CISC3025

Project Report

MAI JIAJUN, D-C1-27853
GUO PENGZE, D-C1-2746

25-APR-2025

Content

Named Entity Recognition

-01-

BRIEF OF PROJECT

1.0 Brief of projecf

This project identifies the names of people in the text through the Maximum Entropy Markov Model (MEMM), relying on feature engineering and the maximum entropy classifier for training and prediction. It is capable of making word-by-word predictions for the input sentences and can save and load the best model for subsequent applications

Frontend

Gradio interface

Gradio is used to quickly build Web interfaces, allowing users to interact with models through sliders, text boxes and buttons

This interface is divided into two tabs. One is for training and visualization, and the other is for sentence prediction.

Backend

This code implements a named entity recognition (NER) system based on the Maximum Entropy Markov Model (MEMM). It can train a model to identify named entities in the text by extracting text features and return the label of each word when making predictions. During the training process, the model will be evaluated based on indicators such as accuracy and recall rate, and ultimately the best model will be saved for subsequent use

-02-

METHODOLOGY

ABOUT METHODOLOGY

early stop

The early stop mechanism is a regularization approach that monitors a certain validation set metric (such as loss or accuracy) during model training. When the performance on the validation set does not improve after several consecutive rounds, the training can be prematurely terminated.

probelm

train method does not support incremental learning
cause it is too old!

how about save and load? ?

No way either.
The training cannot read the previous model parameters

GRADIO UI

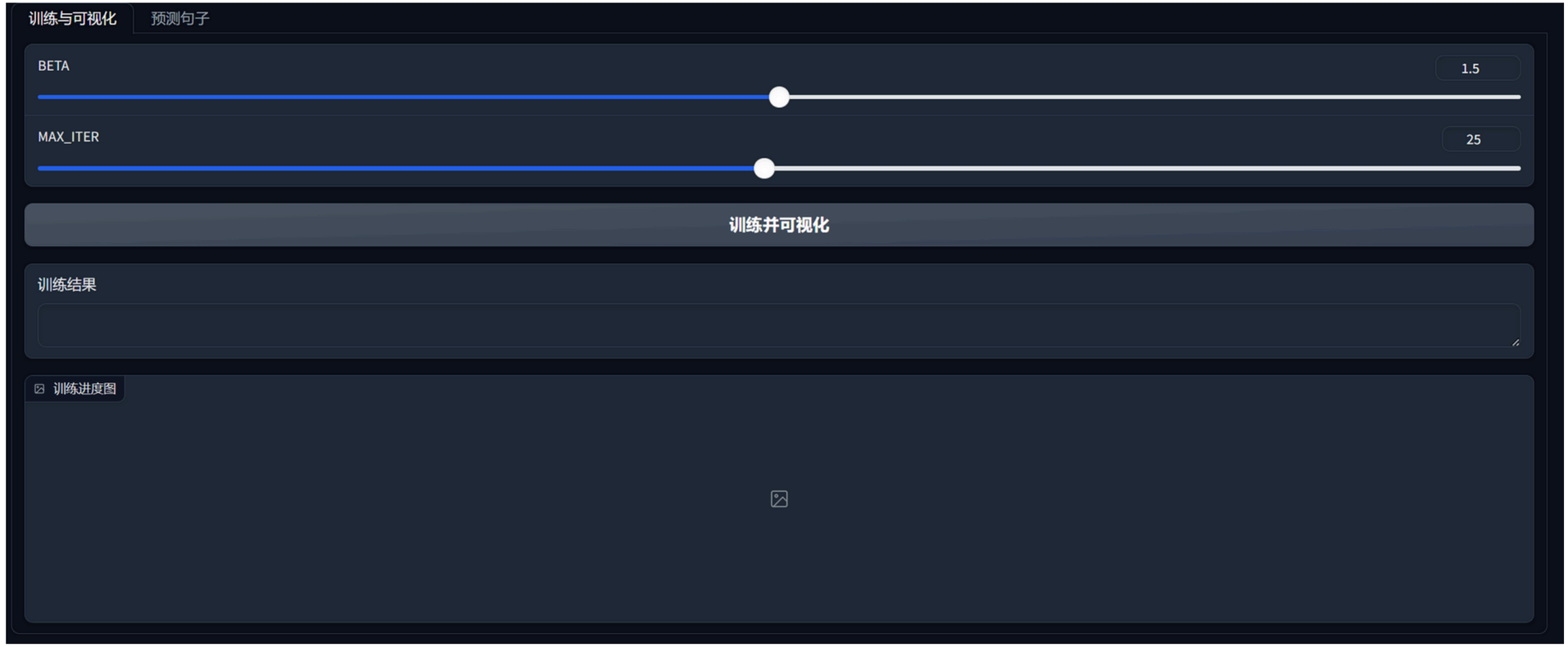
Build & Share D
Machine Learni

Gradio is the

About Gradio

Gradio is an open-source Python library that allows developers to quickly build and share customizable web interfaces for machine learning models and data processing functions. It enables users to interact with models in real-time through a simple GUI, making it ideal for demos, testing, and sharing models without needing complex front-end development.

Train ui



1. slide
2. view of training process

using ui


```
def predict_sentence(self, sentence):
    """
    使用 MEMM 模型进行预测。
    :param sentence: 输入的句子
    """
    words = word_tokenize(sentence)
    predictions = []
    previous_label = "O"

    for i in range(len(words)):
        single_bunch_features = self.features(wo

        current_label = self.classifier.classify
        predictions.append(current_label)

        previous_label = current_label

    return list(zip(words, predictions))
```

gradio train and use the best model
using nltk to split the sentence

-03-

CHALLENGES

3.0 Quick glance on features design

1. Rough features of current word:

```
features = {}
""" Baseline Features """
current_word = words[position]
current_word_lower = current_word.lower()

# Rough features of current word
features['has_(%s)' % current_word] = 1
features['cur_word_len'] = len(current_word) // 2
if current_word_lower in self.nltk_stopwords: features['cur_word_is_stopword'] = 1
```

1. **has** the word?
2. **half of the length** of the current word. (deal with wrong/different spelling)
3. current word is **stop word**?

3.0 Quick glance on features design

2. Rough features of previous word:

```
# Rough features of previous word
prev_word = words[position - 1] if position > 0 else "."

features['prev_word_label'] = previous_label
features['prev_word_is_capitalized'] = prev_word[0].isupper()
features['prev_word_ends_with_punctuation'] = prev_word[-1] in string.punctuation
features['prev_word_is_title'] = any((prev_word.lower().startswith(t) and
                                      len(prev_word.lower()) <= len(t) + 1)
                                     for t in self.titles) # Mr, Mr., Miss., ...
features['prev_word_len'] = len(prev_word) // 2
features['prev_word_is_digit'] = prev_word.isdigit()
```

1. label of previous word
2. previous word first letter Cap? (e.g. The, Why...)
3. End with punctuation? (e.g. Mr., !, 20%...)
4. Is title? (e.g. Mr, Mrs., Miss)
5. half of the length
6. Is digit?

3.0 Quick glance on features design

2. Rough features of previous word:

```
# Rough features of previous word
prev_word = words[position - 1] if position > 0 else "."

features['prev_word_label'] = previous_label
features['prev_word_is_capitalized'] = prev_word[0].isupper()
features['prev_word_ends_with_punctuation'] = prev_word[-1] in string.punctuation
features['prev_word_is_title'] = any((prev_word.lower().startswith(t) and
                                      len(prev_word.lower()) <= len(t) + 1)
                                     for t in self.titles) # Mr, Mr., Miss., ...
features['prev_word_len'] = len(prev_word) // 2
features['prev_word_is_digit'] = prev_word.isdigit()
```

`feature['has_prev_(%s)' % prev_word] = prev_word` **is not good!**
Because this feature can severely reduces the generalisation ability of the model!

3.0 Quick glance on features design

3. Letter cases & has punctuations in current word?

```
# Letter cases
if current_word[0].isupper(): features['Titlecase'] = 1
if current_word.isupper(): features["Allcapital"] = 1
if self.camel_regex.fullmatch(current_word): features["Camelcase"] = 1


# Punctuations
if "'" in current_word: features["Apostrophe"] = 1
if "-" in current_word: features["Hyphen"] = 1
```

1. **Titlecase?** **(Hello, This)**
2. **Allcapital?** **(DC, TOM)**
3. **Camelcase?** **(iPhone, McDonald)**
4. **has - or ’ ?** **(Hi-Rush, O’Steven)**

3.0 Quick glance on features design

4. Record prefix and suffix by Hashing algorithm

```
# Prefix & Suffix hashing
max_record_len = 4

_prefix = current_word[:max_record_len]
_suffix = current_word[-max_record_len:]
features["prefix_hash"] = int(hashlib.md5(_prefix.encode()).hexdigest(), 16) % 100000
features["suffix_hash"] = int(hashlib.md5(_suffix.encode()).hexdigest(), 16) % 100000
```

As directly record the prefix or suffix can cause infinity bucket problem,

We use hashing to **map various prefix/suffix to integer** range from 0 to 9999.

3.1 Non-English Names

French

François-Henri Pinault,
Élodie Fontaine (é, è, û)

German

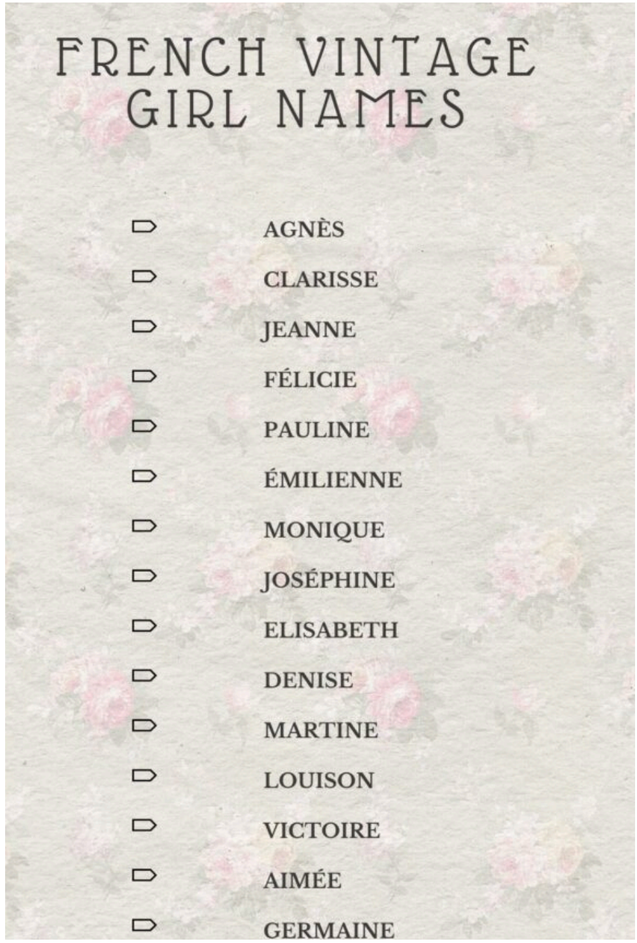
FJürgen Müller, Gisela Schön
(ä, ö, ü)

Spanish

María José García, Sofía Íñiguez
(á, í, ñ)

Chinese

Shen Guofang
Lyu Zhihe, HUO YINGDONG



3.1 Non-English Names

- For French, German and Spanish, etc.
  We add up one special feature to them:
- If any character is non-ASCII Latin letter, mark its feature.

```
@staticmethod
def is_latin_char(check_char):
    try:
        return unicodedata.name(check_char).startswith('LATIN')
    except ValueError:
        return False
...
...
if any(ord(c) > 127 and self.is_latin_char(c) for c in current_word):
    features["Contain_non_ascii_latin"] = 1
...
...
```

3.1 Non-English Names

- For Chinese Pinyin(拼音) names, We are concerned about the correctness of the spelling of the name, so we apply a regular method to filter Chinese Pinyin names.

```
self.pinyin_regex = re.compile(
    r"^("
    r"(a[io]?|ou?|e[inr]?|ang?|ng|[bmp](a[io]?|[aei]ng?|ei|ie?|ia[no]|o|u)|"
    r"pou|me|m[io]u|[fw](a|[ae]ng?|ei|o|u)|fou|wai|[dt](a[io]?|an|e|[aeio]ng|"
    r"ie?|ia[no]|ou|u[ino]?|uan)|dei|diu|[nl](a[io]?|ei?|[eio]ng|i[eu]?|i?ang?|"
    r"iao|in|ou|u[eo]?|ve?|uan)|nen|lia|lun|[ghk](a[io]?|[ae]ng?|e|ong|ou|u[aino]?|"
    r"uai|uang?)|[gh]ei|[jqx](i(ao?|ang?|e|ng?|ong|u)?|u[en]?|uan)|([csz]h?|"
    r"r)([ae]ng?|ao|e|i|ou|u[ino]?|uan)|[csz](ai?|ong)|[csz]h(ai?|uai|"
    r"uang)|zei|[sz]hua|([cz]h|r)ong|y(ao?|[ai]ng?|e|i|ong|ou|u[en]?|uan))"
    r"){1,4}$"
)
self.pinyin_confusion = {"me", "ma", "bin", "fan", "long", "sun", "panda", "china"}
```

- The confusion list is to mark the correct Pinyin that also has English meaning.

3.1 Non-English Names

- But the Chinese letter “吕”, “绿”, or “旅”, they have 3 different spelling in various situation.
- Lu, Lv, and Lyu. So, we add up one feature to record them.

```
if self.pinyin_regex.fullmatch(current_word.lower()):
    features["Pinyin"] = 1
if current_word.lower().endswith("lyu") or current_word.lower().startswith("lyu"):
    features["Pinyin_lyu"] = 1                    # specialize for 吕 written in lyu
if current_word.lower() in self.pinyin_confusion:
    features["Pinyin_confusion"] = 1
```

3.2 The RECALL is too low!!!

- By using **pure MEM**, most of us might noticed that even the f1_score is pretty high, (like 0.95) but the performance is really bad - **almost all words are recognized as 'O'** including most of names.

- That is because the **recall is very low**. We can not only focus on f1_score, but pay more attention on recall.

```
Testing classifier...


f_score=          0.9693
accuracy=         0.9689
recall=           0.7538
precision=        0.9693
```

(The recall of model trained 7-iteration)

3.2 Using beta-f1 score

- By but we can't put too much emphasis on Recall, therefore we use **beta_f1_score** to measure the performance of our model.

$$F_\beta = \frac{\beta^2 + 1}{(\beta^2 \cdot \text{recall}^{-1}) + \text{precision}^{-1}} = \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

- With this, we were able to train models that performed better.

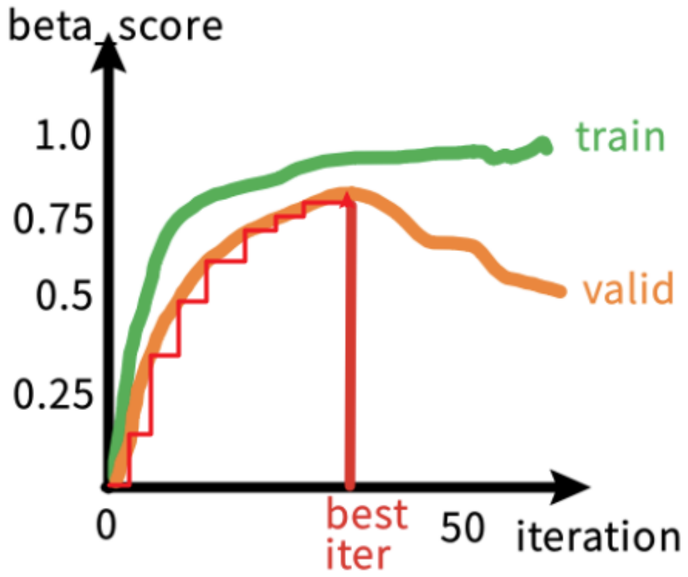
```
Testing classifier...


f_score=          0.9723
accuracy=         0.9840
recall=           0.8845
precision=        0.9723
```

3.3 Training is too slow!!!

- As MaxentClassifier() can only be called and run on a CPU and it cannot do incremental learning! So we need to train one model at a time from scratch.

- If we use the traditional early-stop method to find the best model according to its valid metrics, our time complexity would be *O(Kn^2)*.



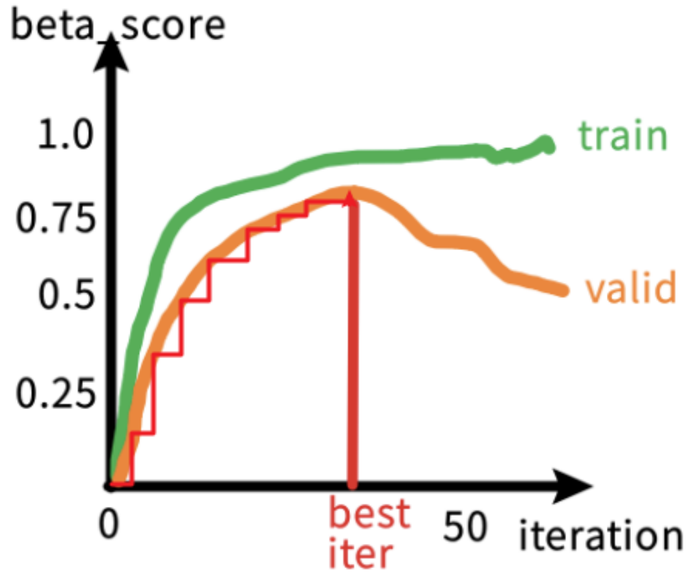
traditional early-stop

3.3 Ternary search algorithm

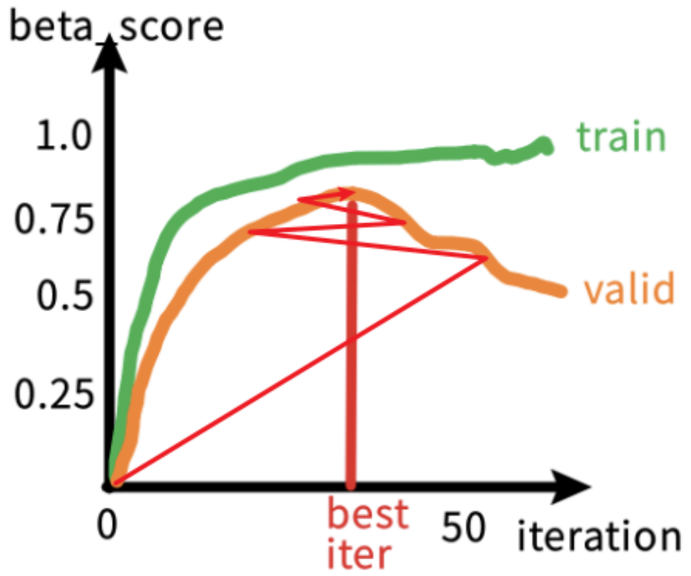
- So we implemented a **ternary search algorithm** to efficiently identify the optimal number of iterations that maximizes performance on the development data.

- This approach assumes that the validation metric follows a unimodal trend—initially increasing until reaching a peak before declining.

- Ternary Search early- stop has Time Complexity of $O(K\,n \log n)$.



traditional early-stop



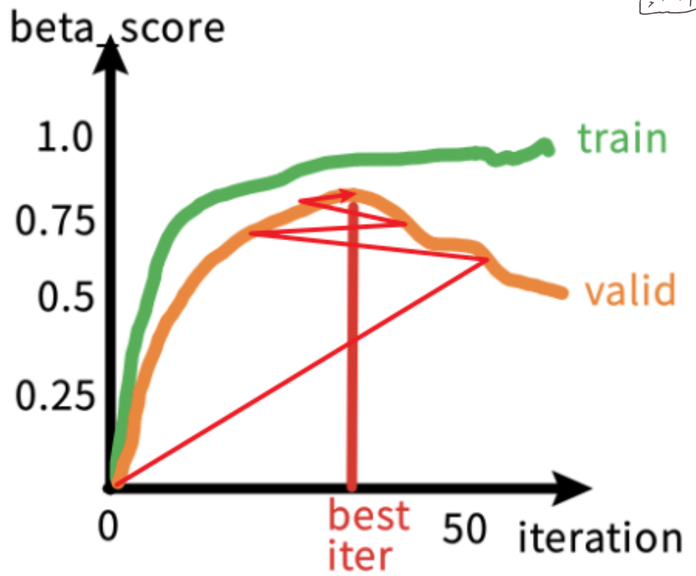
Ternary Search early- stop

3.3 Ternary Search

```
...
left = 0, right = 20
...
while right - left >= 3:
    mid1 = left + (right - left) // 3
    mid2 = right - (right - left) // 3

    # Train and evaluate mid1
    classifier1.train(train_samples, mid1 + 2)
    metrics1 = classifier1.test()
    beta_f1 = f_beta_score(metrics1['precision'], metrics1['recall'])

    # Train and evaluate mid2
    classifier2.train(train_samples, mid2 + 2)
    metrics2 = classifier2.test()
    beta_f2 = f_beta_score(metrics2['precision'], metrics2['recall']
```



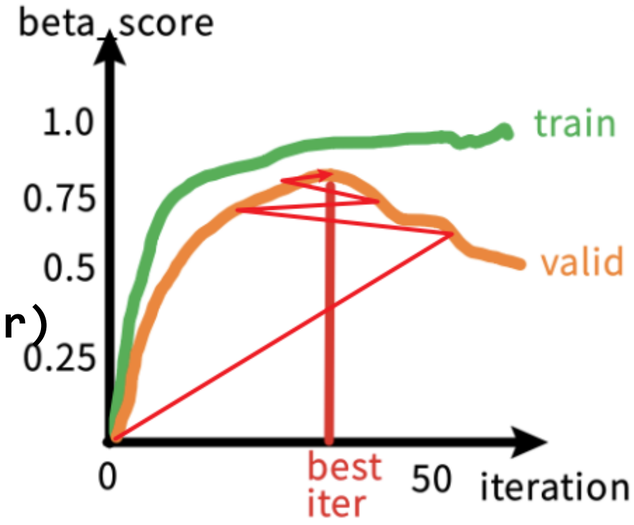
Ternary Search early- stop

3.3 Ternary Search

```
    # Update the best model
    if beta_f1 > best_score:
        best_score = beta_f1
        best_iter = mid1 + 2
        best_classifier = copy.deepcopy(classifier1.classifier)
    if beta_f2 > best_score:
        best_score = beta_f2
        best_iter = mid2 + 2
        best_classifier = copy.deepcopy(classifier2.classifier)

    # Narrow the search range
    if beta_f1 < beta_f2:
        left = mid1
    else:
        right = mid2

# Final evaluation (optional: fine-tune in the remaining range)
classifier.best_classifier = best_classifier
classifier.save_model(classifier.best_classifier)
```

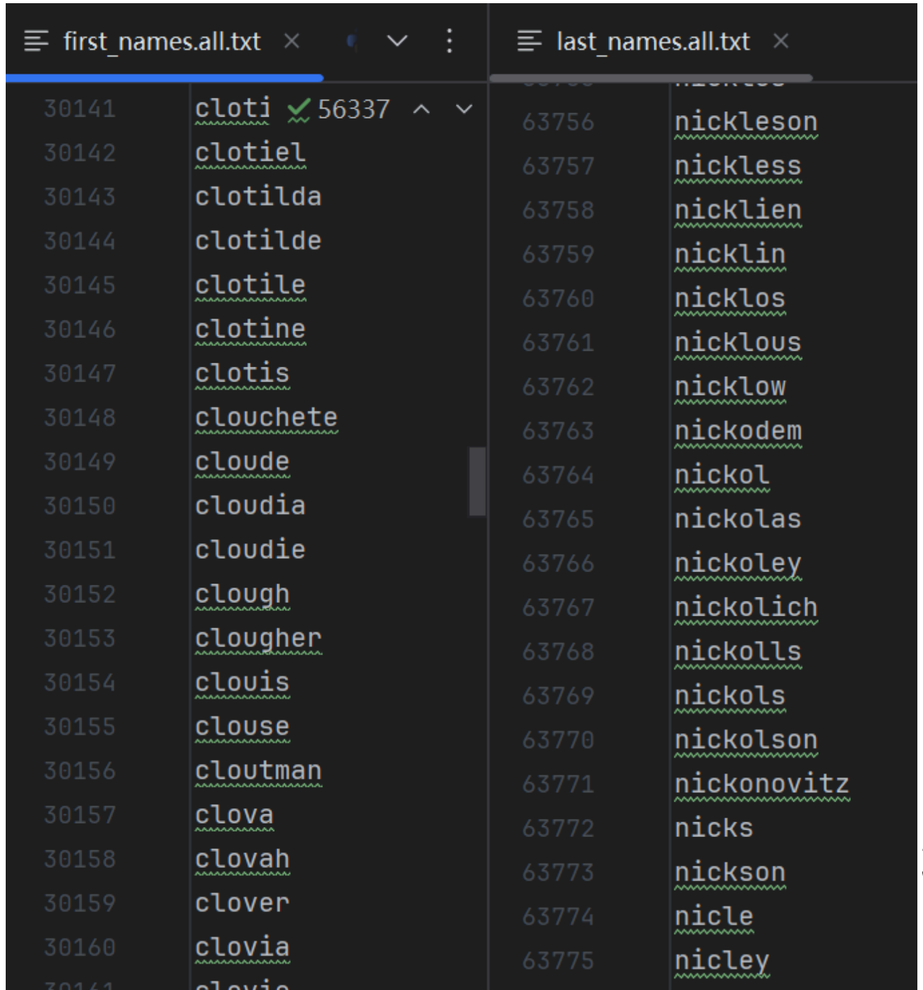


Ternary Search early- stop

3.4 The training set given is too small!!

- To address the **limited coverage of the given training set** for multilingual name identification, we **constructed two comprehensive gazetteers**:

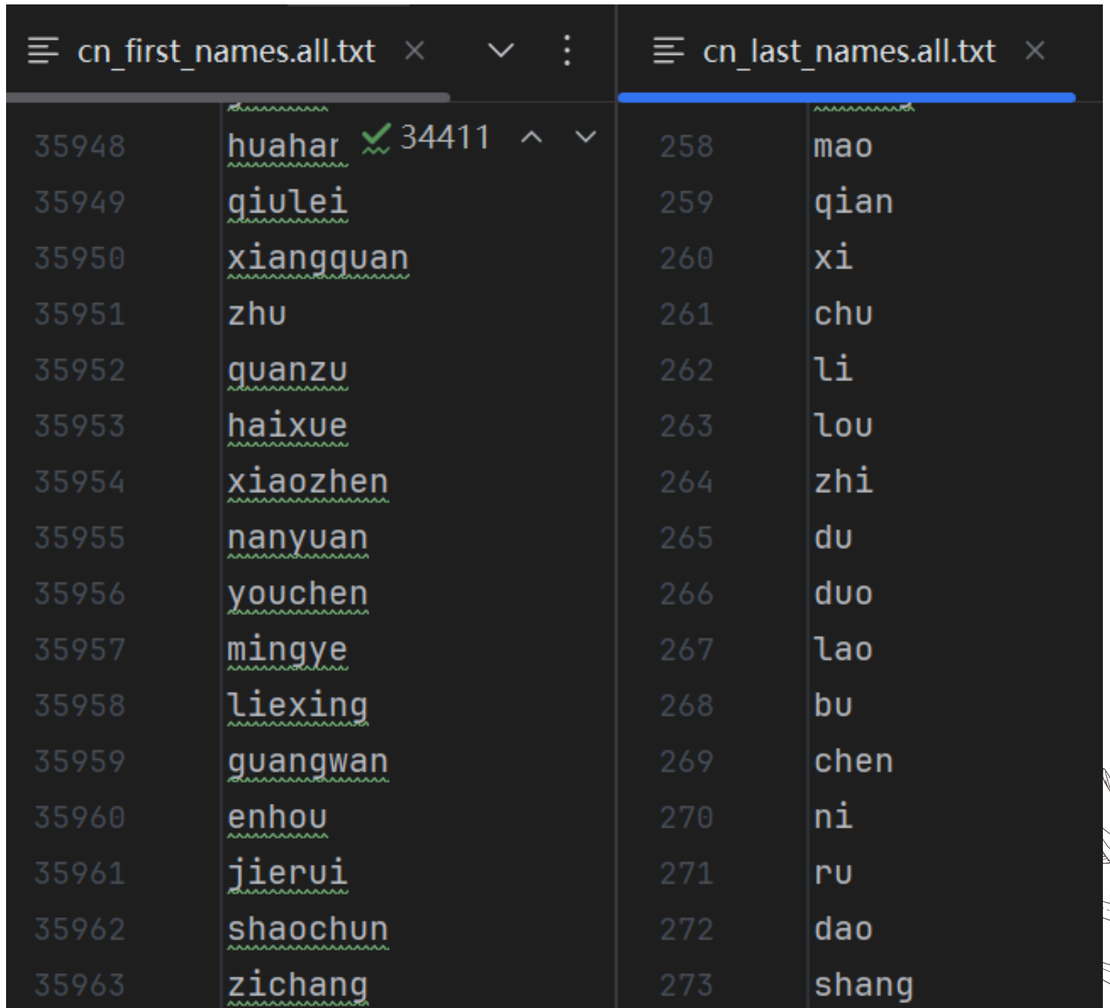
1. **Latin Script Gazetteer**
   - **Contains ~160k first names, ~100k last names (e.g., Smith, García, Muhammad)**
   - **Sources: names.io(https://github.com/Debdut/names.io/)**
   - **Covers 18 languages using Latin script (English, Spanish, Arabic transliterations, etc.)**



3.4 The training set given is too small!!

- To address the **limited coverage of the given training set** for multilingual name identification, we constructed two comprehensive gazetteers:

2. **Chinese Pinyin Gazetteer**
   - **Includes 35k+ common names and 308 family names:**
   - **Compound surnames (复姓, e.g., Ōuyáng 欧阳, Sīmǎ 司马)**
   - **Sources: 中文人名语料库 (https://github.com/wainshine/Chinese-Names-Corpus)**



3.4 The training set given is too small!!

- After building 2 gazetteers, we add up **4 features** for model to better identify the names:

```
# Gazetteer Checking
features['cur_word_in_first_name_gazetteer'] = \
    (current_word_lower in self.first_name_gazetteer)
features['cur_word_in_last_name_gazetteer'] = \
    (current_word_lower in self.last_name_gazetteer)
features['cur_word_in_cn_first_name_gazetteer'] = \
    (current_word_lower in self.cn_first_name_gazetteer)
features['cur_word_in_cn_last_name_gazetteer'] = \
    (current_word_lower in self.cn_last_name_gazetteer)
```

Latin Letters

Chinese Pinyin

3.5 additional mentioned...

- The original **test** method in **MEM.py** (provided on Moodle) was **incorrect** because **it used the true label of the previous word to predict the current word's label**.

```
73      def test(self):
74          print('Testing classifier...')
75          words, labels = self.load_data(self.dev_path) # The 'labels' are the correct answers of test data
76          previous_labels = ["O"] + labels              # But why we use the answers to predict next guessing answer?
77          features = [self.features(words, previous_labels[i], i)
78                      for i in range(len(words))]
79          results = [self.classifier.classify(n) for n in features]
80          '''list of str("O" or "PERSON")'''
81
82          f_score = fbeta_score(labels, results, average='macro', beta=self.beta)
83          precision = precision_score(labels, results, average='macro')
84          recall = recall_score(labels, results, average='macro')
85          accuracy = accuracy_score(labels, results)
```

3.5 additional mentioned...

```
def test(self):
    print('Testing classifier...')
    words, labels = self.load_data(self.dev_path)
    prev_label = "O"
    results = []
    '''list of str("O" or "PERSON")'''

    for i, word in enumerate(words):
        single_bunch_features = self.features(words, prev_label, i)
        pred_label = self.classifier.classify(single_bunch_features)
        results.append(pred_label)
        prev_label = pred_label  # update the previous label predicted

    f_score = fbeta_score(labels, results, average='macro', beta=self.beta)
    precision = precision_score(labels, results, average='macro')
    recall = recall_score(labels, results, average='macro')
    accuracy = accuracy_score(labels, results)
```

- To fix this, we rewrote the method so that it predicts each label based on the **previously predicted label** instead.

-04-

PRESENTATION & CONCLUSION

4.0 presentation on web

4.1 Conclusion

- In this project, our group designed features from the perspective of **different languages** and optimized the **training logic** to improve training efficiency, fully leveraging the advantages of MEMM in NER.

- The final model achieved excellent results (shown in figure below) and proved effective in recognizing **pinyin names** and **Latin variant** names.

- For the web interface, we used **Gradio** and called Python scripts to enable better interaction.

```
Testing classifier...


f_score=          0.9723
accuracy=         0.9840
recall=           0.8845
precision=        0.9723
```

Thank you!