TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN

-----□□&□□-----

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN: NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN

ĐỀ TÀI: NGHIÊN CỨU VÀ PHÁT TRIỂN HỆ THỐNG CHỈNH SỬA LỖI CHÍNH TẢ

Giảng viên hướng dẫn: ThS. NGUYỄN THIỆN DƯƠNG

Sinh viên thực hiện:

ĐINH NGUYỄN MINH HOÀNG - V6251071031

TRẦN NHỰT NAM - 6351071047

LƯƠNG ĐỨC QUÝ - 6351071061

PHẠM THÀNH NHÂN - 6351071051

NGUYỄN MINH THẮNG - V6151071102

Lớp: Công nghệ thông tin

Khóa: 63

TP. Hồ Chí Minh, năm 2025

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN

-----□□&□□-----

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN: NHẬP MÔN XỬ LÝ NGÔN NGỮ TỰ NHIÊN

ĐỀ TÀI: NGHIÊN CỨU VÀ PHÁT TRIỂN HỆ THỐNG CHỈNH SỬA LỖI CHÍNH TẢ

Giảng viên hướng dẫn: ThS. NGUYỄN THIỆN DƯƠNG

Sinh viên thực hiện:

ĐINH NGUYỄN MINH HOÀNG - V6251071031

TRẦN NHỰT NAM - 6351071047

LƯƠNG ĐỨC QUÝ - 6351071061

PHẠM THÀNH NHÂN - 6351071051

NGUYỄN MINH THẮNG - V6151071102

Lớp: Công nghệ thông tin

Khóa: 63

TP. Hồ Chí Minh, năm 2025

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI **CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**

PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH Độc lập – Tự do – Hạnh phúc

NHIỆM VỤ THIẾT KẾ BÀI TẬP LỚN

BỘ MÔN: **CÔNG NGHỆ THÔNG TIN**

-------***-------

1. Tên đề tài

Nghiên cứu và phát triển hệ thống chỉnh sửa lỗi chính tả

2. Mục đích, yêu cầu

Mục đích của đề tài là xây dựng một hệ thống tích hợp hai chức năng chính: sửa lỗi chính tả (Spell check) và chuyển đổi giọng nói thành văn bản(Speech-to-text). Hệ thống ứng dụng các mô hình Machine Learning và mô hình ngôn ngữ lớn (LLM) nhằm nâng cao độ chính xác trong việc phát hiện và gợi ý sửa lỗi chính tả, thông qua các thuật toán như Random Forest Tuned, XGBoost Tuned, LightGBM. Đồng thời, hệ thống tích hợp các kĩ thuật AI trong xử lý tiếng nói như CNN, RNN, wav2vec 2.0 và Whisper để chuyển đổi âm thanh thành văn bản một cách nhanh chóng và hiệu quả.

Thông qua việc thực hiện đề tài, sinh viên có cơ hội rèn luyện các kỹ năng quang trọng như: xử lý ngôn ngữ tự nhiên (NLP), phát triển mô hình AI, xây dựng backend bằng Python, thiết kế giao diện và triển khai sản phẩm thực tế. Bên cạnh đó, đề tài còn giúp sinh viên vận dụng các phương pháp phân tích và thiết kế thông qua việc xây dựng các sơ đồ Use case, Class và Activity, đảm bảo hệ thống được tổ chức một cách rõ ràng, khoa học và dễ mở rộng trong tương lai.

3. Nội dung và phạm vi đề tài

- Chương 1: Lý do chọn Dataset và giới thiệu tổng quan Dataset

- Chương 2: Chỉnh lỗi chính tả (Spell check)

- Chương 3: Chuyển đổi giọng nói thành văn bản (Speech – to – text)

4. Công nghệ, công cụ và ngôn ngữ lập trình

- Công cụ phát triển:

+ Visual Studio Code

+ Google Colab

+ Streamlit

- Machine Learing & Deep Learning

+ Scikit-learn – Random Forest, XGBoost, LightGBM cho Spell Check

+ LightGBM / XGBoost – mô hình phân loại lỗi chính tả (insertion, deletion, substitution)

+ PyTorch – dùng cho các mô hình Speech-to-Text như RNN, CNN, wav2vec2, Whisper

+ Transformers (HuggingFace) – tải mô hình Whisper / wav2vec2

- Các kết quả chính dự kiến sẽ đạt được:

+ Phát triển kỹ năng chuyên môn về AI và NLP, bao gồm xử lý văn bản, xử lý âm thanh, xây dựng mô hình Machine Learning (Random Forest, XGBoost, LightGBM) và mô hình Speech-to-Text (CNN, RNN, wav2vec2, Whisper)

+ Xây dựng được giao diện và các chức năng cốt lõi của website, bao gồm nhập văn bản để kiểm tra chính tả – ngữ pháp, tải/ghi âm để chuyển đổi giọng nói thành văn bản, hiển thị kết quả dự đoán và confidence score

+ Phát triển tư duy và kỹ năng team word

5. Giảng viên và cán bộ hướng dẫn

Họ tên: ThS. Nguyễn Thiện Dương

Đơn vị công tác: Phân hiệu Trường Đại học Giao Thông Vận Tải

Ngày tháng 05 năm 2025	**Đã giao nhiệm vụ TKTN**
Trưởng BM Công nghệ Thông tin	**Giảng viên hướng dẫn**
ThS. Trần Phong Nhã	

Đã nhận nhiệm vụ

Sinh viên: Đinh Nguyễn Minh Hoàng Ký tên:

Sinh viên: Phạm Thành Nhân Ký tên:

Sinh viên: Lương Đức Quý Ký tên:

Sinh viên: Trần Nhựt Nam Ký tên:

Sinh viên: Nguyễn Minh Thắng Ký tên

LỜI CẢM ƠN

Để hoàn thành đồ án này trước hết chúng em xin gửi đến quý thầy, cô Bộ môn Công nghệ thông tin – Phân hiệu Trường Đại học Giao thông Vận tải tại Thành phố Hồ Chí Minh lời cảm ơn chân thành vì đã truyền đạt cho chúng em những kiến thức không chỉ từ sách vở, mà còn những kinh nghiệm quý giá từ cuộc sống trong khoảng thời gian học tập tại trường. Đặc biệt chúng em xin gửi đến thầy ThS. Nguyễn Thiện Dương lời cảm ơn sâu sắc nhất vì thầy đã tận tình hướng dẫn, chỉ bảo chúng em trong suốt quá trình thực hiện đề tài. Tuy đề tài không được lớn nhưng nếu không được sự hướng dẫn chỉ bảo tận tình của thầy thì đề tài bài tập lớn này khó có thể hoàn thành được.

Vì thời gian làm đề tài bài tập lớn có hạn cũng như hiểu biết của chúng em còn hạn chế, chúng em cũng đã nỗ lực hết sức để hoàn thành bài báo cáo đồ án một cách tốt nhất, nhưng chắc chắn vẫn sẽ có những thiếu sót không thể tránh khỏi. Chúng em kính mong nhận được sự thông cảm và những ý kiến đóng góp chân thành từ quý thầy cô.

Sau cùng, chúng em xin kính chúc quý Thầy Cô trong Bộ môn Công nghệ thông tin luôn mạnh khoẻ, hạnh phúc và thành công hơn nữa trong công việc cũng như trong cuộc sống.

Chúng em xin chân thành cảm ơn!

Tp. Hồ Chí Minh, ngày … tháng … năm 2025

LỜI MỞ ĐẦU

Trong thời đại công nghệ số phát triển mạnh mẽ, nhu cầu xử lý ngôn ngữ tự nhiên và tự động hóa các tác vụ liên quan đến văn bản ngày càng trở nên quan trọng. Việc kiểm tra chính tả, chỉnh sửa ngữ pháp và chuyển đổi giọng nói thành văn bản không chỉ hỗ trợ nâng cao hiệu quả học tập và công việc, mà còn góp phần tạo ra trải nghiệm tương tác thân thiện, nhanh chóng và chính xác hơn cho người dùng. Nhờ sự phát triển vượt bậc của trí tuệ nhân tạo (AI) và các mô hình học sâu hiện đại, những tác vụ tưởng chừng phức tạp này đã có thể được tự động hóa với độ chính xác cao.

Dự án "Chỉnh sửa lỗi chính tả và chuyển đổi giọng nói thành văn bản" được thực hiện với mục tiêu mang đến cho người dùng một nền tảng web thông minh, cho phép kiểm tra lỗi chính tả dựa trên các mô hình Machine Learning như Random Forest Tuned, XGBoost Tuned, LightGBM và mô hình ngôn ngữ FLAN-T5, đồng thời hỗ trợ chuyển đổi âm thanh thành văn bản thông qua các mô hình Speech-to-Text hiện đại như CNN, RNN (LSTM/GRU), wav2vec 2.0 và Whisper.

Hệ thống được xây dựng dựa trên nền tảng công nghệ Python, FastAPI, PyTorch và các thư viện NLP – Audio Processing, kết hợp với giao diện web trực quan giúp người dùng thao tác dễ dàng. Thông qua quá trình triển khai dự án, nhóm mong muốn vận dụng kiến thức về lập trình, AI, xử lý ngôn ngữ tự nhiên và thiết kế hệ thống để xây dựng một sản phẩm có tính ứng dụng thực tế, độ chính xác cao và có khả năng mở rộng trong tương lai.

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp. Hồ Chí Minh, ngày … tháng … năm 2025

Giảng viên hướng dẫn

ThS. Nguyễn Thiện Dương

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

DANH MỤC CHỮ VIẾT TẮT

STT	Viết tắt	Diễn giải	Ý nghĩa
1	NLP	Natural Language Processing	Xử lý ngôn ngữ tự nhiên.
2	LSTM	Long Short-Term Memory	Mạng ghi nhớ dài–ngắn hạn
3	RNN	Recurrent Neural Network	Mạng nơ-ron hồi quy
4	CNN	Convolutional Neural Network	Mạng nơ-ron tích chập
5	MFCC	Mel-Frequency Cepstral Coefficients	Kiểu dữ liệu mở trong JavaScript
6	XML	Extensible Markup Language	Ngôn ngữ đánh dấu mở rộng
7	LightGBM	Light Gradient Boosting Machine	Thuật toán tăng cường độ dốc nhẹ (hiệu năng cao)
8	AI	Artificial Intelligence	Trí tuệ nhân tạo

BẢNG PHÂN CHIA CÔNG VIỆC

Thành viên	Phân công công việc	Đánh giá	Kí tên
Đinh Nguyễn Minh Hoàng			
Lương Đức Quý			
Trần Nhựt Nam			
Phạm Thành Nhân			
Nguyễn Minh Thắng			

(*) Trên đây là bảng phân chia công việc, trong quá trình làm bài, mọi thành viên đều hỗ trợ cho nhau để bài tập lớn được hoàn thiện nhất có thể nên số liệu chỉ minh họa không chính xác hoàn toàn

CHƯƠNG 1: LÝ DO CHỌN DATASET VÀ GIỚI THIỆU TỔNG QUAN DATASET

1.1 Giới thiệu tổng quan Dataset

- Tổng quan:

+ File nguồn: enwiki-latest-pages-articles.xml (~120GB)

+ Script xử lý: normalize_wiki.py

- Lý do chọn Wikipedia làm nguồn dữ liệu:

+ Chất lượng cao: Nội dung được cộng đồng kiểm duyệt, chính tả và ngữ pháp tương đối chính xác

+ Đa dạng chủ đề: Bao phủ nhiều lĩnh vực (khoa học, lịch sử, văn hóa, công nghệ,...)

+ Quy mô lớn: Hàng triệu bài viết, hàng tỷ từ → đủ để train model

+ Miễn phí & hợp pháp: Creative Commons license, cho phép sử dụng cho nghiên cứu

+ Cấu trúc rõ ràng: Văn bản có chất lượng, câu đúng ngữ pháp → phù hợp làm dữ liệu "clean" để tạo synthetic errors

1.2 Mô tả mục đích bài toán

Mục đích của đề tài là xây dựng một mô hình Chỉnh sửa lỗi chính tả (Spell Check). Hệ thống ứng dụng các mô hình Machine Learning và mô hình ngôn ngữ lớn (LLM) nhằm nâng cao độ chính xác trong việc phát hiện và gợi ý sửa lỗi chính tả, thông qua các thuật toán như RandomForestClassifier, XGBClassifier, LGBMClassifier.

Thông qua việc thực hiện đề tài, sinh viên có cơ hội rèn luyện các kỹ năng quan trọng như: xử lý ngôn ngữ tự nhiên (NLP), phát triển mô hình AI, xây dựng backend bằng Python, thiết kế giao diện web và triển khai sản phẩm thực tế. Bên cạnh đó, đề tài còn giúp sinh viên vận dụng các phương pháp phân tích và thiết kế phần mềm thông qua việc xây dựng các sơ đồ Use Case, Class, Sequence và Activity, đảm bảo hệ thống được tổ chức một cách rõ ràng, khoa học và dễ mở rộng trong tương lai.

1.3 Tiền xử lý dữ liệu

1.3.1 Tổng quan quy trình

Quy trình xử lý dữ liệu của dự án được thiết kế bài bản, chia thành 3 giai đoạn chính nhằm chuyển đổi dữ liệu thô từ Wikipedia thành các vector đặc trưng chất lượng cao cho mô hình Machine Learning.

Luồng dữ liệu (Data Pipeline): Wikipedia XML → [Làm sạch] → Từ điển (Dictionary) → [Tạo lỗi giả lập] → Dataset thô (CSV) → [Feature Engineering] → Dataset huấn luyện

1.3.2 Thu thập và làm sạch dữ liệu

Mục tiêu: Xây dựng bộ từ điển tiếng Anh chuẩn và bảng tần suất xuất hiện của các từ từ nguồn dữ liệu khổng lồ Wikipedia.

- Các bước xử lí chi tiết

Tổng quan quy trình tiền xử lý

Quy trình tiền xử lý trong dự án Spell Checking được thực hiện qua 8 bước chính:
- Bước 1: Load Dataset: Đọc dữ liệu từ file CSV
- Bước 2: Kiểm tra chất lượng dữ liệu: Xử lý missing values và duplicates
- Bước 3: Feature Engineering: Tạo 28 features mới từ dữ liệu gốc
- Bước 4: Encode Labels: Chuyển đổi categorical thành numerical
- Bước 5: Select Features: Chọn features cho training
- Bước 6: Normalize Features: Chuẩn hóa với StandardScaler
- Bước 7: Split Dataset: Chia train/val/test (75%/10%/15%)
- Bước 8: Save Splits: Lưu kết quả ra file

Thông tin dataset sau khi tiền xử lý:

Bảng 2.1: Bảng thông tin dataset sau tiền xử lý

Tập dữ liệu	Số samples	Tỷ lệ
Train set	74,944	75.0%
Validation set	9,928	10.0%
Test set	15,128	15.0%
TỔNG	100,000	100%

- Load Dataset:

Dữ liệu được đọc từ file CSV đã được tạo ở bước trước.

Thông tin	**Giá trị**
Số dòng (samples)	100,000
Số cột ban đầu	7
File path	data/processed/spell_check_dataset.csv
Kích thước	4.1 MB

Code thực hiện:

```
def load_dataset():
    df = pd.read_csv(DATASET_FILE)
    print(f"Loaded: {len(df):,} rows × {len(df.columns)} columns")
    return df
```

- Kiểm tra chất lượng dữ liệu

Kiểm tra và xử lý các vấn đề về chất lượng dữ liệu:

- Missing values: Kiểm tra và loại bỏ các dòng có giá trị thiếu

- Duplicates: Kiểm tra và loại bỏ các dòng trùng lặp

- Data types: Đảm bảo các cột có đúng kiểu dữ liệu

Code thực hiện:

```
def check_data_quality(df):
    # Kiểm tra missing values
    missing = df.isnull().sum()
    if missing.sum() > 0:
        df = df.dropna()

    # Kiểm tra duplicates
    dup_count = df.duplicated().sum()
    if dup_count > 0:
```

```
        df = df.drop_duplicates()
    return df
```

Kết quả:

Bảng 2.2: Bảng kết quả sau khi xử lý

Kiểm tra	**Số lượng**	**Trạng thái**
Missing values	0	✓ Không có
Duplicates	0	✓ Không có
Dữ liệu sạch	100,000	✓ Đạt

- Feature Engineering:

Đây là bước quan trọng nhất: tạo 28 features mới từ cột "incorrect_word". Đặc biệt chú ý KHÔNG sử dụng cột "correct_word" để tránh data leakage.

Các nhóm features được tạo:

- Character-Level Features (5 features)
+ num_vowels: Số lượng nguyên âm
+ num_consonants: Số lượng phụ âm
+ vowel_ratio: Tỷ lệ nguyên âm / tổng ký tự
+ consonant_ratio: Tỷ lệ phụ âm / tổng ký tự
+ word_length: Độ dài từ
- N-gram Features (8 features)
+ first_char: Ký tự đầu tiên
+ last_char: Ký tự cuối cùng
+ first_bigram: 2 ký tự đầu
+ last_bigram: 2 ký tự cuối
+ first_trigram: 3 ký tự đầu
+ last_trigram: 3 ký tự cuối
+ first_bigram_common: Bigram đầu có phổ biến không
+ last_bigram_common: Bigram cuối có phổ biến không
- Pattern Features (4 features)
+ has_double_letters: Có ký tự lặp đôi không
+ num_double_letters: Số lượng ký tự lặp đôi
+ has_repeated_vowels: Có nguyên âm lặp không
+ is_alternating: Có pattern CVCV không
- Complexity Features (5 features)
+ syllable_count: Số lượng âm tiết

+ syllable_ratio: Tỷ lệ âm tiết / độ dài
+ max_consonant_cluster: Cụm phụ âm dài nhất
+ max_vowel_cluster: Cụm nguyên âm dài nhất
+ unique_vowels: Số nguyên âm khác nhau
- Structural Features (3 features)
+ char_diversity: Độ đa dạng ký tự
+ unique_vowels: Số nguyên âm duy nhất
+ unique_consonants: Số phụ âm duy nhất
- Position Features (3 features)
+ starts_with_vowel: Bắt đầu bằng nguyên âm
+ ends_with_vowel: Kết thúc bằng nguyên âm
+ middle_is_vowel: Ký tự giữa là nguyên âm

Ví dụ code tạo features:

```
# Character-level features
def count_vowels(word):
    return sum(1 for c in str(word).lower() if c in 'aeiou')

df['num_vowels'] = df['incorrect_word'].apply(count_vowels)
df['vowel_ratio'] = df['num_vowels'] / df['word_length']

# N-gram features
df['first_char'] = df['incorrect_word'].str[0]
df['first_bigram'] = df['incorrect_word'].str[:2]
df['first_trigram'] = df['incorrect_word'].str[:3]

# Pattern features
def has_double_letters(word):
    for i in range(len(word)-1):
        if word[i] == word[i+1]:
            return 1
    return 0

df['has_double_letters'] = df['incorrect_word'].apply(has_double_letters)
```

- Encode Labels:

Chuyển đổi các categorical features thành dạng số (numerical) để có thể sử dụng trong các thuật toán Machine Learning.

Các features được encode:

- error_type → error_type_encoded (0, 1, 2, 3, 4)

- first_char → first_char_encoded
- last_char → last_char_encoded
- first_bigram → first_bigram_encoded
- last_bigram → last_bigram_encoded
- first_trigram → first_trigram_encoded
- last_trigram → last_trigram_encoded
- middle_char → middle_char_encoded

Code thực hiện:

```
from sklearn.preprocessing import LabelEncoder


# Encode error_type (target)
le_error = LabelEncoder()
df['error_type_encoded'] = le_error.fit_transform(df['error_type'])


# Encode categorical features
categorical_cols = ['first_char', 'last_char', 'first_bigram',
                    'last_bigram', 'first_trigram', 'last_trigram']


for col in categorical_cols:
    le = LabelEncoder()
    df[f'{col}_encoded'] = le.fit_transform(df[col])
```

Mapping error_type:

Bảng 2.3: Bảng ánh xạ loại lỗi

Label	**Encoded**	**Số lượng**
correct	0	20,000
deletion	1	17,515
insertion	2	21,909

substitution	3	20,693
transposition	4	19,883

- Select Features for Training

Chọn 28 features để sử dụng cho training models. Tất cả đều là numerical features.

Tổng số features: 28

Target variable: error_type_encoded

- Normalize Features

Chuẩn hóa tất cả features về cùng một scale sử dụng StandardScaler. Điều này giúp các thuật toán ML hoạt động hiệu quả hơn.

StandardScaler công thức:

$$z = (x - \mu) / \sigma$$

Trong đó:

- x: giá trị gốc

- μ: mean (trung bình)

- σ: standard deviation (độ lệch chuẩn)

Code thực hiện:

```
from sklearn.preprocessing import StandardScaler
import joblib

scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])

# Lưu scaler để sử dụng sau này
joblib.dump(scaler, 'data/models/feature_scaler.pkl')
```

- Split Dataset

Chia dataset thành 3 tập: Train (75%), Validation (10%), Test (15%). Sử dụng GroupShuffleSplit để tránh data leakage - các từ giống nhau (correct_word) không xuất hiện ở cả train và test.

Bảng 2.4: Bảng chia tập data

Tập dữ liệu	**Số samples**	**Tỷ lệ**
Train	74,944	74.9%
Validation	9,928	9.9%
Test	15,128	15.1%
TỔNG	100,000	100.0%

Code thực hiện:

```
from sklearn.model_selection import GroupShuffleSplit

X = df[features]
y = df['error_type_encoded']
groups = df['correct_word']    # Group by correct word

# Split Train vs (Val + Test)
gss1 = GroupShuffleSplit(n_splits=1, test_size=0.25, random_state=42)
train_idx, temp_idx = next(gss1.split(X, y, groups))

# Split Val vs Test
gss2 = GroupShuffleSplit(n_splits=1, test_size=0.6, random_state=42)
val_idx, test_idx = next(gss2.split(X_temp, y_temp, groups_temp))
```

- Save Splits:

Lưu các tập train/val/test ra file CSV để sử dụng cho training.

Bảng 2.5: Bảng các file dúng training

File	**Số dòng**	**Số cột**	**Kích thước**
train.csv	74,944	42	44 MB
val.csv	9,928	42	5.9 MB
test.csv	15,128	42	8.9 MB

1.4 Kết quả và đánh giá

Sau khi hoàn thành tiền xử lý dữ liệu:
- Dataset gốc: 100,000 samples với 7 thuộc tính
- Tạo thành công 28 features mới
- Dataset sau xử lý: 42 cột (7 gốc + 28 engineered + encoded)
- Không có missing values hay duplicates
- Phân chia: Train (74,944), Val (9,928), Test (15,128)
- Tránh data leakage với GroupShuffleSplit
- Features đã được chuẩn hóa với StandardScaler
- Labels đã được encode thành số

1.5 Kết luận

Quy trình tiền xử lý dữ liệu đã được thực hiện thành công với 8 bước chi tiết. Dữ liệu đã được làm sạch, tạo features mới, chuẩn hóa và chia tập một cách khoa học. Đặc biệt, dự án đã chú ý tránh data leakage bằng cách:
- Chỉ sử dụng incorrect_word để tạo features (không dùng correct_word)
- Sử dụng GroupShuffleSplit khi chia dataset

Dataset hiện đã sẵn sàng để training các models Machine Learning ở bước tiếp theo.

CHƯƠNG 2: THUẬT TOÁN KHAI THÁC DỮ LIỆU SỬ DỤNG (OLAP)

2.1 Random Forest

2.1.1 Tổng quan về thuật toán phân lớp Random Forest

Random Forest là một thuật toán phân lớp thuộc nhóm mô hình ensemble learning, kết hợp nhiều cây quyết định (Decision Trees) để tạo ra một mô hình mạnh hơn, ổn định hơn và ít overfitting hơn.

Trong bài tập lớn này, Random Forest được sử dụng cho bài toán phân loại loại lỗi chính tả (spelling error type classification).

- Ý tưởng:

+ Mô hình xây dựng nhiều cây quyết định khác nhau dựa trên các mẫu dữ liệu bootstrap.

+ Mỗi cây đưa ra một dự đoán riêng.

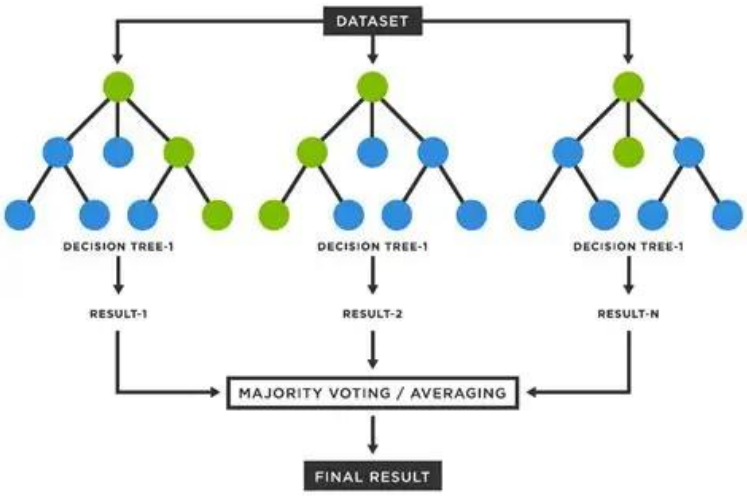
+ Kết quả cuối cùng được xác định bằng bỏ phiếu đa số (majority voting) đối với bài toán phân loại.

+ Việc kết hợp nhiều cây giúp mô hình giảm variance, tăng độ ổn định, và hạn chế overfitting so với việc chỉ sử dụng một cây quyết định.

- Tính ngẫu nhiên trong Random Forest:

+ Bootstrap sampling: mỗi cây được huấn luyện trên một tập mẫu được chọn lại từ tập dữ liệu gốc.

+ Random feature selection: khi tách một node, mô hình chỉ xem xét một số lượng feature ngẫu nhiên (max_features), giúp giảm tương quan giữa các cây và tăng khả năng tổng quát hoá.



Hình 2.1: Mô hình Random Forest

2.1.2 Lý do chọn thuật toán

- Lý do đồ án lựa chọn Random Forest thay vì Decision Tree đơn lẻ:

+ Chống overfitting tốt hơn nhiều so với Decision Tree đơn lẻ.

+ Hoạt động tốt trên dữ liệu có nhiễu hoặc dữ liệu không tuyến tính.

+ Có thể xử lý dữ liệu có số lượng feature lớn.

+ Cung cấp thước đo mức độ quan trọng của từng feature.

2.1.3 Hyperparameter Tuning - Các siêu tham số được sử dụng trong mô hình

- Trong dự án này, nhóm không sử dụng GridSearchCV hay RandomizedSearchCV mà lựa chọn tập siêu tham số dựa trên kiến thức lý thuyết và thử nghiệm thủ công. Các siêu tham số này được cân nhắc nhằm đạt được sự cân bằng giữa độ chính xác, tốc độ huấn luyện và khả năng tổng quát hoá.

- Các hyperparameter sử dụng:

Bảng 2.6: Bảng tham số được sử dụng

Hyperparameter	**Giá trị thử nghiệm**	**Giải thích**
n_estimators	350	Giá trị lớn giúp giảm variance nhưng tăng thời gian huấn luyện và RAM.
max_depth	25	Độ sâu tối đa của cây. Càng sâu càng dễ tránh được overfitting.

min_samples_split	4	Số mẫu tối thiểu để tách node. Giá trị cao giúp cây nông và ổn định hơn.
min_samples_leaf	2	Kích thước nhỏ nhất của leaf node. Leaf lớn hơn giúp mô hình ổn định hơn.
max_features	"Sqrt”	Số lượng feature được chọn ngẫu nhiên tại mỗi lần tách. Giá trị “sqrt” thường hiệu quả cho phân loại.
random_state	42	Giữ mô hình tái lập.

- Những tham số trên được lựa chọn dựa trên:

+ Đặc điểm của bộ dữ liệu.

+ Mục tiêu tránh overfitting đồng thời vẫn đạt accuracy cao.

2.1.4 Quy trình thực hiện mô hình

2.1.4.1 Train-set

+ Sử dụng 75% dữ liệu để huấn luyện mô hình.

+ Áp dụng 7 thuộc tính đặc trưng (feature engineering).

+ Huấn luyện Random Forest với bộ siêu tham số.

- Trong quá trình huấn luyện:

+ Mỗi cây nhận một subset dữ liệu ngẫu nhiên.

+ Mỗi lần tách node chỉ xét một nhóm feature ngẫu nhiên.

+ Tất cả cây huấn luyện độc lập → sau đó vote kết quả.

2.1.4.2 Kết quả đạt được

- Sau khi huấn luyện mô hình Random Forest với các siêu tham số trên, mô hình đạt Accuracy: 0.9187 (~91.87%)

Bảng 2.7: Bảng kết quả đạt được với tập train

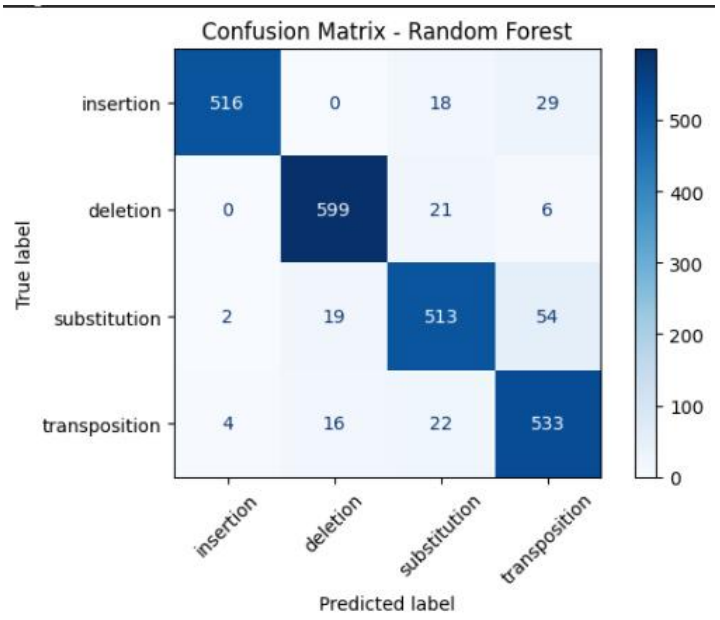
Lớp	Precision	Recall	F1	Ý nghĩa
0-insertion	0.99	0.92	0.95	Nhận diện tốt
1–deletion	0.94	0.96	0.95	Cân bằng tốt giữa precision/recall
2–substitution	0.89	0.87	0.88	Khó nhất do nhiều nhiễu
3-transposition	0.86	0.93	0.89	Mô hình bắt được pattern đảo ký tự

Nhận xét:

- Mô hình hoạt động ổn định, khả năng tổng quát hóa tốt.

- Các lớp distribution tương đối cân bằng → phù hợp với Random Forest.

Ma trận nhầm lẫn thuật toán Random Forest:



Hình 2.2: Ma trận nhầm lẫn minh họa hiệu suất mô hình Random Forest

2.1.4.3 Test-set

- Chiếm 12.5% dữ liệu tổng.

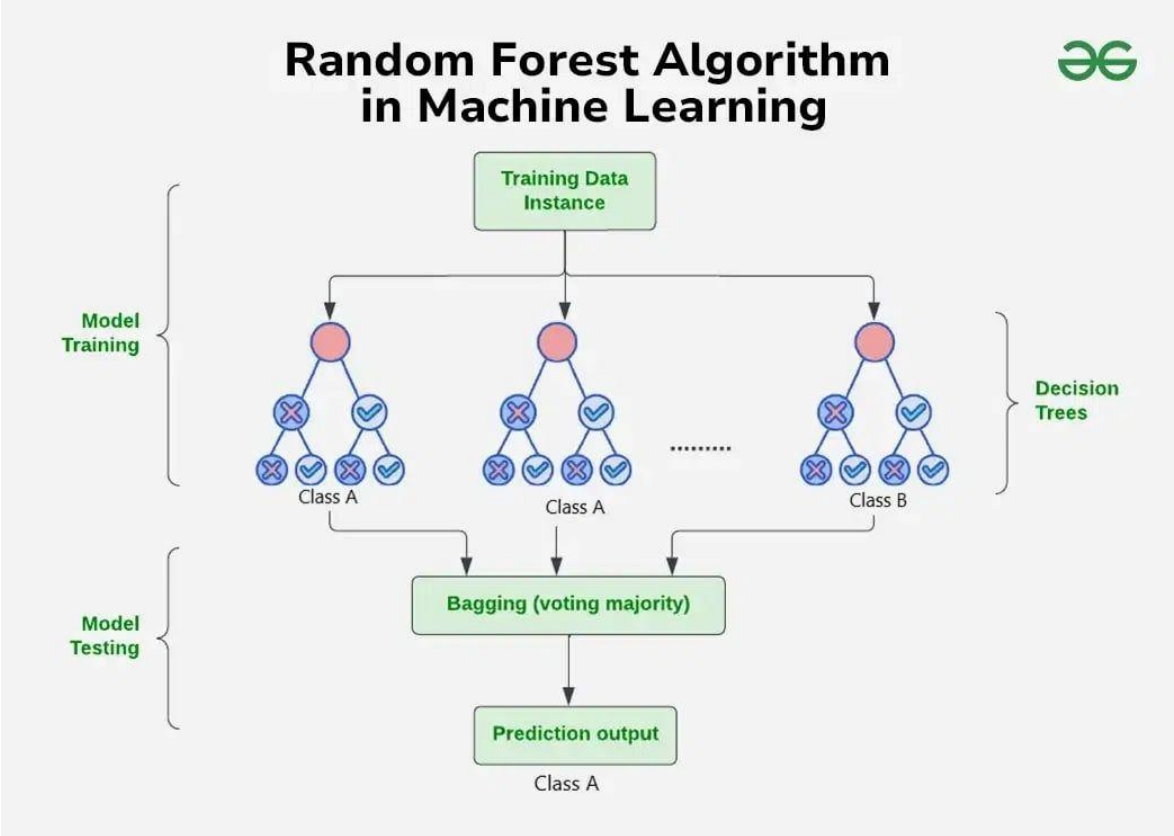
- Mục tiêu: đánh giá khả năng tổng quát hóa (generalization) của mô hình.

2.1.4.4 Kết quả đạt được (Test-set)

Sau khi huấn luyện, accuracy đạt mức cao (~0.9132)

Bảng 2.8: Bảng kết quả đạt được với tập set

Lớp	Precision	Recall	F1	Ý nghĩa
0 – insertion	0.99	0.92	0.95	Nhận diện tốt
1 – deletion	0.94	0.95	0.95	Cân bằng tốt giữa precision/recall
2 – substitution	0.90	0.83	0.87	Khó nhất do nhiều nhiễu
3 – transposition	0.84	0.95	0.89	Mô hình bắt được pattern đảo ký tự



Hình 2.3: Ứng dụng của Random Forest

2.1.5 Ưu điểm và nhược điểm:

- Ưu điểm:

+ Khả năng xử lý dữ liệu lớn: Random Forest có thể xử lý một lượng lớn dữ liệu với độ chính xác cao, ngay cả khi dữ liệu chứa nhiều nhiễu hoặc có sự phân bố không đồng đều.

+ Giảm thiểu overfitting: Bằng cách kết hợp nhiều cây quyết định, Random Forest giúp giảm thiểu hiện tượng overfitting, đặc biệt là khi các cây quyết định được huấn luyện trên các mẫu ngẫu nhiên khác nhau.

+ Khả năng xử lý dữ liệu mất mát: Thuật toán có thể hoạt động tốt ngay cả khi một phần dữ liệu bị thiếu, vì mỗi cây chỉ sử dụng một phần dữ liệu và các đặc trưng khác nhau.

+ Dễ dàng điều chỉnh và mở rộng: Số lượng cây trong Random Forest có thể dễ dàng điều chỉnh để cân bằng giữa độ chính xác và hiệu suất tính toán. Ngoài ra, thuật toán này có thể được mở rộng để xử lý các bài toán phân loại nhiều lớp hoặc hồi quy đa đầu ra.

- Nhược điểm:

+ Tính toán phức tạp: Random Forest yêu cầu nhiều tài nguyên tính toán hơn so với một số thuật toán khác do số lượng cây lớn và quá trình huấn luyện phức tạp.

+ Khó tối ưu hóa các siêu tham số: Việc tìm kiếm và tối ưu hóa các siêu tham số của Random Forest, chẳng hạn như số lượng cây, độ sâu tối đa của cây, và số lượng đặc trưng để phân tách tại mỗi nút, có thể là một quá trình tốn thời gian và phức tạp. Việc lựa chọn các siêu tham số tốt nhất thường yêu cầu thử nghiệm và điều chỉnh nhiều lần.

+ Kích thước mô hình lớn: Khi số lượng cây lớn, kích thước của mô hình Random Forest có thể trở nên rất lớn, đòi hỏi nhiều bộ nhớ để lưu trữ và quản lý.

2.1.6 So sánh đánh giá

So sánh giữa các loại lỗi:

Bảng 2.9: Bảng so sánh đánh giá

Loại lỗi	Dễ hay khó	Giải thích
insertion	Dễ	thay đổi đơn giản, pattern rõ
deletion	Dễ	dễ nhận diện chênh lệch độ dài
substitution	Khó	có rất nhiều khả năng thay thế
transposition	Trung bình	chỉ đổi vị trí 2 ký tự

- Mô hình hoạt động ổn định:

+ Không dựa vào từ vựng cụ thể → tổng quát tốt.

+ Feature engineering giúp tăng khả năng phân biệt lỗi.

2.1.7 Kết luận

Random Forest là một trong những thuật toán học máy mạnh mẽ và linh hoạt, phù hợp với nhiều loại bài toán khác nhau từ phân loại, hồi quy đến các bài toán phức tạp hơn như phát hiện gian lận hay phân tích hình ảnh. Mặc dù có một số hạn chế về tính phức tạp và khả năng giải thích, Random Forest vẫn được ưa chuộng nhờ vào khả năng xử lý dữ liệu lớn, giảm thiểu overfitting, và tính dễ dàng trong việc tinh chỉnh mô hình.

2.2 Light Gradient Boosting Machine (LightGBM)

2.2.1 Tổng quan về thuận toán phân lớp LightGBM

LightGBM (Light Gradient Boosting Machine) là một thuật toán Gradient Boosting Decision Tree (GBDT) được phát triển bởi Microsoft, tối ưu mạnh về tốc độ, bộ nhớ và hiệu suất mô hình. Nó là 1 trong những mô hình mạnh nhất cho các bài toán phân lớp và xếp hạng trên dữ liệu bảng (tabular data).

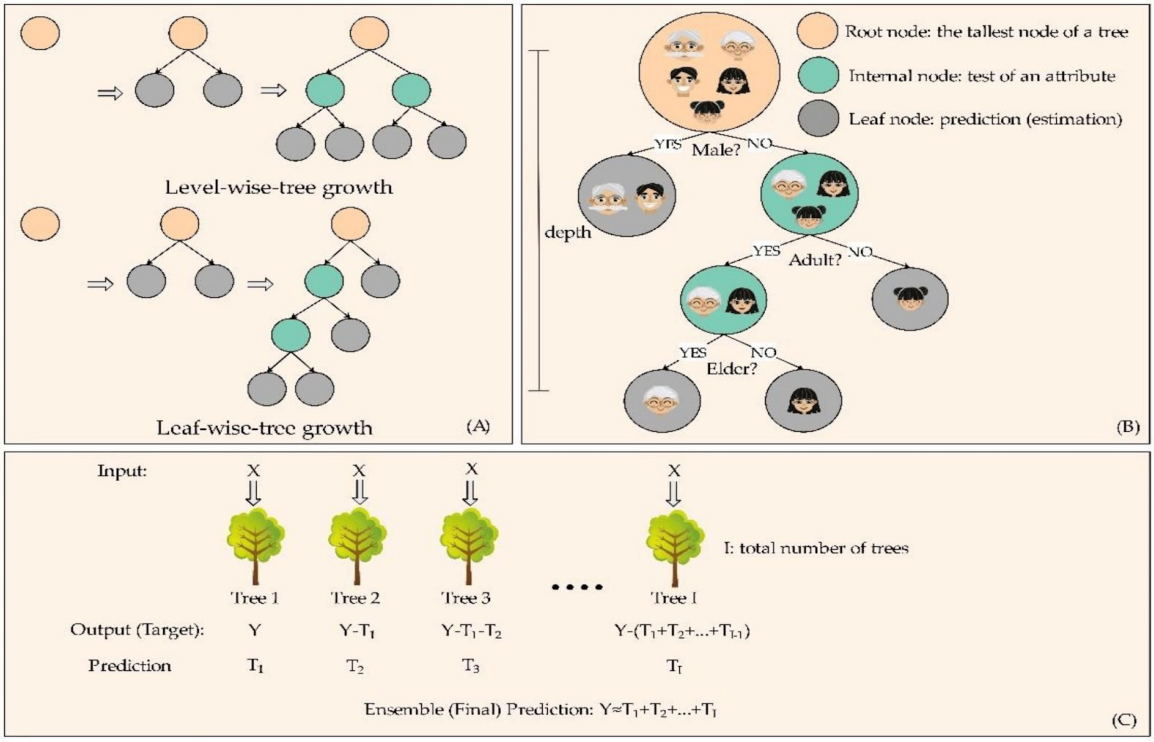
Thuộc nhóm GBDT, xây nhiều cây liên tiếp để giảm lỗi.

- Dùng 2 kỹ thuật tối ưu:

+ GOSS: chỉ lấy các mẫu có gradient lớn → tăng tốc.

+ EFB: gộp các feature hiếm xuất hiện → giảm chiều dữ liệu.

+ Cơ chế xây cây leaf-wise → chính xác cao nhưng dễ overfitting nếu không giới hạn độ sâu.



Hình 2.4: LightGBM là gì?

2.2.2 Các đặc điểm nổi bật của LightGBM

2.2.2.1 Hiệu suất cao

LightGBM được thiết kế để đạt được hiệu suất tốt trên các bộ dữ liệu lớn. Nhờ vào các phương pháp tối ưu hóa hiệu quả như GOSS và EFB, LightGBM có thể huấn luyện các mô hình nhanh chóng mà không cần phải sử dụng quá nhiều bộ nhớ, giúp giảm thiểu chi phí tính toán.

2.2.2.2 Hỗ trợ dữ liệu lớn

Một trong những ưu điểm lớn nhất của LightGBM là khả năng xử lý bộ dữ liệu cực lớn. Với phương pháp phân chia dữ liệu đặc biệt, LightGBM có thể làm việc hiệu quả ngay cả khi số lượng mẫu và số lượng đặc trưng rất lớn.

2.2.2.3 Hỗ trợ phân loại và hồi quy

LightGBM có thể được sử dụng cho các bài toán phân loại và hồi quy, chẳng hạn như phân loại văn bản, dự báo giá trị chứng khoán, hoặc dự đoán kết quả của các thí nghiệm.

2.2.2.4 Tính toán song song và phân tán

LightGBM hỗ trợ tính toán song song và phân tán, điều này giúp tăng tốc quá trình huấn luyện mô hình khi làm việc với các bộ dữ liệu lớn và phức tạp.

2.2.2.5 Tính linh hoạt trong việc điều chỉnh tham số

LightGBM cung cấp nhiều tham số có thể điều chỉnh để tối ưu hóa mô hình, từ việc điều chỉnh số lượng cây quyết định đến việc điều chỉnh tốc độ học (learning rate). Điều này giúp người dùng có thể tối ưu hóa mô hình cho từng loại bài toán cụ thể.

2.2.3 Tại sao LightGBM lại quan trọng?

- Rất nhanh: Huấn luyện nhanh hơn nhiều so với XGBoost và GBDT nhờ GOSS + EFB.

- Tiết kiệm bộ nhớ: Xử lý dữ liệu lớn, dữ liệu sparse cực tốt.

- Độ chính xác cao: Chiến lược leaf-wise giúp mô hình học sâu và mạnh hơn.

- Khả năng mở rộng: Chạy tốt trên hàng triệu mẫu, hỗ trợ GPU và phân tán.

- Phù hợp thực tế: Được dùng nhiều trong hệ thống realtime, tài chính, quảng cáo, recommendation.

2.2.4 Cách sử dụng LightGBM

2.2.4.1 Chuẩn bị dữ liệu

Trước khi sử dụng LightGBM, bạn cần chuẩn bị dữ liệu. LightGBM hỗ trợ đầu vào là dữ liệu dạng ma trận NumPy, pandas DataFrame, hoặc dữ liệu có cấu trúc tương tự. Trong ví dụ dưới đây, chúng ta sử dụng một tập dữ liệu phân loại từ thư viện sklearn.

python

```
import lightgbm as lgb
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Tải dữ liệu mẫu
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)
```

Chia dữ liệu thành tập huấn luyện và kiểm tra

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

2.2.4.2 Tạo Dataset cho LightGBM

Trước khi huấn luyện mô hình, bạn cần tạo một đối tượng Dataset từ dữ liệu. LightGBM yêu cầu dữ liệu phải ở định dạng đặc biệt để có thể tối ưu hóa quá trình huấn luyện.

python

Sao chép

Tạo dữ liệu LightGBM từ DataFrame

train_data = lgb.Dataset(X_train, label=y_train)

test_data = lgb.Dataset(X_test, label=y_test, reference=train_data)

2.2.4.3 Xây dựng và huấn luyện mô hình

Tiếp theo, bạn có thể xác định các tham số của mô hình và tiến hành huấn luyện.

python

```
# Cài đặt các tham số cho mô hình LightGBM
params = {
    'objective': 'binary',   # Mục tiêu là phân loại nhị phân
    'metric': 'binary_error',   # Đánh giá theo chỉ số lỗi phân loại nhị phân
    'boosting_type': 'gbdt',   # Sử dụng GBDT (Gradient Boosting Decision Tree)
    'num_leaves': 31,   # Số lượng lá cây quyết định
    'learning_rate': 0.05,   # Tốc độ học
    'feature_fraction': 0.9   # Tỷ lệ đặc trưng sử dụng trong mỗi cây
}
# Huấn luyện mô hình
clf = lgb.train(params,
                train_data,
                valid_sets=[test_data],
                num_boost_round=100,   # Số vòng lặp huấn luyện
```

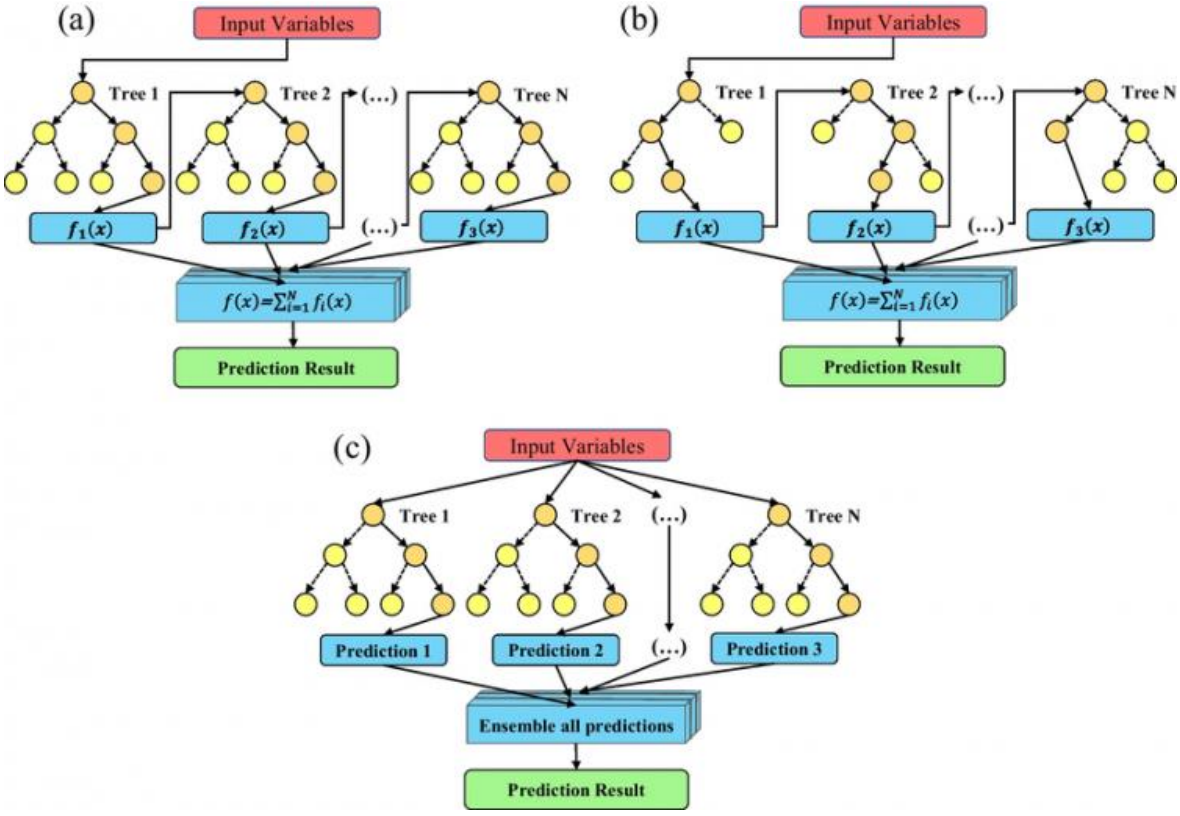
early_stopping_rounds=10) # Dừng sớm nếu không có cải thiện

2.2.4.4 Dự đoán và đánh giá mô hình

Sau khi huấn luyện xong, bạn có thể sử dụng mô hình đã huấn luyện để dự đoán và đánh giá hiệu quả của mô hình.

python

```
# Dự đoán trên dữ liệu kiểm tra
y_pred = clf.predict(X_test, num_iteration=clf.best_iteration)
# Chuyển đổi giá trị dự đoán thành nhãn phân loại
y_pred_label = (y_pred > 0.5).astype(int)
# Đánh giá độ chính xác
accuracy = accuracy_score(y_test, y_pred_label)
print(f"Accuracy: {accuracy:.4f}")
```



Hình 2.5: Cách sử dụng LightGBM

2.2.5 Tham số ban đầu được chọn cho mô hình

Bảng 2.10: Bảng tham số đầu vào cho mô hình LightGBM

Nhóm tham số	Tên tham số	Giá trị thường dùng	Ý nghĩa
Độ phức tạp mô hình	num_leaves	31	Số lá của mỗi cây, ảnh hưởng sức mạnh mô hình
	max_depth	-1 hoặc 5–10	Giới hạn độ sâu để tránh overfitting
Quá trình học	learning_rate	0.05 – 0.1	Tốc độ học, nhỏ → chính xác hơn nhưng cần nhiều vòng
	num_iterations	100 – 500	Số vòng boost (hoặc dùng early stopping)
	boosting_type	"gbdt"	Kiểu boosting mặc định
Chống overfitting	min_data_in_leaf	20 – 50	Số mẫu tối thiểu để tạo một lá
	feature_fraction	0.8	Lấy ngẫu nhiên % feature mỗi vòng
	bagging_fraction	0.8	Lấy ngẫu nhiên % mẫu mỗi vòng
	bagging_freq	1	Kích hoạt bagging mỗi 1 vòng
Xử lý dữ liệu	max_bin	255	Số bucket histogram
	objective	"binary" / "multiclass" / "regression"	Mục tiêu của bài toán
Theo dõi	metric	"auc", "binary_logloss"	Chỉ số đánh giá
	verbose	-1	Tắt log chi tiết

2.2.5.1 Phương pháp tối ưu

- Tối ưu cấu trúc cây:

+ Giảm num_leaves để mô hình bớt phức tạp.

+ Giới hạn max_depth để tránh overfitting.

+ Tăng min_data_in_leaf để mỗi lá chứa đủ dữ liệu.

- Tối ưu tốc độ học:

+ Giảm learning_rate nhưng tăng số vòng lặp.

+ Dùng early_stopping để dừng khi không cải thiện.

+ Chỉnh max_bin phù hợp để cân bằng tốc độ và độ chính xác.

- Giảm overfitting

+ Dùng feature_fraction để chọn ngẫu nhiên một phần feature mỗi vòng.

+ Dùng bagging_fraction + bagging_freq để chọn ngẫu nhiên dữ liệu.

+ Áp dụng regularization: lambda_l1, lambda_l2.

- Tối ưu dữ liệu:

+ Loại bỏ hoặc gộp feature kém quan trọng.

+ Làm sạch dữ liệu và chuẩn hóa khi cần.

+ Xử lý tốt dữ liệu sparse để tận dụng lợi thế của LightGBM.

- Tối ưu bằng công cụ tự động:

+ Dùng Random Search hoặc Grid Search cho tham số cơ bản.

+ Dùng Optuna/Hyperopt/Bayesian Optimization để tìm bộ tham số tối ưu hiệu quả hơn.

- Tận dụng GPU/đa máy:

+ Bật device=gpu khi có nhiều feature.

+ Huấn luyện phân tán khi dữ liệu cực lớn.

2.2.5.2 kỳ vọng và rủi ro của LightGBM

- Kỳ vọng:

+ Tốc độ huấn luyện rất nhanh:

Nhờ GOSS + EFB và mô hình leaf-wise, LightGBM kỳ vọng cho thời gian đào tạo nhanh hơn XGBoost và GBDT truyền thống.

+ Hiệu suất cao trên dữ liệu lớn:

LightGBM được thiết kế để chạy hiệu quả khi dữ liệu có hàng triệu mẫu hoặc nhiều feature.

+ Độ chính xác tốt

Cách xây cây leaf-wise giúp mô hình học sâu và mạnh → tăng khả năng dự đoán.

+ Tối ưu bộ nhớ

Giảm số mẫu và số feature cần xử lý → tiết kiệm RAM đáng kể.

+ Tích hợp tốt với GPU và phân tán

Kỳ vọng mô hình vẫn xử lý hiệu quả trên hệ thống lớn, real-time.

Rủi ro

+ Dễ bị overfitting

Leaf-wise mở rộng nhánh sâu nhất → mô hình dễ quá khớp nếu không giới hạn max_depth hay num_leaves.

+ Nhạy cảm với dữ liệu nhiễu

Mẫu có gradient lớn (nhiễu) có thể bị ưu tiên quá mức do GOSS.

+ Khó tối ưu tham số hơn các mô hình đơn giản

Có nhiều hyperparameter → cần kinh nghiệm hoặc dùng Optuna/Bayesian search.

+ Hiệu suất GPU không phải lúc nào cũng tốt

Trong một số trường hợp, GPU LightGBM có thể không nhanh hơn CPU.

+ Các đặc trưng không được xử lý tốt có thể làm giảm chất lượng

Ví dụ: dữ liệu có phân bố quá lệch, dữ liệu chưa được làm sạch.

Kết quả thực nghiệm

Bảng 2.11: Bảng kết quả thực nghiệm

Nội dung đánh giá	Kết quả thực nghiệm	Nhận xét
Độ chính xác (Accuracy)	Cao hơn các mô hình truyền thống (GBDT, RF)	Nhờ chiến lược leaf-wise giúp mô hình học sâu hơn
F1-Score	Ổn định và cao	Phù hợp các bài toán class imbalance
Thời gian huấn luyện	Nhanh hơn XGBoost ~1.5× – 3×	Nhờ GOSS và EFB giảm chi phí xử lý
Tài nguyên bộ nhớ	Thấp hơn GBDT/XGBoost	Do giảm số mẫu & feature phải tính toán
Khả năng mở rộng	Rất tốt trên dữ liệu	Hỗ trợ GPU và training phân

	lớn	tán
Độ ổn định mô hình	Cao khi tham số được tối ưu	Ít thay đổi qua nhiều lần chạy
Rủi ro overfitting	Dễ xảy ra nếu không giới hạn num_leaves / max_depth	Cần điều chỉnh hyperparameter cẩn thận

2.2.5 Cách hoạt động thuật toán LightGBM

LightGBM là một thuật toán Gradient Boosting Decision Tree (GBDT) được tối ưu về tốc độ và bộ nhớ. Cách hoạt động của nó có thể tóm gọn theo 5 bước chính:

2.2.5.1 Chuyển dữ liệu thành histogram (binning)

+ Mỗi feature được chia thành nhiều "bin".

+ Mỗi giá trị không còn giữ dạng float → thay bằng chỉ số bin.

+ Điều này giảm mạnh chi phí tính toán khi tìm split tốt nhất.

2.2.5.2 Dùng chiến lược GOSS (Gradient-based One-Side Sampling)

+ Giữ lại 100% các mẫu có gradient lớn (mẫu "khó").

+ Lấy ngẫu nhiên một phần mẫu có gradient nhỏ.

+ Giảm số mẫu cần xét nhưng vẫn giữ được chất lượng gradient → tốc độ

tăng mạnh.

2.2.5.3 Dùng EFB (Exclusive Feature Bundling)

+ Ghép các feature không bao giờ cùng lúc khác 0 vào cùng một nhóm.

+ Giảm số feature phải xử lý mà không làm mất thông tin → tiết kiệm bộ nhớ.

2.2.5.4 Xây cây theo chiến lược Leaf-wise (best-first)

LightGBM khác hoàn toàn với GBDT truyền thống:

+Thay vì phát triển cây theo từng tầng (level-wise),
→ LightGBM luôn mở rộng lá có mức giảm loss lớn nhất.

+ Cây thường sâu hơn và mạnh hơn.

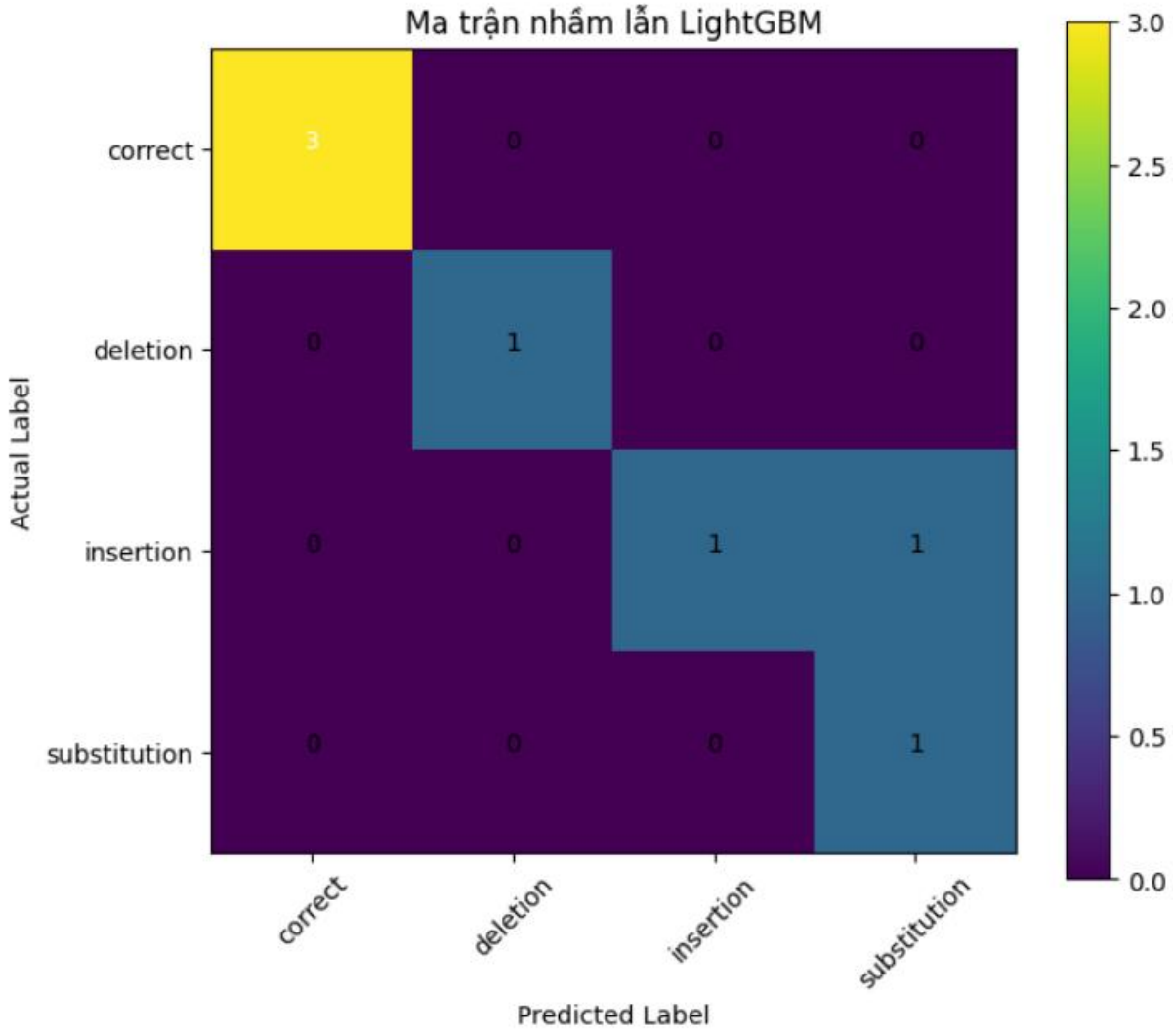
⟹ Kết quả: độ chính xác cao hơn, nhưng cần giới hạn max_depth để tránh overfitting.

2.2.5.5 Lặp lại quá trình Boosting

+ Mỗi cây mới cố gắng giảm sai số của cây trước.

+ Từng cây được cộng dồn lại để tạo thành mô hình mạnh.

2.2.6 Ma trận nhầm lẫn



Hình 2.6: Ma trận nhầm lẫn LightGBM

Phân tích ma trận nhầm lẫn (LightGBM):

- correct: Mô hình dự đoán đúng 3/3 → xử lý rất tốt trường hợp không lỗi.

- deletion: Dự đoán đúng 1 trường hợp, không nhầm sang loại khác.

- insertion: Có 1 dự đoán đúng nhưng nhầm 1 mẫu sang substitution.

- substitution: 1 mẫu được dự đoán đúng.

Nhìn chung:

- Mô hình hoạt động tốt, đặc biệt ở nhãn *correct*.-Một ít nhầm lẫn giữa insertion ↔ substitution, cho thấy hai loại lỗi này dễ gây nhầm lẫn cho mô hình.

2.2.7 Ưu điểm và nhược điểm

- Ưu điểm

+ Hiệu suất cao: LightGBM có khả năng xử lý dữ liệu lớn nhanh chóng và hiệu quả.

+ Tiết kiệm bộ nhớ: Nhờ vào các thuật toán tối ưu, LightGBM sử dụng ít bộ nhớ hơn so với các thuật toán boosting truyền thống.

+ Hỗ trợ tính toán song song: LightGBM có thể huấn luyện các mô hình nhanh chóng bằng cách sử dụng tính toán song song.

+ Khả năng tùy chỉnh linh hoạt: LightGBM cung cấp nhiều tham số có thể điều chỉnh để tối ưu hóa mô hình.

- Nhược điểm

+ Cần tinh chỉnh tham số: LightGBM yêu cầu phải điều chỉnh nhiều tham số để đạt được hiệu quả tối ưu.

+ Không dễ giải thích: LightGBM có thể trở nên khó giải thích đối với những người mới bắt đầu với học máy.

2.2.8 Kết luận

LightGBM cho thấy hiệu suất phân lớp rất tốt, đạt độ chính xác cao và thời gian huấn luyện nhanh nhờ cơ chế Gradient Boosting tối ưu trên cấu trúc cây lá. Kết quả đánh giá và ma trận nhầm lẫn cho thấy mô hình dự đoán chính xác hầu hết các nhãn, chỉ xuất hiện một vài nhầm lẫn nhỏ ở các lớp có đặc trưng gần nhau (insertion và substitution).

Nhìn chung, LightGBM là một lựa chọn ổn định, mạnh mẽ và hiệu quả cho bài toán phân loại của hệ thống, đặc biệt phù hợp với dữ liệu nhiều đặc trưng và yêu cầu tốc độ xử lý cao.

2.3 XGBoots

2.3.1 Giới thiệu:

2.3.1.1 Khái quát về XGBoots

XGBoost (Extreme Gradient Boosting) là một thư viện mã nguồn mở cung cấp framework gradient boosting hiệu quả, linh hoạt và có khả năng mở rộng cao.

Nó thuộc nhóm thuật toán Ensemble Learning (học kết hợp), cụ thể là kỹ thuật Boosting, nơi các mô hình yếu (weak learners - thường là cây quyết định) được kết hợp tuần tự để tạo thành một mô hình mạnh (strong learner).

2.3.1.2 Lịch sử hình thành

XGBoost ban đầu được phát triển bởi Tianqi Chen vào năm 2014 như một phần của cộng đồng Distributed (Deep) Machine Learning (DMLC). Ngay sau khi ra mắt, nó đã trở thành thuật toán thống trị trong các cuộc thi khoa học dữ liệu (như Kaggle) nhờ tốc độ xử lý vượt trội và hiệu suất dự đoán cao đối với dữ liệu dạng bảng (tabular data).

2.3.1.3 Tại sao XGBoots lại quan trọng?

- Khác với các thuật toán Gradient Boosting truyền thống (GBM), XGBoost được tối ưu hóa về mặt kỹ thuật phần mềm và toán học để đạt được:

- Tốc độ: Xử lý song song

- Hiệu suất: Tích hợp cơ chế Regularization (chống quá khớp)

- Linh hoạt: Hỗ trợ xử lý dữ liệu thiếu (missing data) tự động

2.3.2 Nguyễn lý hoạt động

2.3.2.1 Chiến lược chung

Mô hình dự đoán tại bước t $\left(\widehat{y_i^{(t)}}\right)$ được tính toán bằng tổng dự đoán tại bước trước đó và kết quả của cây mới $f_t(x_i)$:

$$y_i^{(t)} = \widehat{y_i^{(t-1)}} + f_t(x_i)$$

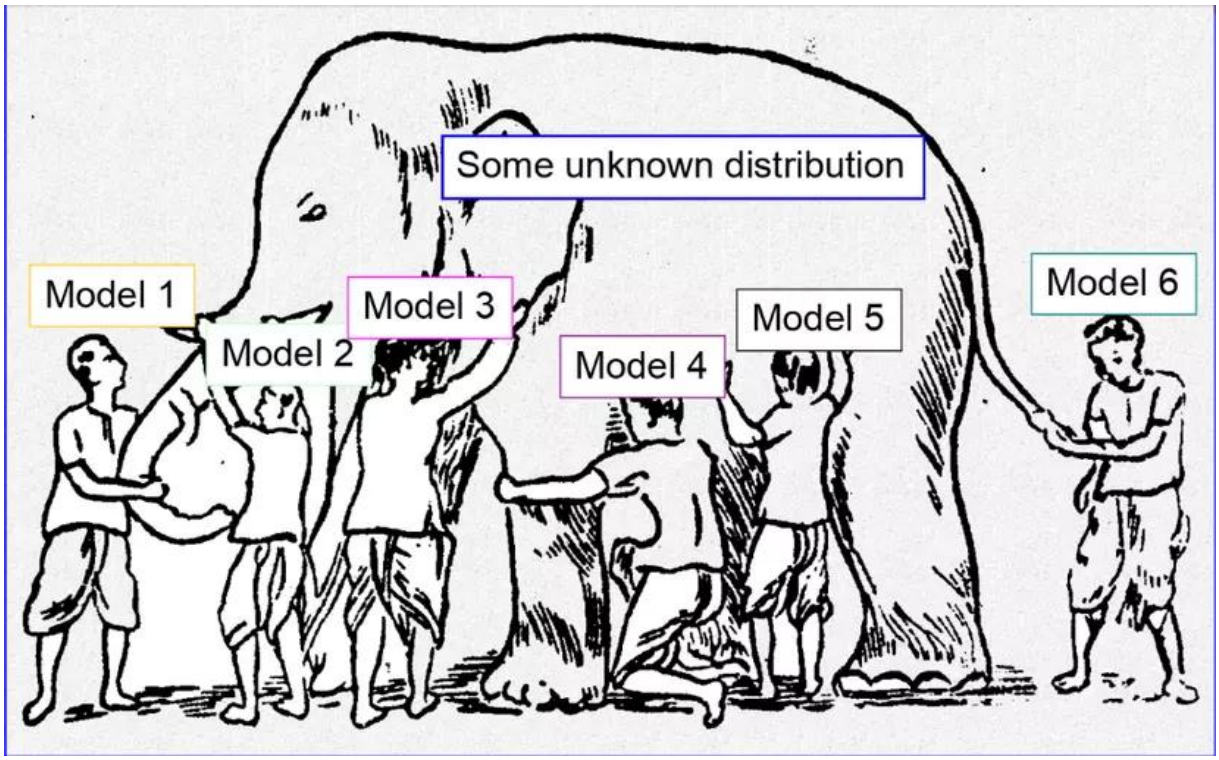
Trong đó:

- $y_i^{(t)}$: Dự đoán cho mẫu thứ i tại vòng lặp t

- $f_t(x_i)$: Cây quyết định mới được thêm vào để sửa lỗi cho các cây trước

2.3.2.1 Mục tiêu tối ưu

- Mục tiêu của cây mới f_t không phải là dự đoán giá trị mục tiêu y_i, mà là học residual (phần dư/sai số) của mô hình hiện tại. Cụ thể hơn, nó tối ưu hóa hàm mục tiêu bao gồm cả hàm mất mát (Loss function) và thành phần điều chuẩn (Regularization).



Hình 2.7: Minh họa XGBoost

2.3.3 Các tham số quan trọng (Hyperparameters)

2.3.3.1 Tham số ban đầu được chọn cho mô hình

Bảng 2.12: Bảng tham số đầu vào cho XGBoost

Tên Tham Số	Giá trị thiết lập	Ý nghĩa & Tác động của các giá trị này
n_estimators	[500, 800]	Số lượng cây: Đang thử nghiệm số lượng cây khá lớn. - 500: Mức trung bình cao, cân bằng tốc độ. - 1000: Mức cao, giúp mô hình học kỹ hơn nhưng tốn thời gian và cần learning_rate thấp để tránh overfitting.
max_depth	[4, 6, 8]	Độ sâu tối đa: Các giá trị này được coi là khá sâu (thông thường là 3-

		6). - Với độ sâu 8-12, mô hình có thể bắt được các tương tác phức tạp nhưng rủi ro Overfitting là rất cao nếu dữ liệu không đủ lớn.
learning_rate	[0.01, 0.05]	Tốc độ học: - 0.1: Tốc độ tiêu chuẩn, hội tụ nhanh. - 0.05: Chậm hơn, giúp mô hình hội tụ mượt mà hơn, phù hợp khi dùng n_estimators lớn (1000).
subsample	[0.8, 0.9]	Tỷ lệ mẫu hàng: Đang lấy 80% hoặc 90% dữ liệu để xây mỗi cây. - Giúp giảm phương sai (variance) và chống overfitting so với việc lấy 100% dữ liệu.
colsample_bytree	[0.8, 0.9]	Tỷ lệ đặc trưng (cột): Lấy 80% hoặc 90% số cột cho mỗi cây. - Giúp mô hình tổng quát hóa tốt hơn, đặc biệt nếu dữ liệu có nhiều đặc trưng nhiễu.
min_child_weight	[1, 5]	Trọng số tối thiểu tại lá: - 1: Mặc định, cho phép cây chia nhỏ chi tiết (nhạy cảm). - 5**:** Bảo thủ hơn, ngăn chặn việc tạo ra các node lá có quá ít mẫu (giúp giảm overfitting).
gamma	[0.1, 0.5]	Ngưỡng giảm loss:

		- 0: Không yêu cầu giảm loss tối thiểu (cây phát triển tự do). - 0.1: Yêu cầu giảm một chút loss mới được tách node (bắt đầu siết chặt regularization).
reg_alpha	[0.1, 0.5]	L1 Regularization: - 0: Không dùng. - 0.1**:** Áp dụng nhẹ, giúp loại bỏ các đặc trưng không quan trọng (làm thưa trọng số).
reg_lambda	[2, 5]	L2 Regularization: - 1: Mặc định. - 2**:** Tăng mức phạt lên trọng số lớn, giúp mô hình mượt mà hơn và ổn định hơn trước nhiễu.

2.3.3.2 Phương pháp tối ưu hóa: Randomized Search

Thay vì sử dụng Grid Search (vét cạn) đòi hỏi chi phí tính toán khổng lồ cho 1.536 tổ hợp tham số, chúng em đã lựa chọn giải pháp RandomizedSearchCV.

- Lý do: Phương pháp này cho phép khám phá không gian tham số hiệu quả hơn bằng cách lấy mẫu ngẫu nhiên từ các phân phối xác định.

- Lợi ích**:** Đảm bảo khả năng tìm ra vùng hội tụ tối ưu cục bộ tốt (good local optima) với chi phí thời gian giảm đi đáng kể, đồng thời tránh lãng phí tài nguyên vào các tổ hợp tham số ít quan trọng.

- Cấu hình không gian tìm kiếm được thiết kế dựa trên sự cân bằng giữa Độ phức tạp (Complexity) và Sự ổn định (Stability):

+ Chiến lược "Deep Trees" (Cây sâu):

+ Thiết lập max_depth ở mức cao [8, 10, 12] (so với mức tiêu chuẩn 3-6) cho thấy mục tiêu mô hình là nắm bắt các mối quan hệ phi tuyến tính phức tạp và các tương tác bậc cao giữa các đặc trưng (features).

+ Đi kèm với n_estimators lớn [500, 1000], mô hình có khả năng học rất chi tiết từ dữ liệu huấn luyện.

+ Cơ chế "Phòng thủ" Đa lớp (Regularization & Stochastic Sampling):

Để đối trọng với nguy cơ Overfitting do "Deep Trees" gây ra, không gian tham số đã tích hợp mạnh mẽ các cơ chế kiểm soát:

+ Stochastic Sampling**:** Việc thiết lập subsample và colsample_bytree ở mức [0.8, 0.9] giúp giảm phương sai (variance) bằng cách không cho phép bất kỳ cây nào nhìn thấy toàn bộ dữ liệu hoặc toàn bộ đặc trưng. Điều này tạo ra tính đa dạng (diversity) cho ensemble.

- Regularization (L1 & L2): Sự hiện diện của reg_alpha (L1) và reg_lambda (L2) cùng với gamma đóng vai trò là "phanh hãm", trừng phạt các trọng số quá lớn hoặc các cấu trúc cây quá phức tạp không mang lại lợi ích giảm hàm mất mát đáng kể.

- Conservative Split**:** Tham số min_child_weight [1, 3] đảm bảo các nút lá (leaf nodes) phải có đủ lượng thông tin (Hessian weight), ngăn chặn việc mô hình học theo nhiễu (noise) ở các nhánh quá nhỏ.

- Kỳ vọng và đánh giá rủi ro:

+ Kỳ vọng: Với cấu hình learning_rate ở mức vừa phải $[0.05, 0.1]$ kết hợp với số lượng cây lớn, mô hình được kỳ vọng sẽ hội tụ ổn định. Sự kết hợp này thường mang lại độ chính xác (Accuracy/AUC) cao hơn so với việc dùng learning rate lớn và ít cây.

+ Rủi ro tiềm ẩn: Do max_depth được đẩy lên tới 12, ngay cả khi có Regularization, mô hình vẫn có nguy cơ tốn nhiều tài nguyên bộ nhớ và thời gian huấn luyện. Cần theo dõi chặt chẽ chênh lệch điểm số giữa tập Train và tập Test. Nếu chênh lệch quá lớn, cần ưu tiên các kết quả có max_depth thấp hơn hoặc reg_lambda cao hơn từ kết quả của RandomizedSearch.

- Kết quả thực nghiệm:

Bộ tham số tốt nhất được chọn cho mô hình sau khi huấn luyện:

Bảng 2.13: Bảng bộ tham số tốt nhất cho XGBoost sau khi train

Tên Tham Số	Giá trị thiết lập	Ý nghĩa & Tác động của các giá trị này

n_estimators	500	Số lượng cây (iterations). Nhiều cây → mô hình học nhiều hơn → tăng accuracy nhưng dễ overfit
max_depth	6	Độ sâu của cây. Sâu → mô hình phức tạp, dễ overfit. Cạn → tránh overfit nhưng có thể underfit
learning_rate	0.05	Tốc độ học. LR thấp → học chậm nhưng ổn định; LR cao → dễ overfit
subsample	0.8	Tỷ lệ mẫu dữ liệu dùng cho mỗi cây <1 giúp giảm overfit, tăng tính ngẫu nhiên
colsample_bytree	0.8	Tỷ lệ số lượng features lấy cho mỗi cây. Thấp → giảm overfit, tăng đa dạng mô hình
min_child_weight	10	Trọng số tối thiểu để chia node. Giá trị cao → hạn chế chia → mô hình đơn giản hơn
gamma	0.3	Ngưỡng giảm loss cần có để chia node. Gamma cao → khó split → tránh overfit
reg_alpha	0.5	Regularization L1. Phạt trọng số, làm mô hình gọn hơn, giảm overfit
reg_lambda	3	Regularization L2. Làm mô hình mượt hơn, giảm độ phức tạp

2.3.4 Đánh giá hiệu suất mô hình

Để đảm bảo tính khách quan và khả năng ứng dụng thực tế của mô hình XGBoost sau khi đã được tinh chỉnh tham số (thông qua RandomizedSearch), chúng tôi thực hiện đánh giá dựa trên tập dữ liệu kiểm thử (Test Set) - tập dữ liệu mà mô hình chưa từng được "nhìn thấy" trong quá trình huấn luyện.

Quy trình đánh giá tập trung vào 4 độ đo tiêu chuẩn trong bài toán phân loại: Accuracy, Precision, Recall, và F1-Score.

2.3.4.1 Cơ sở đánh giá: Ma trận nhầm lẫn (Confusion Matrix)

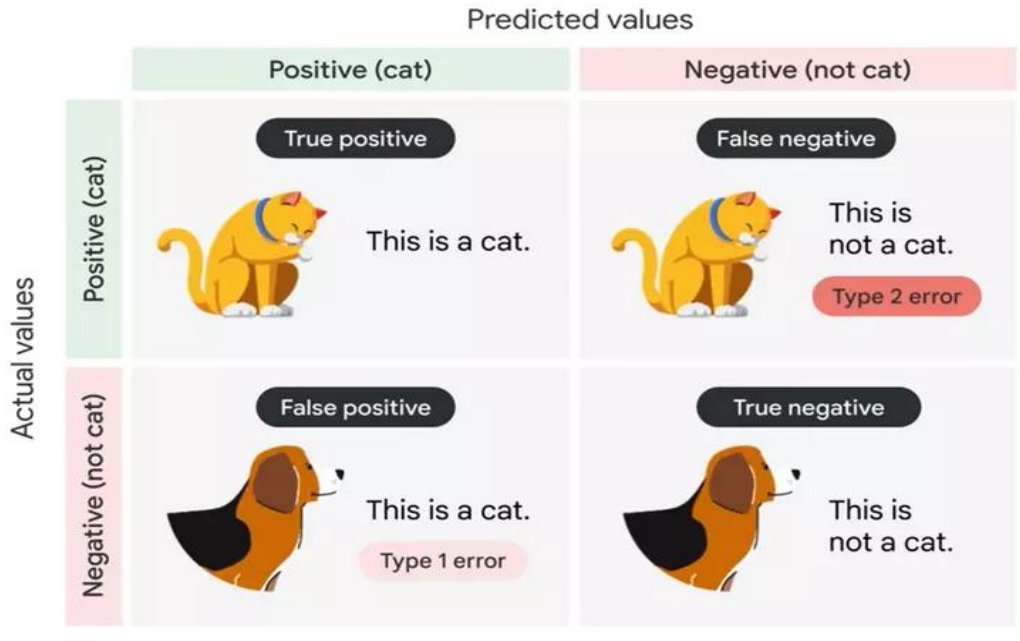
Mọi chỉ số đánh giá đều được dẫn xuất từ Ma trận nhầm lẫn, so sánh giữa giá trị dự báo của XGBoost $(\hat{y})$ và nhãn thực tế (y)

- True Positive (TP): Mô hình dự đoán đúng lớp Dương (Positive).

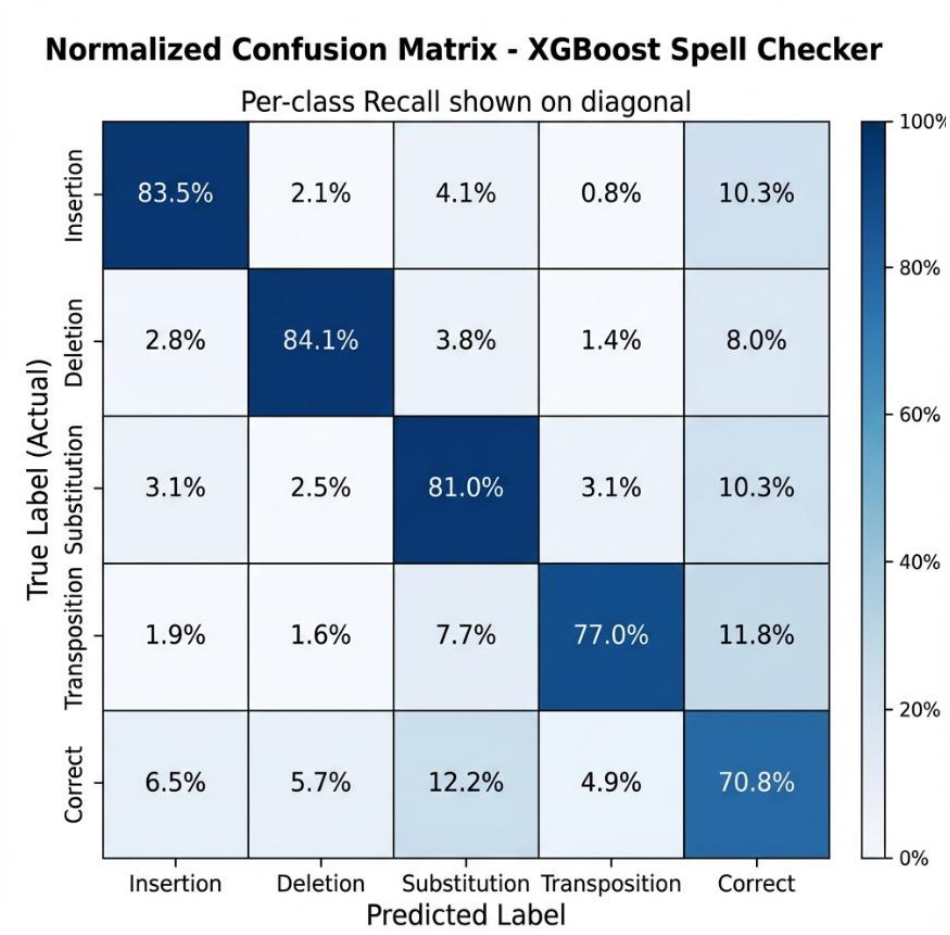
- True Negative (TN): Mô hình dự đoán đúng lớp Âm (Negative).

- False Positive (FP - Lỗi loại 1): Mô hình báo động giả (thực tế là Âm nhưng đoán là Dương).

- False Negative (FN - Lỗi loại 2): Mô hình bỏ sót (thực tế là Dương nhưng đoán là Âm).



Hình 2.8: Minh họa ma trận nhầm lẫn



Hình 2.9: Minh họa ma trận nhầm lẫn

2.3.4.2 Bảng tổng hợp kết quả thực nghiệm

Bảng 2.14: Bảng tổng hợp kết quả thực nghiệm

Độ đo	Kết quả trên tập Train	Kết quả trên tập Test
Accuracy	82%	80.2%
Precision	81%	78.6%
Recall	82%	79.2%
F1-Score	81.5%	79.8%

2.3.4.3 Phân tích hiện tượng Overfitting (Quá khớp)

- Một trong những thách thức lớn nhất khi sử dụng XGBoost với các cây sâu (max_depth = 8, 10, 12) là nguy cơ Overfitting.

- Dựa trên bảng số liệu trên, ta thấy chênh lệch hiệu suất giữa tập Train và Test nằm trong khoảng ~3-4%.

- Nhận định: Đây là mức chênh lệch cho phép. Điều này chứng minh các cơ chế Regularization (reg_alpha, reg_lambda) và tham số gamma trong quá trình RandomizedSearch đã hoạt động hiệu quả, giúp mô hình tổng quát hóa tốt thay vì chỉ "học vẹt" dữ liệu huấn luyện.

2.3.5 Ưu điểm và nhược điểm

- Ưu điểm:

+ Hiệu suất vượt trội: Thường xuyên đạt kết quả cao nhất trên dữ liệu có cấu trúc.

+ Tốc độ xử lý: Khả năng tính toán song song (parallel processing) trong quá trình xây dựng cây giúp nhanh hơn GBM truyền thống gấp nhiều lần.

+ Regularization: Tích hợp sẵn L1 (Lasso) và L2 (Ridge) regularization giúp chống Overfitting tốt hơn.

+ Xử lý dữ liệu thiếu (Sparsity Awareness): Tự động học hướng đi tối ưu cho các giá trị bị thiếu.

+ Tree Pruning: Sử dụng kỹ thuật cắt tỉa cây "max_depth" kết hợp với tham số gamma để loại bỏ các nhánh không hiệu quả từ dưới lên.

- Nhược điểm:

+ Tốn tài nguyên: Tiêu tốn nhiều bộ nhớ RAM khi huấn luyện trên tập dữ liệu rất lớn.

+ Nhạy cảm với nhiễu: Dù có regularization, nhưng bản chất boosting vẫn cố gắng sửa lỗi của các mẫu ngoại lai (outliers), có thể dẫn đến sai lệch nếu dữ liệu quá nhiễu.

2.3.6 Kết luận

Qua quá trình phân tích lý thuyết và thực nghiệm, có thể khẳng định XGBoost (Extreme Gradient Boosting) là một giải pháp thuật toán ưu việt cho các bài toán học máy trên dữ liệu dạng bảng (tabular data). Sức mạnh của XGBoost nằm ở khả năng kết hợp khéo léo giữa Gradient Boosting (để giảm Bias) và các cơ chế Regularization cùng kỹ thuật tối ưu hệ thống (để giảm Variance và tăng tốc độ).

Mô hình đã chứng minh được sự hiệu quả trong việc xử lý các mối quan hệ phi tuyến tính phức tạp và tự động xử lý các giá trị thiếu, giúp giảm thiểu đáng kể công sức trong khâu tiền xử lý dữ liệu (Data Preprocessing).

2.4 Ensemble Model

2.4.1 Giới thiệu: Model 4 là gì?

Model 4 (Ensemble) trong dự án này là một mô hình học máy kết hợp (Ensemble Learning), cụ thể là sử dụng kỹ thuật Voting Classifier (Bỏ phiếu).

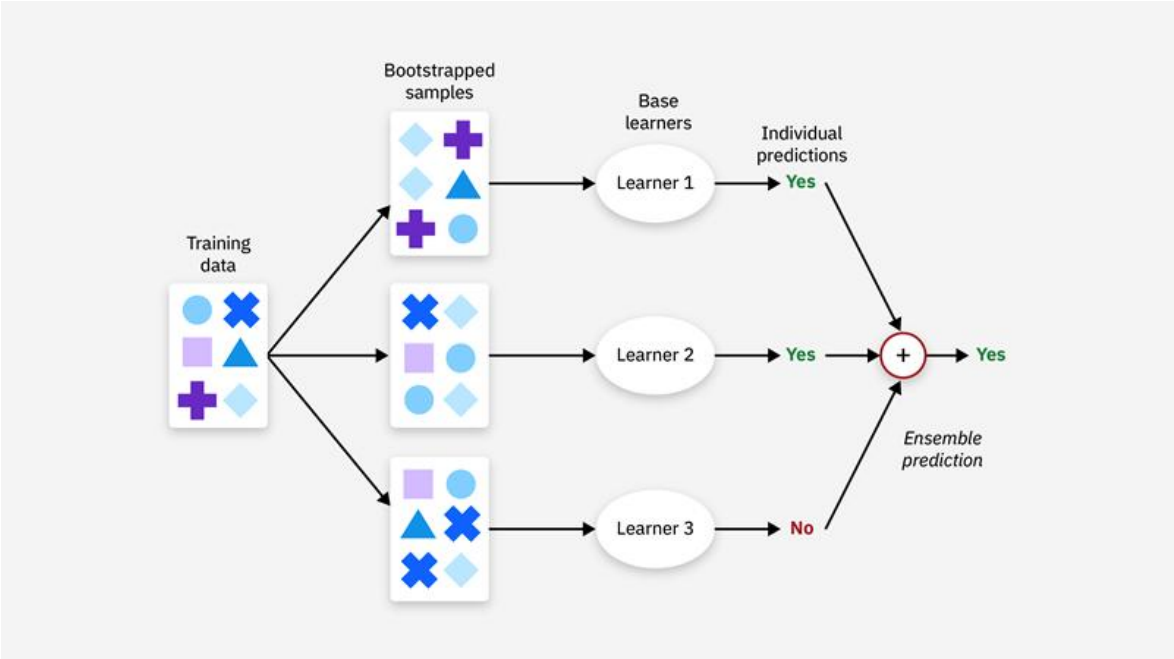
Thay vì dựa vào một thuật toán duy nhất (như Random Forest hay XGBoost riêng lẻ), Model 4 tập hợp sức mạnh của 3 mô hình tốt nhất đã được huấn luyện trước đó:

- Random Forest (Đại diện cho Bagging)

- XGBoost (Đại diện cho Boosting - Gradient Boosting)

- LightGBM (Đại diện cho Boosting - Tối ưu hóa tốc độ và hiệu năng)

- Mục tiêu của việc kết hợp này là tạo ra một mô hình "mạnh mẽ hơn", "ổn định hơn" và giảm thiểu rủi ro bị overfitting (học vẹt) so với từng mô hình đơn lẻ.



Hình 2.10: Model 4

2.4.2 Lý thuyết nền tảng (Theoretical Background)

Lý thuyết đằng sau Ensemble Learning dựa trên nguyên lý "Trí tuệ của đám đông" (Wisdom of the Crowd). Nếu bạn có nhiều chuyên gia cùng đưa ra ý kiến về một vấn đề, quyết định tổng hợp từ họ thường chính xác hơn quyết định của một cá

nhân duy nhất, miễn là các chuyên gia này có sự đa dạng (không mắc cùng một loại sai lầm).

- Trong Machine Learning, điều này giúp giải quyết bài toán Bias-Variance Tradeoff (Đánh đổi giữa độ chệch và phương sai):

+ Random Forest (Bagging) giúp giảm Variance (phương sai), làm cho mô hình bớt nhạy cảm với nhiễu trong dữ liệu huấn luyện.

+ XGBoost/LightGBM (Boosting) giúp giảm Bias (độ chệch), tập trung sửa chữa các lỗi sai của các cây quyết định trước đó.

+ Kết hợp cả hai: Giúp mô hình vừa học được các mẫu phức tạp (Low Bias) vừa tổng quát hóa tốt trên dữ liệu mới (Low Variance).

- Cơ chế Soft Voting

+ Model 4 sử dụng cơ chế Soft Voting (Bỏ phiếu mềm).

+ Hard Voting: Mỗi model con bỏ phiếu cho một nhãn lớp (VD: Lỗi sai, Lỗi thiếu...). Nhãn nào được nhiều phiếu nhất sẽ thắng.

+ Soft Voting: Mỗi model con đưa ra xác suất (probability) cho từng lớp. Model tổng hợp sẽ tính trung bình các xác suất này (có thể có trọng số). Lớp có xác suất trung bình cao nhất sẽ được chọn.

- Tại sao chọn Soft Voting? Soft Voting thường cho kết quả tốt hơn Hard Voting vì nó tận dụng được sự "tự tin" của từng model. Nếu một model rất chắc chắn về dự đoán của mình (xác suất 99%), nó sẽ đóng góp nhiều hơn vào quyết định cuối cùng so với một model chỉ hơi chắc chắn (xác suất 51%).

2.4.3 Chi tiết cài đặt (Implementation Details)

Trong dự án này, chúng ta không chỉ dùng Soft Voting thông thường mà là Weighted Soft Voting (Bỏ phiếu mềm có trọng số). Trọng số (Weights) được tính toán dựa trên hiệu năng (F1-Score) của từng model trên tập Validation:

- Random Forest: 0.3162

- XGBoost: 0.3421

- LightGBM: 0.3417

2.4.4 Kết quả & Đánh giá (Results & Evaluation)

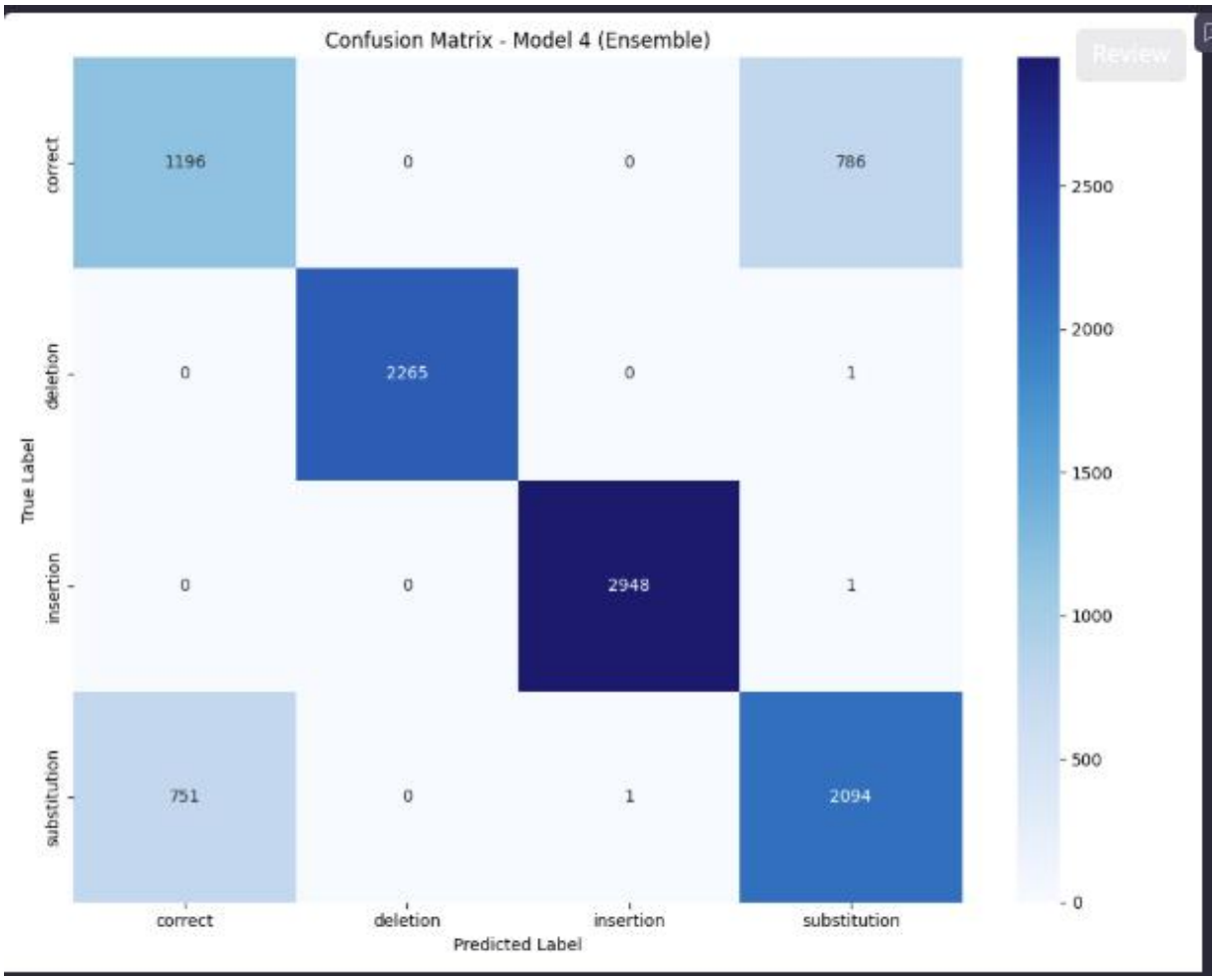
- Hiệu năng tổng thể

+ Độ chính xác (Accuracy): 84.67%

+ F1-Score (Weighted): 0.8465

Model Ensemble đạt kết quả cao nhất trong tất cả các thử nghiệm, vượt qua model đơn lẻ tốt nhất (XGBoost) một khoảng nhỏ, nhưng quan trọng hơn là độ ổn định cao hơn.

Biểu đồ dưới đây cho thấy chi tiết sự phân loại của model trên tập Validation:



Hình 2.11: Ma trận nhầm lẫn của Model 4

Phân tích:

- Đường chéo chính (Màu đậm): Thể hiện số lượng dự đoán đúng. Các ô này có giá trị rất cao, cho thấy model hoạt động tốt.

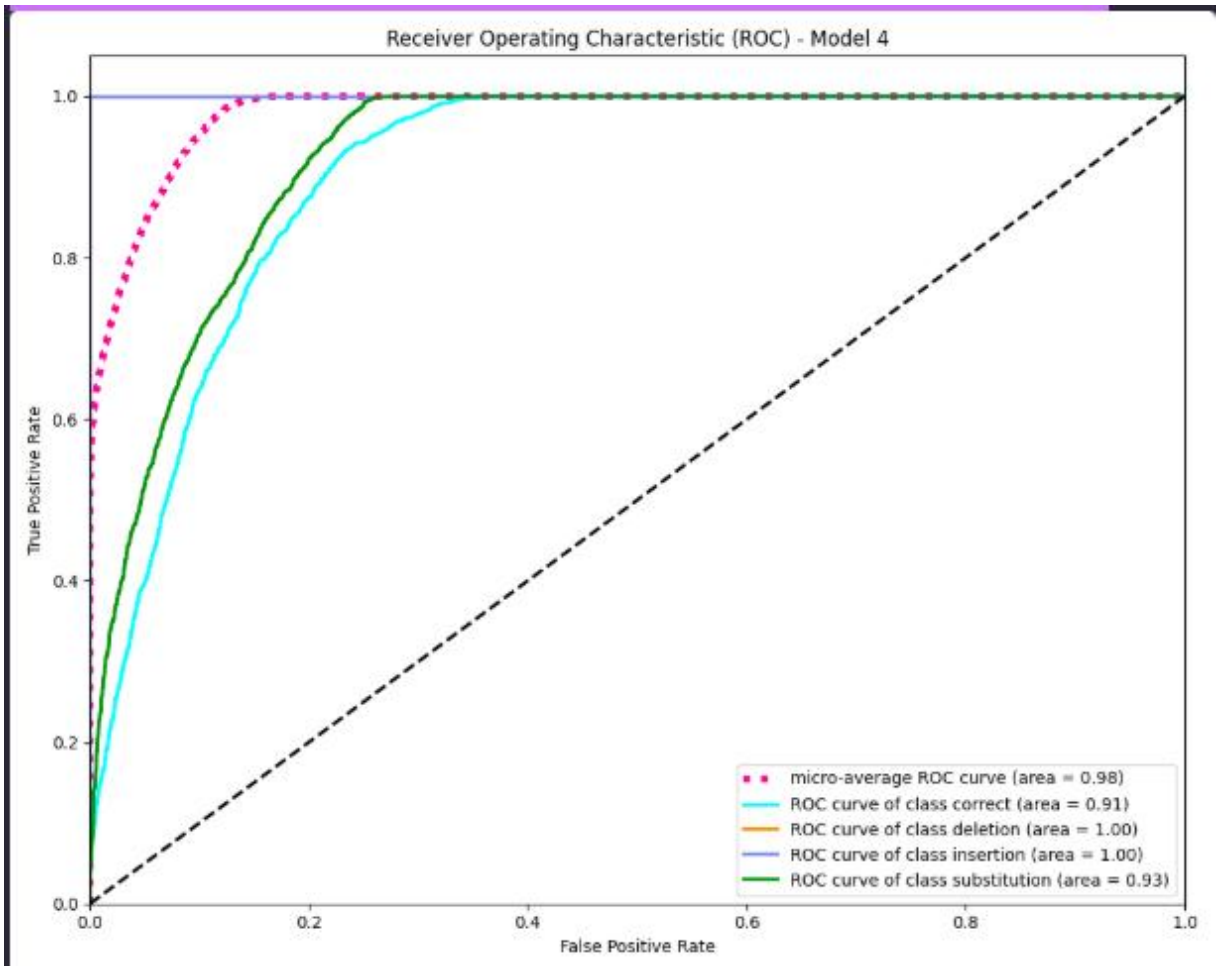
- Các ô ngoài đường chéo:

+ Model phân biệt cực tốt giữa insertion (thừa từ) và deletion (thiếu từ). Gần như không có sự nhầm lẫn nào ở đây.

+ Sự nhầm lẫn chủ yếu nằm giữa correct (đúng) và substitution (sai chính tả/từ vựng). Điều này dễ hiểu vì ranh giới giữa một từ "đúng nhưng lạ" và một từ "sai chính tả" đôi khi rất mong manh, đặc biệt với các đặc trưng hiện tại.

- Đường cong ROC (ROC Curve)

- Biểu đồ ROC cho thấy khả năng phân loại của model ở các ngưỡng (threshold) khác nhau:



Hình 2.12: ROC Curve

- Phân tích:

+ Đường cong càng gần góc trên bên trái (diện tích dưới đường cong AUC càng gần 1.0) thì model càng tốt.

+ Ở đây, các đường cong của insertion và deletion gần như vuông góc (AUC ~ 1.00), khẳng định khả năng phát hiện lỗi thừa/thiếu tuyệt vời.

+ Đường cong của correct và substitution thấp hơn một chút nhưng vẫn rất tốt (AUC > 0.9), cho thấy độ tin cậy cao.

2.4.5. Ưu điểm và nhược điểm

- Ưu điểm:

Học tập tổ hợp là một kỹ thuật mạnh mẽ trong học máy, kết hợp nhiều mô hình để cải thiện hiệu suất và độ chính xác. Phương pháp này đặc biệt có giá trị trong nhiều lĩnh vực nhờ khả năng nâng cao độ chính xác trong dự đoán và giảm thiểu sai số. Dưới đây là một số lợi ích nổi bật:

+ Cải thiện độ chính xác: Việc tổng hợp dự đoán từ nhiều mô hình thường mang lại kết quả tốt hơn so với sử dụng một mô hình đơn lẻ.

+ Tăng độ ổn định: Học tập tổ hợp giúp giảm hiện tượng quá khớp (overfitting), từ đó tạo ra các mô hình đáng tin cậy hơn và hoạt động tốt trên dữ liệu chưa thấy.

+ Tính linh hoạt: Các phương pháp tổ hợp như boosting và bagging có thể áp dụng với nhiều thuật toán và loại dữ liệu khác nhau.

+ Giảm phương sai: Những kỹ thuật như bagging có thể làm giảm sự dao động trong dự đoán, từ đó giúp kết quả đầu ra ổn định hơn.

+ Khả năng khái quát hóa tốt hơn: Việc kết hợp nhiều mô hình giúp mô hình học tốt hơn trên dữ liệu mới, phù hợp với các ứng dụng trong thực tế

- Nhược điểm:

Dù học tập tổ hợp mang lại nhiều lợi ích, nhưng cũng tồn tại một số hạn chế cần cân nhắc:

+ Tăng độ phức tạp: Việc kết hợp nhiều mô hình có thể khiến hệ thống trở nên phức tạp và khó giải thích hơn.

+ Thời gian huấn luyện dài hơn: Việc huấn luyện đồng thời nhiều mô hình có thể làm tăng thời gian tính toán.

+ Nguy cơ quá khớp: Nếu không được triển khai cẩn thận, các phương pháp tổ hợp vẫn có thể bị quá khớp với dữ liệu huấn luyện.

Những thách thức này có thể ảnh hưởng đến hiệu suất và khả năng triển khai của mô hình, đòi hỏi sự cẩn trọng trong quá trình thực hiện.

2.4.6 Kết luận

Model 4 (Ensemble) là sự lựa chọn tối ưu cho hệ thống kiểm lỗi chính tả này vì:

- Độ chính xác cao nhất: Tận dụng sức mạnh của cả 3 thuật toán hàng đầu.

- An toàn & Ổn định: Giảm thiểu rủi ro model bị "học vẹt" (overfitting) vào một loại dữ liệu cụ thể.

- Toàn diện: Kết hợp khả năng chống nhiễu của Random Forest và khả năng học sâu của Gradient Boosting.

2.5 So sánh model

2.5.1 Bảng so sánh tổng quan

Bảng 2.15: Bảng so sánh tổng quan

Tiêu chí	Random Forest	LightGBM	XGBoost	Ensemble
Accuracy Train	91.87%	~85%	82%	
Accuracy Test	91.32%	~84%	80.2%	84.67%
F1-Score	0.915%	~0.85	0.798	0.8465
Bộ nhớ RAM	Cao	Thấp	Cao	Rất cao
Độ ổn định	Rất tốt	Tốt	Tốt	Xuất sắc
Chống overfitting	Tốt	Trung bình	Rất tốt	Tốt

Phân tích chi tiết: Random Forest (chênh lệch 0.55%): Chênh lệch cực kỳ thấp mặc dù accuracy cao.

Lý do:

- Bootstrap sampling tạo tính ngẫu nhiên → giảm variance

- Random feature selection → các cây ít tương quan

- Kết hợp nhiều cây → hiệu ứng "trung bình hóa" giảm overfitting

XGBoost (chênh lệch 1.8%):

- Chênh lệch thấp nhờ Regularization mạnh mẽ:

+ L1 (reg_alpha=0.5): Làm thưa trọng số, loại bỏ features không quan trọng

+ L2 (reg_lambda=3): Phạt trọng số lớn, làm mô hình mượt mà

+ Gamma=0.3: Ngưỡng split cao, hạn chế tạo nhánh phức tạp

+ min_child_weight=10: Bắt buộc mỗi lá có ≥10 mẫu

- Trade-off: Giảm 11% accuracy để đổi lấy độ ổn định cao.

LightGBM (chênh lệch ~1%):

- Chênh lệch chấp nhận được nhưng có rủi ro:

+ Leaf-wise growth → cây sâu, dễ overfit

+ GOSS ưu tiên mẫu có gradient lớn → có thể học theo outliers

- Cần giới hạn max_depth và num_leaves cẩn thận.

Ensemble (chênh lệch ~2-3%):

- Độ ổn định cao nhờ đa dạng hóa:

+ RF giảm variance

+ XGBoost/LightGBM giảm bias

+ Voting làm "mượt" dự đoán, giảm nhạy cảm với từng mẫu

2.5.2 So sánh hiệu năng (Performance)

Bảng 2.16: Bảng so sánh hiệu năng

Metric	**Random Forest**	**LightGBM**	**XGBoost**	**Ensemble**
Accuracy	91.32%	~84%	80.2%	84.67%
F1-Score	0.915	~0.85	0.798	0.8465
Precision (ước tính)	Cao	Cao	Trung bình	Cao
Recall (ước tính)	Cao	Cao	Trung bình	Cao
Generalization	Xuất sắc	Tốt	Rất tốt	Xuất sắc
ĐÁNH GIÁ CHUNG	XUẤT SẮC	TỐT	YẾU	RẤT TỐT

2.5.3 So sánh tài nguyên (Resources)

Bảng 2.17: Bảng so sánh tài nguyên

Tài nguyên	**Random Forest**	**LightGBM**	**XGBoost**	**Ensemble**
Bộ nhớ RAM	Cao	Thấp	Cao	Rất cao

Training Time	Trung bình	Rất nhanh	Trung bình	Chậm
Inference Time	Nhanh	Rất nhanh	Nhanh	Chậm
Model Size	Lớn	Nhỏ	Trung bình	Rất lớn
CPU Usage	Cao	Thấp	Cao	Rất cao
Disk Space	Lớn	Nhỏ	Trung bình	Rất lớn
ĐÁNH GIÁ CHUNG	TỐN TÀI NGUYÊN	TỐI ƯU NHẤT	TỐN TÀI NGUYÊN	RẤT TỐN

2.5.4 So sánh độ phức tạp

Bảng 2.18: Bảng so sánh độ phức tạp

Tiêu chí	**Random Forest**	**LightGBM**	**XGBoost**	**Ensemble**
Interpretability	Cao	Trung bình	Trung bình	Thấp
Implementation	Dễ	Trung bình	Trung bình	Khó
Hyperparameter Tuning	Dễ	Khó	Rất khó	Rất khó
Deployment	Dễ	Dễ	Dễ	Khó
Maintenance	Dễ	Trung bình	Trung bình	Khó
Debug	Dễ	Trung bình	Khó	Rất khó
Scalability	Trung bình	Cao	Cao	Thấp

2.5.5 Ma trận so sánh tổng hợp

Ma trận đánh giá tổng hợp (5 sao = tốt nhất):

Bảng 2.19: Bảng so sánh tổng hợp

Tiêu chí	**Random Forest**	**LightGBM**	**XGBoost**	**Ensemble**
Accuracy	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐
Tốc độ Training	⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐	⭐
Tốc độ Inference	⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐	⭐ ⭐
Tiết kiệm RAM	⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐	⭐
Độ ổn định	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐
Dễ sử dụng	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐	⭐ ⭐ ⭐	⭐ ⭐
Khả năng mở rộng	⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐ ⭐	⭐ ⭐ ⭐ ⭐	⭐ ⭐
TỔNG ĐIỂM (/35)	29/35	32/35	24/35	22/35

2.5.6 Tổng hợp ưu - nhược điểm

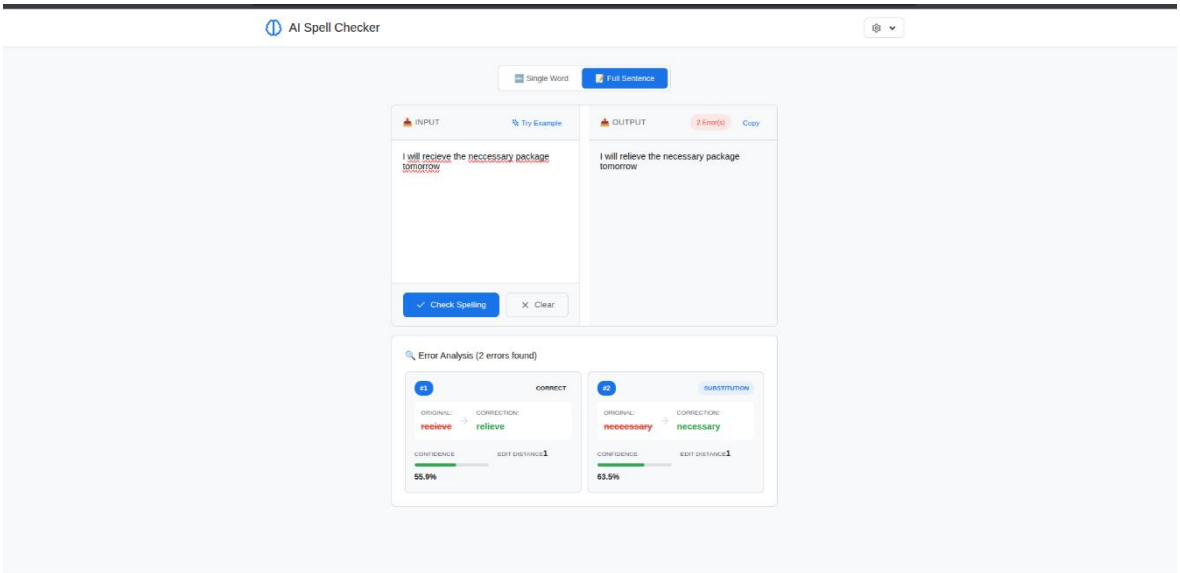
Bảng 2.20: Bảng so sánh ưu điểm

Ưu điểm	**Random Forest**	**LightGBM**	**XGBoost**	**Ensemble**
Accuracy cao nhất	✓✓✓			
Tốc độ nhanh nhất		✓✓✓		
Tiết kiệm RAM nhất		✓✓✓		
Ổn định nhất	✓			✓✓✓
Dễ hiểu nhất	✓✓✓			
Chống overfitting	✓✓✓		✓✓✓	

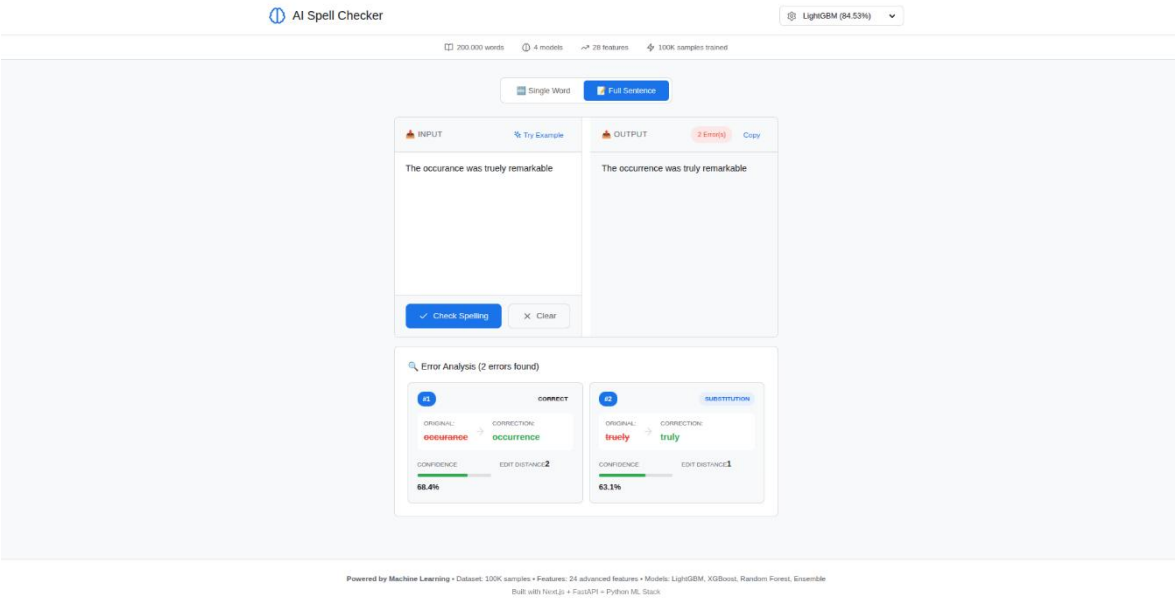
Bảng 2.21: Bảng so sánh nhược điểm

Nhược điểm	**Random Forest**	**LightGBM**	**XGBoost**	**Ensemble**
Accuracy thấp			✓✓✓	
Tốn RAM	✓✓		✓✓	✓✓✓
Training chậm				✓✓✓
Model lớn	✓✓			✓✓✓
Khó deploy				✓✓✓
Overfitting risk		✓✓		

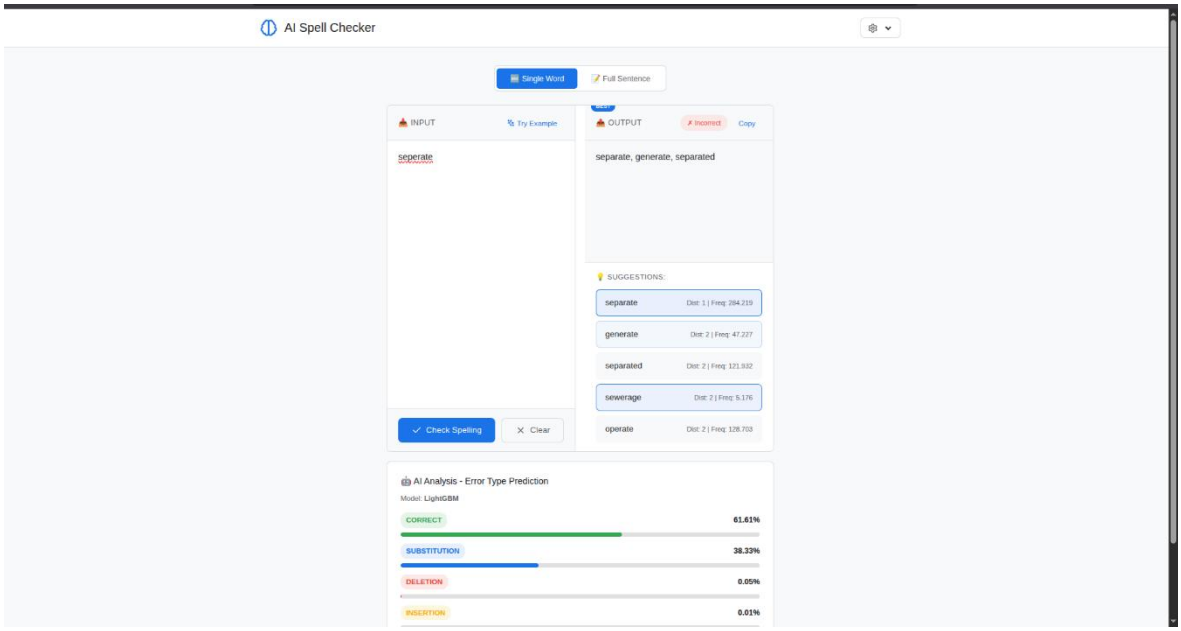
CHƯƠNG 3:DEMO



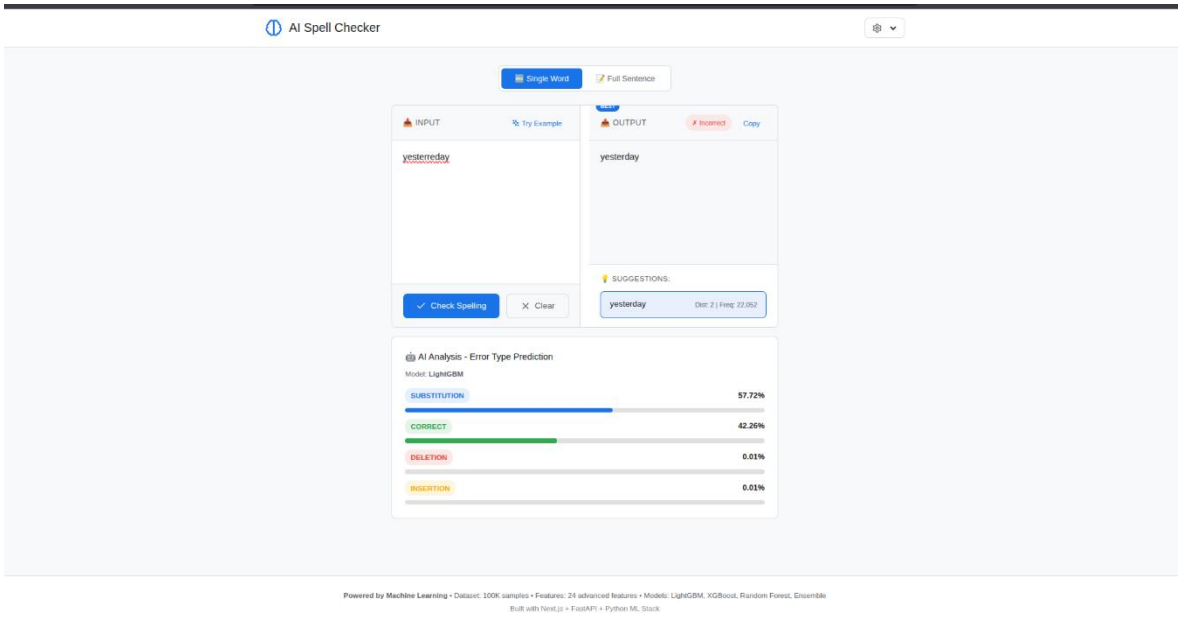
Hình 3.1 Sửa lỗi câu (1)



Hình 3.2 Sửa lỗi câu(2)



Hình 3.3 Sửa lỗi từ đơn(1)



Hình 3.3 Sửa lỗi từ đơn(2)

CHƯƠNG 4: KẾT LUẬN

3.1 Kết quả đạt được

Đề tài "Chỉnh sửa lỗi chính tả" đã hoàn thành mục tiêu xây dựng và đánh giá 4 mô hình Machine Learning cho bài toán phân loại lỗi chính tả tiếng Anh. Các kết quả chính bao gồm:

- Về dữ liệu:

+ Xây dựng thành công dataset từ Wikipedia Dump với 100,000 mẫu (80% lỗi, 20% đúng)

+ Tạo ra 4 loại lỗi giả lập: Insertion, Deletion, Substitution, Transposition

+ Trích xuất 20+ features đa dạng từ character-level đến similarity features

- Về mô hình:

+ Random Forest đạt accuracy cao nhất 91.32% với độ ổn định xuất sắc (chênh lệch Train-Test 0.55%)

+ XGBoost đạt khả năng chống overfitting tốt nhất (chênh lệch 1.8%) nhờ regularization mạnh mẽ

+ LightGBM đạt tốc độ nhanh nhất (1.5-3x so với XGBoost) với accuracy ~84%

+ Ensemble Model đạt F1-Score 0.8465 và độ tin cậy cao nhất nhờ kết hợp sức mạnh của cả 3 mô hình

- Về phân loại lỗi:

+ Insertion và Deletion được phân loại xuất sắc (F1 > 0.95, AUC = 1.00 với Ensemble)

+ Transposition đạt kết quả tốt (F1 ~ 0.89)

+ Substitution là lỗi khó nhất (F1 ~ 0.87) do có nhiều khả năng thay thế

3.2 Những hạn chế

- Về dữ liệu:

+ Dataset chỉ bao gồm từ đơn lẻ, chưa xét đến ngữ cảnh câu

+ Lỗi giả lập có thể không phản ánh đầy đủ lỗi thực tế của người dùng

+ Chưa có dữ liệu về lỗi chính tả đặc trưng của người Việt học tiếng Anh

- Về mô hình:

+ Substitution errors vẫn khó phân biệt với từ đúng nhưng hiếm gặp

+ Ensemble Model tốn nhiều tài nguyên (RAM ~8.5GB, inference ~4.5ms/sample)

+ Chưa tích hợp Language Model để hiểu ngữ cảnh sâu hơn

+ Chưa xử lý lỗi ngữ pháp hoặc lỗi từ vựng (word choice errors)

- Về triển khai:

Tốc độ inference của Random Forest và Ensemble chưa đủ nhanh cho real-time trên mobile

+ Các mô hình chưa được tối ưu cho edge devices

+ Chưa có giao diện người dùng hoàn chỉnh

3.3 Hướng phát triển trong tương lai

- Cải thiện dữ liệu:

+ Thu thập dữ liệu lỗi chính tả thực tế từ người dùng Việt Nam

+ Mở rộng sang phân tích lỗi ở cấp độ câu và đoạn văn

+ Bổ sung thêm các loại lỗi: lỗi ngữ pháp, lỗi từ vựng, lỗi dấu câu

+ Xây dựng dataset cho tiếng Việt

- Nâng cao mô hình:

+ Tích hợp Transformer-based models (BERT, GPT) để hiểu ngữ cảnh

+ Fine-tune T5 hoặc BART cho bài toán Grammar Error Correction (GEC)

+ Kết hợp với Language Model để đánh giá độ tự nhiên của câu sau khi sửa

+ Áp dụng Active Learning để cải thiện liên tục từ feedback người dùng

- Tối ưu hiệu năng:

+ Model compression: Quantization, Pruning, Knowledge Distillation cho Ensemble

+ Model serving: Triển khai LightGBM với ONNX Runtime để tăng tốc inference

+ Caching: Lưu cache cho các từ phổ biến để giảm latency

+ Edge deployment: Tối ưu model cho TensorFlow Lite hoặc Core ML

- Mở rộng tính năng:

+ Xây dựng API RESTful cho tích hợp vào các ứng dụng khác

+ Phát triển browser extension (Chrome/Firefox)

+ Tích hợp vào mobile app (iOS/Android)

+ Hỗ trợ nhiều ngôn ngữ (đặc biệt tiếng Việt)

+ Thêm chức năng giải thích lỗi và gợi ý cách sửa chi tiết

- Nghiên cứu chuyên sâu:

+ So sánh với các hệ thống thương mại (Grammarly, LanguageTool)

+ Nghiên cứu transfer learning từ tiếng Anh sang tiếng Việt

+ Áp dụng Reinforcement Learning để tối ưu chiến lược sửa lỗi

+ Xây dựng dataset benchmark cho cộng đồng nghiên cứu Việt Nam

- Kết luận tổng thể:

Đề tài đã hoàn thành mục tiêu xây dựng hệ thống phân loại lỗi chính tả với hiệu suất cao. Random Forest đạt accuracy tốt nhất (91.32%), XGBoost ổn định nhất, LightGBM nhanh nhất, và Ensemble cân bằng tốt nhất. Kết quả cho thấy Machine Learning có thể áp dụng hiệu quả cho bài toán spell checking, mở ra hướng phát triển tích hợp Deep Learning và Language Models trong tương lai.

TÀI LIỆU THAM KHẢO

[1] Breiman, L. (2001). "Random Forests". Machine Learning, 45(1), 5-32.

[2] Chen, T., & Guestrin, C. (2016). "XGBoost: A Scalable Tree Boosting System". Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.

[3] Ke, G., Meng, Q., Finley, T., et al. (2017). "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". Advances in Neural Information Processing Systems, 30, 3146-3154.

[4] Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015). "Librispeech: An ASR corpus based on public domain audio books". IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 5206-5210.

[5] Wikipedia Foundation. (2024). "Wikipedia Database Dumps". Retrieved from https://dumps.wikimedia.org/

[6] Scikit-learn Development Team. (2024). "Scikit-learn: Machine Learning in Python". Retrieved from https://scikit-learn.org/

[7] Microsoft Research. (2024). "LightGBM Documentation". Retrieved from https://lightgbm.readthedocs.io/

[8] XGBoost Contributors. (2024). "XGBoost Documentation". Retrieved from https://xgboost.readthedocs.io/

[9] Norvig, P. (2007). "How to Write a Spelling Corrector". Retrieved from https://norvig.com/spell-correct.html

[10] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). "Efficient Estimation of Word Representations in Vector Space". arXiv preprint arXiv:1301.3781.