GEOAI CHALLENGE

Location Detection from Social Media Crisis-related Text

Team Patterns

Becka Cammon

Noreen Chihora

Prathamesh Kulkarni

Saswata Rautray

Tejaswi Maruthi

Table of Contents

Table of Figures

Leaderboard Rankings 🏆

Competition Stage leaderboard

Ranking	Team	Submit Time	Score
1	Patterns	2022-11-22 14:04:55	90
2	TetisTextMining	2022-11-23 15:43:29	89
3	GateId	2022-11-29 23:31:37	50

https://geoaichallenge.aiforgood.itu.int/match/matchitem/64

Executive Summary

As technology advances, so does the speed at which information travels around the world. The internet has made it possible for almost anyone with an internet-connected device, regardless of age, to gain access to vast amounts of information. Service providers have developed a system to support and facilitate the sharing of information among people. Facebook, Twitter, and Instagram are some of the most popular online social media services. A wide range of unstructured data can be extracted from social media daily and help us to understand a substantial number of topics e.g., brand reliability, societal concerns, or even real time news like an ongoing crisis or disaster. Unlike other social media platforms or traditional platforms such as newspapers, radios, and magazines, Twitter only allows users to post messages of up to 280 characters. This restriction makes twitter an easier platform to extract information related to any topic.

Using Twitter messages that users have published, this research project seeks to extract toponym (i.e., place/area/street names) information to assist emergency providers to locate people in need of help in the case of a disaster. Crisis-related social media analysis, from tweets, is a well-researched field with a lot of labelled data available for various tasks (Imran et al., 2016; Middleton et al., 2014; Alam et al., 2018). Crisis related data usually involves human annotation. Most of the available human annotated datasets consist of thousands of instances compared to hundreds of thousands in other datasets. This means that crisis datasets are relatively small compared to those available for tasks in other domains. Furthermore, for tasks that can support a new emergent crisis, human-labelled data of a large volume cannot be generated as human annotation is limited. One popular approach in addressing the size limitation of labelled data for a particular task is to leverage unlabeled data. In the field of Natural Language Processing (NLP), the recent advancements in transformer-based architectures, and associated pre-training with large amounts of unlabeled data, have largely been successful in addressing this issue. Transformer-based architectures currently hold state-of-the-art results for many NLP tasks.

Using NLP for spatial data science, one can automate the extraction of locational features from documents, allowing one to discover which locations are present in a text document. The presence of locational features in a document allows for the exploration of spatial patterns and gaining insight from ongoing tasks or historical archives. Identification of location entities in tweets posted during crisis events is vital in extracting actionable situational awareness information. Furthermore, identifying the entities according to a hierarchy of location types can help in geographical location disambiguation and geo-coding. The goal of this project is to create a model that can automate the process of location recognition and geo localization of location mentions in tweets by developing a system for Location Mention Recognition (LMR).

This program will automatically extract toponyms from tweets for responsible authorities to urgently assist in the case of a disaster and reduce response time. In this research project we used different approaches to extract the exact location mentioned in the tweets. We used different NLP approaches that include a BERT model and a Huggingface inspired FLAIR transformer framework. We analyzed the working mechanisms of the machine learning models and determined the best solution to the given problem. The

model performance was judged using the confusion matrix metrics such as F-1 score, precision and recall. Accuracy is not a great metric in this situation because it does not perform well with unbalanced labelled data. The BERT model initially gave an F-1 score of 0.674 The F1 score was then improved by trying different BERT models such as base uncased and based uncased from 0.674 to 0.658. Even though the BERT model is a well-known state-of-the-art transformer-based model we tried the HuggingFace inspired We then tried to use a HuggingFace inspired FLAIR framework that facilitate training and distribution of state-of-the-art NLP models for named entity recognition, part-of-speech tagging, and text classification. The performance metric that we received from the FLAIR model were extraordinary, the F-1 score of 90.5 was reached by the model. This model was then submitted to the GEO-AI competition hosted by QCRI.

Introduction

About Our Client –Qatar Computing Research Institute (QCRI)

QCRI is a research institute founded by Qatar government to support the research work. It helps by tackling large-scale computing challenges for growth and development of the betterment of the world. In doing this, QCRI conducts world-class multidisciplinary computing research that is relevant to the needs of Qatar, and the world. The institution performs cutting-edge research in such areas as Arabic language technologies, social computing, data analytics and cyber security.

Vision of QCRI

- Excellence: Conduct research of the highest quality and publish research results of indisputable value.
- Innovation: Introduce visionary solutions to significant scientific or technological challenges.
- Integrity: Create an environment that fosters dedication to honest and ethical research practices and scientific processes.
- Leadership: Provide leadership in our areas of expertise to global science, international research communities and local society.
- Collaboration: Promote open collaboration among colleagues and with partners.
- Impact: Be result-oriented and provide solutions to emerging problems, ensuring alignment with national priorities.

Stakeholders

The primary stakeholders are:

- Qatari society, which needs a way to retain its top talent in the field of computer science and to attract top graduates from elsewhere in the region to ensure Qatar's sustainability in all fields.

- Qatari industry, which needs advanced applied research solutions before they are available commercially.
- Qatar's government, which needs working solutions that will meet the growing needs of the country and its people.

Location Extraction system architecture

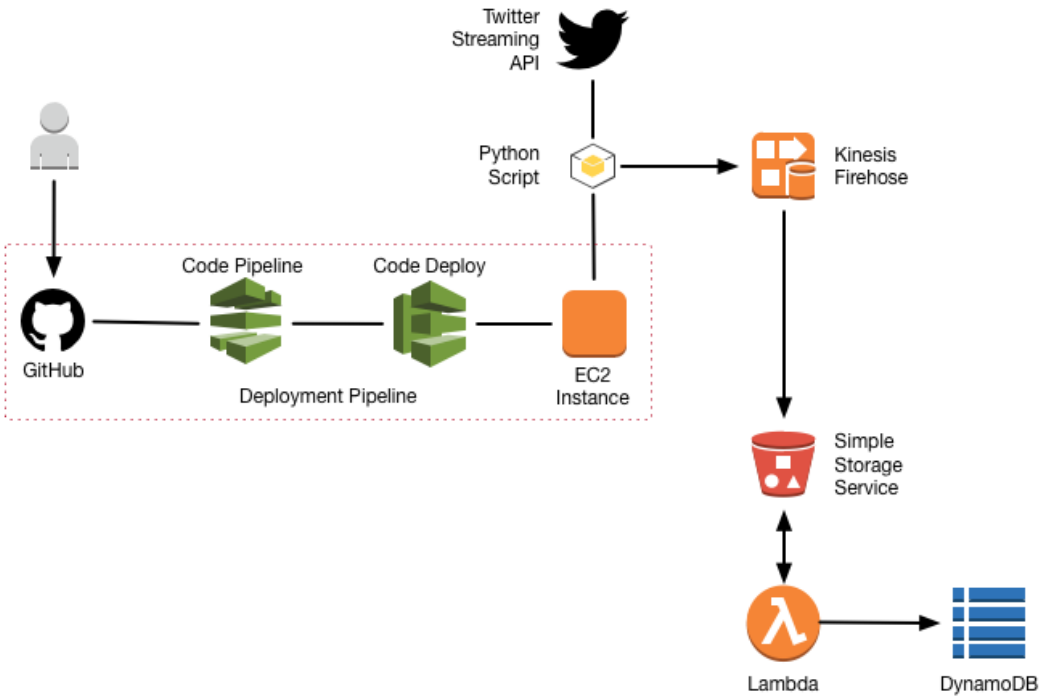

Figure 1. Location extractor NLP pipeline (twitter API based).

It has the following components:

1. Mailbox to be polled for new emails
   Workflow Orchestrator that controls the flow of data:
   Polls the mailbox (1)
   Makes requests to the Prediction Worker (3)
2. Saves the email bodies along with extracted entities to the DB (4)
3. Prediction Worker doing ML by extracting entities applying a model to a given email body
4. Database storing email bodies and extracted entities
5. (Optional) Model Trainer that would:

a. Form a training dataset querying against the DB (4)
b. (Re)train models when needed
c. Validate the model against the test dataset
d. Store the model in the Model Storage (6)
e. Change production model pointer to the new model if validation is successful

6. (Optional) Model Storage in the form of object storage like S3 bucket.

Challenges & Opportunity

At the onset of a disaster, Twitter messages with critical information such as medical assistance, food or shelter needs, reports of trapped people, and severely damaged infrastructure, are not useful for response authorities if they are not geotagged. While tweets with exact GPS coordinates are usually scarce, often they contain toponyms (i.e., place/area/street names), which can be helpful for authorities to estimate the location. However, due to the high volume of Twitter messages, manual extraction of location cues cannot scale. An automated process is required for the recognition and geo localization of location mentioned in tweets. Therefore, the goal of this challenge is to encourage the development of systems for Location Mention Recognition (LMR) from microblogs during emergencies. These automatic systems are anticipated to support the relief activities that are executed by response authorities during disasters.

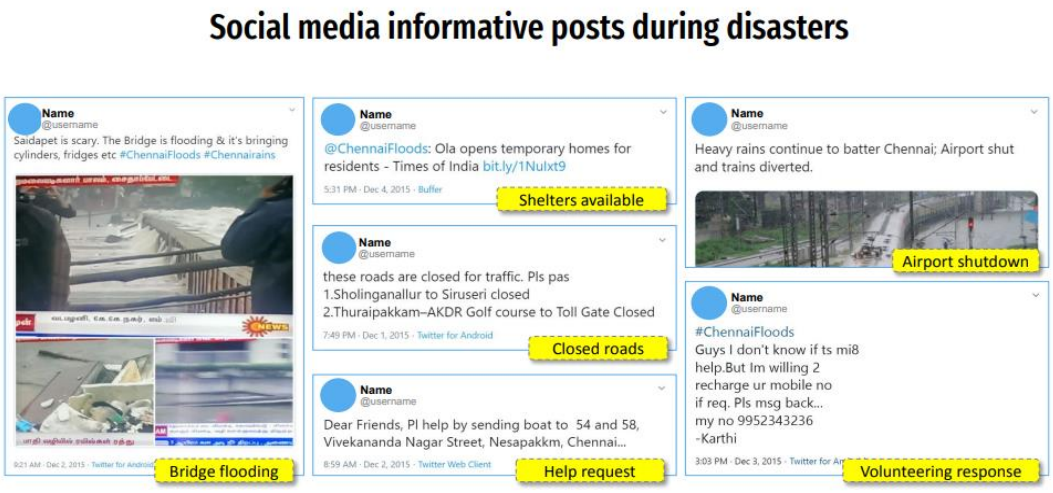

Figure 2. Social media posts during disasters.

The task of LMR is to automatically extract the locations and places mentioned from a corpus of text using text analysis. So, in the tweets provided in the dataset, LMR recognizes the start and end positions of the location in the entire text. For example: Stillwater has just been hit by a severe tornado. The location Stillwater has a start index of 0 and end index of 9.

Since the digitalization and the use of smartphones increased exponentially, GPS tracking and mapping has been easy. This GPS data produced includes crowdsourcing and user generated content (See et al.

2016). Goodchild (2007) terms this wealth of geo data as 'Volunteered Geographic Information' (VGI). This data is usually unstructured and continually updated.

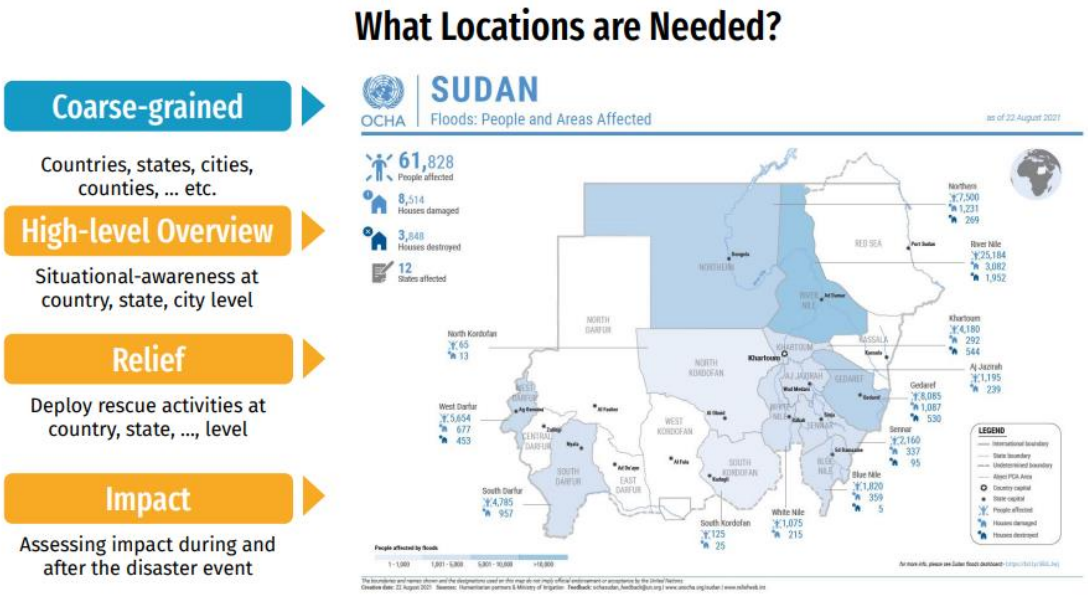

Figure 3. A Geocoded location finder using NLP to find location

One of the platforms that use this VGI data extensively is Open Street Map, a crowd sourced mapping platform (Antoniou et al. 2010). It has geo tagged data from user generated sites like Flickr and twitter and is used to extract information regarding location. One such application is by Gao et al. (2017) where they used the VGI data to create boundaries/cognitive regions by extracting location names from associated tags. Another approach to identify tags like 'citycentre', 'downtown' has been proposed by Hollenstein and Purves (2010). These approaches extract geographic information from Volunteered Geographic Information and gave similar results as manually collected questionnaire data (Gao et al. 2017, Twaroch et al. 2019).

Literature review

Artificial Intelligence (AI) is the domain of technologies where machines are trained to think in parallel to humans. Natural Language Processing is a subpart of AI, where humans focus the attempt to train machines/computers to understand the languages of humans. NLP consists of training machines with large corpus of data and most of this data comes from online social media/web content. There has been extensive research in the field of NLP over the last decade. Especially in the topic of Location mention recognition, NEED4Tweet a twitter bot was presented by Habib and M. van Keulen. NEED4Tweet uses the concepts of Name Entity extraction (NEE) and Name Entity Disambiguation (NED). Then came an automatic location expressions identifier by F. Liu et al., which used large data from social media. The next major study was regarding the potential of NER in location extraction from tweets by J. Lingad et al. Using location extraction using NLP has been successful for disaster management. In the field of docal media

analysis applications, NER experiments were reported on Turkish tweets by D. Küçük and R. Steinberger to determine the factors that are impeding the development of NER system. In NLP muti task learning has been very effective (Ruder, 2017). Multitask learning is popularly implemented by hard parameter sharing where a module shared by all tasks is followed by task-specific modules Caruana (1997). Whereas in soft parameter sharing each task has its own set of layers, but the parameters of the task-specific layers are constrained to be similar across tasks to enforce exchange of information among tasks (Duong et al., 2015; Yang and Hospedales, 2017). In the recent days, for disaster management, a multi modal muti task model was proposed by Wang et al. (2020). The model uses a single muti modal data set consisting of different labels for different tasks. During the same year, for predicting muti token key phrases, a single token keywords identification as an auxiliary task has been proposed by Chowdhury et al. (2020).

Defining the Problem and Identifying the underlying Business Problem

Our inference of the GEO-AI problem is that there needs to be a model that can beat the average BERT based model in performance and help to identify the contextual information in the given text. The current state of the model gives us the average F-1 score of 89.45 for the given disaster data from IDRIS. The models need to good at understanding the incoming data and the extract the location mentioned in the text. The model needs to be trained on the 19-disaster mentioned by the IDRIS GitHub and then based on the trained weights help to identify the location mentioned in the unlabeled data. A faster responsive system can be made using the available deep learning framework such as Pytorch or TensorFlow that can be used further in a deployed machine learning pipeline. We need to provide a novel model that is easy to use and can be easily accommodated with some NLP pipeline that receives twitter data using an API. A simpler model also means that the computing resource needed for the model will limited to basic GPU that was used while training the model.

A faster response system can be then implemented by the government entities to help the areas affected by the disaster swiftly. The loss of precious time due to not knowing the exact coordinate of the location is causing a lot of loss to life, property, and finances in some disaster-prone areas. An NLP-based location extractor would certainly help the Geocoding tool such as Geocoding API by google or other open-source platforms that find the coordinates of a position based on the location name provided. A swift relief service will also help the authorities to better understand the most disaster-sensitive areas and provide the quickest help possible. While this work concentrates solely on the use of geotags and short single phrase tags associated with social media documents to analyze ‘place’ focused geographies, another source of online information that is less frequently considered to have geographic properties is unstructured text, which has the potential to provide an even larger source of geographically focused information. Good results have been reported using basic semantic rules to identify places names found in unstructured text (Moncla et al. 2014), however, these methods have relied on this text almost solely containing place names as entities.

According to a gritta article from 2017 the NER model was trained on hinders, name extraction, place, and identification of place names within texts. The Goo Mordecai 2 uses a NER tagger from the SpaCy Python library. The SpaCy Python library provides many entities, like name, places, and 4 C. As well as GPE, LOC, and FAC. (Geopolitical Entity, Location, Facility). The problem with these three entities is that they do not take into consideration that a place's name can be geo-located. When thinking about place names, they do not directly relate to geographic locations. Due to this, there has not been a lot of research done that considered focusing on place names in new data since it is very time-consuming to report on.

Current NER models are now defined by " widely used annotated corpora; some work has considered the need to identify spatial entities."SpatialML is defined as "a natural language annotation scheme that presents the place tag for any mention of a location." According to the semantic evaluation workshop, built tasks are related to the spatial language that the ISO-Space annotation specification described. To properly consider geography when while parsing unstructured texts, you need to build models from scratch.

Data Overview

To deal with the location entity task the data used was from the largest-scale publicly available Twitter Location Mention Recognition (LMR) dataset, called IDRIS-R that is available both in English and Arabic language. The data is named after Muhammad Al- Idris, who is one of the pioneers and founders of the advanced geography. The "R" refers to the recognition task. IDRISI-R contains 41 disaster events of different types (e.g., floods, earthquakes, fires, etc.) that occurred in a wide geographical area of English- and Arabic-speaking countries across continents. The annotated LMs were also labeled for different coarse- (e.g., country, city) and fine-grained location types (e.g., streets, POIs). Detailed statistics are provided below.

IDRISI-R is released in two ***versions***:

- gold: around 20K and 4.6K human-labeled tweets for English and Arabic, respectively. The annotation was collected using crowd and in-house workers for English and Arabic, respectively.
- silver: around 57K and 1.2M automatically annotated English and Arabic tweets, respectively. The annotation was done using the best performing LMR models.

The tweet datasets naturally support two processing ***setups*** that are:

- time-based: tweets are chronologically ordered.
- random: tweets are shuffled randomly while discarding their timestamps.

We release the datasets in both JSONL and BILOU ***formats***:

- BILOU: NER token-based annotation with 5 classes: Beginning, Inside, Last, Outside, & Unit.
- JSONL: every line corresponds to one tweet with the following properties.

Event	*Date Range*	*# Twt*	*# Twt$_{gold}$*	*# Twt$_{LM0}$*	*# LMs (uniq)*
Ecuador Earthquake	2016/04/17 - 2016/04/18	1,594	1,153	205	1,178 (116)
Canada Wildfires	2016/05/06 - 2016/05/27	2,259	1,300	277	1,333 (165)
Italy Earthquake	2016/08/24 - 2016/08/29	1,240	590	360	284 (66)
Kaikoura Earthquake	2016/09/01 - 2016/11/22	2,217	1,231	375	1,133 (227)
Hurricane Matthew	2016/10/04 - 2016/10/10	1,659	855	62	1,132 (119)
Sri Lanka Floods	2017/05/31 - 2017/07/03	575	457	88	521 (105)
Hurricane Harvey	2017/08/25 - 2017/09/01	9,164	1,299	719	753 (182)
Hurricane Irma	2017/09/06 - 2017/09/17	9,467	1,298	563	951 (354)
Hurricane Maria	2017/09/16 - 2017/10/02	7,328	1,299	357	1,165 (217)
Maryland Floods	2018/05/28 - 2018/06/07	747	422	28	693 (89)
Kerala Floods	2018/08/17 - 2018/08/31	8,056	1,300	331	1,656 (367)
Hurricane Florence	2018/09/11 - 2018/09/18	6,359	1,300	559	1,245 (403)
California Wildfires	2018/11/10 - 2018/12/07	7,444	1,300	455	1,075 (181)
Cyclone Idai	2019/03/15 - 2019/04/16	3,944	1,300	376	1,744 (268)
Midwest. US Floods	2019/03/25 - 2019/04/03	1,930	1,076	92	1,592 (189)
Total	**2016/04/17 - 2019/09/26**	**77,196**	**20,514**	**5,723**	**21,879 (3,830)**

Figure 4. Data table

The training dataset is heavily imbalanced relative to the target feature “hi_flag”. With about 95.6 percent of observations being housing secure and only a meagre 4.4 percent of the observations being housing insecure.

Model Objectives

Precision: This is the number of accurately predicted location words to the total number of predicted location words. It is computed as given in equation.

The range of precision varies between 0 and 1, where 1 is the best and 0 is the worst value.

The performance metrics were selected as below for the model performance.

- Precision: This is the number of accurately predicted location words to the total number of predicted location words. It is computed as given in equation. The range of precision varies between 0 and 1, where 1 is the best and 0 is the worst value.

$$\text{Precision} = \frac{\text{Number of accurately predicted location words}}{\text{Total number of predicted location words}} = \frac{|\, yi \cap y\hat{}i \,|}{|\, y\hat{}i \,|}$$

- Recall: This is the number of accurately predicted location words to the total number of actual location words in the tweet. It is computed as given in equation. The range of recall varies between 0 and 1, where 1 is the best and 0 is the worst value.

$$\text{Recall} = \frac{\text{Number of accurately predicted location words}}{\text{Total number of actual location words}} = \frac{|\, yi \cap y\hat{}i \,|}{|\, y\hat{}i \,|}$$

- F1-score: This is the harmonic mean between Precision and Recall, which gives the balanced evaluation between them. It can be represented by equation. The range of F1-score varies between 0 and 1, where 1 is the best and 0 is the worst value.

$$\text{F1-score} = 2 \text{ X } \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Methodology

Figure 1 gives an overview of the model and data processing pipeline used in our paper. This section first outlines the computational infrastructure used. The data collection and data processing are then described, obtaining a corpus of Wikipedia articles for locations in Great Britain with place names labelled. This dataset was then used to train custom NER models of various architectures, which were evaluated using separate test data against each other and popular prebuilt NER models. We then selected our BERT transformer model to extract all place names from the full corpus of Wikipedia articles, as this model performed well as indicated by its test F1 score, despite its smaller size.

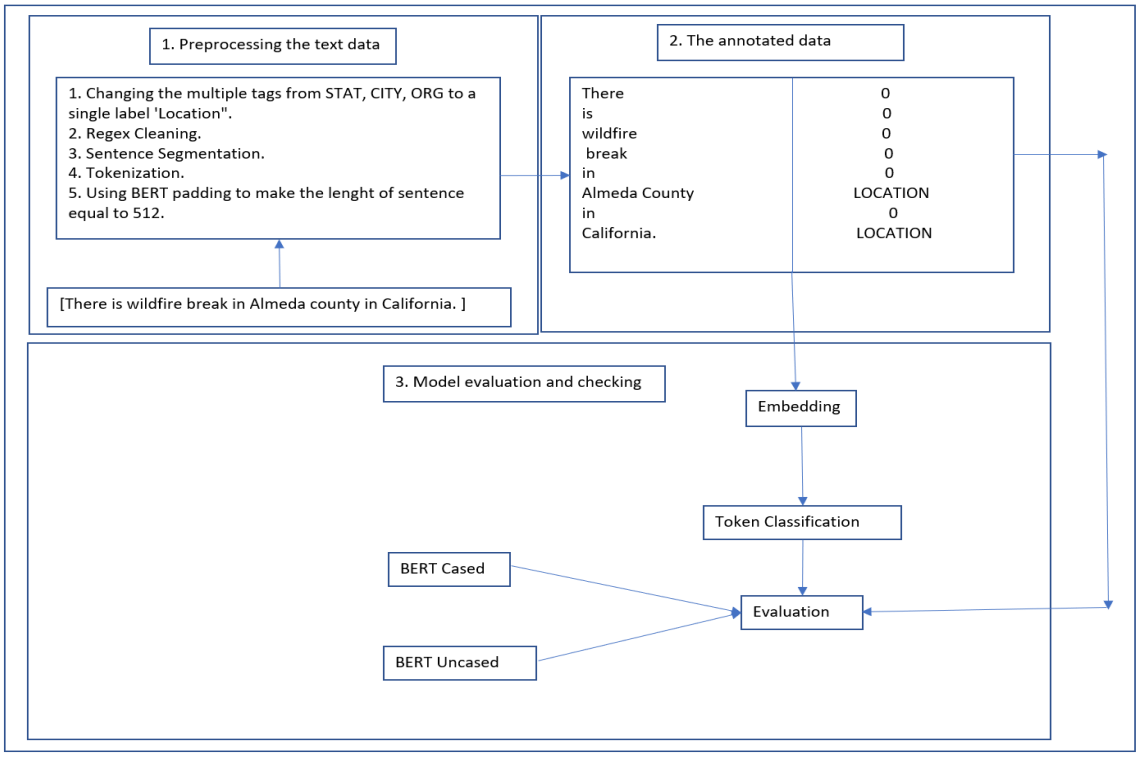

Figure 4. Steps followed in BERT implementation

BERT Model

The BERT model uses a transformer which is an "attention mechanism that learns contextual relations between words (or sub-words) in a text." In its regular form, a transformer has two separate mechanisms. The first is an encoder that can read the text. The second is a decoder that can produce a prediction for the task. The overall goal of a BERT model is to create a language model, therefore only the encoder is necessary. You can read more about the transformer on google.

What's different about the transformer compared to other models is that the transformer encoder will read the entire word sequence at once. This is considered non-directional. This allows the model to receive the word based on the word's surroundings (the words left and right of the target word). The input is put as a sequence of tokens that are within the vectors and then they are processed into the neural network. The vectors are put into the size of H, which each vector corresponds to a input token that has the same index. Directional models only read texts sequentially.

While training language models, the main challenge to keep in mind is defining a prediction goal. So many models will predict the next word within a sequence, which is the directional approach. The problem with this is that is limits context learning.

To overcome this challenge, BERT uses two training strategies:

- Masked LM (MLM)
- Next sentence prediction (NSP)

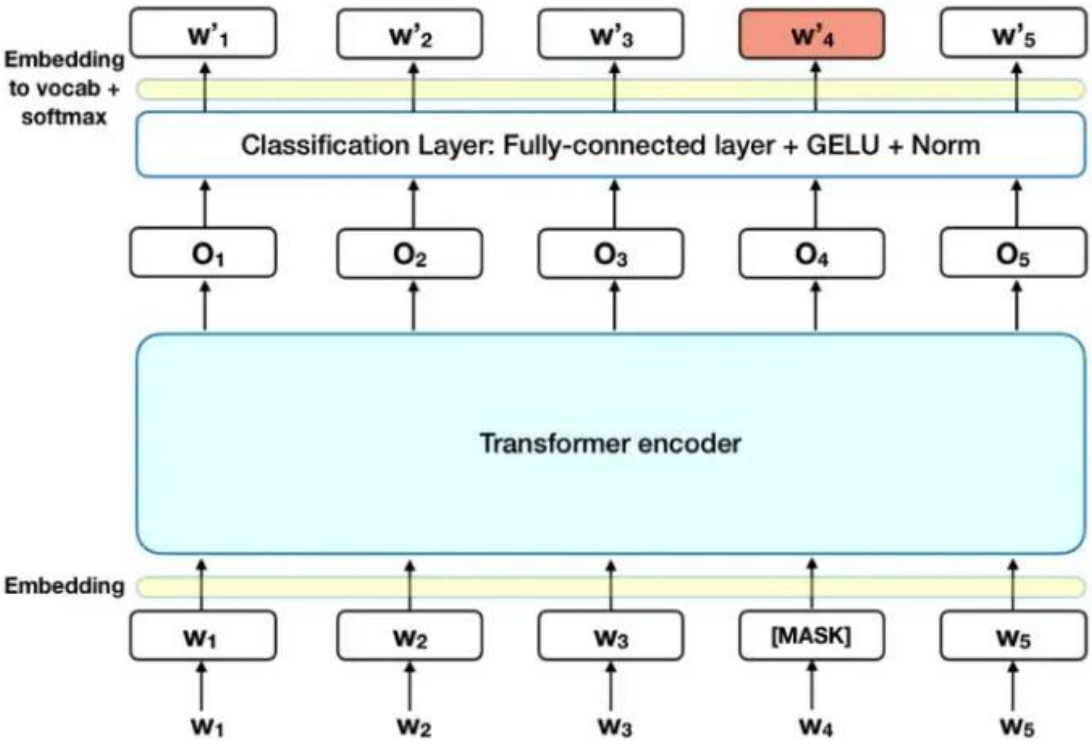

Figure 5. BERT architecture

The BERT loss function considers, just the prediction of masked values and ignores the non-masked words. So, the overall model converges much slower than regular directional models. However, this is offset by its increase in context awareness.
The difference between text classification for BERT and NER problems is the output for the model. For BERT we only use the "embedding output vector output" from the special token.

Data Preprocessing

Data Preprocessing includes two parts: tokenization and adjusting the label to match the tokenization.

Tokenization

Tokenization can be easily implemented with BERT, as we can use BertTokenizerFast class from a pretrained BERT base model with HuggingFace.

We provided several arguments when calling `tokenizer` method from `BertTokenizerFast` class above:

- `padding` : to pad the sequence with a special **[PAD]** token to the maximum length that we specify. The maximum length of a sequence for a BERT model is 512.
- `max_length` : maximum length of a sequence.
- `truncation` : this is a Boolean value. If we set the value to True, then tokens that exceed the maximum length will not be used.
- `return_tensors` : the tensor type that is returned, depending on machine learning frameworks that we use. Since we're using PyTorch, then we use `pt` .

```
print(text_tokenized)

>>> {'input_ids': tensor([[  101,  3460,  2110,   144,  6851,  1197, 11679,  2881,  1162,  1144,
          3347,  1106, 13133,  1137,  1840,  1111,  1346,  3212,   119,   102,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
                                  ..............................
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0]]),
    'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              ..............................
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0]]),
    'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              ..............................
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Figure 6. BERT input (input_ids, token_type_ids, attention_mask)

As we can see, the output that we get from the tokenization process is a dictionary, which contains three variables:

- input_ids: The id representation of the tokens in a sequence. In BERT, the id 101 is reserved for the special **[CLS]** token, the id 102 is reserved for the special **[SEP]** token, and the id 0 is reserved for **[PAD]** token. [15]
- token_type_ids: To identify the sequence in which a token belongs to. Since we only have one sequence per text, then all the values of token_type_idswill be 0. [15]
- attention_mask : To identify whether a token is a real token or padding. The value would be 1 if it's a real token, and 0 if it's a **[PAD]** token. [15]

From the input_ids above, we can decode the ids back into the original sequence with decode method. We got our original sequence back after implementing decode method with the addition of special tokens from BERT such as **[CLS]** token at the beginning of the sequence, **[SEP]** token at the end of the sequence, and a bunch of **[PAD]** tokens to fulfill the required maximum length of 512. [15]

Adjusting Label After Tokenization

This is a very important step that we need to do after the tokenization process. This is because the length of the sequence is no longer matching the length of the original label after the tokenization process. The BERT tokenizer uses the so-called word-piece tokenizer under the hood, which is a sub-word tokenizer. This means that BERT tokenizer will likely to split one word into one or more meaningful sub-words. There are two problems that we need to address after tokenization process:[15]

- The addition of special tokens from BERT such as **[CLS]**, **[SEP]**, and **[PAD]**
- The fact that some tokens are splitted into sub-words.

The consequence of this word piece tokenization and the addition of special tokens from BERT is that the sequence length after tokenization is no longer matching the length of the initial label. [15]

From the example above, now there are in total 512 tokens in the sequence after tokenization, while the length of the label is still the same as before. Also, the first token in a sequence is no longer the word '*Prime*', but the newly added **[CLS]** token, so we need to shift our label as well. To solve this problem, we need to adjust the label such that it has the same length as the sequence after tokenization. [15] These word_ids will be very useful to adjust the length of the label by applying either of these two methods:

1. We only provide a label to the first sub-word of each splitted token. The continuation of the sub-word then will simply have '-100' as a label. All tokens that don't have word_ids will also be labeled with '-100'.

2. We provide the same label among all of the sub-words that belong to the same token. All tokens that don't have word_ids will be labeled with '-100'.

```
>>> [-100, 16, 16, 6, 6, 6, 14, 14, 14, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, -100, -100, -100,
>>> ['[CLS]', 'Prime', 'Minister', 'G', '##ei', '##r', 'Ha', '##ard', '##e', 'has', 'refused', 'to
```

Figure 7. Realigned text after padding as input

Model Building

We have used a pretrained BERT base model from HuggingFace. Since we're going to classify text in the token level, then we need to use BertForTokenClassification class. [15]

BertForTokenClassification class is a model that wraps BERT model and adds linear layers on top of BERT model that will act as token-level classifiers.

Approach 1

In this approach we tried to use the raw data with same "BILOU" tagging for various location mention. We tried using a different pretrained BERT model from Huggingface. We tried using 'bert-base-cased" from "Huggingface" as a new model. The BERT base cased was used by the QCRI as the base model for prior LMR task and had the average F-1 score of 89.44. Therefore, we thought of using the same but with different parameters. The BERT uncased basically differs from regular BERT model in the way it does not lowercase the text before the text is feed into the wordpiece tokenizer. Therefore, we retain the same base BERT architecture.

```
my_config = DistilBertConfig.from_pretrained("distilbert-base-cased", activation="relu", attention_dropout=0.4)
```

The model configuration change code for BERT training file

The F-1 score that we got from the model was about 0.674 and the model had the loss function of cross-entropy loss, as we tried to keep data as multilabel tagged.

Approach 2

In this approach we tried to run the model with the same "BILOU" tagging with multi location tagging. The model that we chose for this approach was 'bert-base-uncased'. This model differs from the regular bert model in the sense it tries to lowercase the text before being put into the word piece tokenizer. Also the bert base uncased trips out any accent markers in text giving the model more uniform homogenous corpus to train on. Though generally for NER task BERT cased is a preferred model we got some different result using the bert base uncased on this disaster dataset.

```
my_config = DistilBertConfig.from_pretrained("distilbert-base-uncased", activation="relu", attention_dropout=0.4)
```

The model configuration change code for BERT training file

The F-1 score of 0.658 and the loss function of the model was Cross Entropy loss as we had multilabel data.

FLAIR Model

- **A powerful NLP library.** Flair allows you to apply state-of-the-art natural language processing (NLP) models to text, such as named entity recognition (NER), part-of-speech tagging (PoS), classification, with support for a rapidly growing number of languages.
- **A text embedding library.** Flair has simple interfaces that allow to use and combine different word and document embeddings, including proposed Flair Embedding BERT embeddings and ELMo embeddings.
- **A PyTorch NLP framework.** The framework builds directly of Pytorch, making it easy to train our own models and experiment with new approaches using Flair embeddings and classes.

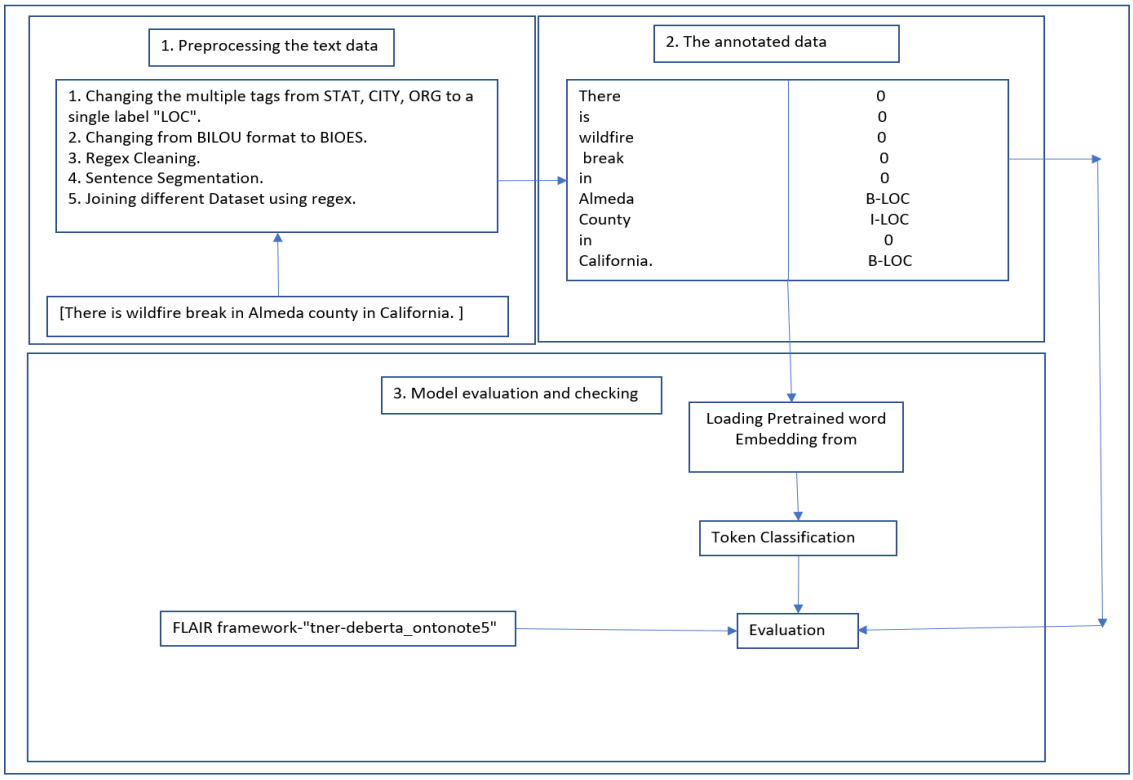

Figure 8. Steps implementation for Flair model

Data Preprocessing

The Data used from IDRIS-R is labelled in accordance with the BILOU notation. Therefore, each label corresponds to a special character level representation. The labels have predefined word position such as Before, Intermediate, Other, unit and last for the location words. Due to the predefined word notation, we can get the contextual information from the given text as the transformer model can better understand the word meaning and then assign the closest vector embedding for the given word. The Flair model also tries to use different embedding type using the pretrained embedding that leads to more deeper understanding of the word in each sentence. The accuracy of the model can be improved by using the vector embedding that represent the closest representation of the given word.

We uploaded data into two files the training data and the validation data. We combined the training data using a python script and make sure that there is no concurrent whitespace in between the sentences as the model understands that as the end of the data. The total number of training sentences is 14392 and while test/validation data is 2056.

Approach 1

From the first run, we confirmed that flair uses BIOES tag formatting for the "NER" task, therefore, we tried changing it to the format of BILOU, and needs to be formatted in the form of BIOES. Using a regex script, we tried to replace the "L" to "E" and "U" to "S". The changing of the tagged labels was also followed by some more hyperparameters optimization such as changing the learning rate, changing the pre-trained embedding and the max epochs. Our deduction from approach 1 was that we didn't have the required computing resources for epochs exceeding 5 and learning rate exceeding 1.0 e-5. As the GPU would run out of memory on Google Colab platform. We also tried to change the pretrained document embedding from "tner/roberta-v3-large-ontonotes5" to "tner/deberta-v3-large-ontonotes5" as the deberta model had some advantages, the first is the disentangled attention mechanism where each word is represented using two vectors that encode its content using the entangled matrices on their contents and relative positions. Second, an enhanced mask decoder is used to replace the output softmax layer to predict the masked tokens for model pretraining.

The model performed relatively well with a F-1 score of 86.84 and the following hyperparameters were used for the model training.

- Pretrained embedding: "tner/deberta-v3-large-ontonotes5"
- Max epochs: 3
- Learning rate: 5.0e-6
- Optimizer: AdamW

The lowering of the number of epochs as well as the learning rate helped the model to converge with optimized speed to better fit the loss function. The training time was reduced from 12 hours to 6 hours. But to further improve the model we needed a more robust technique. We further explored if combining another annotated dataset would help to increase the F-1 score. That was the third approach that we took.

Approach 2

In this approach we tried to combine the dataset from different NER dataset. The dataset that we referred were such as WNUT17 and CoNLL 2003. This are the latest updated version of the famous dataset. The dataset is formatted using a universal NER type. Each word has been put on a separate line and there is an empty line after each sentence. The first item on each line is a word, the second a part-of-speech (POS) tag, the third a syntactic chunk tag and the fourth the named entity tag. The chunk tags and the named entity tags have the format I-TYPE which means that the word is inside a phrase of type TYPE.

```
U.N.          NNP  I-NP  I-ORG
official      NN   I-NP  O
Ekeus         NNP  I-NP  I-PER
heads         VBZ  I-VP  O
for           IN   I-PP  O
Baghdad       NNP  I-NP  I-LOC
.             .    O     O
```

Figure 9. The training data format for flair model

The further regex script was used to change the tagging from BILOU to BIOES. The second and third columns were also dropped from the required data. This prepared dataset was then joined with the other IDRIS-R dataset. The combined dataset was then used to train model keeping the same parameters as above.

The model performed relatively well with a F-1 score of 87.84 and the following hyperparameters were used for the model training.

- Pretrained embedding: "tner/deberta-v3-large-ontonotes5"
- Max epochs: 3
- Learning rate: 5.0e-6
- Optimizer: AdamW

The further development was also needed to further improve the score from 90.38 to above the threshold limit of 89.4 on the IDRIS-R dataset. We tried to improve the model performance by further understanding the mechanism of the loss function used. We tried to change to the loss function form "Crossentropy" to "Binarylosslogit" but there was no significant improvement in the F-1 Score. What we tried to further improve was to change the classification type from the multi label classification to the single class classification. That's where we tried the approach 4.

Approach 3

The further analysis of the loss function helped us to devise that the model can be further improved by using a single label token for all location irrespective of having special labels such as CITY, STAT, COUNTRY etc. The BIOES format of the labels was retained but the labelling of special ids was changed to "LOC". The

flair model then identified the labelled data as the single class classification problem rather than multi class classification problem. The hyperparameter were kept as before but the model performance increased using this newly annotated data.

- Pretrained embedding: "tner/deberta-v3-large-ontonotes5"
- Max epochs: 3
- Learning rate: 5.0e-6
- Optimizer: AdamW

The F-1 score of 90.56 was achieved using this special labelled data. This model was then saved as a model.pt file that would be submitted in the docker submission. The model has the precision of 0.8920 recall 0.9184 The model has outperformed the threshold F-1 score on the IDRIS-R dataset of 89.44 and was selected for final submission. The next task that we did was to create a docker submission report for the trained model with all of the environment dependency.

Performance and Evaluation

The model performance was judged using the performance metrics of F-1 score, precision and recall. The models performed relatively well on the chosen dataset and the model with highest F-1 score was finalized. The model performance for the model with different changes can be summarized as below.

The performance metrics used are:

1. Precision
2. Recall
3. F-1 Score

Sr.No	Model	Type	F-1 Score	Precision	Recall	Changes
1	BERT	bert-base-cased	0.674	0.774	0.594	Raw data used with BILOU format. Multi label classification.
2	BERT	bert-base-uncased	0.658	0.781	0.566	Data used was raw along with BILOU format.
3	Flair	tner/deberta-v3-large-ontonotes5	0.8664	0.8770	0.8560	Raw data changed from BILOU format to BIOES.
4	Flair	tner/deberta-v3-large-ontonotes5	0.9038	0.8975	0.9102	Used more data from WNUT-17 & CoNLL 2003. The data was merged with the IDRIS data to create a bigger corpus.
5	Flair	tner/deberta-v3-large-ontonotes5	0.9050	0.8920	0.9184	Data has special token called "LOC" created in BIOES tagging.

A comprehensive study of different NLP model for NER task.

The following are some of individual examples of the model identifying the locations correctly:

1.
```
Sentence: "George Washington went to Washington ." → ["Washington"/LOC]
The following NER tags are found:
Span[4:5]: "Washington" → LOC (0.9703)
```

2.
```
Sentence: "Rajesh went to buy ice cream from hotel California ." → ["California"/LOC]
The following NER tags are found:
Span[8:9]: "California" → LOC (0.9968)
```

3.
```
Sentence: "There has been a large fire break in Santa Fe near China Wok bar ." → ["Santa Fe"/LOC]
The following NER tags are found:
Span[8:10]: "Santa Fe" → LOC (0.9602)
```

Conclusion

From the modelling, we found out that in the specific NER task Flair generally outperforms the other BERT based models. Also, a lot of improvement in the model was attributed to the preprocessing of the data in terms of tagging. A more uniform tagging produced better result. Also, the belief the more the data the better the result was not really found true in our research. The addition of more data led to in turn decrease the model performance as false positive were identified in the model due to disassociated tags with some of the terms.

Future Scope

The Flair based NER model we have developed can be said state-of-the-art for the "location mention recognition" (LMR) task. Further improvement is also possible by using an inbuilt Flair function called "StackedEmbedding" that help to combine various embedding such as word embedding "Word2Vec", "FastText", document embedding "dslim/bert-base-uncased", "tner/deberta-v3-large-ontonotes5". Combining different embedding is generally preferred as to use both traditional embeddings together with contextual string embeddings. The StackedEmbeddings class is instantiated by passing a list of embeddings that we wish to combine. For instance, combining classic GloVe embeddings with forward and backward Flair embeddings. A combination of embeddings often gives best results.

Deployment

Code was expected to be submitted through the docker image. The container should be able to run on a Linux or Mac based docker-engine.

Using containerization technology, Docker enables us to execute a Docker image containing an application on any computer. A Docker container is a compilation of a Base OS (such as Alpine, Ubuntu, etc.) and other required software dependencies that may be specified inside a Docker image. Now, without having to worry about environment management, you can use that image to build a container and run your application on many computers or even on cloud platforms (AWS, GCP, Azure, etc). (Development, testing, and production). The portability, security, and performance of Docker make it one of the most widely used DevOps solutions. The steps are as follows:

Step 1: Create a Dockerfile

A Dockerfile must be made in order to get your code into a container since it instructs Docker what your application needs. A Dockerfile is a text file that includes all the command-line options a user may use to put together an image. A read-only base image is the first layer of a Docker image, which is then built up with additional dependencies by subsequent read-only layers. The container receives instructions on how to activate your model in the end.

We ensured not to transfer the source code into Dockerfile without first adding requirements.txt file. This prevents Docker from running the pip install command repeatedly even if the library dependencies haven't changed when the code is modified, and container is rebuilt. Instead, it will reuse the cached layer up to the installed packages.

Step 2 : Build an Image with your Dockerfile

In the next step we created a container image after Dockerfile was completed. According to the directions in the Dockerfile, an image is created by docker build. We instructed the Docker daemon to retrieve the Dockerfile located in the current directory (the. at the end does this).

Using the command line option docker build -t USERNAME/IMAGE NAME:TAG, we taged the image as it builds. Alternatively, after it was constructed, it can be explicitly tagged using “docker tag SOURCE IMAGE:TAG TARGET IMAGE:TAG”.

Now that the model has been integrated into a container, it may be run everywhere Docker is present. We ensured that, using the CMD command from the docker file, the test script (main.py) executes automatically when the container starts. We mounted the folder geoai containing the input.jsonl. The test script reads the test data from the file /geoai/input.jsonl and output results to the file /geoai/output.jsonl.

GitHub: Data and Code

The data and codes that support the findings of this study are available on the public GitHub (https://github.com/osu-msba/ban5753-fall2022-team-patterns). Instructions for using the data and code are provided as a README within the GitHub repository.

References

1. Antoniou, V., Morley, J., and Haklay, M., 2010. Web 2.0 geotagged photos: assessing the spatial dimension of the phenomenon. Geomatica, 64, 99–110.

2. Auer, S., et al., 2007. DBpedia: a nucleus for a web of open data. In: D. Hutchison, et al., eds. The semantic web. Vol. 4825. Berlin, Heidelberg: Springer Berlin Heidelberg, 722–735.

3. Barbieri, F., et al., 2020. TweetEval: unified benchmark and comparative evaluation for Tweet classification. arXiv:2010.12421 [cs]. Buscaldi, D., 2011. Approaches to disambiguating toponyms. SIGSPATIAL Special, 3 (2), 16–19. Couclelis, H., 2010. Ontologies of geographic information. International Journal of Geographical Information Science, 24 (12), 1785–1809.

4. DeLozier, G., Baldridge, J., and London, L., 2015. Gazetteer-independent toponym resolution using geographic word profiles. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence (AAAI'15). New York: AAAI Press, 2382–2388.

5. Devlin, J., et al., 2019. BERT: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805 [cs].

6. Dror, R., and Reichart, R., 2018. Appendix – recommended statistical significance tests for NLP tasks. arXiv:1809.01448 [cs].

7. Gao, S., et al., 2017. A data-synthesis-driven method for detecting and extracting vague cognitive regions. International Journal of Geographical Information Science, 31 (6), 1–27.

8. Gao, S., et al., 2013. Towards platial joins and buffers in place-based GIS. In: Proceedings of the first ACM SIGSPATIAL international workshop on computational models of place (COMP '13). New York: Association for Computing Machinery, 42–49.

9. Gardner, M., et al., 2018. AllenNLP: a deep semantic natural language processing platform. In: Proceedings of workshop for NLP open source software (NLP-OSS), Melbourne, Australia. Association for Computational Linguistics, 1–6.

10. Goodchild, M.F., 2007. Citizens as sensors: the world of volunteered geography. GeoJournal, 69 (4), 211–221.

11. Chiu, J. P., & Nichols, E. (2015). Named entity recognition with bidirectional lstm-cnns. arXiv preprint arXiv:1511.08308, .

12. Chowdhury, S. R., Imran, M., Asghar, M. R., Amer-Yahia, S., & Castillo, C. (2013). Tweet4act: Using incident-specific profiles for classifying crisis-related messages. In ISCRAM. Citeseer.

13. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. Journal of Machine Learning Research, 12, 2493–2537.

14. Daniken, P., & Cieliebak, M. (2017). Transfer learning and sentence level features for named entity recognition on tweets. In ¨ Proceedings of the 3rd Workshop on Noisy User-generated Text (pp. 166–171).

15. Winastwan, Ruben. “Named Entity Recognition with BERT in PyTorch - towards Data Science.” Medium, Towards Data Science, 3 May 2022, towardsdatascience.com/named-entity-recognition-with-bert-in-pytorch-a454405e0b6a.