from Spain. Remarkably, the UnicodeCNN even understands verb conjugations it has never seen before. The Spanish verb `apresar`[2] does not appear in our training data in either the vosotros form (`apreséis`) or the ustedes form (`apresen`). Nevertheless, our model correctly predicts that the Castillian Spanish phrase

`No me apreséis`

is from Spain:

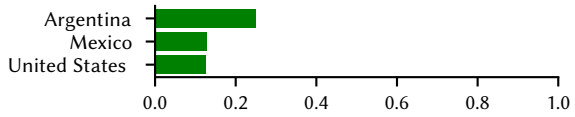


[2] The word `apresar` translates as `to arrest`. The phrases `No me apreséis`/`No me apresen` translate as `Don't arrest me`.

Similarly, our model predicts that the American Spanish phrase

No me apresen

is from the Americas:



There are many other geographical variations in Spanish verb conjugation patterns which the UnicodeCNN exploits, but no previous work studying Spanish language geolocation exploits these patterns [17, 21, 38, 54]. Exploiting these patterns is difficult for methods using word-level features because word-level feature extraction requires explicit tokenization, and the tokenization procedure would somehow need to understand these conjugation patterns. Special purpose tokenizers for tweet analysis have been built for English [41] and are widely used in existing tweet analysis work. Similar tools could be designed for Spanish, but many other less common (and less studied) languages also have sophisticated verb conjugation systems, and it would be difficult to develop special tokenizers for all of these languages. The UnicodeCNN avoids this difficulty because it operates at the character level rather than the word level and so does not need an explicit tokenization step. The UnicodeCNN is able to learn conjugations in all languages in a simple, unified manner.

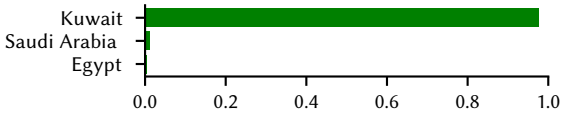
Example 2. This example shows how the UnicodeCNN can learn geographic knowledge in one language and transfer that knowledge to other languages. Consider the following tweet sent from Kuwait written in a mixture of English and Arabic:

I'm at عراش مماطملا in تيوكلا.

Translated fully into English, this tweet reads:

I'm at a street restaurant in Kuwait.

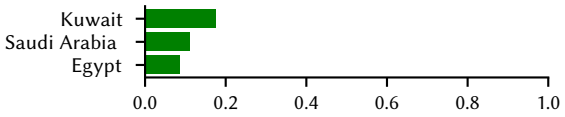
For this tweet, there is no need to analyze subtle linguistic clues to determine where the tweet was sent from because the location is written directly in the text. To identify this tweet's location, all we need to do is understand that the Arabic word تيوكلا refers to the country Kuwait. The UnicodeCNN successfully learns this fact. Its top three predicted countries and associated probabilites are:



Now consider the following plausible scenario: A Japanese tourist visits Kuwait and sends a similar tweet but with the Arabic portions written in Japanese. The translated tweet is:

I'm at 通りレストラン in クウェート.

A good geolocation model should still predict this tweet was written in Kuwait despite the language change, and our model does (but with lower confidence). Its output is



This result is remarkable because the Japanese word for Kuwait (クウェート) never appears in our training data. The UnicodeCNN learned that クウェート is Japanese for Kuwait because this word sounds similar to the Arabic and English words for Kuwait, and these words are in the training data. Similar transfers of knowledge happen for many other location and language combinations, and more generally for loan words adopted from foreign languages. This knowledge transfer is due to the novel way we encode the input Unicode text, and no previous work has observed similar transfers.

In this example, we specifically chose to mix Japanese and English in a single tweet to demonstrate another advantage of the UnicodeCNN. Japanese text does not uses spaces to mark word boundaries, and so the only prior work on multilingual tweet geolocation [21] required a specialized tokenizer to extract words from Japanese tweets. This tokenizer works only on pure Japanese tweets, and can fail for tweets containing a mixture of Japanese and other languages. Many other Asian languages also require specialized tokenizers to extract words from tweets, but these languages were ignored by [21] due to the additional complexity of supporting them. The UnicodeCNN, in contrast, does not require tokenization because it constructs features from characters rather than words. It therefore works with all languages and supports arbitrary combinations of languages in the same sentence.

3 OUR MODEL

The structure of our geolocation model is shown in Figure 2. The input text is passed to the UnicodeCNN, which generates features and passes those features to the output layers. The UnicodeCNN generates its features in four stages: a character encoder, convolutional layers, a language estimator, and a feature mixing layer. The convolutional and feature mixing layers are inspired by the *character level convolutional neural network* (CLCNN) [62]. The CLCNN was designed only for English language text, and the character encoder and language estimator are the key improvements which let the UnicodeCNN support all languages. The experiments in Section 4 evaluate three versions of the UnicodeCNN with different hyperparameter settings. We call these the Small, Large, and Huge models, and Table 1 summarizes their differences. The output of the UnicodeCNN is a set of application agnostic features. For our geolocation application we use a standard cross entropy layer to estimate the country and our novel mixture of von Mises-Fisher layer to estimate the GPS location. Other output layers can easily be used for other applications, and we open source our model's weights to facilitate transfer learning between applications.

3.1 The UnicodeCNN

We first present necessary background from the Unicode standard [53], including a summary of shortcomings of previous Unicode-aware deep learning systems. Then we describe the four stages of the UnicodeCNN feature generator.

Unicode Background. The fundamental building block of Unicode strings is called a *code point*. A code point is a number (written as U+ the number in hexadecimal) that represents either a character or formatting command. For example, the code point U+004F

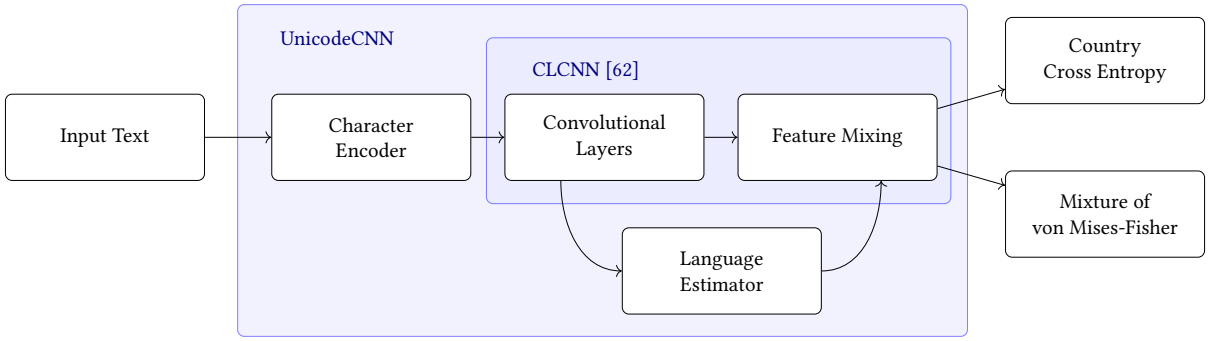


Figure 2: The UnicodeCNN adds a novel character embedding and language estimator to the *character level convolutional neural network* (CLCNN) [62]. We use two outputs for the geolocation task: the standard cross entropy to predict the country, and a novel mixture of von Mises-Fisher distributions to predict the GPS coordinates.

represents the Latin uppercase O. Complex characters can be encoded directly or as a combination of simple letters plus formatting commands called *marks*. For example, the Vietnamese character Ố can be represented by the single code point U+1EDA, or by the sequence of code points U+004F (Latin uppercase O) U+031B (combining horn) U+0301 (combining acute accent), or a number of other strings. Each of these strings is *semantically equivalent*, because they represent the same character.

For each code point, the Unicode standard [53] associates important linguistic information that the UnicodeCNN's character encoder will exploit. For example, *normalization strategies* are methods for converting between semantically equivalent strings; *transliterations* are methods for converting between different alphabets; and *character properties* is the generic Unicode term for other information associated with a code point. This information represents the combined expertise of hundreds of linguistic experts working for nearly 30 years.

Previous Unicode-aware deep learning systems have ignored these features of code points and instead operated at the encoding level. An *encoding* associates each code point with a binary representation. UTF-8 is a popular encoding that uses 1 byte to represent Latin characters, 2 bytes to represent most other European and Arabic characters, and 3 bytes to represent most Asian characters. Gillick et al. [15] and Plank et al. [44] are the only two previous works developing neural networks that accept Unicode input, and their models work directly on UTF-8 encoded strings. This method of incorporating Unicode inputs is simple but has two disadvantages. First, strings written in English have simpler UTF-8 encodings than strings written in other languages. and so the learning pipeline will be biased to favor English text. Second, Unicode's linguistic knowledge of normalization, transliteration, and character properties is not used in the learning pipeline. Our character encoder works at the code point level instead of the encoding level. It is more complicated than directly using the UTF-8 encoding, but has reduced bias for English language strings and takes full advantage of Unicode's many features.

Character Encoder. The character encoder converts the input text into a binary matrix. This conversion is necessary because deep learning systems require numeric inputs and cannot work directly on text. The original CLCNN model used an encoding matrix constructed by 1-hot encoding the input characters. A 1-hot encoding was feasible because the CLCNN only supported the Latin alphabet. Unicode supports more than one million code points, however, so a 1-hot encoding of a Unicode string is infeasible.[3] We use an alternative to 1-hot encoding that greatly reduces the encoding size and ensures that similar inputs have similar encoding matrices. This latter property facilitates the UnicodeCNN's robustness to spelling mistakes and the transfer of language knowledge observed in Section 2 above. For example, the construction ensures that the English word Kuwait, the Arabic word كويت (which means Kuwait), and the Japanese word クウェート (which also means Kuwait) use similar encoding matrices.

The character encoder produces a $280 \times d$ encoding matrix, where each entry is either 0 or 1, and d is a hyperparameter called the *encoding size*. The number of rows is set to 280 because this is the maximum number of characters in a tweet. Most Unicode characters are encoded as a single row in the encoding matrix, but certain complex characters (e.g. Chinese, Japanese, and Korean characters) get encoded into multiple rows. Twitter limits the size of tweets in these languages to only 140 characters, so tweets in these languages still typically fit in the 280 row encoding matrix.

The first step to generating the encoding matrix is to normalize the string's representation using Unicode's NFKC normalization strategy. NFKC normalization converts the input string into the smallest possible semantically equivalent string. This step ensures that all semantically equivalent input tweets will have the same character encoding matrix.

Next, we loop through each code point in the normalized string and update the corresponding rows in the encoding matrix. We determine the number of rows a code point occupies as follows: If the code point is not a letter, then it gets a single row. Otherwise, we first transliterate the letter into the Latin alphabet. The number of characters in the transliteration is the number of rows the code point gets in the matrix. For example, the Vietnamese character Ố is transliterated into O and so gets only a single row, but the

[3] The sparsity of the encoding doesn't help because the total number of parameters used in future stages of the model is proportional to the size of the encoding matrix and not the sparsity. A hypothetical 1-hot encoding of Unicode code points also wouldn't take advantage of similar code points sharing parameters as our encoding does.

Chinese character 東 gets transliterated into dong and so occupies 4 rows. For each row, the column values are assigned as follows.

Columns 1-7: These columns are a 1-hot encoding of the code point's *Unicode category*. The Unicode category can be either: letter, mark, number, punctuation, symbol, separator, or other.

Column 8: This column is set to one if the code point is either an upper case or title case character, and zero otherwise.

Columns 9-13: These columns are a 1-hot encoding of the code point's *Unicode directionality*. The directionality can be either: strongly left-to-right (e.g. Latin letters), strongly right-to-left (e.g. Arabic letters), weak (e.g. numbers), neutral (e.g. paragraph separators), or an explicit formatting command.

Columns 14-29: These columns encode diacritic marks on letter code points as follows. The code point is decomposed into a ready-made character and combining marks using the NFD normalization scheme. Each mark in the decomposition is assigned a number between 0 and 15 by first multiplying the code point's value by a large prime, then taking the remainder mod 16. Then, column $14 + r$ is set to 1 for each mark. There are thousands of marks defined in the Unicode standard, so many marks will have the same representation. This procedure can be thought of as an advanced form of feature hashing [59] at the mark level.

Columns 30-31: Column 30 is set to 1 if the character is #, and column 31 is set to 1 if the character is @. These characters have their own columns because they have special meaning in Twitter messages (indicating hashtags and mentions).

Columns 32-57: These columns encode the Latin transliteration of the code point. Column 32 is set to 1 if the transliteration of the code point is a, column 33 if the transliteration is b, and so on until column 57 if the transliteration is z.

Column 58: This column is set to 1 if the character is actually from the Latin alphabet.

Column 59: This column is set to 1 if the character is the first character in a transliteration string.

Columns 60-d: If the character is from the Latin alphabet (i.e. Column 58 is 1), then all these bits are set to 0. Otherwise, the code point is multiplied by a large prime and the remainder r with respect to $d - 60$ is taken. Column $60 + r$ is then set to 1 and all others to 0. As with columns 14-29, these columns will have many collisions and can be thought of as feature hashing [59] at the character level. As d increases, the number of collisions decreases, and we expect better training performance. As shown in Table 1, the small and large models use $d = 128$ and the huge model uses $d = 256$.

Convolutional Layers. The character encoding matrix is passed as the input to a series of six temporal convolutional layers. The purpose of these convolutional layers is to find important patterns in the input encoding matrix. Temporal convolution is a standard deep learning method, and a full description is beyond the scope of this paper. Here, we briefly describe the intuition behind these convolutional layers and state the parameters that we use. We refer the reader to the *Deep Learning Book* [18] and the original paper on CLCNNs [62] for mathematical details.

Intuitively, a convolutional layer generates a new "higher level" set of features from the input "low level" features. These higher level features are known to be robust to small variations in spelling. They are formed by convolving a suitable *filter* with the input matrix, then optionally performing *max-pooling*. There are two important parameters: the number of channels and the width. The number of channels determines the number of filters that are applied in parallel, each creating its own set of output features. Increasing the number of channels increases the model's ability to detect important linguistic features, but it also increases the computational complexity of the model and the model's ability to overfit the data. Our Large and Small models use the same number of channels as the Large and Small models of the original CLCNN [62], and our Huge model uses twice as many channels (see Table 1). Our training dataset is orders of magnitude bigger than the datasets used to evaluate the original CLCNN, so we are able to train bigger models without overfitting. The width of a convolutional or max-pooling layer determines the number of input features that get combined into a single output feature. All three sizes of our model use the same size widths as the original CLCNN. The parameters for each convolutional layer are shown in the table below:

Hyperparameter	Small	Large	Huge
encoding size (d)	128	128	256
convolutional channels	256	1024	2048
feature mixing layer size	1024	2048	4096

Table 1: Hyperparameters for the three variants of the UnicodeCNN.

Layer	Kernel Width	Max-Pooling Width
1	7	3
2	7	3
3	3	N/A
4	3	N/A
5	3	N/A
6	3	3

It is possible to replace the CLCNN convolutional layers we use with alternative architectures designed for character-level processing. Alternative architectures have proposed using deeper convolutional layers with resnet connections [10], recurrent neural networks [8], or complex combinations of convolutional and recurent networks [30, 31]. Each of these techniques requires considerable processing resources, however, so we do not exhaustively compare these techniques against each other. (Training the Huge UnicodeCNN requires an estimated 90 GPU-weeks as described in Section 4, and the alternative models are similarly computationally expensive.) We chose to base our model off of the CLCNN's convolutional layers because it had the best performance on initial tests using a small held-out subset of tweets. We speculate that convolutions perform better than recurrent networks on twitter analysis because tweets are guaranteed to be short, but recurrent networks are designed for arbitrary length inputs.

Language Encoder. The output of the convolutional layers is passed to two fully connected *rectified linear unit* (ReLU) layers. For all three model sizes we set the width of these layers to 1024. The

fully connected layers are then passed to a softmax layer which predicts the language of the tweet.

Training the model requires that the text be labeled with the language in some way. For unlabeled text, standard language predictors such as `langid.py` [36] can be used, but the Twitter API associates a language with each tweet and we use this label. The Twitter API labels each tweet with one of 65 officially supported languages or `Unknown` if it cannot determine a language. For example, Croatian is not an officially supported language, and so tweets written in Croatian get classified as `Unknown`. Twitter's language labels are rather noisy. Short tweets are likely to get mislabelled, and many tweets in unsupported languages get misclassified as a supported language. For example, tweets written in Catalan get classified as Spanish, and tweets written in Malay get classified as Indonesian. Our fully trained system estimates the Twitter API's language labels with about 95% accuracy for all three UnicodeCNN sizes, and we suspect this is close to optimal.

Why not just directly use labels provided by the Twitter API or a language classifier? Two reasons: First, we believe our learned language labels are more accurate than the Twitter API's. Second, language classifiers like `langid.py` are slow. Language classifiers use their own alternate pipeline to generate features from text, but we can simply reuse the feature generation pipeline we've already created to avoid duplicated work.

Feature Mixing. The feature mixing stage combines the output of the previous stages to generate the final features. It consists of another two fully connect ReLU layers. The size of these layers depends on the model size and is shown in Table 1. The input to these ReLU layers is a concatenation of the output of the convolutional layers and the softmax language estimate. The output of the feature mixing layer is the features generated by the UnicodeCNN, and these are passed to the output layers.

3.2 Model Output

To geolocate tweets, we connect two types of output layers to the UnicodeCNN that model location at different levels of granularity.

Country Cross Entropy. This output predicts the tweet's country of origin. Determining this country is a standard classification problem, and we therefore use the standard softmax cross entropy loss. Our primary goal is to estimate the exact GPS coordinates using the Mixture of von Mises-Fisher output described below, but we observed that jointly optimizing both the country and the GPS coordinates accelerated training. Using multiple related outputs is commonly called *multitask learning* and is a well known technique for accelerating training [63]. No previous work on tweet geolocation used multitask learning.

Mixture of von Mises-Fisher Distributions. This output layer was introduced by [29] to predict the exact GPS locations of images, and we adapt this work to predict the GPS location of tweets instead. Previous work attempting to predict the exact GPS coordinates of tweets [14] modeled GPS coordinates in Euclidean space. GPS coordinates live in a non-Euclidean manifold, however, and the Euclidean approximation can fail in two ways. First, two points near the poles can have small distance but large difference in GPS coordinates. Second, two points on opposite sides of the international date line will have small distance but large difference in GPS coordinates. The Mixture of von-Mises Fisher distributions does not suffer from these problems because it accounts for the Earth's spherical geometry. This is the first method for tweet geolocation to account for this geometry.

The *von Mises-Fisher* (vMF) distribution is one of the standard distributions in the field of *directional statistics*, which is the study of distributions on hyperspheres. The vMF can be considered the spherical analogue of the Gaussian distribution [e.g. 39] and enjoys many of the Gaussian's nice properties. Thus, the *mixture of vMF* (MvMF) distributions can be seen as the spherical analogue of the commonly used mixture of Gaussian distributions.

We now formally describe the MvMF layer. Let $\mathbb{S}^2 = \{\mathbf{x} \in \mathbb{R}^3 : \|\mathbf{x}\|_2 = 1\}$ denote the unit sphere. Then the density of the vMF distribution is given by

$$\text{vMF}(\mathbf{x}; \mu, \kappa) = \frac{\kappa}{\sinh \kappa} \exp(\kappa \mu^\top \mathbf{x}), \tag{1}$$

where $\mathbf{x}$ is any point in $\mathbb{S}^2$, $\mu \in \mathbb{S}^2$ is called the *mean direction*, $\kappa \in \mathbb{R}$ is called the *concentration parameter*. The density of a mixture of q vMF distributions is given by

$$\text{MvMF}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\kappa}, \mathbf{w}) = \prod_{i=1}^{q} \text{vMF}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\kappa}_i)^{\mathbf{w}_i} \tag{2}$$

where $\boldsymbol{\mu}$ is an array of q mean directions, $\boldsymbol{\kappa}$ is an array of q concentration parameters, and $\mathbf{w}$ is a vector of mixture weights satisfying $\mathbf{w}_i \in (0, 1)$ and $\sum_{i=1}^{q} \mathbf{w}_i = 1$. The associated log loss is

$$\ell(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\kappa}, \mathbf{w}) = -\log \sum_{i=1}^{q} \mathbf{w}_i \text{vMF}(\mathbf{x}; \boldsymbol{\mu}_i, \boldsymbol{\kappa}_i), \tag{3}$$

and this is the expression we optimize. A linear function maps the features generated by the UnicodeCNN onto the $\mathbf{w}_i$ mixture components. Thus, the features extracted from the text determine the relative weight of each component in the mixture, but does not determine the mean direction or the concentration parameters of the components. Each κ_i is initialized to 10, which we empirically observed to correspond to a standard deviation about the size of a large city. Each μ_i is initialized by setting it equal to the location on the sphere of the world's ith most populated city. In our model, we use $q = 10000$ mixture components. The number of components can easily be increased at the expense of more computation, but we found $q = 10000$ to be a good compromise between accuracy of prediction and computational expense.

After training, the values of each $\boldsymbol{\mu}_i$ and $\boldsymbol{\kappa}_i$ are fixed. Inputing a text string fixes the UnicodeCNN's features, which fixes the $\mathbf{w}_i$, and so all the parameters of the distribution are completely determined. Figure 1 plots the density of an example distribution. A point estimate can be computed using the maximum likelihood:

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x} \in \mathbb{S}^2} \text{MvMF}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\kappa}, \mathbf{w}). \tag{4}$$

In our experiments below, we measure the quality of our point estimate by measuring the distance between the tweet's true location and $\hat{\mathbf{x}}$ on the surface of the sphere using Vincenty's numerically stable distance formula [56].

model	average distance (km)	accuracy @50km	@100km	@500km	@1000km	@2000km	@3000km	@country
`lang`	2715.577	0.125	0.169	0.347	0.456	0.632	0.704	0.635
`lang+bow`	2150.998	0.187	0.257	0.465	0.577	0.711	0.762	0.719
UnicodeCNN (Small)	1945.779	0.181	0.246	0.468	0.587	0.741	0.797	0.737
UnicodeCNN (Large)	1826.648	0.192	0.261	0.487	0.608	0.758	0.812	0.752
UnicodeCNN (Huge)	**1783.320**	**0.196**	**0.267**	**0.494**	**0.614**	**0.765**	**0.818**	**0.757**

Table 2: The UnicodeCNN features give the best results across all measures, with the larger UnicodeCNN's giving better results than the smaller ones. The baseline `lang+bow` model generalizes methods from previous work [6, 9, 13, 20, 34, 37, 45, 46, 61] to our setting of multilingual input and GPS location output. Recall that smaller values indicate better performance for average distance and larger values are better for accuracy.

4 EXPERIMENTS

We perform three experiments to validate our UnicodeCNN and MvMF techniques. In our main experiment, we introduce a new multilingual dataset that is orders of magnitude larger than all previous geolocation datasets. We show on this dataset that our UnicodeCNN and MvMF techniques work well for all tweets written in all languages sent from anywhere in the world. The second experiment uses a standard English-only benchmark dataset. We show that our techniques not only work on all languages, but they even outperform specialized English-only techniques on an English-only benchmark task. The third experiment introduces an artificial dataset that helps demonstrate that our UnicodeCNN models are able to learn geographic textual cues in any language.

4.1 Main Experiment

Dataset. Our dataset contains all tweets with geolocation information sent between 26 October 2017 and 08 July 2018. The dataset contains more than 900 million tweets written by 3.0 million users in over 100 languages. Unlike previous work, we perform no filtering to remove "hard" tweets from the dataset, and as a result our dataset is orders of magnitude larger than previously used datasets.

The largest previously used dataset for geolocating tweets was the WORLD+ML dataset [21], which contains only 23 million tweets written by 2.1 million users. WORLD+ML includes tweets in all languages, but removes hard-to-geolocate tweets that are not close to major cities. Whereas the WORLD+ML dataset contained only 47% non-English tweets, our dataset contains 58% non-English tweets. Other datasets such as WNUT [22], WORLD [19], and NA [47] additionally remove non-English tweets and so are considerably smaller.

For each tweet in our dataset, we associate a country and GPS coordinates using the Twitter API. The Twitter API defines a total of 247 unique country codes that a tweet can be sent from. This number is larger than the number of sovereign states recognized by the United Nations (206) because many non-countries are given country codes (e.g. Puerto Rico and Hong Kong). The process of assigning GPS coordinates is slightly more complicated. User privacy settings allow Twitter to share the exact GPS coordinates of approximately 16% of tweets in our dataset. For the remaining tweets, Twitter only provides the city of origin. For these tweets, we take the city's center of mass to be the true GPS location of the tweet. There are approximately 3 million unique city-level locations in our dataset. The WNUT [22], WORLD [19], and WORLD+ML [21] datasets generated class labels using a complex procedure to: (i) select approximately the 3000 most populated of these cities, (ii) combine them with nearby cities into a single metropolitan area which serves as the class label, and (iii) discard all tweets not from these metropolitan areas. Clearly, significant amounts of information is lost when creating class labels in this way as the transformation is not reversible. Our method of using the GPS coordinates preserves as much information from the tweet as possible and creates a more challenging prediction problem.

Baseline Models. We compare our UnicodeCNN with three baseline models. All three models use multitask learning to predict both the country and gps coordinates a tweet was sent from as described in Section 3.2. The only difference is the way the features are generated.

The first model, `lang` uses only the language of the tweet 1-hot encoded as an input feature. This is the simplest reasonable model for the multilingual geolocation problem. This model is able to capture the fact that different countries use different languages, but is unable learn patterns about how different regions use different dialects of the same language.

The final model, `lang+bow` adds bag-of-words features to the model generated from the text. Bag-of-words features are a classic method for analyzing English-language text corpora, and our model follows best practices. In particular, we use feature hashing with L1 regularization to automatically select the best features (which correspond to the most location indicative words). Bag-of-words features are the most popular features used in content-based tweet geolocation, and they have been used in all the following papers: [6, 9, 13, 17, 20, 34, 37, 38, 45, 46, 54, 61]. Our results are not directly comparable to the results in these papers because we perform no filtering of hard tweets in our dataset, and we use the MvMF output layer to predict exact GPS coordinates.

Many methods augment their bag-of-words models with additional features. For example, Zhang and Gelernter [61] use *gazeteers* (databases that map place names to GPS coordinates) to improve the quality of prediction. Because these gazeteers are only available in English, they are not applicable in our highly multilingual domain, and we do not compare against this method. Other methods use the social graph or metadata associated with each tweet to improve performance [e.g. 21, 24, 51]. We do not compare against these methods because our focus is only on content-based geolocation. Considering other non-content features would obscure the impact of the tweet's text on the predicted location.

Training and Evaluation Procedure. We train all models using the Adam optimizer [32] and evaluate performance online. That is, we perform only a single pass over the dataset. For each tweet, we first compute the loss of the model on that tweet; then we update the model's parameters. The total loss is then the sum of the individual tweet losses. Online learning is a standard learning paradigm [e.g. 52], and we emphasize that at all time steps our evaluated losses are unbiased estimates of the model's true loss. We choose online evaluation rather than the more popular batch evaluation (i.e. with a train/test split) for three reasons. First, training and evaluating the bigger UnicodeCNN models is extremely computationally expensive (many weeks using parallel computing as described below). Many of the computations between the Adam parameter update and loss computation can be shared, and so the total amount of computation is reduced in the online setting. Second, online evaluation is a natural model for tweet geolocation because new data can easily be obtained. The train/test split experimental method was designed for situations with limited training data where obtaining new evaluation data is difficult. Finally, our dataset is large enough that a single pass is sufficient to train the models to convergence. If multiple passes were needed, then the evaluated losses would no longer be unbiased estimates of the true loss after the first pass.

We use several losses to evaluate the models' performance. The *distance* measures the distance between a tweet's true location and the point estimate given by the MvMF output. The *accuracy @50km* is the fraction of tweets whose distance error is less than 50km. The accuracy @100km, @500km, 1000km, 2000km, and 3000km measures are similar. Finally, the *accuracy @country* measures the fraction of tweets whose country of origin was correctly predicted. Table 2 shows numerical results for all baseline and all UnicodeCNN models evaluated on the full dataset. The Huge UnicodeCNN model geolocates tweets on average 367km closer to their true location than the baseline `lang+bow` model. Table 3 compares the performance of these two models over different languages. For all languages, the Huge UnicodeCNN significantly outperforms the `lang+bow` model. The difference is most striking for those tweets that the Twitter API classifies as having an Unknown language, where the UnicodeCNN geolocates tweets an average of 1563km closer. Recall that a tweet gets labelled as Unknown if the language is so infrequently used that it is not officially supported by Twitter, or the text is too short. These are the hardest tweets to geolocate, and the tweets worst served by current state-of-the-art. They comprise approximately 7.4% of all tweets, which is about 40 million tweets per day.

To train the UnicodeCNN models, we use a single machine with 16 CPUs, 64 GB of RAM, and 6 NVidia Titan x80 GPUs. We train using the Adam optimizer [32] with a learning rate of 5×10^{-4} and a batch size of 600. The character encoding is computed on the CPUs, and all other parts of the model are computed on the GPUs. We use data parallelism, so that each GPU processes 100 tweets per batch, and then the gradients are averaged together. The batch size of 100 tweets/GPU is the largest batchsize we could fit on a GPU with the huge model. We observed about a 5-fold speed up using all 6 GPUs, and in total, training the large model took about 4 weeks with this setup. The results reported for the huge model are on approximately 20% of the data after 3 weeks of training. We estimate that training the huge model on the full dataset would take about 15 weeks. We expect the huge model's performance would continue to slightly improve given more training time. An ablative study comparing all possible character encodings and model architectures discussed in Section 3 is clearly infeasible when model training is so expensive.

language	fraction of dataset	model	average distance (km)	accuracy @country
English	0.425	`lang+bow`	2708.620	0.703
		UnicodeCNN (Huge)	**2326.873**	**0.745**
Portuguese	0.139	`lang+bow`	785.798	0.940
		UnicodeCNN (Huge)	**734.632**	**0.951**
Spanish	0.102	`lang+bow`	2757.163	0.479
		UnicodeCNN (Huge)	**2343.886**	**0.535**
Unknown	0.074	`lang+bow`	5120.919	0.429
		UnicodeCNN (Huge)	**3557.385**	**0.525**
Japanese	0.060	`lang+bow`	337.931	0.964
		UnicodeCNN (Huge)	**332.421**	**0.968**
Arabic	0.030	`lang+bow`	723.563	0.522
		UnicodeCNN (Huge)	**707.136**	**0.562**
Turkish	0.029	`lang+bow`	323.291	0.901
		UnicodeCNN (Huge)	**296.059**	**0.943**
Indonesian	0.025	`lang+bow`	1060.114	0.734
		UnicodeCNN (Huge)	**899.193**	**0.824**
Tagalog	0.024	`lang+bow`	1086.997	0.788
		UnicodeCNN (Huge)	**983.495**	**0.872**
Other	0.090	`lang+bow`	1738.139	0.731
		UnicodeCNN (Huge)	**1379.444**	**0.767**

Table 3: The performance of the best baseline and UnicodeCNN models across the ten most popular languages as labelled by the Twitter API. The Huge UnicodeCNN significantly outperforms the `lang+bow` baseline model on all languages. Notice the dramatic improvement for tweets with unknown language.

All baseline models were trained using Adam [32] and random hyperparameter search [3]. Random hyperparameter search is a state-of-the-art hyperparameter tuning method that requires no manual intervention and works well when there is more than one hyperparameter to tune. For each model, we randomly selected a learning rate, L2 regularization strength, and (for the `lang+bow` model only) L1 regularization strength in the range 10^{-6} to 10^{0} distributed uniformly over the logarithm. We trained 20 versions of each model and report only the best model.

4.2 English Benchmark

The 2016 Workshop on Noisy User-generated Text (WNUT2016) hosted a tweet geolocation challenge [22]. The dataset for the challenge consists of 12.8 million tweets sent between 2013 to 2015. In total, 419 million geotagged tweets were sent during this time period, but all tweets not written in English and not sent from a major city were discarded from the dataset. That is, 97% of available tweets were excluded from the competition because they were thought to be too difficult to geolocate. In this experiment, we will show that our UnicodeCNN+MvMF model achieves state-of-the-art performance on this dataset of easily geolocated tweets. In other

model	accuracy	median (km)	mean (km)
Chi et. al., 2016 [7]	0.121	3105.8	4867.5
`lang`	0.041	2678.4	3609.8
`lang+bow`	0.071	1710.5	3292.4
UnicodeCNN (Small)	0.119	1621.3	3140.5
UnicodeCNN (Large)	0.131	1562.1	2898.3
UnicodeCNN (Huge)	**0.133**	**1532.8**	**2802.1**

Table 4: **Evaluation on the WNUT2016 test set for English language content based tweet geolocation. The UnicodeCNN outperforms the previous state-of-the-art.**

words, this experiment shows that our methods beat existing methods at the small subset of tweets those methods were designed for, plus our methods generalize this performance to all tweets.

Table 4 compares our methods with the existing state-of-the-art on the WNUT2016's validation dataset. In this table, the "accuracy" represents the probability that the tweet was geolocated to the correct city, and the "median" and "mean" distance represents the distance between the estimated city location and the true city location. All prior work using the WNUT2016 dataset has focused on optimizing the accuracy of prediction but acknowledged that the median/mean distance more accurately captures desired real world performance.

Chi et. al., 2016 [7] have the previous best reported performance using content-based geolocation techniques only.[4] Their method uses a special feature extraction specialized to the English language, and a classification strategy to select the most probable city. Remarkably, our `lang` baseline method has better median and mean distances even though it uses no input features! (The only feature, language, is the same for all tweets.) The reason is that it uses the MvMF loss, which is specifically designed to select locations that have low distance. This shows that the MvMF loss is clearly better than classification-based strategies when our goal is to minimize distance. The UnicodeCNN has better features than Chi et. al.'s method, and has both slightly better accuracy and median/mean performance almost twice as good.

4.3 Artificial Data

Our final experiment shows that the UnicodeCNN learns features related to geographical words in the tweet's text, and is able to transfer these features to other languages. It is a quantitative version of the examples in Section 2 and helps to explain why the UnicodeCNN performs well in the linguistically diverse main experiment.

We generated two artificial datasets according to the following procedure. First, we created an English-language dataset using a list of the world's 100 "most prominent" cities.[5] For each city we generated an artificial tweet containing the text "city, country" and with GPS position at the center of the city. For example, for the city of Tokyo, we generated a tweet with text

```
Tokyo, Japan
```

[4]Other techniques have taken advantage of the social graph or metadata embedded within the tweet. These methods have better performance, but they are not comparable to our text-only method and so not reported here.

[5]As provided by https://simplemaps.com/data/world-cities

model	English	other
`lang`	0.140	0.082
`lang+bow`	0.270	0.163
UnicodeCNN (Small)	0.710	0.250
UnicodeCNN (Large)	0.920	0.346
UnicodeCNN (Huge)	**0.940**	**0.382**

Table 5: **Accuracy at predicting country location on synthetic dataset of city names.**

located at 35.685N, 139.7514E.

Then we generate a second dataset by translating the tweets in the first dataset into 20 other languages using Google Translate. We selected the following languages for their linguistic diversity: Arabic, Burmese, Chinese (Simplified), Czechoslovakian, Farsi, French, German, Greek, Hebrew, Indonesian, Italian, Japanese, Korean, Pashto, Punjabi, Portuguese, Spanish, Tagalog, and Turkish. In this other-language dataset, the Tokyo tweet from before is represented in Spanish as

```
Tokio, Japón
```

and in Japanese as

```
東京、日本
```

Notice that this dataset contains tweets written in many scripts, and even languages using the latin script represent cities and countries in different ways. The vast majority of these translations do not exist anywhere in our training data.

Table 5 shows the results of predicting the country of origin for these tweets. The `lang` model on the English dataset has no features to use, and achieves 0.14 accuracy simply be guessing the United States for each location. The UnicodeCNN is able to predict the correct country for almost all English language tweets. This indicates that the model has learned features that capture the geographic meaning of most city and country names. Some cities (such as Pyongyang, DPRK) have essentially no training data in our dataset, however, and so the UnicodeCNN is not able to learn good features for these cities.

5 DISCUSSION

The UnicodeCNN is the first deep learning model that supports any input language and exploits the Unicode standard's many features. Many text processing pipelines could be made multilingual simply by replacing the existing feature extraction methods with the UnicodeCNN. Unfortunately, training the UnicodeCNN from scratch is computationally expensive. We ameliorate this problem by open sourcing our learned model weights. Transfer learning can then be used to reuse these weights, learning only a new application-specific output layer [26, 57]. Transfer learning is known to be especially effective where the original model was trained on a large dataset like ours.

ACKNOWLEDGEMENTS

Vagelis Papalexakis was supported by NSF grants OAC-1808591 and IIS-1746031 and DON grant N00174-17-1-0005. Vassilis Tsotras was supported by NSF grants IIS-1447826 and IIS-1527984.

REFERENCES

[1] Dolan Antenucci, Michael Cafarella, Margaret Levenstein, Christopher Ré, and Matthew D Shapiro. 2014. *Using social media to measure labor market flows*. Technical Report. National Bureau of Economic Research.

[2] Pablo Barberá. 2014. Birds of the same feather tweet together: Bayesian ideal point estimation using Twitter data. *Political Analysis* 23, 1 (2014), 76–91.

[3] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *JMLR* (2012).

[4] Su Lin Blodgett, Lisa Green, and Brendan O'Connor. 2016. Demographic dialectal variation in social media: A case study of African-American English. *arXiv preprint arXiv:1608.08868* (2016).

[5] Su Lin Blodgett and Brendan O'Connor. 2017. Racial Disparity in Natural Language Processing: A Case Study of Social Media African-American English. *arXiv preprint arXiv:1707.00061* (2017).

[6] Zhiyuan Cheng, James Caverlee, and Kyumin Lee. 2010. You are where you tweet: a content-based approach to geo-locating twitter users. In *CIKM*.

[7] Lianhua Chi, Kwan Hui Lim, Nebula Alam, and Christopher J Butler. 2016. Geolocation prediction in Twitter using location indicative words and textual features. In *WNUT*. 227–234.

[8] Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. 2016. A character-level decoder without explicit segmentation for neural machine translation. *arXiv preprint arXiv:1603.06147* (2016).

[9] Ryan Compton, David Jurgens, and David Allen. 2014. Geotagging one hundred million twitter accounts with total variation minimization. In *Big Data*.

[10] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. 2017. Very deep convolutional networks for text classification. In *EACL*.

[11] Michael Conover, Jacob Ratkiewicz, Matthew R Francisco, and Bruno Gonçalves. 2011. Political polarization on twitter.. In *ICWSM*.

[12] Kate Crawford and Megan Finn. 2015. The limits of crisis data: analytical and ethical challenges of using social and mobile data to understand disasters. *GeoJournal* (2015).

[13] Mark Dredze, Miles Osborne, and Prabhanjan Kambadur. 2016. Geolocation for Twitter: Timing Matters.

[14] Nghia Duong-Trung, Nicolas Schilling, and Lars Schmidt-Thieme. 2016. Near real-time geolocation prediction in twitter streams via matrix factorization based regression. In *CIKM*.

[15] Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2015. Multilingual language processing from bytes. *arXiv preprint arXiv:1512.00103* (2015).

[16] Bruno Gonçalves, Lucía Loureiro-Porto, José J Ramasco, and David Sánchez. 2017. The fall of the empire: The americanization of English. *arXiv preprint arXiv:1707.00781* (2017).

[17] Bruno Gonçalves and David Sánchez. 2015. Learning about Spanish dialects through Twitter. *arXiv preprint arXiv:1511.04970* (2015).

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[19] Bo Han, Paul Cook, and Timothy Baldwin. 2012. Geolocation prediction in social media data by finding location indicative words. *COLING* (2012).

[20] Bo Han, Paul Cook, and Timothy Baldwin. 2013. A Stacking-based Approach to Twitter User Geolocation Prediction.

[21] Bo Han, Paul Cook, and Timothy Baldwin. 2014. Text-based twitter user geolocation prediction. *JAIR* (2014).

[22] Bo Han, Afshin Rahimi, Leon Derczynski, and Timothy Baldwin. 2016. Twitter geolocation prediction shared task of the 2016 workshop on noisy user-generated text. In *WNUT*.

[23] Eszter Hargittai. 2018. Potential Biases in Big Data: Omitted Voices on Social Media. *Social Science Computer Review* (2018), 0894439318788322.

[24] Brent Hecht, Lichan Hong, Bongwon Suh, and Ed H Chi. 2011. Tweets from Justin Bieber's heart: the dynamics of the location field in user profiles. In *SIGCHI*.

[25] Lichan Hong, Gregorio Convertino, and Ed H Chi. 2011. Language Matters In Twitter: A Large Scale Study.. In *ICWSM*.

[26] Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned Language Models for Text Classification. *arXiv preprint arXiv:1801.06146* (2018).

[27] Yuan Huang, Diansheng Guo, Alice Kasakoff, and Jack Grieve. 2016. Understanding US regional linguistic variation with Twitter data analysis. *Computers, Environment and Urban Systems* 59 (2016), 244–255.

[28] Muhammad Imran, Prasenjit Mitra, and Carlos Castillo. 2016. Twitter as a lifeline: Human-annotated twitter corpora for NLP of crisis-related messages. *arXiv preprint arXiv:1605.05894* (2016).

[29] Mike Izbicki, Vagelis Papalexakis, and Vassilis Tsotras. 2019. Exploiting the Earth's Spherical Geometry to Geolocate Images. *ECML-PKDD* 3, 2 (2019), 223–239.

[30] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410* (2016).

[31] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-Aware Neural Language Models.

[32] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *ICLR* (2014).

[33] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2016. Fully character-level neural machine translation without explicit segmentation. *arXiv preprint arXiv:1610.03017* (2016).

[34] Rui Li, Shengjie Wang, Hongbo Deng, Rui Wang, and Kevin Chen-Chuan Chang. 2012. Towards social user profiling: unified and discriminative influence model for inferring home locations. In *SIGKDD*.

[35] Alejandro Llorente, Manuel Garcia-Herranz, Manuel Cebrian, and Esteban Moro. 2015. Social media fingerprints of unemployment. *PloS one* 10, 5 (2015), e0128692.

[36] Marco Lui and Timothy Baldwin. 2012. langid. py: An off-the-shelf language identification tool. In *ACL*.

[37] Jalal Mahmud, Jeffrey Nichols, and Clemens Drews. 2014. Home location identification of twitter users. *TIST* (2014).

[38] Wolfgang Maier and Carlos Gómez-Rodríguez. 2014. Language variety identification in Spanish tweets. In *EMNLP*.

[39] K.V. Mardia and P.E. Jupp. 2009. *Directional Statistics*.

[40] Quynh C Nguyen, Suraj Kath, Hsien-Wen Meng, Dapeng Li, Ken R Smith, James A VanDerslice, Ming Wen, and Feifei Li. 2016. Leveraging geotagged Twitter data to examine neighborhood happiness, diet, and physical activity. *Applied Geography* (2016).

[41] Brendan O'Connor, Michel Krieger, and David Ahn. 2010. TweetMotif: Exploratory Search and Topic Summarization for Twitter.. In *ICWSM*.

[42] Pavlos Paraskevopoulos and Themis Palpanas. 2015. Fine-grained geolocalisation of non-geotagged tweets. In *ASONAM*.

[43] Michael J Paul, Mark Dredze, and David Broniatowski. 2014. Twitter improves influenza forecasting. *PLoS Currents* (2014).

[44] Barbara Plank, Anders Søgaard, and Yoav Goldberg. 2016. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. *arXiv preprint arXiv:1604.05529* (2016).

[45] Afshin Rahimi, Trevor Cohn, and Timothy Baldwin. 2015. Twitter user geolocation using a unified text and network prediction model. *arXiv preprint arXiv:1506.08259* (2015).

[46] Afshin Rahimi, Trevor Cohn, and Timothy Baldwin. 2017. A neural model for user geolocation and lexical dialectology. *arXiv preprint arXiv:1704.04008* (2017).

[47] Stephen Roller, Michael Speriosu, Sarat Rallapalli, Benjamin Wing, and Jason Baldridge. 2012. Supervised text-based geolocation using language models on an adaptive grid. In *EMNLP*.

[48] Koustav Rudra, Siddhartha Banerjee, Niloy Ganguly, Pawan Goyal, Muhammad Imran, and Prasenjit Mitra. 2016. Summarizing situational tweets in crisis scenario. In *HT*.

[49] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. 2010. Earthquake shakes Twitter users: real-time event detection by social sensors. In *WWW*.

[50] Mauricio Santillana, André T Nguyen, Mark Dredze, Michael J Paul, Elaine O Nsoesie, and John S Brownstein. 2015. Combining search, social media, and traditional data sources to improve influenza surveillance. *PLoS Computational Biology* (2015).

[51] Axel Schulz, Aristotelis Hadjakos, Heiko Paulheim, and Johannes Nachtwey. 2013. A Multi-Indicator Approach for Geolocalization of Tweets.

[52] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning*. Cambridge University Press.

[53] The Unicode Consortium. [n. d.]. The Unicode Standard.

[54] Antonio Ruiz Tinoco. 2017. Variation of the second person singular of the simple past tense in twitter: hiciste vs. hicistes" you did". *Dialectologia: Revista Electrònica* (2017).

[55] Zeynep Tufekci. 2014. Big Questions for Social Media Big Data: Representativeness, Validity and Other Methodological Pitfalls.. In *ICWSM*.

[56] Thaddeus Vincenty. 1975. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review* (1975).

[57] Dong Wang and Thomas Fang Zheng. 2015. Transfer learning for speech and language processing. *arXiv preprint arXiv:1511.06066* (2015).

[58] Joonatas Wehrmann, Willian Becker, Henry EL Cagnini, and Rodrigo C Barros. 2017. A character-based convolutional neural network for language-agnostic Twitter sentiment analysis. In *IJCNN*.

[59] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *ICML*.

[60] Michael J Widener and Wenwen Li. 2014. Using geolocated Twitter data to monitor the prevalence of healthy and unhealthy food references across the US. *Applied Geography* (2014).

[61] Wei Zhang and Judith Gelernter. 2014. Geocoding location expressions in Twitter messages: A preference learning method. *Jour. of Spatial Info. Science* (2014).

[62] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*.

[63] Yu Zhang and Qiang Yang. 2017. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114* (2017).