ĐẠI HỌC QUỐC GIA HÀ NỘI
Trường Đại học Công nghệ

BÁO CÁO BÀI TẬP LỚN HỌC MÁY
PHÂN TÍCH BÌNH LUẬN

21021645 - Mai Thanh Tùng

Tóm tắt nội dung

Báo cáo trình bày kết quả bài tập lớn, nội dung bao gồm lý thuyết về học máy, tập dữ liệu, mô hình học máy đã được sử dụng và tối ưu tham số cho mô hình học máy. Nội dung bài tập lớn đề tài Phân tích bình luận, tập trung vào bài toán nhận dạng thực thể có tên (Named Entity Recognition), với đầu vào là một đoạn văn bản, thông qua mô hình học máy (XGBOOST) đưa ra nhãn gán cho từng từ trong đoạn văn bản đó. Chi tiết sẽ được trình bày cụ thể trong các phần của mục lục dưới đây.

Mục lục

Chương 1

Giới thiệu bài toán

Named Entity Recognition (NER) là bài toán xác định vị trí và phân loại các đối tượng quan trọng trong một văn bản, ví dụ như xác định địa danh, tên người nổi tiếng, các công ty lớn. NER được ứng dụng vào trong trích xuất thông tin, trong bài toán hỏi đáp và trong trình dịch máy [5]. Các thực thể được đặt tên (Named Entity) là các từ hoặc cụm từ được đặt hoặc phân loại theo các chủ đề nhất định. Thông thường những thực thể đó được phân loại theo ba lớp chính bao gồm lớp chỉ người (PER), lớp chỉ địa danh (LOC) và lớp chỉ một tổ chức (ORG), ngoài ra tùy theo mục đích và chủ đề hoàn toàn có thể phân lớp các thực thể sao cho phù hợp (MIC/MISC). Các nhãn BIO đi kèm theo các lớp thường được sử dụng cho các ghi nhãn theo trình tự, trong đó:

- B: (Beginning) bắt đầu ký tự chỉ một thực thể.
- I: (Inside) ký tự nằm trong một thực thể nhưng không phải ký tự bắt đầu.
- O: (Outside) ký tự nằm ngoài một thực thể.

1.1 Phương pháp dựa trên quy tắc (Rule - Based)

Đây là cách tiếp cận sớm nhất và vẫn còn được sử dụng trong một số hệ thống. Các phương pháp dựa trên quy tắc sử dụng các quy tắc ngữ pháp, từ điển và mẫu cấu trúc để nhận diện thực thể. Phương pháp này tỏ ra hiệu quả trong một số

trường hợp nhất định, đồng thời dễ hiểu và có thể giải thích được kết quả. Tuy nhiên các quy tắc và từ vựng ban đầu được khởi tạo thủ công dẫn đến việc mở rộng và duy trì mô hình cho bài toán có các quy mô lớn, hoặc cho một ngôn ngữ mới trở nên khó khăn.

1.2 Phương pháp dựa trên thống kê

Các phương pháp này sử dụng các mô hình học máy để học từ dữ liệu được gắn nhãn trước. Mô hình thống kê bao gồm hai thành phần chính:

- Dữ liệu là các văn bản đã được xác định các thực thể và gán nhãn tương ứng.
- Mô hình được huấn luyện dựa trên dữ liệu, từ đó thể hiện các đặc trưng của tập dữ liệu.

Một số mô hình thường được sử dụng cho phương pháp này như Hidden Markov Models, Conditional Random Fields, một số mô hình học máy như Support Vector Machines, Decision Trees. Cách tiếp cận này có khả năng tổng quát hóa tốt hơn so với các phương pháp dựa trên quy tắc, có thể xử lý các ngữ cảnh phức tạp. Tuy nhiên đánh đổi lại là yêu cầu nhiều tài nguyên hơn cho tính toán, đồng thời khó giải thích kết quả hơn và yêu cầu nhiều thời gian cho gán nhãn dữ liệu.

1.3 Phương pháp dựa trên học sâu

Mô hình học sâu với cấu trúc gồm nhiều mạng nơ ron, giúp cho khả năng tìm ra đặc trưng của dữ liệu trở nên chính xác hơn so với các mô hình học máy, từ đó có thể ứng dụng vào các bài toán ở mức tổng quát cao với độ tin cậy tốt hơn so với mô hình học máy. Một số mô hình học sâu được phát triển cho bài toán xử lý ngôn ngữ tự nhiên (NLP) và NER như RNN, LSTM, và gần đây là các mô hình Transformer như BERT và GPT. Ưu điểm của những mô hình học sâu này là có độ chính xác cao, có thể xử lý được các tập dữ liệu phức tạp. Tuy nhiên cũng giống như các phương pháp dựa trên thống kê, dữ liệu huấn luyện cần được gán nhãn, yêu cầu nhiều tài nguyên và khó có thể giải thích được kết quả.

Chương 2

Tập dữ liệu

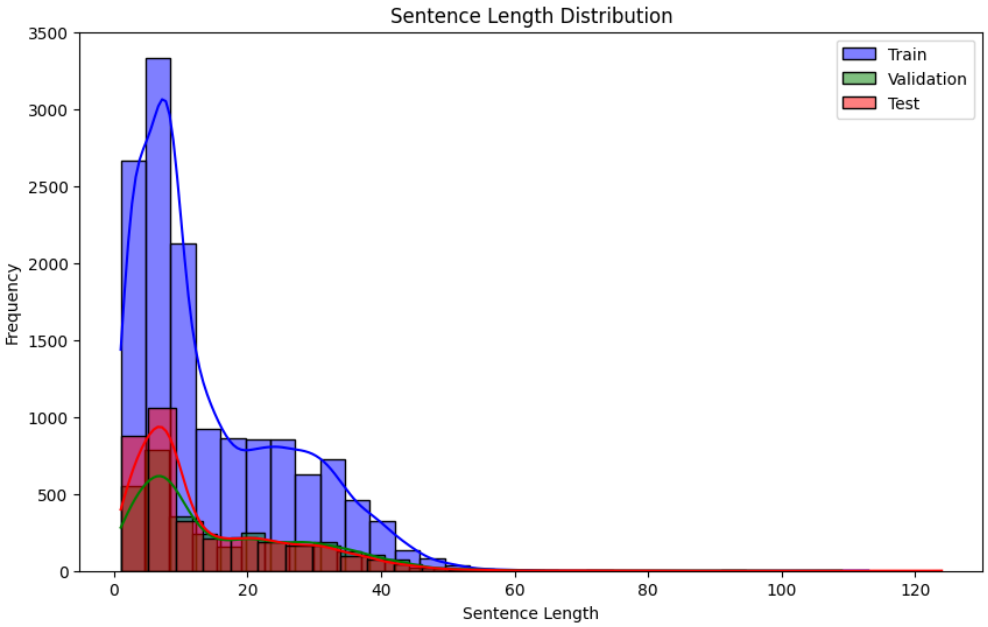
Tập dữ liệu được sử dụng cho mô hình là CoNLL-2003 [6], là một trong những tập dữ liệu nổi tiếng và phổ biến nhất được sử dụng trong lĩnh vực nhận diện thực thể (Named Entity Recognition - NER). Lần đầu được giới thiệu trong cuộc thi Shared Task của Hội nghị về Ngôn ngữ Tính toán và Học máy (Conference on Natural Language Learning - CoNLL) vào năm 2003.

Tập dữ liệu này bao gồm dữ liệu văn bản tiếng Anh và tiếng Đức được lấy từ Tin tức Reuters đối với tiếng Anh, trong khi văn bản tiếng Đức được lấy từ các bài báo của tờ Frankfurter Rundschau. Trong báo cáo này sẽ sử dụng tập dữ liệu là tiếng Anh. Nhãn của tập dữ liệu tuân theo định dạng BIO, các nhãn cho dữ liệu bao gồm:

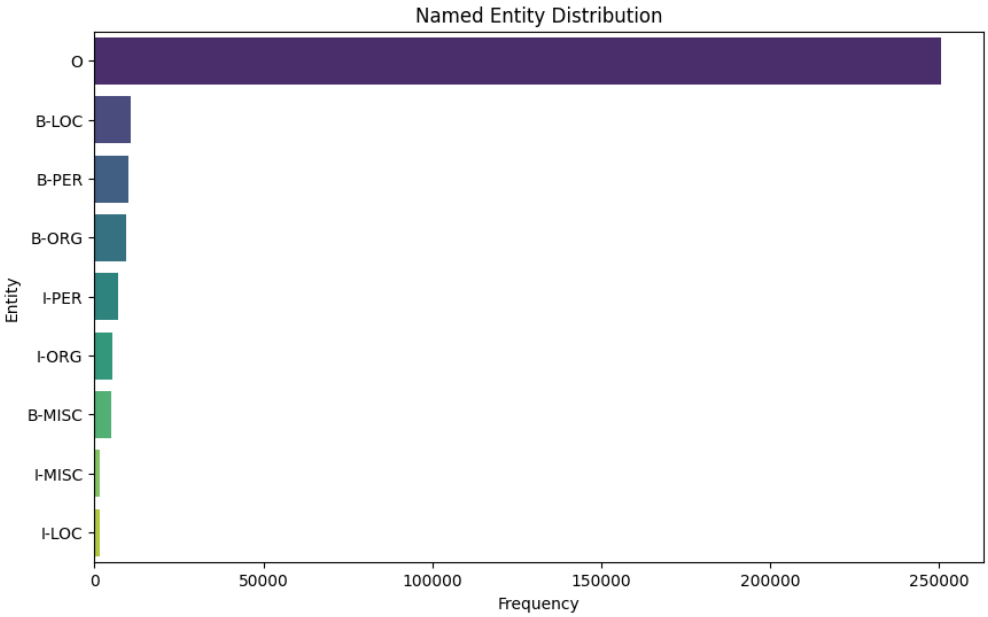
- PER (Person): chỉ người người.
- LOC (Location): chỉ địa điểm.
- ORG (Organization): chỉ tổ chức.
- MISC (Miscellaneous): Các loại khác

Tập dữ liệu tiếng anh được chia thành ba phần Train, Val, Test với lần lượt 14041, 3250, 3453 câu.

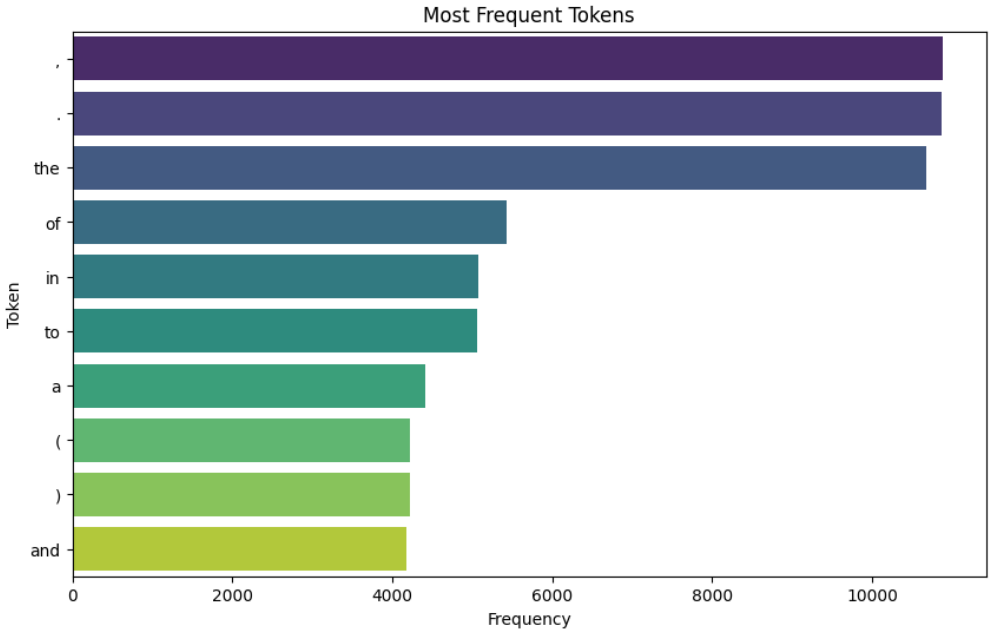
Dựa theo Table 5 [6], baseline precision cho tập dữ liệu CoNLL2003 tiếng Anh là 71.91%.



Hình 2.1: Phân bố của độ dài câu trong tập dữ liệu



Hình 2.2: Phân bố của các thực thể trong tập dữ liệu



Hình 2.3: Tần suất các ký tự trong tập dữ liệu

Chương 3

Mô hình học máy

3.1 Giới thiệu về Boosting

Boosting là một phương pháp xây dựng mô hình học máy dựa trên việc kết hợp tuần tự nhiều mô hình dự đoán có độ chính xác thấp hơn (weak learner), trong đó các mô hình sau sẽ hoàn thiện các thiếu sót của mô hình trước. Sự kết hợp đó tạo thành một mô hình dự đoán có độ chính xác cao hơn (strong learner)[4].

3.2 Gradient boosting

Gradient boosting xây dựng mô hình tổng hợp $F(x)$ dưới dạng các mô hình đơn giản $h_m(x)$ như sau:

$$F(x) = \sum_{m=1}^{M} \beta_m h_m(x)$$

Trong đó M là số lượng các weak leaners, β_m là trọng số của các mô hình $h_m(x)$

Tại mỗi bước m, ta cập nhật mô hình $F_m(x)$ bằng cách thêm một mô hình đơn giản $h_m(x)$ vào mô hình hiện tại $F_{m-1}(x)$:

$$F_m(x) = F_{m-1}(x) + \beta_m h_m(x)$$

Cập nhập phần dư $r_i^{(m)}$ tại bước m được tính toán dựa trên gradient của hàm mất mát L đối với dự đoán hiện tại $F_{m-1}(x_i)$:

$$r_i^{(m)} = -\left[\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}\right]$$

Với hàm lỗi bình phương $L(y, F) = \frac{1}{2}(y - F)^2$, phần dư sẽ là:

$$r_i^{(m)} = y_i - F_{m-1}(x_i)$$

Huấn luyện mô hình đơn giản $h_m(x)$ để dự đoán phần dư $r_i^{(m)}$.

Cập nhật mô hình tổng hợp với một hệ số học β_m:

$$F_m(x) = F_{m-1}(x) + \eta\beta_m h_m(x)$$

3.3 XGBOOST

XGBOOST là một mô hình được phát triển dựa trên mô hình Gradient Boosting và có khả năng mở rộng cho các tập dữ liệu lớn tốt hơn[3].

Hàm mục tiêu của XGBoost kết hợp giữa hàm mất mát và một thuật toán regularization để ngăn chặn overfitting:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Hàm regularization được định nghĩa như sau:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Giả sử tại bước t, mô hình dự đoán hiện tại là:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i)$$

Mô hình mới f_t được thêm vào để tối thiểu hóa hàm mục tiêu. Tổng dự đoán sau khi thêm mô hình mới là:

$$\hat{y}_i^{(t+1)} = \hat{y}_i^{(t)} + f_t(x_i)$$

Để áp dụng thuật toán tối ưu hóa, XGBoost sử dụng đạo hàm bậc nhất (gradient) và đạo hàm bậc hai (Hessian) của hàm mất mát. Cụ thể, với mỗi mẫu i:

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{(t)})}{\partial \hat{y}_i^{(t)}}$$

$$h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t)})}{\partial (\hat{y}_i^{(t)})^2}$$

XGBoost sử dụng xấp xỉ bậc hai của hàm mất mát để tối ưu hóa. Hàm mục tiêu xấp xỉ là:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \Omega(f_t)$$

Bỏ qua các hằng số không ảnh hưởng đến tối ưu hóa, ta có:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

Tối ưu hóa hàm mục tiêu cho một cây quyết định dựa trên việc chọn các split points (điểm phân chia) để giảm thiểu lỗi.

3.4 Ứng dụng XGBOOST vào trong bài toán NER

Chuẩn bị tập dữ liệu cho mô hình, do tập dữ liệu CoNLL2003 đã chia sẵn các phần Train, Val, Test do đó chỉ cần tải các dữ liệu này rồi gán cho các biến train_tokens, val_tokens, test_tokens. Sau đó dữ liệu văn bản được chuyển đổi về dạng dữ liệu số thông qua hàm TfidfVectorizer. Các nhãn của dữ liệu được mã hóa thông qua hàm LabelEncoder.

```
# data loader
train_tokens, train_labels = flatten_conll_dataset(ner_dataset['train'])
test_tokens, test_labels = flatten_conll_dataset(ner_dataset['test'])
val_tokens, val_labels = flatten_conll_dataset(ner_dataset['validation'])

#data vectorizer
vectorizer = TfidfVectorizer(max_features=10000)
X_ner_train = vectorizer.fit_transform(train_tokens)
X_ner_test = vectorizer.transform(test_tokens)
X_ner_val = vectorizer.transform(val_tokens)

#encode label
label_encoder = LabelEncoder()
y_ner_train = label_encoder.fit_transform(train_labels)
y_ner_test = label_encoder.transform(test_labels)
y_ner_val = label_encoder.transform(val_labels)

label_names = label_encoder.inverse_transform(
                                    range(len(label_encoder.classes_)))
```

Tải mô hình XGBOOST và thiết đặt các tham số ban đầu như sau:

- n_estimators = 100: số lượng cây quyết định con tối đa được tạo ra và thiết lập vào mô hình.
- device = 'cuda': lựa chọn tài nguyên cho học máy là GPU.
- tree_method: 'hist': thuật toán lựa chọn cây quyết định, thuộc tính 'hist' dùng khi lựa chọn tài nguyên là GPU.

Tiến hành huấn luyện mô hình với các tham số và dữ liệu đã thiết lập:

```
import xgboost as xgb
n_estimators = 100
ner_model = xgb.XGBClassifier(tree_method = 'hist',
                              device="cuda",
                              n_estimators = n_estimators)
```

```
ner_model.fit(X_ner_train, y_ner_train)
y_ner_val_predict = ner_model.predict(X_ner_val)
print(precision_score(y_ner_val, y_ner_val_predict, average = 'weighted'))
print(classification_report(y_ner_val, y_ner_val_predict,
                            target_names=convert()))
```

Kết quả sau khi huấn luyện mô hình:

	precision	recall	f1-score	support
O	0.87	1.00	0.93	42759
B-PER	0.90	0.19	0.31	1842
I-PER	0.79	0.05	0.09	1307
B-ORG	0.73	0.15	0.24	1341
I-ORG	0.78	0.11	0.19	751
B-LOC	0.92	0.41	0.57	1837
I-LOC	0.68	0.43	0.53	257
B-MISC	0.93	0.41	0.57	922
I-MISC	0.70	0.33	0.45	346
accuracy			0.87	51362
macro avg	0.81	0.34	0.43	51362
weighted avg	0.87	0.87	0.83	51362

Hình 3.1: Kết quả (Val_data)

Macro average tính trung bình các chỉ số hiệu suất cho từng lớp mà không xem xét đến số lượng mẫu trong mỗi lớp. Weighted average tính trung bình các chỉ số hiệu suất cho từng lớp, có xem xét đến số lượng mẫu trong mỗi lớp. Hai thông số này cho biết precision với từng ngữ cảnh khác nhau, tuy nhiên kết quả của hai tham số này đều đã lớn hơn baseline của dataset là 0.7191.

Chức năng nhập một câu bất kỳ và đưa ra phân tích cho câu đó:

```
def preprocess_text(text):
  tokens = re.findall(r'\b\w+\b', text)
  return tokens

def predict_ner(text):
```

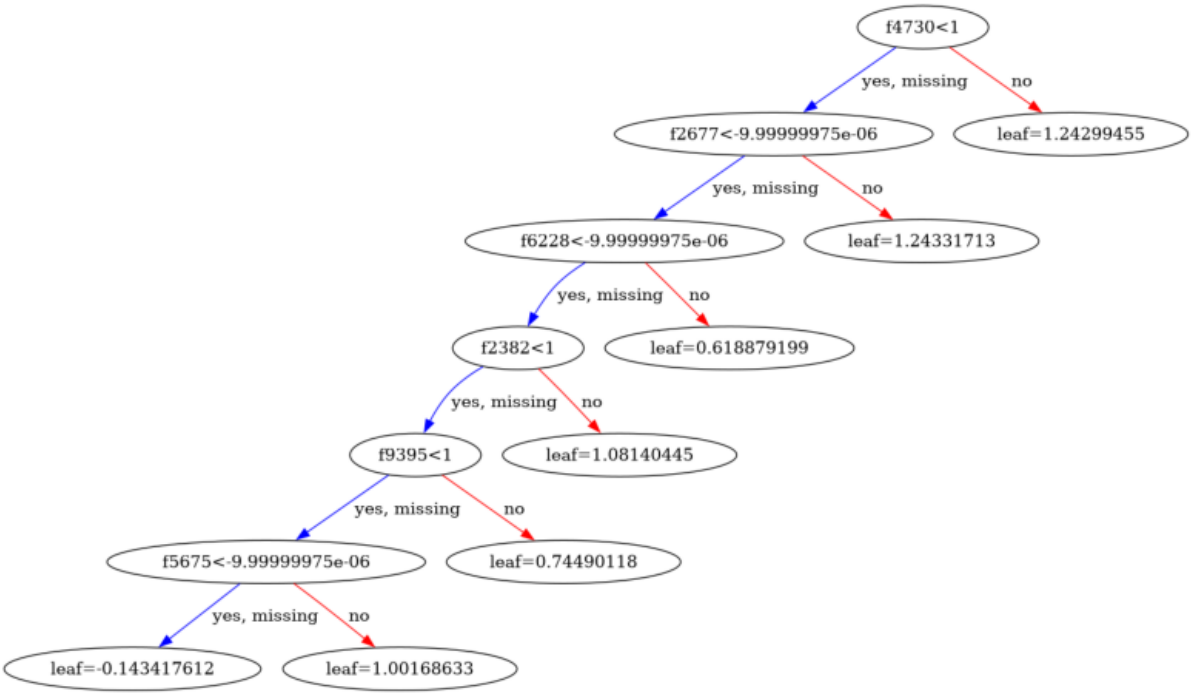
```
  tokens = preprocess_text(text)
  X_input = vectorizer.transform(tokens)
  y_pred = ner_model.predict(X_input)


  tag = {'O': 0, 'B-PER': 1, 'I-PER': 2, 'B-ORG': 3, 'I-ORG': 4,
  'B-LOC': 5, 'I-LOC': 6, 'B-MISC': 7, 'I-MISC': 8}
  reverse_tag = {v: k for k, v in tag.items()}
  predicted_labels = [reverse_tag[label] for label in y_pred]

  result = list(zip(tokens, predicted_labels))
  return result

input_text = input("text here ")
input_text = 'George Washington was the first president of the United States'
predictions = predict_ner(input_text)
print(predictions)
```

Kết quả:

```
[('George', 'O'), ('Washington', 'B-LOC'), ('was', 'O'), ('the', 'O'),
('first', 'O'), ('president', 'O'), ('of', 'O'), ('the', 'O'),
('United', 'O'), ('States', 'I-LOC')]
```

Cây quyết định của mô hình:

```
fig, ax = plt.subplots(figsize=(10, 10))
xgb.plot_tree(ner_model, num_trees=4, ax=ax, rankdir = 'LB')
plt.show()
```



Hình 3.2: Cây quyết định mô hình

Chương 4

Điều chỉnh siêu tham số

Điều chỉnh tham số là quá trình tìm kiếm các giá trị tối ưu cho các siêu tham số nhằm cải thiện hiệu suất của mô hình học máy nhằm cải thiện hiệu suất của mô hình. Mô hình XGBOOST có thể điều chỉnh các tham số sau:

- max_depth: độ sâu cây tố đa trong một mô hình.
- learning_rate: tốc độ học hay hệ số điều chỉnh mức độ đóng góp của mỗi cây vào kết quả cuối cùng.
- n_estimators: số lượng cây được xây dựng.
- gamma: tham số thể hiện có cần thiết phải phân chia nhánh trong cây hay không.
- min_child_weight: trọng số tối thiểu của một lá, tránh tạo ra các lá có ít giá trị cho cây.
- subsample: Tỷ lệ mẫu ngẫu nhiên được sử dụng để xây dựng mỗi cây.
- colsample_bytree: Tỷ lệ các đặc trưng được lấy mẫu ngẫu nhiên cho mỗi cây.
- reg_alpha: Tham số điểu chỉnh L1 regularization.
- reg_lambda: Tham số điều chỉnh L2 regularization.

4.1 Hyperopt Optimization

Hyperopt[2] là phương pháp tối ưu siêu tham số, phương pháp này sử dụng ba thuật toán tối ưu là:

- Random Search.
- Tree - structured Parzen Estimator (TPE).
- Adaptive TPE.

Tree - structured Parzen Estimator (TPE) là một thuật toán tối ưu hóa được tiếp cận theo phương pháp Bayesian[1]. Cách tiếp cận này coi hàm mục tiêu $f(x)$ cần tối ưu hóa với x là vector siêu tham số. TPE chia không gian các siêu tham số x thành hai vùng dựa trên ngưỡng γ, khi đó có thể biểu diễn tập hợp các siêu tham số như sau[7]:

$$l(x) = p(x|y < \gamma) \tag{4.1}$$

$$g(x) = p(x|y \geq \gamma) \tag{4.2}$$

Trong đó 4.1 thể hiện xác suất có điều kiện của các siêu tham số với giá trị hàm mục tiêu $f(x)$ có giá trị nhỏ hơn γ, 4.2 thể hiện xác suất có điều kiện của các siêu tham số với giá trị hàm mục tiêu $f(x)$ có giá trị lớn hơn hoặc bằng γ. TPE chọn các giá trị siêu tham số tiếp theo bằng cách tối đa hóa tỉ lệ:

$$\arg\max_x \left(\frac{l(x)}{g(x)}\right)$$

.

Tối ưu hóa mô hình XGBOOST sử dụng hyperopt:

```
space = {
    'max_depth': hp.quniform('max_depth', 3, 15, 1),
    'learning_rate': hp.loguniform('learning_rate', -3, 0),
    'n_estimators': hp.quniform('n_estimators', 50, 1000, 10),
    'gamma': hp.quniform('gamma', 0, 0.5, 0.01),
    'min_child_weight': hp.quniform('min_child_weight', 1, 6, 1),
```

```
    'subsample': hp.uniform('subsample', 0.5, 1),
    'colsample_bytree': hp.uniform('colsample_bytree', 0.5, 1),
    'reg_alpha': hp.loguniform('reg_alpha', -3, 1),
    'reg_lambda': hp.loguniform('reg_lambda', -3, 1)
}

def objective(params):
    params['max_depth'] = int(params['max_depth'])
    params['n_estimators'] = int(params['n_estimators'])

    model = xgb.XGBClassifier(
        max_depth=params['max_depth'],
        learning_rate=params['learning_rate'],
        n_estimators=params['n_estimators'],
        gamma=params['gamma'],
        min_child_weight=params['min_child_weight'],
        subsample=params['subsample'],
        colsample_bytree=params['colsample_bytree'],
        reg_alpha=params['reg_alpha'],
        reg_lambda=params['reg_lambda'],
        objective='multi:softprob',
        eval_metric='mlogloss',
        use_label_encoder=False,
        device='cuda',
        tree_method = 'approx'
    )

    model.fit(X_ner_train, y_ner_train)
    y_pred = model.predict(X_ner_test)
    accuracy = accuracy_score(y_ner_test, y_pred)

    return {'loss': -accuracy, 'status': STATUS_OK}

trials = Trials()
best = fmin(fn=objective,
            space=space,
            algo=tpe.suggest,
            max_evals=100,
            trials=trials)
```

```
print("Best hyperparameters:", best)
```

Siêu tham số sau khi được điều chỉnh bằng phương pháp hyperopt:

```
{'colsample_bytree': 0.8036618670709809,
'gamma': 0.16,
'learning_rate': 0.5325818109045147,
'max_depth': 13.0,
'min_child_weight': 1.0,
'n_estimators': 580.0,
'reg_alpha': 0.10715360359894781,
'reg_lambda': 0.1854558606786038,
'subsample': 0.999984226036394}
```

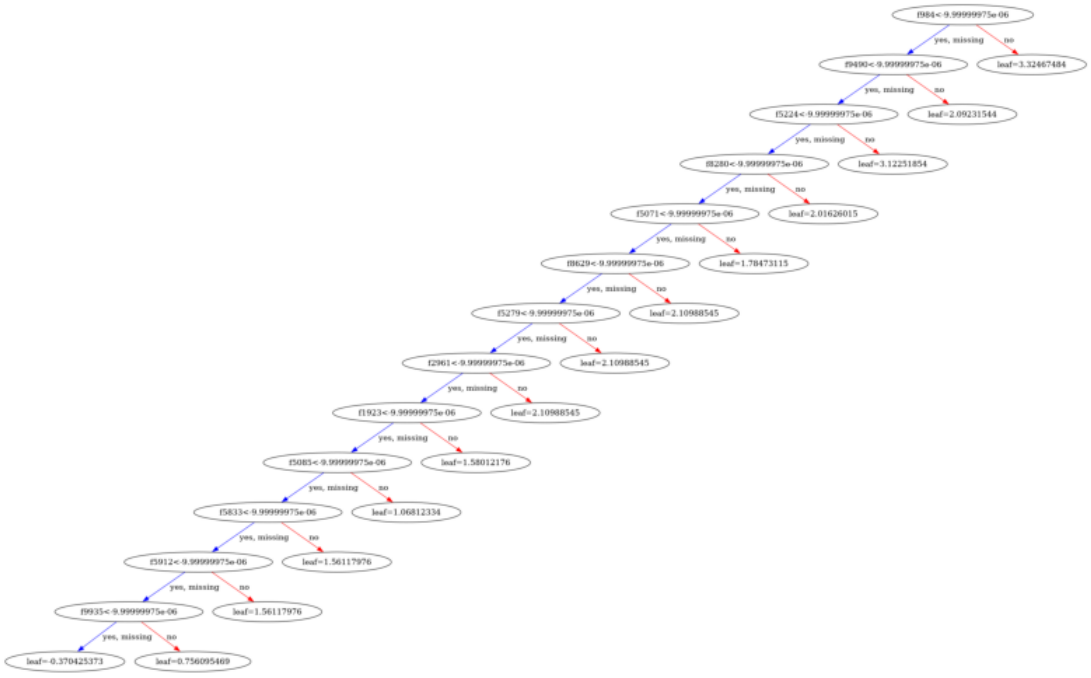
Sử dụng mô hình với các siêu tham số trên:

```
best['max_depth'] = int(best['max_depth'])
best['n_estimators'] = int(best['n_estimators'])
model = xgb.XGBClassifier(
    max_depth=best['max_depth'],
    learning_rate=best['learning_rate'],
    n_estimators=best['n_estimators'],
    gamma=best['gamma'],
    min_child_weight=best['min_child_weight'],
    subsample=best['subsample'],
    colsample_bytree=best['colsample_bytree'],
    reg_alpha=best['reg_alpha'],
    reg_lambda=best['reg_lambda'],
    objective='multi:softprob',
    eval_metric='mlogloss',
    use_label_encoder=False,
    device = 'cuda',
    tree_method = 'approx'
).fit(X_ner_train, y_ner_train)
```

Kết quả:

	precision	recall	f1-score	support
O	0.91	1.00	0.95	42759
B-PER	0.85	0.36	0.51	1842
I-PER	0.65	0.17	0.27	1307
B-ORG	0.73	0.42	0.53	1341
I-ORG	0.68	0.22	0.34	751
B-LOC	0.87	0.65	0.74	1837
I-LOC	0.67	0.51	0.58	257
B-MISC	0.86	0.62	0.72	922
I-MISC	0.70	0.34	0.46	346
accuracy			0.90	51362
macro avg	0.77	0.48	0.57	51362
weighted avg	0.89	0.90	0.88	51362

Hình 4.1: Kết quả sau khi điều chỉnh siêu tham số (val_data)



Hình 4.2: Cây quyết định

4.2 Đánh giá điều chỉnh siêu tham số

Dựa trên bảng kết quả và cây quyết định của mô hình XGBOOST ban đầu và sau khi điều chỉnh siêu tham số với tập dữ liệu test, cả hai đều vượt qua được baseline

	precision	recall	f1-score	support
O	0.86	1.00	0.92	38323
B-PER	0.94	0.13	0.22	1617
I-PER	0.70	0.03	0.05	1156
B-ORG	0.80	0.11	0.19	1661
I-ORG	0.66	0.09	0.15	835
B-LOC	0.90	0.37	0.53	1668
I-LOC	0.70	0.34	0.46	257
B-MISC	0.87	0.34	0.49	702
I-MISC	0.56	0.46	0.51	216
accuracy			0.86	46435
macro avg	0.78	0.32	0.39	46435
weighted avg	0.85	0.86	0.81	46435
	precision	recall	f1-score	support

Hình 4.3: Kết quả với mô hình ban đầu (test_data)

	precision	recall	f1-score	support
O	0.89	0.99	0.94	38323
B-PER	0.85	0.25	0.38	1617
I-PER	0.66	0.06	0.11	1156
B-ORG	0.74	0.32	0.45	1661
I-ORG	0.66	0.26	0.37	835
B-LOC	0.83	0.66	0.74	1668
I-LOC	0.61	0.37	0.46	257
B-MISC	0.78	0.50	0.61	702
I-MISC	0.57	0.48	0.52	216
accuracy			0.88	46435
macro avg	0.73	0.43	0.51	46435
weighted avg	0.87	0.88	0.86	46435

Hình 4.4: Kết quả với mô hình đã điều chỉnh siêu tham số (test_data)

của tập dữ liệu (0.7191)[6]. Nhìn chung về mặt tổng thể, mô hình đã điều chỉnh siêu tham số hoạt động tốt hơn do accuracy, recall và F1 score tăng. Sự sụt giảm Macro avg và tăng lên của Weighted avg cho thấy mô hình đang có dấu hiệu hoạt động không tốt với các lớp có nhãn hiếm gặp, nhưng dự đoán tốt hơn với các lớp có nhãn phổ biến. Độ sâu của cây quyết định tăng lên nhiều, cho thấy mô hình đã trở nên phức tạp hơn, tuy nhiên cũng có khả năng mô hình gặp phải tình trạng over fitting.

Chương 5

Kết luận

Báo cáo bài tập lớn học máy với đề tài Phân tích bình luận đã trình bày những những nội dung của bài toán Named Entity Recognition với tập dữ liệu CoNLL2003, sử dụng mô hình XGBOOST và thuật toán điều chỉnh tham số Hyperopt. Kết quả ban đầu cho thấy mô hình XGBOOST đã có kết quả dữ đoán khá tốt, tuy nhiên báo cáo vẫn chưa có cơ hội cải thiện mô hình dựa trên các thuật toán điều chỉnh siêu tham số khác. Tổng kết lại báo cáo đã trình bày về bài toán NER, là một cơ sở cho phát triển các mô hình xử lý ngôn ngữ tự nhiên sau này.

Tài liệu tham khảo

[1] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. Advances in neural information processing systems, 24, 2011.

[2] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In International conference on machine learning, pages 115–123. PMLR, 2013.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pages 785–794, 2016.

[4] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön. Machine learning: a first course for engineers and scientists. Cambridge University Press, 2022.

[5] B. Mohit. Named entity recognition. In Natural language processing of semitic languages, pages 221–245. Springer, 2014.

[6] E. F. Sang and F. De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. arXiv preprint cs/0306050, 2003.

[7] S. Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. arXiv preprint arXiv:2304.11127, 2023.