ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

ĐỒ ÁN CUỐI KỲ

GÁN NHÃN TỪ LOẠI VĂN BẢN TIẾNG VIỆT

Môn học: CS211.O11

Giảng viên lý thuyết: Nguyễn Trọng Chỉnh

Giảng viên thực hành: Đặng Văn Thìn và Nguyễn Đức Vũ

Nhóm sinh viên thực hiện:

Đỗ Việt Bách	19521230
Huỳnh Phước Lân	21521064
Lê Thị Như Ý	21522818

TP.HCM, ngày 02 tháng 2 năm 2024

MỤC LỤC

1. Giới thiệu đề tài

Gán nhãn từ loại, hay còn gọi là Part-of-speech tagging hoặc POS tagging, đề cập đến việc đánh dấu mỗi từ trong văn bản (corpus) tương ứng với một loại từ nhất định, dựa trên định nghĩa và bối cảnh ngôn ngữ của từ đó. Đây là một bài toán quan trọng trong lĩnh vực Xử lý Ngôn ngữ Tự nhiên. Việc phân biệt các từ loại trong câu giúp chúng ta hiểu rõ hơn về ý nghĩa và là một bước tiền xử lý quan trọng cho các nhiệm vụ khác như phân tích cú pháp, tìm kiếm văn bản, nhận dạng thực thể, và nhiều nhiệm vụ khác. Tuy nhiên, bài toán gán nhãn từ loại thường đối mặt với nhiều thách thức khó khăn. Một trong những thách thức đầu tiên là sự nhập nhằng (ambiguity), hiển thị qua việc một từ có thể được gán nhiều loại từ tùy thuộc vào ngữ cảnh của văn bản.

Ví dụ, trong câu "Con ngựa đá con ngựa đá," từ "đá" đầu tiên được gán là động từ, trong khi từ "đá" còn lại được xem là tính từ. Bên cạnh đó, việc xử lý từ không xuất hiện trong ngữ liệu huấn luyện tạo ra những thách thức bổ sung khi xây dựng mô hình gán nhãn.

Trong phạm vi đề tài này, nhóm nghiên cứu trình bày phương pháp gán nhãn từ loại sử dụng mô hình Hidden Markov kết hợp với thuật toán Viterbi so sánh với phương pháp Trích xuất đặc trưng và mô hình Support Vector Machine để thực hiện việc gán nhãn từ loại cho văn bản tiếng Việt.

2. Thu thập dữ liệu

2.1. Nguồn dữ liệu

Dữ liệu được thu thập từ nhiều trang báo uy tín của Việt Nam thuộc nhiều lĩnh vực khác nhau như đơi sống, công nghệ, pháp luật,… Được lưu trữ trong file data NLP.txt.

Một số nguồn như https://dantri.com.vn/ , https://thanhnien.vn/ , https://vnexpress.net/.

2.2. Nhận xét dữ liệu

Bộ dữ liệu được lưu trữ trong file data NLP.txt:

- Mỗi dòng là một câu, cuối câu kết thúc bằng dấu chấm "." hoặc dấu ba chấm "…" hoặc dấu chấm hỏi "?"
- Các từ được phân cách với nhau bởi dấu khoảng cách " "
- Số lượng câu: 147
- Chứa một số từ viết tắt cho tiếng Việt và tiếng Anh (VD: VQG, NATO, TP.HCM, …), tên riêng nước ngoài (VD: NATO, Washington, Lavrov, …)
- Chứa các dạng số nguyên được ngăn cách bởi dấu chấm (VD: 1.000.0000), các dạng ngày tháng (VD: 30/4, 1/5, …), số thập phân.
- Tiền xử lý: kiểm tra và loại bỏ lỗi chính tả, kiểm tra cấu trúc, ngữ pháp của câu.

3. Tách từ

3.1. Giới thiệu

Tách từ là quy tắc ngữ pháp quan trọng trong tiếng Việt, đóng vai trò quyết định đến sự hiểu biết chính xác và truyền đạt ý nghĩa của câu. Quy tắc này giúp chúng ta phân biệt được các thành phần câu, từ đó xây dựng được cấu trúc ngữ pháp rõ ràng.

3.2. Phương pháp thủ công

Phương pháp tách từ trong tiếng Việt là một quy tắc ngữ pháp giúp chúng ta xác định ranh giới giữa các từ trong câu để hiểu rõ ý nghĩa và cấu trúc ngữ pháp. Dưới đây là một số phương pháp chính để tách từ trong tiếng Việt:

- Dựa vào Dấu Cách: Một trong những phương pháp đơn giản là dựa vào dấu cách giữa các từ để phân biệt chúng. Mỗi từ thường được đặt sau một dấu cách.
- Dựa vào Dấu Ngữ Cảnh: Một số trường hợp tách từ dựa vào ngữ cảnh. Ví dụ, khi gặp một từ mới mà chưa biết, có thể dựa vào ngữ cảnh xung quanh để xác định ranh giới của từ đó.
- Dựa vào Tiền-Đề và Hậu-Đề: Tách từ còn dựa vào việc xác định tiền-đề (phần trước từ) và hậu-đề (phần sau từ) của một từ. Việc này thường dựa vào kiến thức về ngữ pháp và ý nghĩa của từ trong câu.
- Dựa vào Dấu Chấm, Dấu Phẩy, Dấu Chấm Phẩy: Các dấu chấm, dấu phẩy, và dấu chấm phẩy thường được sử dụng để phân tách các đơn vị ngữ pháp trong câu. Sau mỗi dấu chấm, dấu phẩy, hoặc dấu chấm phẩy, thường sẽ tách từ mới.
- Dựa vào Tiếng Đặt Câu và Tần Số: Người viết thường sử dụng ngôn ngữ tự nhiên và theo tiếng đặt câu để tạo ra sự hài hòa và dễ hiểu. Sự tách từ cũng thường phản ánh tiếng đặt câu tự nhiên và tần suất xuất hiện của từ trong văn bản.

- Dựa vào Kiến Thức Ngôn Ngữ: Sự hiểu biết về ngữ pháp và cấu trúc ngôn ngữ cũng giúp người viết tách từ một cách chính xác. Kiến thức này bao gồm việc nhận biết các loại từ (danh từ, động từ, tính từ, trạng từ) và cách chúng kết hợp trong câu.

Ngoài những đặc điểm trên, nhóm chúng em cũng tham khảo bộ từ điển Từ điển Tiếng Việt. Vietnamese Dictionary. (vtudien.com) để có thể tách từ một cách tốt nhất.

3.3. Phương pháp Longest matching

Longest Matching là thuật toán dựa trên chiến lược tham lam. Ý tưởng của thuật toán là xét các tiếng từ trái sang phải, các tiếng đầu tiên dài nhất xuất hiện trong từ điển sẽ được tách thành một từ. Thuật toán sẽ dừng khi xét hết các tiếng trong câu.

Nhóm sử dụng bộ dữ liệu từ điển gồm 31158 lưu trong file Dictionary.txt. Nhóm sẽ chia bộ dữ liệu thành các file bi_grams.txt chỉ chứa các từ có 2 tiếng, file tri_grams.txt chỉ chứa các từ có 3 tiếng, file quadri_grams.txt chỉ chứa các từ có 4 tiếng và file penta_grams.txt chi chứa các từ có 5 tiếng. Sau đó tiến hành cài đặt thuật toán.

Ưu điểm:

- Thời gian xử lý tương đối nhanh.

- Cài đặt đơn giản.

- Độ chính xác tương đối cao.

Nhược điểm:

- Độ chính xác phụ thuộc hoàn toàn vào tính đầy đủ và tính chính xác của từ điển.

- Khó có thể xử lý được các tình huống nhập nhằng.

- Không thể nhận ra các từ ghép ngoài từ điển.

3.4. So sánh

Nhóm so sánh khả năng tách từ của phương pháp Longest matching, thủ công và thư viện VNcoreNLP như sau:

```
count_manual_compounds = 0
for sentence in  manual_tokenize_sentences:
    for word in sentence.split():
        if '_' in word: count_manual_compounds += 1
print('Số lượng từ ghép khi tách từ thủ công:', count_manual_compounds)
```

```
Số lượng từ ghép khi tách từ thủ công: 1069
```

```
[ ] count_vncore_compounds = 0
for sentence in  vncore_tokenize_sentences:
    for word in sentence.split():
        if '_' in word: count_vncore_compounds += 1
print('Số lượng từ ghép khi tách từ bằng thư viện VNcoreNLP:', count_vncore_compounds)
```

```
Số lượng từ ghép khi tách từ bằng thư viện VNcoreNLP: 1025
```

```
[ ] count_longest_compounds = 0
for sentence in  longest_tokenize_sentences:
    for word in sentence.split():
        if '_' in word: count_longest_compounds += 1
print('Số lượng từ ghép khi tách từ bằng Longest matching:', count_longest_compounds)
```

```
Số lượng từ ghép khi tách từ bằng Longest matching: 907
```

4. Gán nhãn từ loại

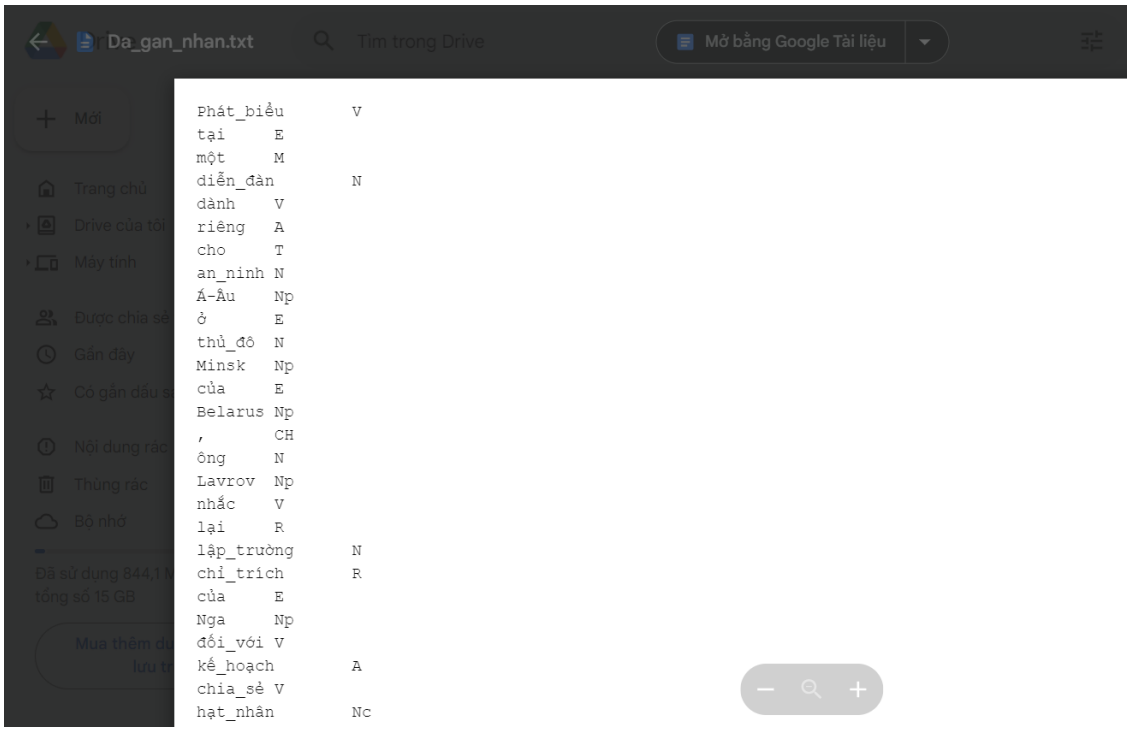
4.1. Danh sách nhãn gán

Sau khi thực hiện tách từ thủ công, nhóm sẽ dùng bộ dữ liệu này để tiến hành phân công gán nhãn từ loại thủ công, dựa trên từ điển https://vtudien.com/ . Phát biểu bài toán: Đầu vào là một câu hay đoạn văn bản tiếng Việt đã tách từ, đầu ra là tập nhãn phù hợp nhất cho các từ trong câu hoặc đoạn văn bản đó.

STT	Nhãn	Tên nhãn	Ví dụ
1	N	Danh từ	đồng bằng, đường sắt, khoa học, …
2	Np	Danh từ riêng	Việt Nam, Đức, Trị An,…
3	Nc	Danh từ phân loại	con, cái, ngôi, đứa, tấm, …
4	Nu	Danh từ đơn vị	mét, cân, xu, đồng, USD
5	Ny	Danh từ viết tắt	HCV, THCS, THPT, IELTS, …

6	V	Động từ	nghiên cứu, có, tuyển, viết, đọc, bơi lội, …
7	A	Tính từ	tốt, xấu, đẹp, cao, thấp, …
8	P	Đại từ	tôi, tớ, ta, chúng tôi, chúng ta, …
9	R	Phó từ	đã, cũng, sẽ, chẳng, chưa,…
10	M	Số từ	0, 1, 2.000, một, trăm, triệu, …
11	L	Định từ	những, mỗi, từng, mấy,…
12	E	Giới từ	trên, dưới, trong, ngoài, trừ,…
13	C	Liên từ	vì vậy, tuy nhiên, ngược lại, …
14	Cc	Liên từ đẳng lập	và, hoặc, cùng, …
15	Nb	Từ mượn nước ngoài	Tivi, website,..
16	T	Trợ từ	ngay, chính, thì, là, chỉ, …
17	X	Không xác định	
18	Z	Yếu tố cấu tạo từ	bất, vô, phi
19	CH	Nhãn cho các loại dấu	.!?,(){}/\:";[]

Kết quả gán nhãn thủ công được ghi lại trong file Da_gan_nhan.txt.



Thông tin bộ dữ liệu:

- Các âm tiết của từ ghép được phân cách bởi dấu gạch dưới "_"
- Nhãn được phân tách với từ bởi dấu tab "\t"
- Mỗi dòng là 1 từ kèm nhãn của nó
- Mỗi câu được ngăn cách bởi một dòng trống "\n"
- Số lượng câu: 147
- Số lượng từ kèm nhãn: 3719

4.2. Tiền xử lý

- Nhóm thực hiện phân chia tập dữ liệu thành tập train và test theo tỷ lệ 7:3.

```
[ ]  train_gold = open('/content/drive/MyDrive/Do_an_CS211/data/train_gold.txt', encoding='utf-8').readlines()
     print('Số lượng từ trong tập train_gold:', len(train_gold))
     train_gold[0:5]
```

```
Số lượng từ trong tập train_gold: 2849
['Phát_biểu\tV\n', 'tại\tE\n', 'một\tM\n', 'diễn_đàn\tN\n', 'dành\tV\n']
```

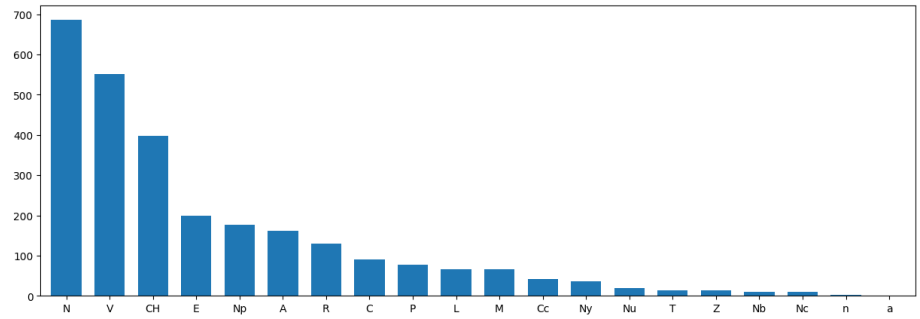
```
[ ]  test_gold = open('/content/drive/MyDrive/Do_an_CS211/data/test_gold.txt', encoding='utf-8').readlines()
     print('Số lượng từ trong tập test_gold:', len(test_gold))
     test_gold[0:5]
```

```
Số lượng từ trong tập test_gold: 870
['phân_bổ\tN\n', 'số\tN\n', 'tiền\tN\n', 'hơn\tA\n', '100\tM\n']
```

- Sau đó thực hiện insights để kiểm tra các từ vựng ở trong tập train và test không có trong tập dữ liệu từ điển dictionary, và nhận ra rằng phần lớn nằm trong nhãn Np và Ny. Các nhãn unknown này có thể ảnh hưởng đến kết quả dự đoán.

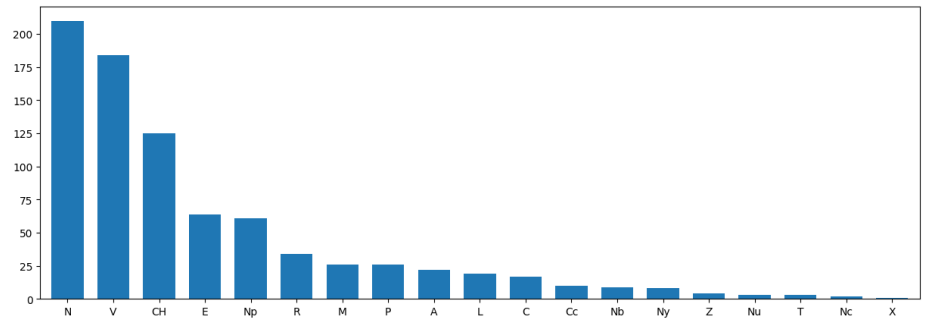
```
[ ] print('Các từ không nằm trong vocabs', end=': ')
    k = 0
    for word_tag, word in zip(train_gold, train_words):
        if word == '--unk--':
          k += 1
          print(word_tag.split()[0], end=', ')
    plot_tag_counts(train_gold)
    print('\n')
    print(k)
```

```
Các từ không nằm trong vocabs: Phát_biểu, Á-Âu, Minsk, Belarus, Lavrov, khóa, Ngoại_trưởng, Lavrov, đáp_trả, dọa, bom_hạt_nhân, Bỉ, Hà_Lan, Thổ_Nhĩ_Kỳ, Một_số, Ba_Lan, khóa, Moskva, Ngoại_trưởng, Lavrov, phương_Tây, Ngoại_trưởng, La

290
```




```
[ ] print('Các từ không nằm trong vocabs', end=': ')
    k = 0
    for word_tag, word in zip(test_gold, test_words):
        if word == '--unk--':
          k += 1
          print(word_tag.split()[0], end=', ')
    plot_tag_counts(test_gold)
    print('\n')
    print(k)
```

```
Các từ không nằm trong vocabs: MTTQ, Nghị_định, 2021, Đến, Chia_sẻ, Cựu, Thủ_tướng, Mykola_Azarov, Chẳng_hạn, Ngoại_trưởng, Antony_Blinken, Trước_tiên, hủy, Tiếp, Moskva, Và, Kiev, Thủ_tướng, Ukraine_Azarov, Kiev, Kiev, 0, Azarov, s

95
```

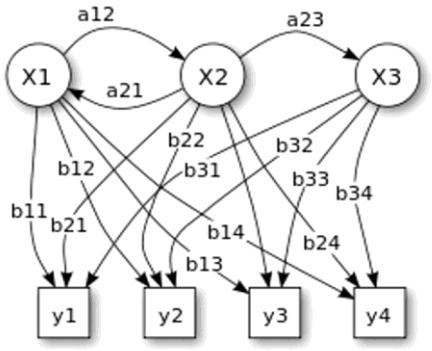


4.3. Mô hình Hidden Markov kết hợp với thuật toán Viterbi

4.3.1. Tổng quan về mô hình Hidden Markov

- HMM (Hidden Markov Models) là một trong những thuật toán được sử dụng phổ biến nhất trong Xử lý ngôn ngữ tự nhiên và là nền tảng cho nhiều kỹ thuật học sâu. Ngoài gán nhãn từ loại, HMM còn được dùng để nhận dạng giọng nói, tổng hợp giọng nói, …
- Mô hình Markov sử dụng ma trận chuyển trạng thái A (Transition Matrix). Mô hình Markov ẩn thêm một ma trận thể hiện B (Emission Matrix) mô tả xác suất của một quan sát có thể nhìn thấy khi ta ở một trạng thái cụ thể. Trong trường hợp này, các quan sát là các từ. Trạng thái, thứ được xem là ẩn (Hidden) chính là nhãn của từ đó.

Hidden Markov Model



- Nhóm sẽ triển khai xây dựng Transition Matrix A và Emission Matrix B, dựa vào các từ điển transition_counts, emission_counts và tag_counts.
  + Từ điển transition_counts thể hiện số lần chuyển đổi giữa các nhãn.
  + Tử điển emission_counts thể hiện số lần thể hiện nhãn của từ loại.
  + Từ điển tag_counts thể hiện số lần xuất hiện của nhãn trong tập test.

```
[ ]  print("Transition examples: ")
     for example in list(transition_counts.items())[:15]:
         print(example)
```

```
Transition examples:
(('--s--', 'V'), 16)
(('V', 'E'), 50)
(('E', 'M'), 9)
(('M', 'N'), 41)
(('N', 'V'), 127)
(('V', 'A'), 47)
(('A', 'T'), 1)
(('T', 'N'), 4)
(('N', 'Np'), 75)
(('Np', 'E'), 13)
(('E', 'N'), 90)
(('E', 'Np'), 16)
(('Np', 'CH'), 78)
(('CH', 'N'), 84)
(('Np', 'V'), 43)
```

```
[ ]  print("Emission examples: ")
     for example in list(emission_counts.items())[:15]:
         print (example)
```

```
Emission examples:
(('V', '--unk--'), 21)
(('E', 'tại'), 6)
(('M', 'một'), 18)
(('N', 'diễn_đàn'), 1)
(('V', 'dành'), 2)
(('A', 'riêng'), 3)
(('T', 'cho'), 8)
(('N', 'an_ninh'), 5)
(('Np', '--unk--'), 137)
(('E', 'ở'), 22)
(('N', 'thủ_đô'), 1)
(('E', 'của'), 32)
(('CH', ','), 188)
(('N', 'ông'), 8)
(('V', 'nhắc'), 1)
```

```
[ ]  tag_counts
```

```
defaultdict(int,
            {'V': 551,
             'E': 199,
             'M': 66,
             'N': 687,
             'A': 162,
             'T': 13,
             'Np': 176,
             'CH': 398,
             'R': 130,
             'Nc': 9,
             'Ny': 36,
             '--s--': 100,
             'Cc': 41,
             'L': 67,
             'P': 78,
             'C': 90,
             'Z': 13,
             'n': 3,
             'a': 1,
             'Nu': 19,
             'Nb': 10})
```

- Transition Matrix A: với mỗi phần tử a_{ij} thể hiện xác suất chuyển từ nhãn i đến nhãn j với ràng buộc $\sum_{j=1}^{n} a_{ij} = 1$ (Tổng của mỗi hàng = 1) với mọi i và $a_{ij} \geq 0$ với mọi i, j. Xác suất này được tính theo công thức sau:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i) + \alpha}{C(t_{i-1}) + \alpha * N}$$

Trong đó:

- N: tổng số nhãn

- $C(t_{i-1}, t_i)$: số lượng bộ (prev_tag, tag) trong transition_counts
- $C(t_{i-1})$: số lượng nhãn prev_tag trong tag_counts
- α: tham số làm mịn (smoothing), ở đây lấy α = 0.1

```
#Hàm tạo ma trận chuyển đổi trạng thái A (từ transition_counts)
def create_transition_matrix(alpha, tag_counts, transition_counts):
    all_tags = sorted(tag_counts.keys())
    num_tags = len(all_tags)

    A = np.zeros((num_tags, num_tags))
    trans_keys = set(transition_counts.keys())

    for i in range(num_tags):
        for j in range(num_tags):
            count = 0
            key = (all_tags[i], all_tags[j])
            if key in transition_counts:
              count = transition_counts[key]

            count_prev_tag = tag_counts[all_tags[i]]
            A[i, j] = (count + alpha) / (count_prev_tag + alpha * num_tags)
    return A


alpha = 0.1 #Laplace smoothing
for i in range(len(states)):
  tag_counts.pop(i, None)

A = create_transition_matrix(alpha, tag_counts, transition_counts)
df = pd.DataFrame(
    A[:10, :10],
    index = states[:10],
    columns = states[:10]
)

df
```

	--s--	A	C	CH	Cc	E	L	M	N	Nb
--s--	0.000979	0.049951	0.098923	0.059745	0.000979	0.079334	0.030362	0.030362	0.363369	0.020568
A	0.000609	0.043266	0.055454	0.329677	0.037172	0.128580	0.024985	0.049360	0.122486	0.000609
C	0.001086	0.011944	0.055375	0.174810	0.001086	0.077090	0.011944	0.022801	0.218241	0.001086
CH	0.247688	0.035241	0.047738	0.072732	0.012747	0.045239	0.015246	0.015246	0.210197	0.005249
Cc	0.002320	0.048724	0.002320	0.025522	0.002320	0.002320	0.095128	0.002320	0.187935	0.025522
E	0.000497	0.020388	0.005470	0.045251	0.005470	0.015415	0.060169	0.045251	0.448036	0.010443
L	0.001447	0.001447	0.015919	0.001447	0.001447	0.001447	0.015919	0.001447	0.811867	0.015919
M	0.001468	0.001468	0.001468	0.074890	0.001468	0.001468	0.001468	0.060206	0.603524	0.001468
N	0.001596	0.095922	0.019010	0.124946	0.014657	0.097373	0.010303	0.017559	0.214918	0.000145
Nb	0.008264	0.008264	0.008264	0.173554	0.008264	0.008264	0.008264	0.008264	0.173554	0.090909

- Emission Matrix B: với mỗi phần tử b_{ij} thể hiện xác suất một từ ở vị trí i được gán nhãn j. Xác suất này được tính theo công thức sau:

$$P(w_i|t_i) = \frac{C(t_i, w_i) + \alpha}{C(t_i) + \alpha * N}$$

Trong đó:

- N: số lượng từ trong từ điển
- $C(t_i, w_i)$: số lượng cặp (tag, word) thứ i trong emission_counts
- $C(t_i)$: số lượng nhãn thứ i trong tag_counts
- α: tham số làm mịn (smoothing), ở đây lấy α = 0.1

```
[ ]  #Xây dựng ma trận thể hiện / phát xạ B (dựa trên emission_counts)
     def create_emission_matrix(alpha, tag_counts, emission_counts, vocabs):
         all_tags = sorted(tag_counts.keys())
         num_tags = len(tag_counts)
         num_words = len(vocabs)

         B = np.zeros((num_tags, num_words))
         emis_keys = set(list(emission_counts.keys()))

         for i in range(num_tags):
             for j in range(num_words):
                 count = 0
                 key = (all_tags[i], vocabs[j])
                 if key in emission_counts.keys():
                   count = emission_counts[key]

                 count_tag = tag_counts[all_tags[i]]
                 B[i, j] = (count + alpha) / (count_tag + alpha * num_words)
         return B

     cidx  = ['Đồng_bằng', 'sông', 'Cửu_Long', 'được', 'ưu_tiên', 'bố_trí', 'vốn', 'đầu_tư', 'các', 'tuyến', 'cao_tốc', '--unk--']
     rvals = ['N', 'V', 'CH', 'Cc', 'A', 'L', 'Np', 'Nc', 'CH']
     cols = [vocabs_dict[word] for word in cidx]
     rows = [states.index(tag) for tag in rvals]

     for i in range(len(states)):
       tag_counts.pop(i, None)
     B = create_emission_matrix(alpha, tag_counts, emission_counts, list(vocabs_dict))

     df = pd.DataFrame(B[np.ix_(rows, cols)], index=rvals, columns=cidx)
     df
```

	Đồng_bằng	sông	Cửu_Long	được	ưu_tiên	bố_trí	vốn	đầu_tư	các	tuyến	cao_tốc	--unk--
N	0.000016	0.000016	0.000016	0.000016	0.000016	0.000178	0.000178	0.000178	0.000016	0.000016	0.000016	0.011006
V	0.000017	0.000017	0.000017	0.004974	0.000017	0.000182	0.000017	0.000017	0.000017	0.000017	0.000017	0.003487
CH	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000865
Cc	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018
A	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000724
L	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.006304	0.000018	0.000018	0.000198
Np	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.024151
Nc	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000018	0.000381
CH	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000017	0.000865

4.3.2. Tổng quan về thuật toán Viterbi

- Thuật toán Viterbi là phương pháp để ước lượng xác suất chuỗi trạng thái cực đại xác suất của mô hình đưa ra dãy các quan sát.
- Nhóm sẽ tiến hành sử dụng thuật toán Viterbi kết hợp với hai ma trận A và B của mô hình Hidden Markov để tiến hành xác định chuỗi trạng thái ẩn, sử dụng chiến lược quy hoạch động. Quy trình được chia thành 3 bước: Khởi tạo, Forward, Backward.
- Bước 1: Khởi tạo

- Khởi tạo hai ma trận cùng chiều:
  + best_probs: mỗi phần tử chứa xác suất đi từ một nhãn sang một từ.
  + best_paths: ma trận giúp tìm đường đi tốt nhất.
- Cả hai ma trận đều được khởi tạo bằng 0, trừ cột đầu tiên của best_probs sẽ được khởi tạo với giả định rằng từ đầu tiên của ngữ liệu được đặt trước bởi một ký tự bắt đầu (‘--s--’).
- Mã giả:
  if A[s_idx, i - 1] != 0:
  best_probs[i, 0] = ln(A[s_idx, i]) + ln(B[i, index]) (*)
  else:
  best_probs[i, 0] = float(“-inf”)
- Ở đây s_idx là chỉ số của nhãn ‘--s--’ trong tập trạng thái (s_idx = 0 theo hình 4.6), index là chỉ số của từ đầu tiên trong corpus xuất hiện trong từ điển. Corpus được sử dụng là tập train_words và test_words.
- Phép tính ở (*) thực chất là phép nhân xác suất:

  *best_probs[i, 0] = A[s_idx, i] * B[i, index]*

- Các giá trị xác suất trong 2 ma trận A và B có giá trị rất nhỏ, khi nhân lại sẽ càng nhỏ. Để tránh điều này, ta dùng logarit tự nhiên đưa chúng thành tổng 2 log.

```
[ ]  best_probs_train, best_paths_train = viterbi_initialize(states, tag_counts, A, B, train_words, vocabs_dict)
     print('best_probs_train[0, 0]:', best_probs_train[0, 0])
     print('best_paths_train[0, 0]:', best_paths_train[0, 0])
```

```
best_probs_train[0, 0]: -17.844316952902375
best_paths_train[0, 0]: 0
```

```
[ ]  best_probs_train.shape
```

```
(21, 2849)
```

```
[ ]  best_paths_train.shape
```

```
(21, 2849)
```

```
[ ]  best_probs_test, best_paths_test = viterbi_initialize(states, tag_counts, A, B, test_words, vocabs_dict)
     print('best_probs_test[0, 0]:', best_probs_test[0, 0])
     print('best_paths_test[0, 0]:', best_paths_test[0, 0])
```

```
best_probs_test[0, 0]: -17.844316952902375
best_paths_test[0, 0]: 0
```

```
[ ]  best_probs_test.shape
```

```
(21, 870)
```

```
[ ]  best_paths_test.shape
```

```
(21, 870)
```

- Bước 2: Forward

- Tiến hành điền vào hai ma trận best_probs và best_paths đã khởi tạo.
- Triển khai:

+ Duyệt tất cả từ trong corpus (dùng biến i), trừ từ đầu tiên.

+ Với mỗi từ, duyệt tất cả nhãn có thể của từ đó (dùng biến j).

+ Duyệt tất cả nhãn có thể của từ đứng trước nó (dùng biến k).

+ Với mỗi k, tính xác suất để từ hiện tại có nhãn j và từ trước đó có nhãn k:

prob = best_probs[k, i - 1] + log(A[k, j]) + log(B[j, index])

Trong đó:

- best_probs[k, i -1]: Xác suất lớn nhất để từ trước nó có nhãn k
- A[k, j]: Xác suất để nhãn j xuất hiện sau nhãn k
- B[j, index]: Xác suất từ đang xét được gán nhãn j

+ Cập nhật best_probs[j, i] và best_paths[j, i]

+ Trả về hai ma trận best_probs và best_paths

- Mã giả:

```
def viterbi_forward(A, B, corpus, best_probs, best_paths, vocabs_dict):
    num_tags = best_probs.shape[0] #Hoặc = len(tag_counts)
for i in range(1, len(corpus)):        for j in range(num_tags):
        best_prob_i = float('-inf')
        best_path_i = Null
for k in range(num_tags):
        index = vocabs[corpus[i]] #Ví trị từ thứ i của corpus trong vocabs
prob = best_probs[k, i - 1] + log(A[k, j - 1]) + log(B[j - 1, index])

if prob > best_prob_i:
```

best_prob_i = prob

best_path_i = k

best_probs[j, i] = best_prob_i

best_paths[j, i] = best_path_i

return best_probs, best_paths

```
[ ] best_probs_train, best_paths_train = viterbi_forward(A, B, train_words, best_probs_train, best_paths_train, vocabs_dict)
    print('best_probs_train[0, 1]:', best_probs_train[0, 1])
    print('best_paths_train[0, 4]:', best_paths_train[0, 4])

    best_probs_train[0, 1]: -22.87754878624839
    best_paths_train[0, 4]: 8
```

```
[ ] best_probs_test, best_paths_test = viterbi_forward(A, B, test_words, best_probs_test, best_paths_test, vocabs_dict)
    print('best_probs_test[0, 1]:', best_probs_test[0, 1])
    print('best_paths_test[0, 4]:', best_paths_test[0, 4])

    best_probs_test[0, 1]: -28.95774541385704
    best_paths_test[0, 4]: 1
```

- Bước 3: Backward

- Sử dụng best_probs và best_paths trả về danh sách các nhãn được dự đoán cho mỗi từ trong corpus.
- Triển khai:
  + Duyệt tất cả nhãn của từ cuối cùng trong best_probs, tìm nhãn có giá trị lớn nhất.
  + Dùng best_paths từ vị trí từ cuối cùng, tìm nhãn có giá trị cao nhất cho từ trước nó.
  + Xuất ra mảng kết quả dự đoán nhãn.
- Mã giả cho hàm viterbi_backward:

  def viterbi_backward(best_probs, best_paths,

  corpus, states): m = best_paths.shape[1] #Hoặc =

  *len(corpus) z, pred = [None] * m*

  best_prob_for_last_word = float('-inf') num_tags

  = best_probs.shape[0] #Hoặc = len(tag_counts)

  for k in range(num_tags): if best_probs[k, m -

  1] > best_prob_for_last_word:

  best_prob_for_last_word = best_probs[k, m - 1]

  z[m - 1] = k pred[m - 1] = states[z[m - 1]] for i

in range(m - 1, -1, -1): z[i - 1] =

best_paths[z[i], i] pred[i - 1] = states[z[i - 1]]

return pred

```
[ ] train_pred = viterbi_backward(best_probs_train, best_paths_train, train_words, states)
    test_pred = viterbi_backward(best_probs_test, best_paths_test, test_words, states)
    m = len(test_pred)

    print('Dự đoán cho test_pred[0:20]:')
    print(test_words[:20])
    print(test_pred[:20])
```

```
Dự đoán cho test_pred[0:20]:
['phân_bổ', 'số', 'tiền', 'hơn', '100', 'tỷ', 'đồng', 'hỗ_trợ', 'nạn_nhân', 'vụ', 'cháy', 'chung_cư', 'mini', '.', '--n--', 'Theo', 'lãnh_đạo',
['V', 'N', 'N', 'A', 'M', 'Nu', 'Nu', 'V', 'N', 'N', 'V', 'N', 'A', 'CH', 'N', 'V', 'V', 'N', 'Np', 'Ny']
```

```
[ ] print('Dự đoán cho test_pred[0:20]:')
    print(test_words[:20])
    print(test_pred[:20])
```

```
Dự đoán cho test_pred[0:20]:
['phân_bổ', 'số', 'tiền', 'hơn', '100', 'tỷ', 'đồng', 'hỗ_trợ', 'nạn_nhân', 'vụ', 'cháy', 'chung_cư', 'mini', '.', '--n--', 'Theo', 'lãnh_đạo',
['V', 'N', 'N', 'A', 'M', 'Nu', 'Nu', 'V', 'N', 'N', 'V', 'N', 'A', 'CH', 'N', 'V', 'V', 'N', 'Np', 'Ny']
```

4.3.3. Kết quả

Kết quả của mô hình Hidden Markov kết hợp thuật toán Viterbi trên tập train:

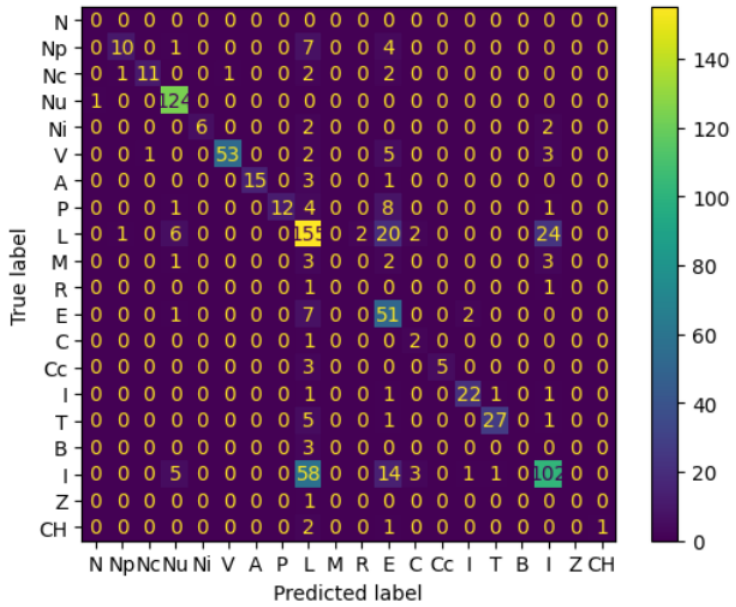
	precision	recall	f1-score	support
A	0.99	0.86	0.92	162
C	0.97	0.74	0.84	90
CH	1.00	0.99	0.99	398
Cc	1.00	1.00	1.00	41
E	0.95	0.93	0.94	199
L	0.97	0.97	0.97	67
M	1.00	0.88	0.94	66
N	0.83	0.96	0.89	687
Nb	1.00	0.50	0.67	10
Nc	1.00	0.22	0.36	9
Np	0.73	0.83	0.77	176
Nu	1.00	0.89	0.94	19
Ny	1.00	0.47	0.64	36
P	0.96	0.94	0.95	78
R	1.00	0.88	0.93	130
T	0.00	0.00	0.00	13
V	0.94	0.93	0.94	551
Z	1.00	0.54	0.70	13
a	0.00	0.00	0.00	1
n	0.00	0.00	0.00	3
accuracy			0.91	2749
macro avg	0.82	0.68	0.72	2749
weighted avg	0.91	0.91	0.91	2749

Kết quả của mô hình Hidden Markov kết hợp thuật toán Viterbi trên tập test:

	precision	recall	f1-score	support
--s--	0.00	0.00	0.00	0
A	0.83	0.45	0.59	22
C	0.92	0.65	0.76	17
CH	0.89	0.99	0.94	125
Cc	1.00	0.60	0.75	10
E	0.98	0.83	0.90	64
L	1.00	0.79	0.88	19
M	1.00	0.46	0.63	26
N	0.60	0.74	0.66	210
Nb	0.00	0.00	0.00	9
Nc	0.00	0.00	0.00	2
Np	0.46	0.84	0.60	61
Nu	0.29	0.67	0.40	3
Ny	1.00	0.62	0.77	8
P	0.88	0.85	0.86	26
R	0.93	0.79	0.86	34
T	0.00	0.00	0.00	3
V	0.74	0.55	0.63	184
X	0.00	0.00	0.00	1
Z	1.00	0.25	0.40	4
accuracy			0.72	828
macro avg	0.63	0.50	0.53	828
weighted avg	0.75	0.72	0.72	828

```
[ ] from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
    tag_list = ['N', 'Np', 'Nc', 'Nu', 'Ni', 'V', 'A', 'P', 'L', 'M', 'R', 'E', 'C', 'Cc', 'I', 'T', 'B', 'I', 'Z', 'CH']
    cm = confusion_matrix(y_true_test, y_pred)
    ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=tag_list).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7e886f151ed0>
```



4.3.4. Đánh giá

• Accuracy của thuật toán là 72%, không tệ so với bộ dữ liệu dành cho Tiếng Việt. Nhưng mô hình HMM + Viterbi chưa thể dự đoán chính xác các nhãn Np (danh từ riêng) và Ny (danh từ viết tắt).

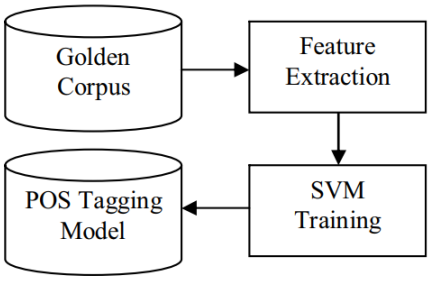
• Theo hướng dẫn của anh Nguyễn Đức Vũ, thì để giải quyết vấn đề trên cần xây dựng bộ từ điển riêng cho vấn đề này. Nhưng trong khuôn khổ môn học việc thu thập bộ dữ liệu riêng cho 2 nhãn này khá mất thời gian, nên cần có tính toán phù hợp.

4.4. Trích xuất đặc trưng và mô hình Support Vector Machine

4.4.1. Tổng quát

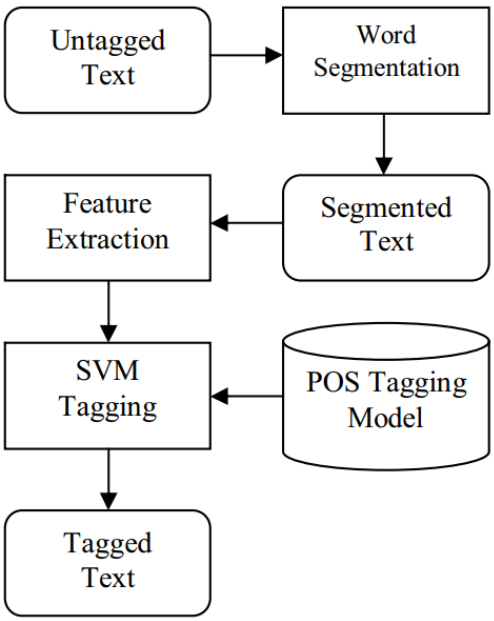
Phương pháp này sử dụng tận train và test tương tự như phương pháp Hidden Markov kết hợp với thuật toán Viterbi.

Quá trình training



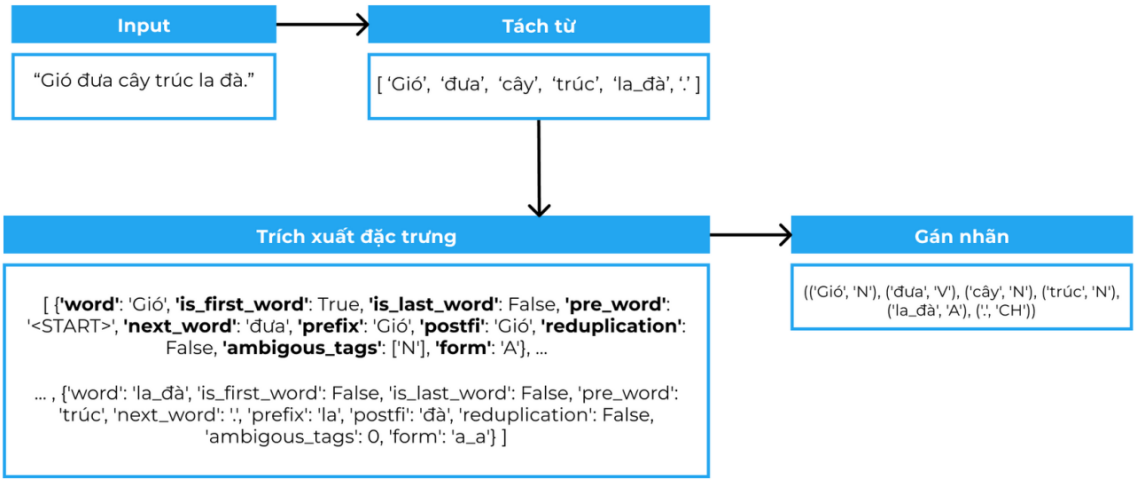
Dữ liệu tập train sẽ được đưa đi trích xuất đặc trưng. Với mỗi từ sẽ được biến thành một vector đặc trưng. Sau đó dùng tập vector này để train mô hình SVM. Cuối cùng là dùng mô hình đã train để thực hiện gán nhãn.

Quá trình gán nhãn



Input văn bản đầu vào đầu tiên sẽ được Tách từ, ở đây nhóm chúng em sử dụng thưu viện VNcoreNLP để thực hiện tách từ. Sau đó là bước trích xuất đặc trưng và gán nhãn từ loại.

Minh họa quá trình gán nhãn



4.4.2. Trích xuất đặc trưng

Bước này giúp biến văn bản từ input đầu vào thành một tập các vector đặc trưng để mô hình SVM có thể xử lý.

Những đặc trưng chúng ta trích xuất có thể chia thành 2 nhóm:

Những đặc điểm cơ bản:

- Từ hiện tại: đây là đặc trưng dễ dàng nhất, chính là lấy từ ngữ hiện tại làm một đặc trưng.
- Ngữ cảnh: từ trước và sau từ hiện tại.
- Đặc trưng không rõ ràng (Ambiguous features): được lấy từ một từ điển nhãn gán, tự điển sẽ cung cấp một danh sách các loại nhãn gán mà từ đó có thể có. Giúp giới hạn lại số lượng nhãn gán cần dự đoán cho một từ. Từ điển có thể được trích xuất từ dữ liệu training.
- Đặc trưng chính tả: cho biết từ đó được viết hoa như thế nào, nó là từ đơn hay từ ghép.

Những đặc trưng đặc biệt:

Thường xuất hiện ở tiếng Việt, một số tiếng châu Á và không có ở tiếng Anh.

- Sao chép (Reduplication): trong tiếng Việt có hệ thống các từ láy hoàn toàn hoặc từ láy chứa các từ rất tương đồng.

Ví dụ:

"xanh xanh" nó sẽ mang ý nghĩa là màu xanh nhẹ.

"mạnh mẽ".

“người người” sẽ đông người hơn là một từ “người”.

- Tiền tố và hậu tố: nhiều từ ghép chứa tiền tố và hậu tố đặc biệt trong tiếng Việt.

Ví dụ:

“bán” là một nửa, “nguyệt” là mặt trăng => “bán nguyệt” là trăng khuyết.

Tiền tố “phi” khiến từ ghép chứa nó mang ý nghĩa phủ định: “phi chính phủ”, “phi nghĩa”, “phi lý”.

Hậu tố “gia” biến cụm từ thành một chức danh: “triết học” => “triết học gia”.

Hậu tố “niềm” biến cụm từ thành danh từ: “vui” là động từ => “niềm vui” là danh từ.

Ví dụ trích xuất đặc trưng:

WORD	AMBIGOUS TAGS	ORTHO -GRAPHIC	ORTHO -GRAPHIC FORM	PREFIX	POSTFI X	REDUPLI -CATION	TAG
Bình thường	J	FirstCap	A_a	bình	thường	No	J
không	R-Q-J-N	Letters	a	không	không	No	R
người	N	Letters	a	người	người	No	N
đàn ông	N	Letters	a_a	đàn	ông	No	N
nào	P-M-U-R	Letters	a	nào	nào	No	P
gọi	V	Letters	a	gọi	gọi	No	V
vợ	N	Letters	a	vợ	vợ	No	N
như	C-R	Letters	a	như	như	No	C
thế	P-M-N	Letters	a	thế	thế	No	P
.	.	Punctuation	_	.	.	No	.

Với câu input: “Bình thường không người đàn ông nào gọi vợ như thế.”

Sau khi tách từ: “Bình_thường không người đàn_ông nào gọi vợ như thế.”

Với mỗi từ đã được tách từ, chúng ta có một vector đặc trưng với 7 thuộc tính như sau:

- Word: từ hiện tại (“Bình_thường”)
- Ambigous tags: các nhãn gán được dự đoán có thể có của từ hiện tại, thuộc tính này được lập dựa trên một từ điển nhãn gán là một đối tượng thuộc kiểu dictionary với mỗi thẻ là một từ và trong đó là một list các nhãn gán được tìm thấy của từ đó trong tập dữ liệu train (đối với từ “Bình_thường” nhãn dự đoán chỉ có “J”)
- Ortho graphic: cho biết từ hiện tại có phải là năm đầu câu hay cuối câu không (“Bình_thường” là từ đầu câu)
- Form: cho biết hình thái của từ hiện tại (“Bình _thường” là từ đầu câu nên được viết hoa “A_a”)
- Prefix: cho biết tiền tố của từ (“Bình_thường” có tiền tố “bình”)
- Postfix: cho biết hậu tố của từ (“Bình_thường” có hậu tố “thường”)
- Reduplication: cho biết từ có phải từ láy hoàn toàn không.

Thực hiện cài đặt hàm trích xuất đặc trưng:

```
def feature_extraction(sentence):
  features = []
  for word in sentence:
    i = sentence.index(word)
    feature = {
        'word': word,
        'is_first_word': i == 0,
        'is_last_word': i == len(sentence) - 1,
        'pre_word': sentence[i-1] if i > 0 else '<START>',
        'next_word': sentence[i+1] if i < len(sentence)-1 else '<END>',
        'prefix': refix(word),
        'postfi': postfi(word),
        'reduplication': reduplicate(word),
        'ambigous_tags': ambigous(word),
        'form': form_of_word(word)
    }
    features.append(feature)
  return features #một list các dict
```

Hàm tìm từ bắt đầu của từ ghép

```
[ ]  def refix(word):
       if "_" in word:
         i = word.index('_')
         return word[0:i]
       else: return word
```

Hàm tìm từ cuối cùng của từ ghép

```
[ ]  def postfi(word):
       if "_" in word:
         i = word.rfind('_')
         return word[i+1:len(word)]
       else: return word
```

Hàm xác nhận từ là từ láy hoàn toàn

```
[ ]  def reduplicate(word):
       word = word.lower()
       if word.count("_") == 1: #từ 2 âm tiết giống i nhau
         i = word.index('_')
         if word[0:i] == word[i+1:len(word)]:
           return True
       return False
```

Hàm tìm trong Dictionary

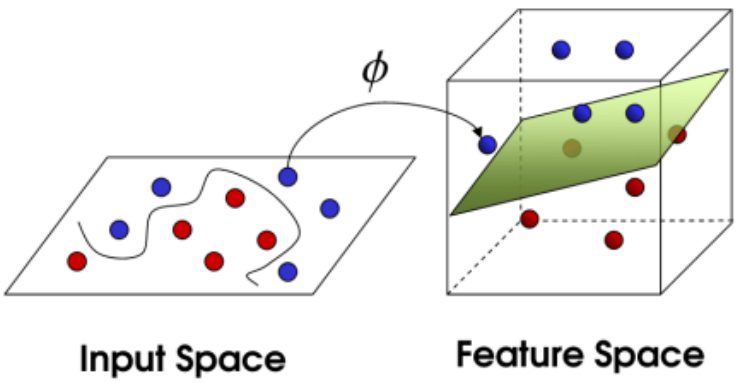
```
[ ]  def ambigous(word):
       word = word.lower()
       if word in dictionary:
         return dictionary[word]
       else: return 0
```

4.4.3. Mô hình Support Vector Machine

Giới thiệu:

Support Vector Machine (SVM) là thuật toán học có giám sát dùng trong các bài toán phân lớp (Classification) hoặc hồi quy (Reggression).

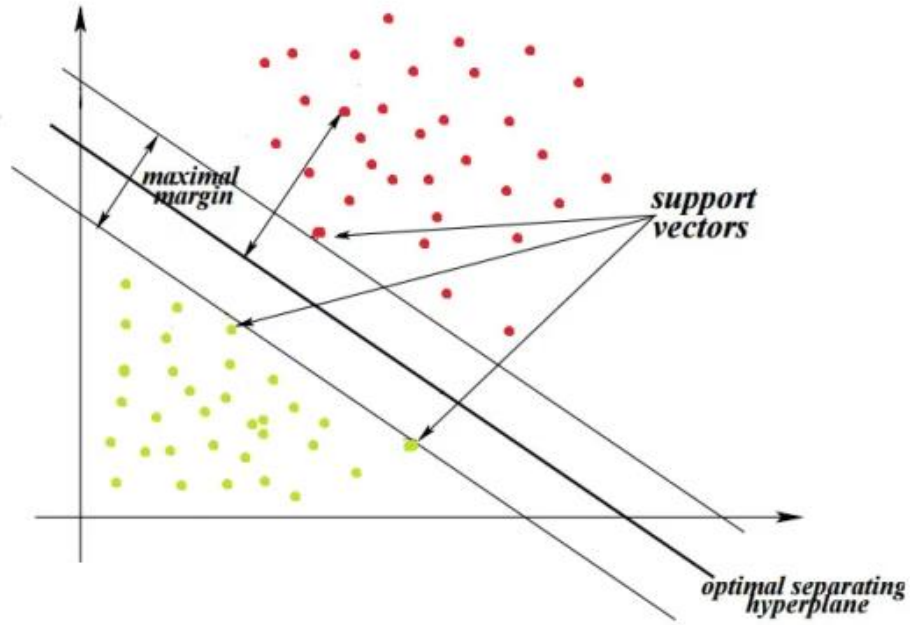
Mục tiêu của SVM là tìm ra siêu phẳng có kích thước n-1 chiều trong không gian N chiều để chia dữ liệu thành các phần tương ứng với lớp của chúng.



Các điểm dữ liệu nằm trên hoặc gần nhất với siêu phẳng được gọi là support vector, chúng ảnh hưởng đến vị trí và hướng của siêu phẳng.

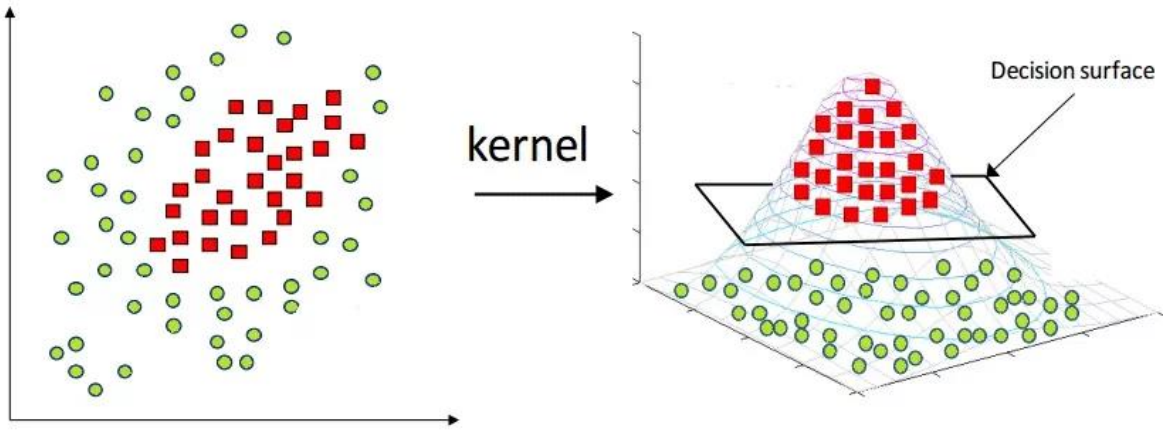
Các vector này được sử dụng để tối ưu hóa lề và nếu xóa các điểm này, vị trí của siêu phẳng sẽ thay đổi.

Một điểm lưu ý nữa đó là các support vector phải cách đều siêu phẳng.



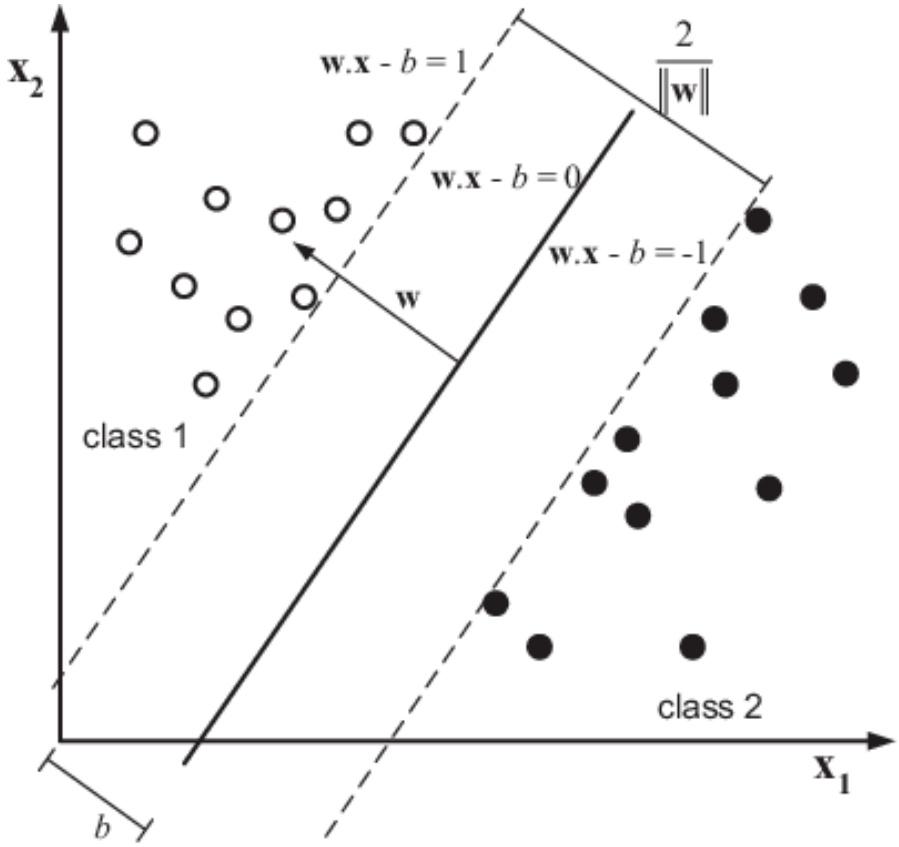
Có 2 phương pháp để giải quyết đối với dữ liệu phi tuyến tính là lề mềm (soft margin) và thủ thuật kernel.
Trong đó thủ thuật kernel là phương án tối ưu hơn.

Có 2 phương pháp để giải quyết đối với dữ liệu phi tuyến tính là lề mềm (soft margin) và thủ thuật kernel.
Trong đó thủ thuật kernel là phương án tối ưu hơn.



Siêu phẳng được biểu diễn bởi hàm số W.X - b = 0.
Để tìm siêu phẳng tối ưu nhất, ta cần tìm được 2 siêu phẳng lề H1, H2 cách đều H0 có dạng:
H1: W.X + b = -1
H2: W.X + b = 1
Khoảng cách từ H1, H2 tới siêu phẳng H0 gọi là mức lề m (margin) được tính trong bài toàn tối ưu SVM có dạng:

$$\text{Margin} = \min_i \left(\frac{y_i (w \cdot x_i - b)}{||w||} \right)$$



Ưu nhược điểm:

Ưu điểm:

Hiệu suất tốt đối với tập dữ liệu phức tạp và dữ liệu lớn.

Có khả năng tích hợp kernel để xử lý dữ liệu phi tuyến tính.

Nhược điểm:

Dễ bị nhiễu bởi tập dữ liệu có margin quá nhỏ.

Khó khăn trong việc tính toán và tốc độ xử lý chậm.

Cài đặt:

```
[ ]  from sklearn.feature_extraction import DictVectorizer
     from sklearn.svm import SVC
     from sklearn.pipeline import Pipeline
```

```
[ ]  vectorizer = DictVectorizer(sparse=False)

     # Fit the vectorizer and transform the features
     X_train = vectorizer.fit_transform(X_train)

     # Create an SVM model
     svm_model = SVC(kernel='linear')

     # Train the SVM model
     svm_model.fit(X_train, Y_train)
```

Kết quả:

Độ đo Accuracy đạt được trên tâọ test là 0.7581. Đây là một kết quả khá tốt và ấn tượng.

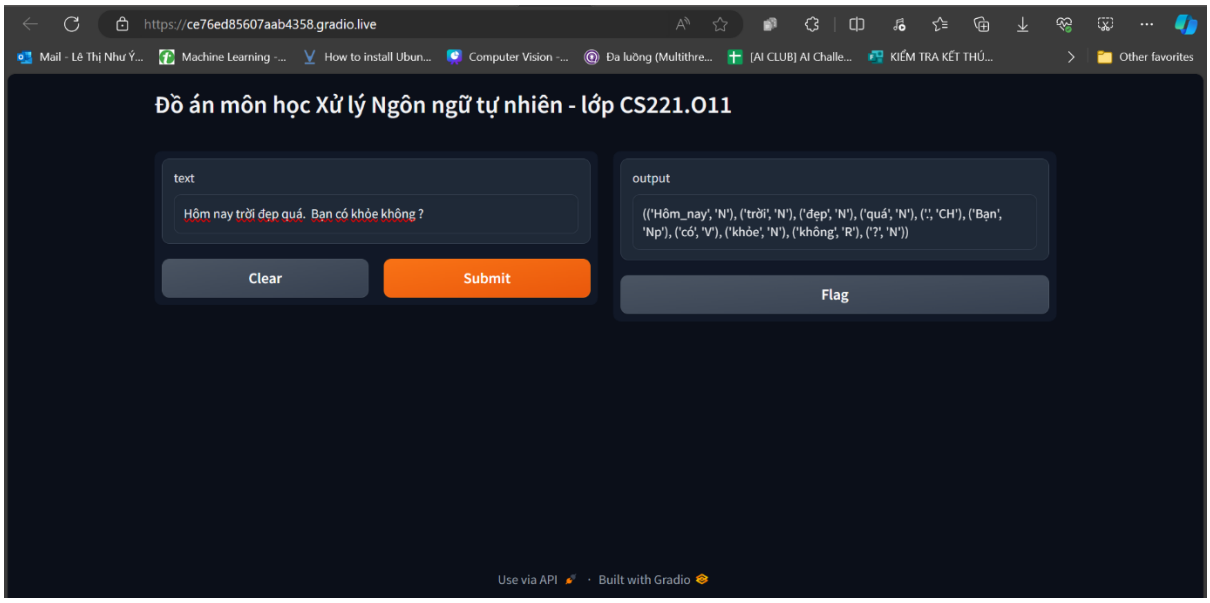
```
Accuracy on test set: 0.7581
Classification Report on Test Set:
              precision    recall  f1-score   support

      1.00       1.00      1.00        38
           A       0.57      0.42      0.48        19
           C       0.78      0.82      0.80        17
          CH       0.58      1.00      0.73       117
          Cc       1.00      0.70      0.82        10
           E       0.96      0.89      0.92        62
           L       1.00      0.79      0.88        19
           M       1.00      0.68      0.81        22
           N       0.66      0.74      0.70       188
          Nb       0.00      0.00      0.00         9
          Nc       0.00      0.00      0.00         2
          Np       0.75      0.90      0.82        58
          Nu       0.00      0.00      0.00         1
          Ny       1.00      1.00      1.00         6
           P       1.00      0.92      0.96        25
           R       1.00      0.84      0.91        31
           T       1.00      0.33      0.50         3
           V       0.96      0.52      0.68       166
           X       0.00      0.00      0.00         1
           Z       1.00      0.50      0.67         4

    accuracy                           0.76       798
   macro avg       0.71      0.60      0.63       798
weighted avg       0.80      0.76      0.75       798
```

5. Kết luận

Trong đề tài này, nhóm đã trình bày về hai bài toán tách từ tiếng Việt và gán nhãn từ loại tiếng Việt. Đối với bài toán tách từ tiếng Việt, đây là bước tiền xử lý quan trọng cho bài toán gán nhãn từ loại. Nhóm đã tìm hiểu hướng tiếp cận theo từ điển bằng thuật toán Longest Matching, triển khai trên bộ dữ liệu đã thu thập và so sánh kết quả với hai thư viện là VnCoreNLP và gán nhãn thủ công. Từ đó nhóm đưa ra kết luận: thuật toán Longest Matching tuy dễ cài đặt và có thời gian thực thi nhanh, nhưng không thể tách các từ không nằm trong bộ từ điển (đặc biệt là với các tên riêng), đồng thời không giải quyết được vấn đề nhập nhằng. Đối với bài toán gán nhãn từ loại tiếng Việt, nhóm tiến hành tìm hiểu về mô hình Hidden Markov cũng như thuật toán Viterbi, tuy nhiên mô hình vẫn còn phụ thuộc nhiều vào bộ dữ liệu từ vựng và tập dữ liệu train. Vì vậy nhóm đã kết luận mô hình bị overfitting và đưa ra một số hướng cải thiện bằng cách tăng cường dữ liệu train và hoàn thiện đầy đủ hơn bộ từ vựng. Cuối cùng, nhóm tiến hành thử nghiệm gán nhãn từ loại với phương pháp Trích xuất đặc trưng và mô hình Suport Vector Machine (SVM) để so sánh với mô hình HMM. Nhóm kết luận rằng phương pháp này cho kết quả gán nhãn từ loại tốt hơn. Lý dó SVM cho kết quả tốt hơn theo nhóm nhận xét vì SVM là phương pháp đi trực tiếp vào ngữ cảnh của từ, mô hình học được các đặc trưng về từ ngữ thay vì phương pháp HMM chỉ tiếp cần vào xác xuất gán nhãn của từ. Sau khi tìm hiểu đề tài, nhóm đã nắm được kiến thức về bài toán tách từ và gán nhãn từ loại, hiểu được cách hoạt động của mô hình Hidden Markov và thuật toán Viterbi. Từ nền tảng trên, nhóm có thể tiếp tục tìm hiểu các hướng tiếp cận khác cho bài toán này, cũng như tiếp cận các bài toán nâng cao hơn

6. Demo

Nhóm sử dụng https://www.gradio.app/ để thực hiện một web app demo cho phương pháp Trích xuất đặc trưng và SVM. Để chạy demo này, chạy file Demo SVM.ipynb.



7. Bảng phân chia công việc

Thành viên	Công việc	Tỷ lệ công việc
Đỗ Việt Bách	- Tiền xử lý dữ liệu. - Gán nhãn từ loại với mô hình HMM + Viterbi	*33.33%*
Huỳnh Phước Lân	- Gán nhãn từ loại với mô hình SVM - Xây dựng UI	*33.33%*
Lê Thị Như Ý	- Thực hiện tách từ với Longest Matching và VncoreNLP - Gán nhãn từ loại thủ công	*33.33%*

8. Tài liệu tham khảo

[1] W.Li, C.Zhang, "Markov Chain Analysis" in International Encyclopedia of Human Geography, ScienceDirect, https://www.sciencedirect.com/topics/earth-andplanetarysciences/markov-chain , 2009, pp. 455-460.

[2] "Hidden Markov Model" Github. https://mmz33.github.io/Hidden-MarkovModel/.

[3] "Speech and Language Processing." https://web.stanford.edu/~jurafsky/slp3/ .

[4] "VnCoreNLP: A Vietnamese natural language processing toolkit" GitHub. https://github.com/vncorenlp/VnCoreNLP .

[5] Minh Nghiem, Dien Dinh, Mai Nguyen, "Improving Vietnamese POS tagging by integrating a rich feature set and Support Vector Machines", https://ieeexplore.ieee.org/document/4586344.