|  |  |
| --- | --- |
| A logo on a black background  Description automatically generated | |
| **Faculty of Science and Technology**  **CISC3025 – Natural Language Processing** | |
| **Project 3: Implementation of a Maximum Entropy Model for Named Entity Recognition** | |
| **Group Members:** | |
| Huang Yanzhen | DC126732 |
| Chen Zirui | DC127901 |

Table of Contents

[1 Introduction 3](#_Toc164120015)

[2 Methods and Implementation 3](#_Toc164120016)

[2.1 Features Selection 3](#_Toc164120017)

[2.1.1 Internal Pattern Features 3](#_Toc164120018)

[2.1.2 Library Features 4](#_Toc164120019)

[2.1.3 Contextual Features 4](#_Toc164120020)

[2.2 Server Implementation 5](#_Toc164120021)

[3 Evaluations and Discoveries 5](#_Toc164120022)

[3.1 Trial and Errors 5](#_Toc164120023)

[3.2 Advantages and Drawbacks 5](#_Toc164120024)

[3.3 Interesting Phenomenon 5](#_Toc164120025)

[4 Conclusion 5](#_Toc164120026)

# 1 Introduction

Being a part of the syntax level processing techniques in NLP, Named-Entity Recognition is a mechanism for a subtask of information retrieval that’s been applied in various fields, such as search engines.

To be more specific, Named Entity Recognition involves assigning a class label for entities within a piece of plain text. Those classes could include person names, firm or organization names, locations, dates, and more customed tags as you want. This project made use of the Maximum Entropy Classifier from *scikit-learn* to implement a simple NER task, which is to recognize person names.

The initial task for building a MaxEnt model is to select proper features. Once the features are selected, the training set, which are the word tokens pre-assigned a label, are passed along with the features to train the model. The model will assign an importance weight to each feature for each class. Lastly, to test the model, for each entity, the model calculates the vote of each class using the features and the weights, finally deciding the class label.

In this project, the training part is done by the encapsulated model in *scikit-learn.* The main workload of this process is to process training data to select features, and to build a website that demonstrates the performance of the model, which runs at the backend in the local server.

# 2 Methods and Implementation

## 2.1 Features Selection

Feature selection aims to find representative features of a word that could produce the maximum entropy of the classification, that is, to best distinguish the classes. There are three types of features, which are Internal Pattern Features, Library Features, and Contextual Features. For the binary case, there are positive and negative features. Positive features indicate that a word is more likely to be a human name. Negative features, on the contrary, unrecommends to assign the label of human name to the word.

### 2.1.1 Internal Pattern Features

Internal Pattern Features put their scopes in the word itself. It analyses the patterns of a word and assumes that some specific pattern can distinguish a person’s name from others.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table.** Internal Pattern Features | | | | |
|  | **Feature Name** | **Feature Description** | **Reason for selection** | **Match Examples** |
| + | p\_cap\_low | Start with Capital letter, and the rest letters are lowercase. There may be a prime after the first capital letter. There may be a second capital letter in the third letter’s space. | In English, names always start with a capital letter. There are also some special name styles, like MacArthur, and O’Brien. | Jenny, MacArthur, O’Brien |
| + | p\_cap\_period | A single capital letter followed by a period. | In English, human name initials | D., J., M., … |
| - | p\_noun\_like | Have some noun-like ending patterns, like *-tion, -sion, -ance,* etc. | Human names doesn’t always ends with the same pattern as some special nouns that are derived from verbs or adjectives. | mention, motion, addiction, expansion, allowance, … |

The corresponding regular expressions of these features are as follows.

|  |  |  |
| --- | --- | --- |
| **Table.** Regular expressions for internal patterns | | |
|  | **Feature Name** | **Regular Expression** |
| + | p\_cap\_low | ^[A-Z](\'[A-Z]?|[a-z][A-Z])?[a-z]+ |
| + | p\_cap\_period | ^[A-Z]\.$ |
| - | p\_noun\_like | (([aio]?tion|ment|ness|ship|hood|\w+age|[ae]nce)[sd]?$)/i |

### 2.1.2 Library Features

There are a pack of words that is likely not to be classified into human names. For instance, week names like *Monday* and *Tuesday* are not likely to be human names. Country names, like *Korea, China, Japan* or *Britain* are also not likely to be classified to human names. Therefore, we added some libraries that contains these kinds of words and combined them into one evaluation criteria. That is to say, as long as a word belongs to one of the libraries in this set, it will be assigned a negative feature. The negative library set includes the following.

|  |  |
| --- | --- |
| **Table.** Negative feature libraries | |
| **Libraries** | **Python Packages** |
| Week Names | *self-defined* |
| Month Names | *self-defined* |
| Country Names | *geonamescache* |
| City Names | *geonamescache* |
| Stop Words | *nltk.stopwords* |

On the opposite side, there are pack of words that are likely be classified into names. One of the most important packs are, exactly, names. Therefore, an *nltk* corpora is used to identify whether a token is always presented as names. Likewise, if a token belongs to this library, it will be assigned a positive feature.

### 2.1.3 Contextual Features

Contextual Features looks beyond the word itself, and seeks evidence in the contextual environment where the word is in. For instance, a name tends to be in the first and second place of a sentence. A name also tends to be in the main reference of an attributive clause. Unlike Library Features, Contextual Features are discrete, that is each pattern matches only one feature, unlike in Library Features, one feature is matched to numerous patterns.

One of the most useful features is that, the target of a restricted attribute clause is always a human name. For instance, “Joe, who was the president…” is an attributive clause and Joe is a human name.

|  |  |  |
| --- | --- | --- |
| **Table.** Contextual Features | | |
|  | **Feature Name** | **Regular Expression** |
| + | is\_start\_of\_sentence |  |
| + | is\_target\_of\_clause |  |
| - | is\_after\_status |  |

## 2.2 Server Implementation

We selected *Django* as our backend framework, since it is written in python and thus have better compatibility to our model. The front-end is written in plain HTML and styled using plain CSS, for simplicity. Additional JavaScript is written to handle the XML HTTP request, aiming to handle the JSON response from the backend and display the results in the browser without leaving the current page.

*Django* is a web app framework using Model-Template-View software design pattern. Our NER model, in this case, makes use of the *View* component. Given a URL, it accesses the backend resources in the corresponding view.

Below shows the abstract directory hierarchy of this project for better reference.

|  |
| --- |
| **Code.** Directory hierarchy of this project |
| CISC3025-Name-Entity/  |- name\_entity\_server/ --> Server directory  |- name\_entity\_server/ --> Default app of this server  |- NER\_app/ --> Custom app of server, in this case, the NER app  |- migrations/  |- NER/ --> Project Main Directory, i.e. from Moodle  |- data  |- \_\_init\_\_.py --> For pre-downloading nltk corpora  |- MEM.py --> MaxEnt Model  |- playground.py --> Prediction function, connect model with backend.  |- run.py  |- templates/  |- nerUI.html --> Webpage UI  |- \_\_init\_\_.py  |- views.py --> Defines the JSON response of backend.  |- \_\_init\_\_.py  |- manage.py --> To start server |

To run the server, it is required to change directory into the first name\_entity\_server directory. After that, run command python -m manage runserver to invoke the manage.py file, running the server. More instructions and exception handling is described in the Readme.md document in the repository.

# 3 Evaluations and Discoveries

## 3.1 Trial and Errors

### 3.1.1 Over-Fitting Problem

First of implement, a number of features were applied:

***Negative features****:* All capital letters, Numeric expressions, Location name abbreviation, etc.

Too many features which make it comes to a over-fitting problem, we can’t recognize any name but Johnson. It is predicted that this model was over-fitted.

|  |
| --- |
| **Code. Negative features applied at first** |
| - Person status prefix. e.g. Mr., Ms., Mrs., ...  'p\_name\_prefix': re.compile(r'M[a-z]{1,3}\.'),  - All capital letters.  'p\_all\_cap': re.compile(r'^[A-Z]+$'),  # - Possessive case. e.g. 's, ....  'p\_possessive\_like': re.compile(r'\'s$'),  # - Location name abbreviation. U.S., U.K., D.C., ...  'p\_country\_abbrev\_like': re.compile(r'([A-Z]\.){2,3}'),  # - Numeric expressions.  'p\_num\_slash': re.compile(r'(\d+-)+\d+|\+\d+|\d+|\d+\.\d+')  """  if previous\_label == 'PERSON':  features['is\_previous\_person'] = 1    if previous\_label == 'O':  features['is\_previous\_other'] = 1  # ------------- Position Related -------------  Is around the first place in a sentence.  if (position > 0 and words[position-1] == ".") or (position > 1 and words[posit  features['is\_around\_first'] = 1    Is the last word  if position == len(words) - 1:  features['is\_last\_word'] = 1  # Is after name prefix  if position < len(words) - 1 and re.match(r'M[a-z]{1,3}\.', words[position-1]):  features['is\_after\_name\_prefix'] = 1  # + Is in possessive case  if position < len(words) -1 and words[position+1] == "'s":  features['is\_possessive'] = 1    if words[position+1] == "verb":  features['is\_after\_verb'] = 1    if words[position - 1] in stop\_words:  features['is\_after\_stop\_word'] = 1    if not position >= len(words) - 1 and words[position + 1] in stop\_words:  features['is\_before\_stop\_word'] = 1  """ |

So we brought up some improvements about this model, we cut useless features (Features are useless if the evaluations of this model doesn’t work better after we comment the feature). Besides, some of those features are also merged.

### 3.1.2 Over-Reliance on Name Corpus

To overcome our over-fitted problem, we import a name library that can optimize our model effect. It occurs to an improved name entity display. But it also cause over-reliance on name corpus problem, if we comment our name library, model effect will have a great discount.

## 3.2 Advantages and Drawbacks

*Perform bad for non-english names, good if it is after non-english names.*

*Good for attributive clause. Good for multiple names together.*

**3.2.1 Advantages**

Our model works very well in an attributive clause because of our train data is decomposed from a text and contextual features such as *is\_start\_of\_sentence, is\_target\_of\_clause, is\_after\_status* also play an important role. As figures showed below, there is an evident model effect improvement. More names were recognized was labelled as name after using attributives clause.

|  |
| --- |
| **Works well in an attributive clause** |
|  |
|  |
|  |

**3.2.2 Drawbacks**

**a. Unseen names can only be recognized if they are after the names that’s seen in the corpus.**

|  |
| --- |
| **Unseen name is after a name seen *Figure01*** |
|  |
| **Unseen name is before a name seen *Figure02*** |
|  |

As test figure01 and figure02 showing, “George Washington” is marked as name if it after a seen name such as “Alex”. Otherwise, “George Washington” won’t be marked as a name.

## 3.3 Interesting Phenomenon

|  |
| --- |
| **Edison were in our name library, but cannot be recognized as name.** |
|  |

# 4 Conclusion