ĐẠI HỌC BÁCH KHOA HÀ NỘI

BÁO CÁO BÀI TẬP LỚN NHẬP MÔN HỌC MÁY VÀ KHAI PHÁ DỮ LIỆU

IT3190 – 141320

PHÂN LOẠI EMAIL (PHISHING EMAIL DETECTION)

Nguyễn Trần Chung - 20204520
Chung.nt204520@sis.hust.edu.vn

Nguyễn Duy Thành - 20204691
Thanh.nd204691@sis.hust.edu.vn

Nguyễn Tiến Đạt - 20204640
Dat.nt204640@sis.hust.edu.vn

Trần Nguyễn Anh Tuấn – 20200565
Tuan.tna200565@sis.hust.edu.vn

Đào Văn Thiều - 20204610
Thieu.dv204610@sis.hust.edu.vn

Giảng viên kiểm tra: PGS.TS Thân Quang Khoát
Khoa: Khoa học máy tính
Trường: Công nghệ thông tin và Truyền thông

Hà Nội, ngày 6 tháng 7 năm 2023

1. Giới thiệu chung

1.1 Giới thiệu về bài toán phân loại email, thực trạng

Bài toán phân loại email là một bài toán quan trọng trong lĩnh vực xử lý ngôn ngữ tự nhiên và máy học. Nhiệm vụ của bài toán này là phân loại các email đến vào các nhóm khác nhau dựa trên nội dung và tính chất của email.

Thực trạng của bài toán phân loại email là ngày càng trở nên quan trọng do sự gia tăng về lượng email gửi và nhận hàng ngày. Các tổ chức, doanh nghiệp và cá nhân đều phải xử lý một số lượng lớn email, trong đó có cả email thư rác (spam) và email hợp lệ.

1.2 Động lực chọn đề tài

Các email thư rác, hay còn gọi là spam, tạo ra vấn đề cho người nhận email. Chúng gây phiền hà, lãng phí thời gian và tài nguyên, và có thể chứa nội dung độc hại hoặc lừa đảo. Do đó, việc phân loại email để loại bỏ thư rác là một yêu cầu cần thiết và được đặt ra.

Ngày nay, dù cho các phương pháp phân loại email đã và đang phát triển, vấn đề phân loại chính xác các email vẫn là một thách thức. Các kẻ tấn công spam ngày càng sáng tạo hơn và sử dụng các kỹ thuật để tránh phát hiện. Do đó, việc sử dụng các phương pháp tiên tiến hơn và kết hợp nhiều phương pháp lại là cần thiết để giải quyết bài toán phân loại email một cách hiệu quả.

1.3 Các kĩ thuật chính trong đề tài sử dụng

1.3.1 Các phương pháp trích xuất đặc trưng

- Content-based
- Bag Of Words in terms of Frequency
- TF-IDF
- Word2vec

1.3.2 Model dùng huấn luyện

- Gaussian Naive Bayes
- Support Vector Machine Classifier
- K-nearest Neighbor
- Random Forest

1.4 Quy trình thực hiện phân loại Email

- Tiền xử lý dữ liệu:

Thao tác 1: Lemmatization là quá trình chuyển đổi một từ về dạng cơ bản hoặc từ điển của nó, được gọi là lemma. Ví dụ, lemma của từ "running" là "run", và lemma của từ "better" là "good".

Thao tác 2: Loại bỏ các stopwords: i, me, my, myself, we, our, ours,...

Thao tác 3: Lowercase tất cả các từ.

Thao tác 4: Loại bỏ các kí tự đặc biệt: ‘!”#$%&'()*+,-./:;?@[\]^_`{|}~’

Thao tác 4: Loại bỏ các chữ số

```
#text pre processing
train_X=[]
for i in range(0, len(data0['text'])):
    review = re.sub('[^a-zA-Z]', ' ', data0['text'][i])
    review = review.lower()
    review = review.split()
    review = [lemmatizer.lemmatize(word) for word in review if not word in set(stopwords)]
    review = ' '.join(review)
    train_X.append(review)
train_X
```

- Trích xuất các đặc trưng: Sử dụng các phương pháp như: content-based, Bag of Words, TF-IDF và Word2vec
- Huấn luyện mô hình: Naive Bayes với GaussianNB, SVM, KNN, Randomforest
- Tinh chỉnh siêu tham số của các mô hình
- Đánh giá hiệu quả qua độ đo: Accuracy, Recall, Precision, F1-score

Các bước trên được lặp đi lặp lại để đưa ra những giải pháp cho một số vấn đề gặp phải trong quá trình phân loại.

2. Tóm tắt lý thuyết sử dụng trong Project

2.1 Bag of words

Kỹ thuật Bag of Words (BOW) là một phương pháp được sử dụng trong xử lý ngôn ngữ tự nhiên và trích xuất đặc trưng. Nó được sử dụng để biểu diễn và xử lý các văn bản trong các bài toán như phân loại văn bản, phát hiện cảm xúc, tìm kiếm thông tin, và nhiều ứng dụng khác.

Kỹ thuật BOW cho phép chúng ta biểu diễn văn bản dưới dạng số để sử dụng trong các thuật toán máy học. Tuy nhiên, nó chỉ quan tâm đến sự xuất hiện của các từ và không xem xét thứ tự hay ngữ nghĩa của chúng. BOW là một phương pháp đếm tần suất xuất hiện của các từ trong một văn bản và biểu diễn chúng dưới dạng vector. Với BOW, không quan tâm đến thứ tự từ và cấu trúc câu trong văn bản, chỉ quan tâm đến các từ và số lần xuất hiện của chúng. Ví dụ, văn bản "This is a cat" sẽ được biểu diễn thành vector [1, 1, 1, 0], với 4 từ "this", "is", "a", và "cat".

Ý tưởng chính của kỹ thuật BOW là chuyển đổi mỗi văn bản thành một vector đặc trưng dựa trên sự xuất hiện của các từ trong văn bản đó. Quá trình này bao gồm các bước sau:

Bước 1: Xây dựng từ điển: Đầu tiên, ta tạo ra một tập hợp các từ khác nhau trong toàn bộ tập huấn luyện. Các từ này sẽ tạo thành từ điển của chúng ta.

Tạo một danh sách các từ trong “bag”. Sử dụng CountVectorizer để tokenize các từ trong văn bản, từ đó ta có một tập các từ đã biết.

Bước 2: Biểu diễn văn bản: Tiếp theo, mỗi văn bản trong tập huấn luyện sẽ được biểu diễn dưới dạng một vector. Mỗi phần tử trong vector tương ứng với số lần xuất hiện của từ tương ứng trong văn bản đó. Ta sẽ đánh số lượng các từ xuất hiện trong mỗi văn bản.

Tại đây, chúng ta có thể dừng lại ở bước này vì đã xác định được đầu vào cụ thể cho thuật toán học của chúng ta.

Một số hạn chế của Bag-of-Words (BoW) trong xử lý ngôn ngữ tự nhiên là:

- Mất thông tin về cấu trúc câu: BOW không bao gồm thông tin về cấu trúc câu, nghĩa là nó không biết về thứ tự từ và mối quan hệ giữa chúng. Do đó, nó không thể hiểu được một câu có thể có nhiều ý nghĩa khác nhau dựa trên cấu trúc của nó.
- Mất thông tin về ngữ nghĩa: BOW không hiểu được ý nghĩa của các từ trong câu. Một từ có thể có nhiều ý nghĩa khác nhau và BoW không thể phân biệt được chúng.
- Mất thông tin về từ loại và ngữ cảnh: BOW không phân biệt được các từ loại khác nhau và không hiểu được ngữ cảnh xung quanh các từ.
- Kích thước lớn: Nếu không được xử lý cẩn thận, biểu diễn BoW có thể trở nên rất lớn và tốn nhiều bộ nhớ, đặc biệt khi thao tác với dữ liệu lớn.
- Không xử lý được dữ liệu bị thiếu: Nếu một từ không xuất hiện trong từ điển mà BoW đang sử dụng, nó sẽ bị mất đi khi biểu diễn. Điều này dẫn đến việc mất mát thông tin quan trọng.

2.2 TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF là một phương pháp tính toán mức độ quan trọng của một từ trong một văn bản. TF-IDF tính toán giá trị của mỗi từ dựa trên tần suất xuất hiện của từ đó trong văn bản (Term Frequency) và tần suất xuất hiện của từ đó trong toàn bộ tập văn bản (Inverse Document Frequency). Giá trị TF-IDF càng cao cho biết từ đó càng quan trọng trong văn bản đó.

Vì vậy, khác biệt chính giữa BOW và TF-IDF là BOW chỉ đếm tần suất xuất hiện của các từ trong văn bản và không quan tâm đến quan trọng của từ đó trong toàn bộ tập văn bản, trong khi TF-IDF tính toán mức độ quan trọng của từ dựa trên cả tần suất xuất hiện trong văn bản và tần suất xuất hiện trong toàn bộ tập văn bản.

- TF (Term Frequency): Tần suất của từ trong văn bản. Nó được tính bằng cách đếm số lần xuất hiện của từ trong văn bản.
- IDF (Inverse Document Frequency): Đánh giá sự quan trọng của từ trong tập dữ liệu lớn hơn. Nó được tính bằng cách lấy logarit tỉ lệ giữa số lượng văn bản trong tập dữ liệu chia cho số lượng văn bản chứa từ đó.

Kết hợp giữa TF và IDF, TF-IDF đánh giá sự quan trọng của một từ trong văn bản mà cân nhắc cả đến tần suất xuất hiện trong văn bản cụ thể cũng như sự phổ biến của từ trong toàn bộ tập dữ liệu. Điều này giúp cải thiện độ chính xác trong việc phân loại văn bản, tìm kiếm thông tin hoặc xử lý ngôn ngữ tự nhiên.

Như vậy, về bản chất thì TF-IDF là sự nối tiếp trong khi sử dụng BOW. Ngoài ra, khi sử dụng TF-IDF ta phải có bước chuẩn hóa giá trị TF-IDF. Chuẩn hóa có ý nghĩa

- Tăng tính nhất quán: Chuẩn hóa giúp đảm bảo tính nhất quán giữa các văn bản hoặc từ khóa. Điều này làm cho kết quả phân tích và so sánh giữa các văn bản ngắn/dài trở nên công bằng hơn.

- Tránh hiện tượng quá khớp: Trong một số trường hợp, các giá trị tf-idf có thể rất khác nhau giữa các văn bản. Điều này có thể dẫn đến hiện tượng quá khớp, khi mà các văn bản có giá trị tf-idf lớn hơn sẽ có ảnh hưởng quá mức đến kết quả phân tích. Chuẩn hóa giúp giảm thiểu hiện tượng này và làm cho kết quả phân tích công bằng hơn

2.3 Word2vec (Word Embedding)

Kỹ thuật WordEmbedding giúp các mô hình hiểu được ý nghĩa và mối quan hệ giữa các từ và ngữ cảnh. Word embedding giữ được thông tin về mối quan hệ ngữ nghĩa và cú pháp giữa các từ. Các từ có ý nghĩa tương đồng sẽ có các biểu diễn gần nhau trong không gian vector. Điều này giúp mô hình có thể xử lý các từ mới mà chưa được thấy trong dữ liệu huấn luyện một cách hiệu quả. Ngoài ra, word embedding còn giúp giảm chiều dữ liệu. Thay vì biểu diễn mỗi từ bằng một vector one-hot lớn (có kích thước bằng số lượng từ trong từ điển), word embedding biểu diễn từng từ bằng một vector có số chiều thấp hơn. Điều này giúp giảm đáng kể kích thước của không gian biểu diễn và giúp mô hình học nhanh hơn và hiệu quả hơn.

Thay vì tính toán và lưu trữ thông tin của toàn bộ dataset, chúng ta có thể tạo ra một mô hình có thể học cách mã hóa xác suất của một từ khi biết ngữ cảnh của nó. Các tham số của mô hình này sẽ là các vector từ ngữ. Sau đó, huấn luyện mô hình này với mục đích tạo ra một biểu diễn mới cho vecto từ ngữ đó. Mỗi vòng lặp chúng ta sẽ chạy mô hình, đánh giá lỗi và tuân theo luật cập nhật các tham số để hạn chế lỗi. Trong project, nhóm sẽ sử dụng thuật toán continuous bag of word để dự đoán 1 từ trung tâm từ các word vector xung quanh với ngữ cảnh và sử dụng negative sampling làm phương pháp huấn luyện.

Trong CBOW, mục tiêu là dự đoán từ hiện tại dựa trên các từ xung quanh nó trong câu. Mô hình CBOW sẽ lấy các từ xung quanh từ hiện tại làm input và cố gắng dự đoán từ hiện tại. Negative sampling là một phương pháp để tính xác suất dự đoán từ. Thay vì tính xác suất của tất cả các từ trong từ điển, negative sampling chỉ tính xác suất của một số lượng nhỏ các từ âm bằng cách lấy mẫu ngẫu nhiên từ một phân phối xác suất đã được lấy mẫu trước đó. Sau đó, ta tính toán cross-entropy loss giữa kết quả dự đoán của mô hình CBOW với các từ ngữ tham chiếu và các từ âm đã lấy mẫu. Mục tiêu là tối thiểu hóa loss này để cải thiện khả năng dự đoán của mô hình CBOW.Bằng cách kết hợp CBOW và negative sampling, ta có thể giảm đáng kể chi phí tính toán trong quá trình huấn luyện và đồng thời cải thiện hiệu suất của mô hình word2vec.

2.4 PCA

Phân tích thành phần chính, hay PCA, là một phương pháp giảm chiều vector thường được sử dụng để giảm chiều của các tập dữ liệu lớn, bằng cách chuyển đổi một tập hợp lớn các biến thành một biến nhỏ hơn mà vẫn chứa hầu hết thông tin trong tập hợp lớn.

Việc giảm số lượng chiều của một tập dữ liệu đương nhiên phải trả giá bằng độ chính xác, nhưng ta có thể đánh đổi một chút độ chính xác để lấy sự đơn giản. Bởi vì các tập dữ liệu nhỏ hơn sẽ dễ khám phá và trực quan hóa hơn, đồng thời giúp việc phân tích các điểm dữ liệu trở nên dễ dàng và nhanh hơn nhiều đối với các thuật toán máy học mà không cần xử lý các biến ngoại lai.

Có 5 bước để thực hiện PCA:

- **Bước 1**: Chuẩn hóa miền giá trị liên tục của các biến. Bước này nhằm để tránh sự tác động quá mức của những dữ liệu có miền giá trị lớn.
- **Bước 2**: Tính ma trận hiệp phương sai để biết sự tương quan giữa các biến. Đôi khi các biến có liên hệ chặt với nhau nhưng lại chứa các thông tin dư thừa. Ma trận hiệp phương sai có kích thước bằng số biến trong dữ liệu ban đầu. Các phần tử của ma trận hiệp phương sai cho biết mức độ tương quan giữa các biến.
- **Bước 3:** Tính toán các vector riêng và các giá trị riêng. Các vector riêng đại diện cho các hướng chính của phương sai trong dữ liệu, và các giá trị riêng cho biết mức độ phương sai hợp lệ trong các hướng đó.
  Các thành phần chính là các biến mới được xây dựng dưới dạng tổ hợp tuyến tính hoặc hỗn hợp của các biến ban đầu. Các tổ hợp này được thực hiện theo cách sao cho các biến mới (nghĩa là các thành phần chính) không tương quan với nhau và hầu hết thông tin trong các biến ban đầu được ép hoặc nén vào các thành phần đầu tiên.
- **Bước 4**: Tạo một vector đặc trưng để lưu những thành phần chính quan trọng
- **Bước 5:** Mục đích là sử dụng vectơ đặc trưng được hình thành bằng cách sử dụng các vectơ riêng của ma trận hiệp phương sai, để định hướng lại dữ liệu từ các trục ban đầu sang các trục được biểu thị bởi các thành phần. Điều này có thể được thực hiện bằng cách nhân chuyển vị của tập dữ liệu gốc với chuyển vị của vectơ đặc trưng.

2.5 GridSearch

Tìm các giá trị được tối ưu hóa cho siêu tham số của các mô hình học sao cho chúng được tối ưu hóa để mang lại hiệu suất mô hình cao. Quá trình trên còn được gọi là 'Search'.

GridSearch cho phép xác định một tập hợp các giá trị hyperparameter và thử nghiệm tất cả các kết hợp có thể có của chúng. Điều này giúp đảm bảo rằng bạn không bỏ sót bất kỳ cấu hình hyperparameter nào có thể cải thiện hiệu suất của mô hình.

Ta chia miền của siêu tham số thành một lưới rời rạc. Sau đó, thử mọi kết hợp giá trị của lưới này, tính toán một số chỉ số hiệu suất bằng cách sử dụng xác thực chéo. Điểm của lưới tối đa hóa giá trị trung bình trong cross-validation, là sự kết hợp tối ưu của các giá trị cho siêu tham số.

Hạn chế lớn là nó rất chậm. Việc kiểm tra mọi sự kết hợp của không gian đòi hỏi rất nhiều thời gian mà đôi khi không có sẵn. Mà mọi điểm trong lưới đều cần xác thực chéo k lần, yêu cầu k bước đào tạo. Vì vậy, điều chỉnh các siêu tham số của một mô hình theo cách này có thể khá phức tạp và tốn kém.

2.6 Độ đo Recall, Precision, F1-score

Độ chính xác có thể không phản ánh đúng mức độ hiệu quả của mô hình khi dữ liệu mất cân bằng nên ta cần sử dụng thêm một số độ đo khác để đánh giá

- **Precision:** Đo lường khả năng của mô hình trong việc dự đoán đúng các trường hợp dương tính (positive cases). Precision được tính bằng tỉ lệ giữa số lượng dự đoán dương tính đúng và tổng số lượng dự đoán dương tính. Precision cao cần được đảm bảo trong quá trình xây dựng mô hình để đảm bảo tính chính xác và đáng tin cậy của kết quả dự đoán.
- **Recall:** Đo lường khả năng của mô hình trong việc tìm ra tất cả các trường hợp dương tính. Recall được tính bằng tỉ lệ giữa số lượng dự đoán dương tính đúng và tổng số lượng trường hợp dương tính. Recall cao cho thấy mô hình có khả năng tìm ra nhiều dữ liệu thuộc lớp cần dự đoán.
- **F1-score:** F1-score là trung bình điều hòa giữa Precision và Recall. Nó được sử dụng để tổng hợp độ chính xác và độ phủ sóng của mô hình. F1-score càng cao, mô hình càng đạt hiệu suất tốt hơn. F1-score là một lựa chọn tốt khi dữ liệu bị mất cân bằng và đánh giá hiệu suất của mô hình không chỉ dựa trên độ chính xác

Precision đo lường tỷ lệ các trường hợp dự đoán đúng trong số các trường hợp dự đoán là positive (đúng tích cực). Trong khi đó, recall đo lường tỷ lệ các trường hợp dự đoán đúng trong số tất cả các trường hợp positive có thể dự đoán. Precision cao chỉ ra rằng mô hình có khả năng dự đoán đúng một cách chính xác các trường hợp positive. Tuy nhiên, điều này không đảm bảo rằng mô hình đang dự đoán đúng tất cả các trường hợp positive có thể có. Điều này có thể dẫn đến recall thấp, tức là mô hình bỏ sót nhiều trường hợp positive. Tương tự, recall cao chỉ ra rằng mô hình có khả năng tìm ra nhiều trường hợp positive có thể dự đoán. Tuy nhiên, điều này không đảm bảo rằng mô hình đang dự đoán đúng tất cả các trường hợp positive. Điều này có thể dẫn đến precision thấp, tức là mô hình đưa ra nhiều dự đoán positive sai.

3. Các phương án được sử dụng để phân loại email

3.1 Content-based

3.1.1 Cách cài đặt

Các đặc trưng trích xuất được phân loại dựa trên:

- Hình thức email:

Nhiều đặc trưng có thể được trích xuất có thể được coi là đặc trưng cơ sở của hình thức email. Trong số đó, đề cập dưới đây đã được xem xét cho project này.

- Phần mở: Số các từ dear, hi,...
- Bố cục : các đặc trưng về Chuyển tiếp, reply

Mỗi đặc trưng này được trình bày và code ở dưới:

```
def haveFormalWord(email):
    formal_words = ['hi', 'dear', 'best', 'wishes', 'regard','dict','reply']
    for word in formal_words:
        if word in email.lower():
            return 0
    return 1
```

```
def response(email):
    response_words = ['re', 'fw', 'fwd','cc', 'to']
    for word in response_words:
        if word in email.lower():
            return 0
    return 1
```

- Từ ngữ:

Nhiều đặc trưng có thể được trích xuất thuộc danh mục này. Trong số đó, đề cập dưới đây đã được xem xét cho project này.

- Chính tả trong từ điển tiếng Anh
- Blacklist:{verify, account, password,access, limitted, risk, security, service, suspend, allow, click, information, risk, credit, minutes, only, last,review,cost, fail, immediate, now, winner, best,any, passion, interest, late, soon, help, update, confirm }
- Kí tự đặc biệt: {%,$,#,?,!}
- Con số: số lượng các số
- Chữ: số lượng các chữ
- Từ: Số lượng các từ

Mỗi đặc trưng này được trình bày và code ở dưới:

```
# Check number of misspelled words
def misspelledWord(email):
    spell = SpellChecker()
    words = re.findall(r'\w+', email)
    misspelled = spell.unknown(words)
    return len(misspelled)
```

```
 # Check if the email has typos or bad grammar
def badGrammar(email):
    if len(re.findall(r'[\w]+', email)) > 2:
        return 1
    else:
        return 0
```

```
# Extract common phishing keywords
def keyword(email):
    keywords = [ "login",'account', 'alert', 'verify', 'bank', 'price', 'security', "suspend",'fraud', 'urgent', 'warning', 'limi
    for keyword in keywords:
        if keyword in email.lower():
            return 1
        else:
            return 0
```

```
# Check if the email asks for personal or financial information
def requestInfo(email):
    # Check if the email asks for personal or financial information
    personal_info = ['password', 'pin', 'ssn', 'social security', 'credit', 'debit', 'card number', 'card expiration', 'cvv']
    for info in personal_info:
        if info in email.lower():
            return 1
        else:
            return 0
```

- URL:

Nhiều đặc trưng có thể được trích xuất thuộc danh mục này. Trong số đó, đề cập dưới đây đã được xem xét cho project này.

- Tên miền: Số hay là chữ
- Độ dài miền: Ngắn/dài
- Chuẩn HTTP: có/ không
- Số dot trong domain
- Redirection "//" trong URL
- Ký tự '.' trong miền

```
# Check the length of the email
def longEmail(email):
    if len(email) > 500:
```

```
def numLink(email):
    # Check if number of links
    num_of_links = len(re.findall(r"http", email))
    return num_of_links
```

```
def suspiciousLink(email):
    # Check if the email has suspicious link
    # links = re.find(r"(http?://\S+)", email)
    # if links:
    #     for link in links:
    #         if  '-' in link or 'paypal' in link or 'bank' in link or 'irs' in link or 'amazon' in link or 'ebay' in link:
    #             return 1
    #         else:
    #             return 0
    # else:
    #     return 0
    num_suspicious_link = 0
    if 'http' in email and 'https' not in email:
        num_suspicious_link +=1
    return num_suspicious_link
```

```
def numDot(email):
    num_dot = 0
    # Check if the link in email has dot
    links = re.findall(r"http", email)
    for link in links:
        num_dot += len(re.findall(r".", link))
    return num_dot
```

Mỗi đặc trưng này được trình bày và code ở dưới:

Ta sẽ biểu diễn các email thành các vector với từng phần tử là từng đặc trưng nêu trên được tổng hợp lại. Như vậy, ta đã có đầu vào cho các thuật toán học. Tạo một danh sách và một hàm gọi các hàm khác và lưu trữ tất cả các đặc trưng của email trong danh sách. Nhóm sẽ trích xuất các đặc trưng của từng email và thêm vào danh sách này.

```
def extract_features(email,label):
    features = []

    features.append(haveFormalWord(email))
    features.append(response(email))
    features.append(misspelledWord(email))
    features.append(badGrammar(email))
    features.append(keyword(email))
    features.append(requestInfo(email))
    features.append(senseUrgency(email))
    features.append(senseThreat(email))
    features.append(longEmail(email))
    features.append(numSpecialChar(email))
    features.append(numDigit(email))
    features.append(numChar(email))
    features.append(numWord(email))
    features.append(numLink(email))
    features.append(suspiciousLink(email))
    features.append(numDot(email))
    features.append(label)

    return features
```

```
#Extracting the feautres & storing them in a list
features_list = []
# for email in data0:
for i in range(5728):
    email = data0['text'][i]
    # Add the email features to the list of all features
    features_list.append(extract_features(email, data0['spam'][i]))
```

Sau đó, ta sẽ tạo tệp dữ liệu mới tương ứng với các email. Trước đó, nhóm cũng đã thể hiện phân phối của các đặc trưng trong toàn bộ dataset (đã trình bày ở jupyter notebook).

```
# Convert the list of features into a pandas DataFrame
feature_names = ['have_formal_word','have_respone', 'num_misspelled_words','bad_grammar', 'keyword', 'requests_personal_info', 's
df = pd.DataFrame(features_list, columns= feature_names)
df.head()
```

```
# Storing the extracted email fatures to csv file
df.to_csv('email_processed.csv', index= False)
```

```
df.shape
```

```
(5728, 17)
```

```
df.describe()
```

	have_formal_word	have_respone	num_misspelled_words	bad_grammar	keyword	requests_personal_info	sense_of_urgency	sense_of_threat	long_
count	5728.000000	5728.000000	5728.000000	5728.000000	5728.000000	5728.000000	5728.000000	5728.000000	5728.0
mean	0.220496	0.019553	11.654330	0.999651	0.006983	0.020426	0.001920	0.111732	0.5
std	0.414617	0.138471	13.119121	0.018684	0.083281	0.141465	0.043784	0.315064	0.4
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.000000	0.000000	4.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.0
50%	0.000000	0.000000	9.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.0
75%	0.000000	0.000000	15.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.0
max	1.000000	1.000000	220.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

Tiếp đây, ta đã có thể thực hiện việc huấn luyện các mô hình học máy sử dụng đầu vào là những vector đặc trưng của từng email. Ta sẽ cần phải chia tập dữ liệu thành tập training và tập test theo tỉ lệ 80/20.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2, random_state = 12)
X_train.shape, X_test.shape
```

```
((4582, 16), (1146, 16))
```

Sau đó, sử dụng các tập này để huấn luyện và đánh giá qua các mô hình Naive Bayes (Gaussian NB), SVM, KNN, RF.

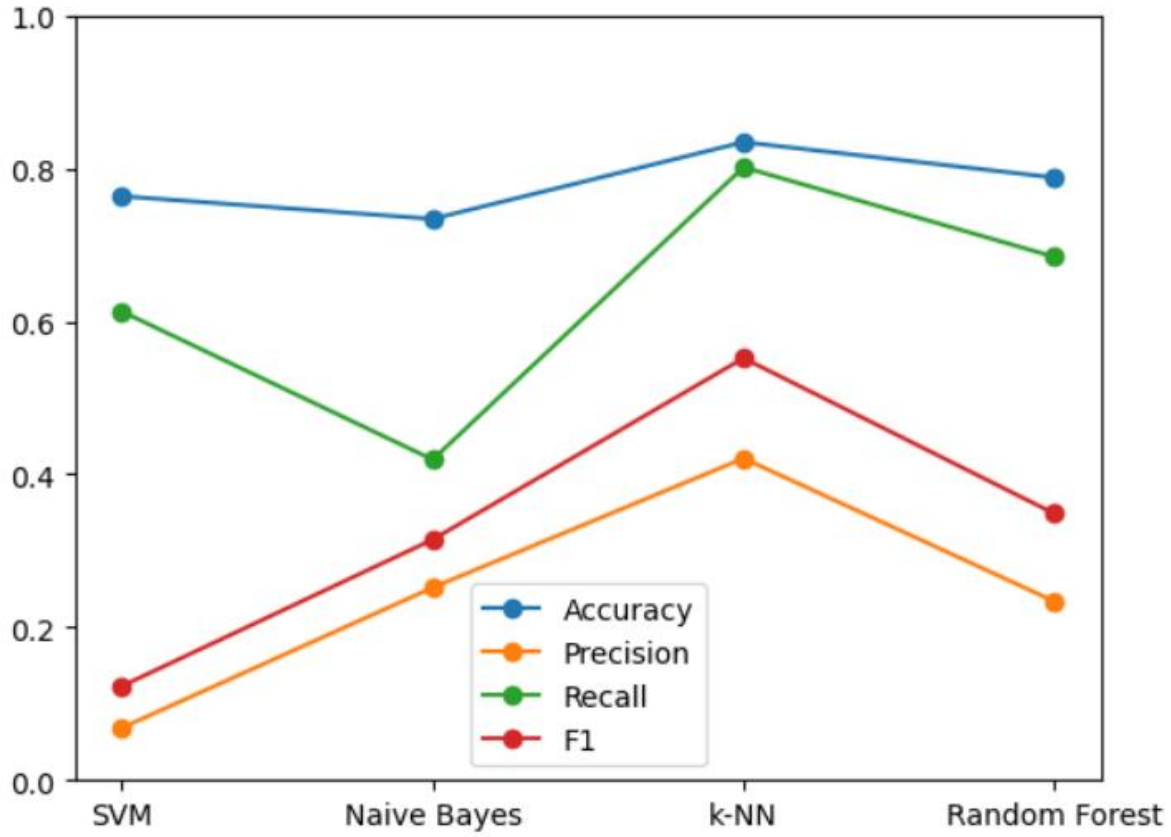
```
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
```

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train, y_train)
```

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100,max_depth=9,max_leaf_nodes=9,max_features=None,n_jobs=-1)
rf_model = rf.fit(X_train, y_train)
y_pred_rf=rf_model.predict(X_test)
```

Tuy nhiên, kết quả huấn luyện lại không thực sự tốt



	Accuracy	Precision	Recall	F1
SVM	0.763525305410 1222	0.612903225806 4516	0.068345323741 0072	0.122977346278 31716
Gaussian Naive Bayes	0.733856893542 7574	0.251798561151 0791	0.419161676646 70656	0.314606741573 0337
KNN (K=4)	0.834205933682 3735	0.420863309352 518	0.801369863013 6986	0.551886792452 8301
Random Forest	0.787958115183 2461	0.233812949640 28776	0.684210526315 7895	0.348525469168 90076

3.1.2 Một số vấn đề

Việc sử dụng các đặc trưng về nội dung hay về hành vi trong các email không mang lại kết quả cao. Một số yếu tố có thể xuất hiện như

- Tốn công: Việc tìm hiểu và tổng hợp các kiến thức liên quan về phishing email còn hạn chế.
- Thiếu các đặc trưng đại diện: Ngoài các đặc điểm nhóm đã đề xuất ở trên, chúng ta có thể kiểm tra thêm các đặc trưng về đường dẫn URL. Nhưng các URL trong data lại bị tokenize hóa nên không thể xử lý thêm các thông tin từ đường dẫn. Quá trình tiền xử lý dữ liệu cũng đã làm mất mát đi những đặc trưng của đường dẫn, hình thức email.
- Các đặc trưng không đa dạng : Quá trình xây dựng các đặc trưng có thể chưa phù hợp với dataset của nhóm, dẫn đến sự thiếu tổng quá hóa bởi underfiiting.
- Việc cập nhật, mở rộng với những email khác sẽ khó khăn. Nhóm sẽ lại phải tìm ra những đặc trưng khác để cải thiện dần mô hình. Do đó, tính linh hoạt và khả năng nập nhật tương đối kém.

Như vậy, cần phải có một cách giải quyết khác phù hợp với dataset của nhóm, những phương pháp này có thể tìm thấy những đặc trưng ẩn trong dataset của nhóm một cách tự động. Nhóm đã thử tiếp cận cách phương pháp trích xuất đặc trưng bằng cách tiếp cận theo xử lý ngôn ngữ tự nhiên NLP.

3.2 Bag of Word

3.2.1 Cách cài đặt

Quy trình chung

- Khởi tạo từ điển từ ngữ trong tập huấn luyện
- Biểu diễn các email trong tập huấn luyện thành dạng các vector, mỗi chiều tương ứng với tần suất xuất hiện từ trong từ điển.

Công cụ:

- CountVectorizer được cung cấp bởi sklearn.

CountVectorizer tạo một ma trận trong đó mỗi từ duy nhất được biểu thị bằng một cột của ma trận và mỗi email là một hàng trong ma trận. Giá trị của mỗi ô là số lượng từ trong 1 email cụ thể.

```
#Step 1: Initialize a dictionary about the trainning data set.
#
from sklearn.feature_extraction.text import CountVectorizer
count_vector = CountVectorizer(ngram_range=(1, 1), lowercase = True , stop_words =  'english')

count_vector.fit(X_train_email)
# what are the words in our dictionary?
print(sorted(count_vector.vocabulary_))

#Step 2: Represent our email in training set in form of vector (encoding)
#
# Fit the training data and then return the matrix
X_train = count_vector.fit_transform(X_train_email)
# to get the the list of feature words that are lowercase with no puttutaion or stop words
#Since you fit transform X data training
# they convert data to integer data

# Transform testing data and return the matrix. Note we are not fitting the testing data into the CountVectorizer()
X_test = count_vector.transform(X_test_email)
#Since you transform X data testing
# they convert data to integer data
```

- Sử dụng các mô hình học để phân loại: Gaussian Naive Bayes, SVM, KNN, Random forest

```
#Step 3: Using model for prediction.
# We are using Naive-Bayes, SVM, KNN, Randomforest
#
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train.toarray() , y_train)
```

```
#Create a svm Classifier
clf = svm.SVC(C=100, gamma=0.0001,kernel='rbf')

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
knn = KNeighborsClassifier(n_neighbors = 4, weights = 'distance',metric = 'minkowski')
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

```
rf = RandomForestClassifier(n_estimators=100,max_depth=9,max_leaf_nodes=9,max_features=None,n_jobs=-1)
rf_model = rf.fit(X_train, y_train)
y_pred_rf=rf_model.predict(X_test)
```

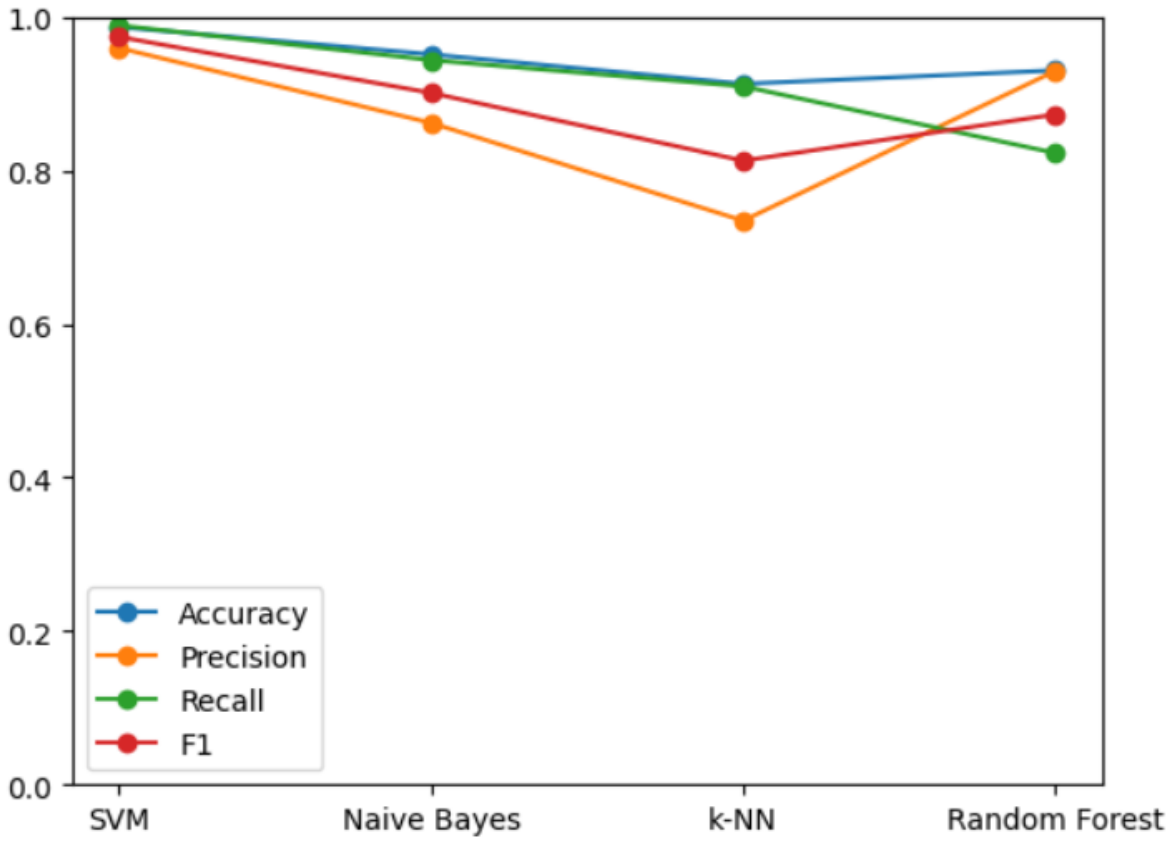
- Tinh chỉnh các siêu tham số của các mô hình: SVM, KNN và RF

```
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
param_grid_SVM = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf','linear']}


gs_svm = GridSearchCV(SVC(),param_grid_SVM,cv=10)
gs_svm.fit(X_train, y_train)
```

```
#Defining a number of folders for GridSearchCV and assigning TT.

param_grid = {'n_neighbors': list(range(1,9)),'weights' : ['uniform','distance'],'metric' : ['minkowski','euclidean','manhattan']
gs_knn = GridSearchCV(KNeighborsClassifier(),param_grid,cv=10)
gs_knn.fit(X_train, y_train)
```

```
from sklearn.ensemble import RandomForestClassifier
param_grid_RF = {'n_estimators': [25, 50, 100, 150],'max_features': ['sqrt', 'log2', None],'max_depth': [3, 6, 9],'max_leaf_nodes
gs_rf = GridSearchCV(RandomForestClassifier(),param_grid_RF,cv=10)
gs_rf.fit(X_train, y_train)
```

- Đánh giá hiệu quả các mô hình học



	Accuracy	Precision	Recall	F1
SVM	0.986731843575419	0.959349593495935	0.988826815642458 1	0.973865199449 7937
Gaussian Naive Bayes	0.951117318435 7542	0.861788617886 1789	0.943620178041 5431	0.900849858356 9406

KNN (K=4)	0.9127094972067039	0.7344173441734417	0.9093959731543624	0.8125937031484257
F1	0.9301675977653632	0.9295392953929539	0.8225419664268585	0.8727735368956742

3.2.2 Một số vấn đề
Nhận xét tổng quan về một số vấn đề:

Cụ thể hơn khi làm việc với từ điển sinh bởi tập dữ liệu huấn luyện của nhóm, nhóm quan sát thấy một số từ không nằm trong tập dữ liệu huấn luyện. Khi xây dựng từ điển sử dụng Bag-of-Words (BOW), việc có một số từ không xuất hiện trong dữ liệu huấn luyện có thể gây ra một số vấn đề.

- Mất mát thông tin: Nếu một từ không xuất hiện trong dữ liệu huấn luyện, thông tin liên quan đến từ đó sẽ bị mất đi. Điều này có thể gây ra sự thiếu sót trong việc hiểu các văn bản chứa từ này trong quá trình phân loại hoặc phân tích văn bản.
- Chệch lệch trong việc phân loại: Nếu từ không xuất hiện trong dữ liệu huấn luyện nhưng xuất hiện trong dữ liệu kiểm tra, mô hình có thể không biết cách xử lý nó. Điều này có thể dẫn đến sự chệch lệch trong kết quả phân loại, vì mô hình không được huấn luyện với các dữ liệu chứa từ này.
- Kích thước từ điển lớn: Nếu có quá nhiều từ không xuất hiện trong dữ liệu huấn luyện, điều này có thể làm tăng kích thước của từ điển. Điều này có thể làm tăng độ phức tạp và thời gian xử lý của mô hình, và cũng có thể gây ra hiện tượng quá khớp (overfitting) nếu kích thước từ điển quá lớn so với lượng dữ liệu huấn luyện.

Nhóm tìm hiểu được một số phương án để giải quyết vấn đề trong từ điển. Ngoài đề xuất thu thập thêm dữ liệu để đảm bảo rằng các từ xuất hiện trong dữ liệu kiểm tra cũng xuất hiện trong dữ liệu huấn luyện. Điều này có thể giúp cung cấp thông tin đầy đủ cho mô hình trong quá trình huấn luyện và đánh giá. Song, nhóm muốn thử một cách tiếp cận mới trong xử lý ngôn ngữ tự nhiên và mang lại kết quả cao là:

- Sử dụng kỹ thuật word embedding: Thay vì sử dụng BOW, có thể sử dụng các phương pháp word embedding như Word2Vec để biểu diễn từ vựng. Các phương pháp này có thể giúp mô hình hiểu được mối quan hệ giữa các từ ngay cả khi chúng không xuất hiện trong dữ liệu huấn luyện.

3.3 TF-IDF (Term Frequency-Inverse Document Frequency)

3.3.1 Cách cài đặt
Quy trình chung

- Khởi tạo từ điển từ ngữ trong tập huấn luyện.

- Tính toán các giá trị TF và IDF. Biểu diễn các email trong tập huấn luyện thành dạng các vector, mỗi chiều tương ứng với giá trị TF-IDF của từ trong từ điển đối với email.

```
#tf idf
tf_idf = TfidfVectorizer(ngram_range=(1, 1), lowercase = True , stop_words =  'english')
#applying tf idf to training data
X_train_email_tf = tf_idf.fit_transform(X_train_email)
#applying tf idf to training data
#X_train_email_tf = tf_idf.transform(X_train_email)

print("n_samples: %d, n_features: %d" % X_train_email_tf.shape)
```

```
n_samples: 4296, n_features: 26736
```

```
#transforming test data into tf-idf matrix
X_test_email_tf = tf_idf.transform(X_test_email)
print("n_samples: %d, n_features: %d" % X_test_email_tf.shape)
```

```
n_samples: 1432, n_features: 26736
```

- Sử dụng các mô hình học để phân loại: Gaussian Naive Bayes, SVM, KNN, Random forest

```
#naive bayes classifier
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train_email_tf.toarray(), y_train)
```

```
GaussianNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
predictions = naive_bayes.predict(X_test_email_tf.toarray())
predictions
```

```
#Create a svm Classifier
clf = svm.SVC(C=1, gamma=1,kernel='linear')

#Train the model using the training sets
clf.fit(X_train_email_tf, y_train)

#Predict the response for test dataset
y_pred_svm = clf.predict(X_test_email_tf)
```

```
knn = KNeighborsClassifier(n_neighbors = 8, weights = 'distance',metric = 'minkowski')

knn.fit(X_train_email_tf, y_train)
y_pred_knn = knn.predict(X_test_email_tf)
```

```
rf = RandomForestClassifier(n_estimators=150,max_depth=9,max_features= None,max_leaf_nodes=9,n_jobs=-1)
rf_model = rf.fit(X_train_email_tf, y_train)
y_pred_rf=rf_model.predict(X_test_email_tf)
```

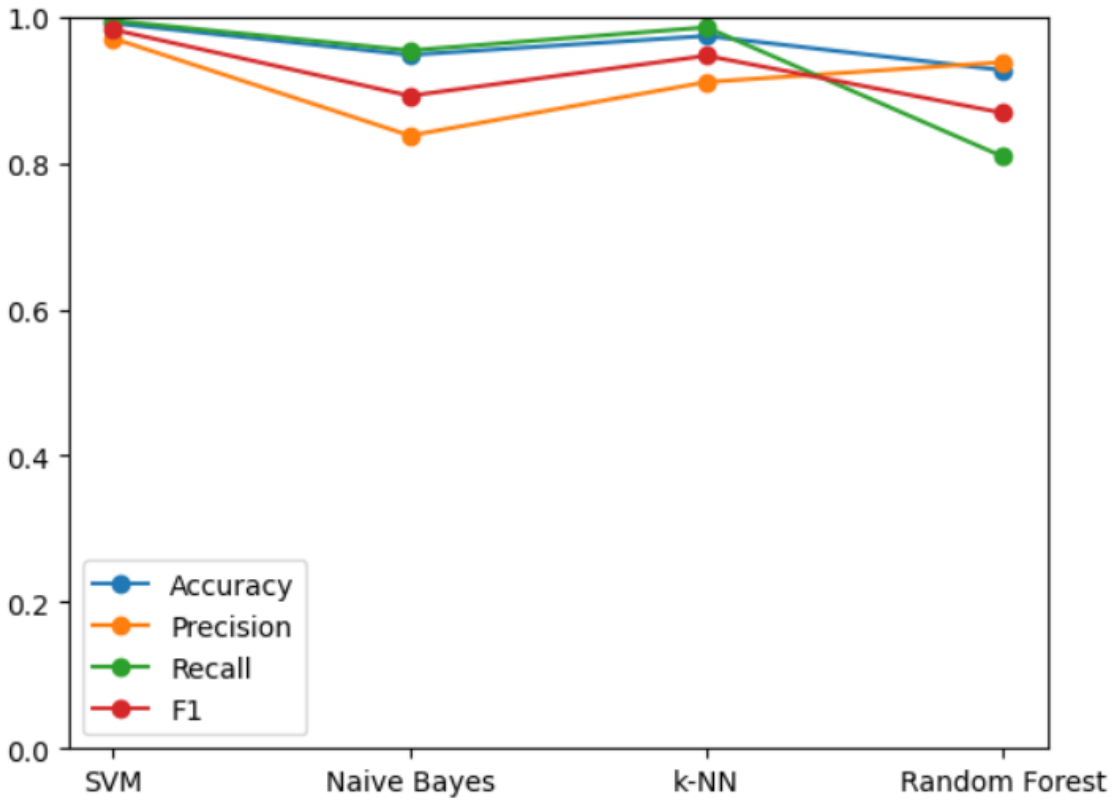
- Tinh chỉnh các siêu tham số

```
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
param_grid_SVM = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf','linear']}


gs_svm = GridSearchCV(SVC(),param_grid_SVM,cv=4)
gs_svm.fit(X_train_email_tf, y_train)
```

```
from sklearn.ensemble import RandomForestClassifier
param_grid_RF = {'n_estimators': [25, 50, 100, 150],'max_features': ['sqrt', 'log2', None],'max_depth': [3, 6, 9],'max_leaf_nodes
gs_rf = GridSearchCV(RandomForestClassifier(),param_grid_RF,cv=5)
gs_rf.fit(X_train_email_tf, y_train)
```

```
from sklearn.neighbors import KNeighborsClassifier
param_grid = {'n_neighbors': list(range(1,9)),'weights' : ['uniform','distance'],'metric' : ['minkowski','euclidean','manhattan']
gs_knn = GridSearchCV(KNeighborsClassifier(),param_grid,cv=3)
gs_knn.fit(X_train_email_tf, y_train)
```

- Đánh giá hiệu quả các mô hình học:



	Accuracy	Precision	Recall	F1
SVM	0.9909217877094972	0.9944444444444445	0.9701897018970189	0.9821673525377229
Gaussian Naive Bayes	0.9476256983240223	0.8373983739837398	0.9537037037037037	0.8917748917748918
KNN (K=8)	0.9734636871508380	0.9105691056910569	0.9853372434017595	0.9464788732394366
F1	0.9266759776536313	0.9376693766937669	0.8084112149532711	0.8682559598494354

3.3.2 Một số vấn đề
Nhận xét tổng quan về một số vấn đề:

- Mặc dù TF-IDF cải thiện được vấn đề về tính quan trọng của các từ ngữ trong tập dữ liệu nhưng nó vẫn không khắc phục được ngữ nghĩa: Phương pháp này chỉ dựa trên tần suất xuất hiện của các từ trong tài liệu mà không đánh giá về ngữ nghĩa của từ đó. Điều này có thể dẫn đến việc đánh giá cao các từ phổ biến nhưng không mang tính quyết định trong việc phân loại tài liệu.
- Đôi khi F1-Score của TF-IDF kém hơn so với BOW thông thường. Việc tiền xử lý dữ liệu có thể làm mất đi một số thông tin ta qua tâm qua các từ nối “like”, “likely”, những từ này có ý nghĩa nhất định trong thể hiện ngữ nghĩa của một câu. Điều này dẫn đến việc IDF sẽ làm giảm ý nghĩa của thông tin ta muốn truyền đạt, do vậy làm việc phân loại bị sai.

3.4 Word2vec

3.4.1 Cách cài đặt
Quy trình chung:

- Tokenize các từ ngữ trong các văn bản

```
x_tokenized = [[w for w in sentence.split(" ") if w != ""] for sentence in data0['text']]
x_tokenized[0]
```

- Khởi tạo thuật toán Word2vec sử dụng mô hình CBOW. Điều này giúp chúng ta dự đoán những từ có nghĩa tương đồng.

```
import gensim
from gensim.models import Word2Vec


model = gensim.models.Word2Vec(x_tokenized,
                 vector_size=100
                 # Size is the length of our vector.
                )
```

- Xây dựng một lớp Sequencer để chuyển đổi từ ngữ, tại thành chuỗi các từ nhúng.
  Hàm khởi tạo nhận 4 tham số: all_words,max_words,seq_length,embedding_matrix
  All_words: list các tokens của dataset ở bước 2
  Max_words: Giới hạn những từ đặc biệt trong dataset
  Sequence Length: Độ dài giới hạn của một câu (phải xác định đối với các thuật toán)
  Embedding_matrix: Ma trận nhúng của các từ trong Word2vec

```
class Sequencer():

    def __init__(self,
                 all_words,
                 max_words,
                 seq_len,
                 embedding_matrix
                ):

        self.seq_len = seq_len
        self.embed_matrix = embedding_matrix
        """
        temp_vocab = Vocab which has all the unique words
        self.vocab = Our last vocab which has only most used N words.

        """
        temp_vocab = list(set(all_words))
        self.vocab = []
        self.word_cnts = {}
        """
        Now we'll create a hash map (dict) which includes words and their occurencies
        """
        for word in temp_vocab:
            # 0 does not have a meaning, you can add the word to the list
            # or something different.
            count = len([0 for w in all_words if w == word])
            self.word_cnts[word] = count
            counts = list(self.word_cnts.values())
            indexes = list(range(len(counts)))
```

```
        # Now we'll sort counts and while sorting them also will sort indexes.
        # We'll use those indexes to find most used N word.
        cnt = 0
        while cnt + 1 != len(counts):
            cnt = 0
            for i in range(len(counts)-1):
                if counts[i] < counts[i+1]:
                    counts[i+1],counts[i] = counts[i],counts[i+1]
                    indexes[i],indexes[i+1] = indexes[i+1],indexes[i]
                else:
                    cnt += 1

        for ind in indexes[:max_words]:
            self.vocab.append(temp_vocab[ind])

    def textToVector(self,text):
        # First we need to split the text into its tokens and learn the length
        # If length is shorter than the max len we'll add some spaces (100D vectors which has only zero values)
        # If it's longer than the max len we'll trim from the end.
        tokens = text.split()
        len_v = len(tokens)-1 if len(tokens) < self.seq_len else self.seq_len-1
        vec = []
        for tok in tokens[:len_v]:
            try:
                vec.append(self.embed_matrix[tok])
            except Exception as E:
                pass
```

```
        last_pieces = self.seq_len - len(vec)
        for i in range(last_pieces):
            vec.append(np.zeros(100,))

        return np.asarray(vec).flatten()
```

```
sequencer = Sequencer(all_words = [token for seq in x_tokenized for token in seq],
              max_words = 1200,
              seq_len = 14,
              embedding_matrix = model.wv
             )
```

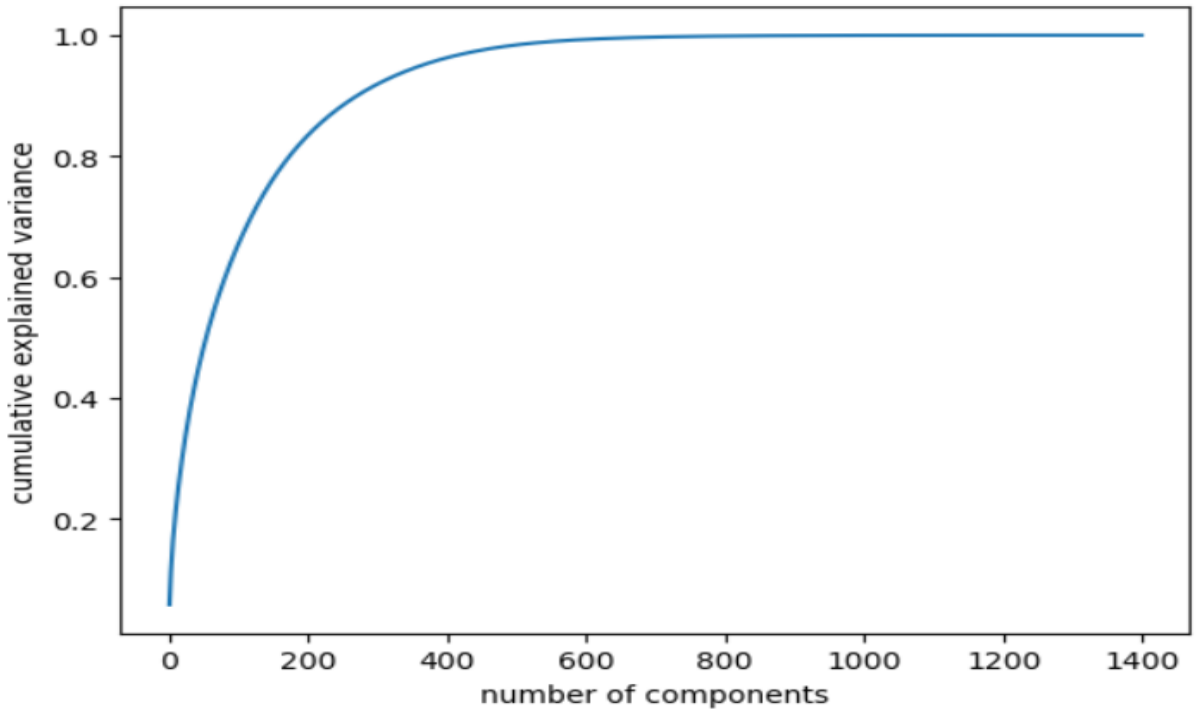
- Thực hiện giảm chiều dữ liệu với PCA:
  Chọn component sao cho variance cao nhất (khoảng 90-99%)

```
from sklearn.decomposition import PCA
pca_model = PCA(n_components=450)
pca_model.fit(x_vecs)
print("Sum of variance ratios: ",sum(pca_model.explained_variance_ratio_))
```

```
Sum of variance ratios:  0.9749782502721588
```

```
x_comps = pca_model.transform(x_vecs)
x_comps.shape
```

Nhìn đồ thị ta có thể thấy n_components ở khoảng 350-500.



- Chuẩn hóa miền giá trị: Sử dụng MinMaxSacler() để miền giá trị trở về (0,1)

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# print(scaler.fit(x_test_transformed))
print(scaler.fit(np.concatenate((x_train,x_test))))

print(scaler.data_max_)
print(scaler.transform(x_train))
print(scaler.transform(x_test))
X_train_W2vec=scaler.transform(x_train)
X_test_W2vec=scaler.transform(x_test)
```

- Sử dụng các mô hình học để phân loại

```
from sklearn.svm import SVC
#Create a svm Classifier

clf = svm.SVC(C=10, gamma=0.1,kernel='rbf')

#Train the model using the training sets
clf.fit(X_train_W2vec,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test_W2vec)
```

```
knn = KNeighborsClassifier(n_neighbors = 2, weights = 'uniform',metric = 'minkowski')
knn.fit(X_train_W2vec,y_train)
y_pred_knn = knn.predict(X_test_W2vec)
```

```
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train_W2vec,y_train)
predictions = naive_bayes.predict(X_test_W2vec)
predictions
```

```
rf = RandomForestClassifier(n_estimators=100,max_depth=6,max_leaf_nodes=9,max_features=None,n_jobs=-1)
rf_model = rf.fit(X_train_W2vec,y_train)
y_pred_rf=rf_model.predict(X_test_W2vec)
```

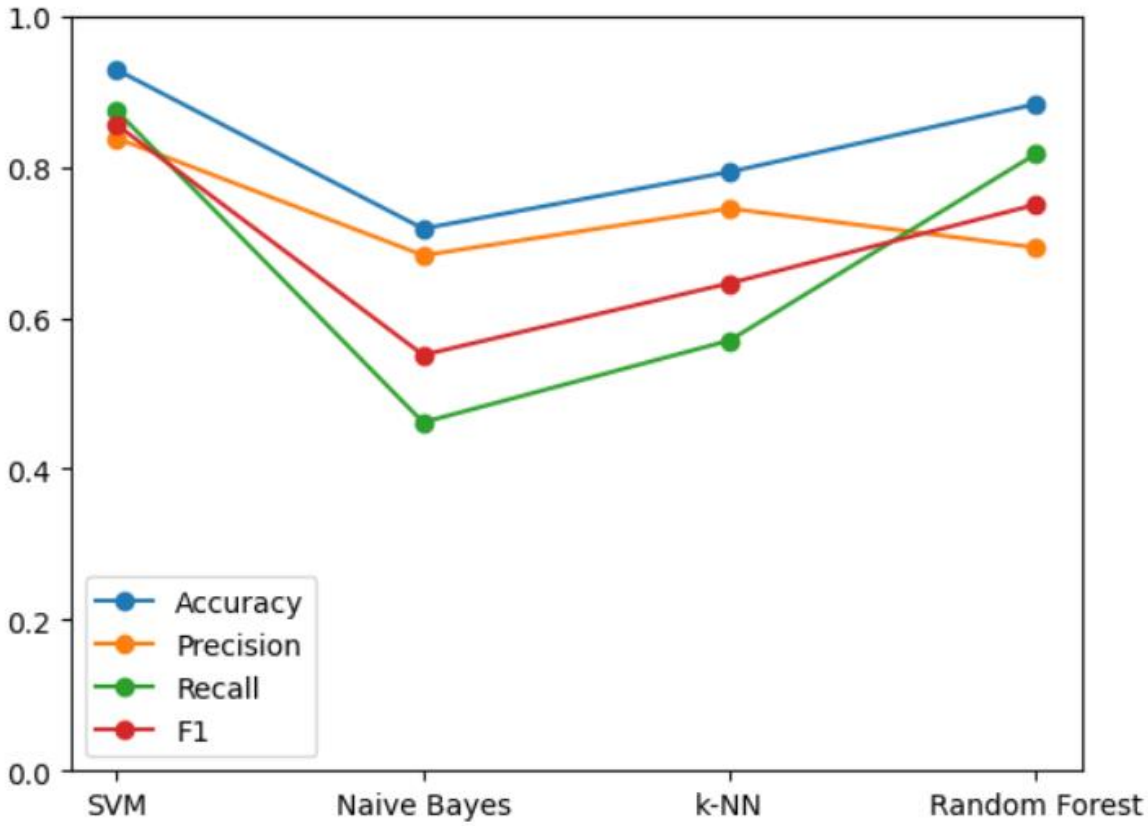
- Tinh chỉnh các siêu tham số:

```
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
"""
param_grid_SVM = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf','linear']}


gs_svm = GridSearchCV(SVC(),param_grid_SVM,cv=5)
gs_svm.fit(X_train_W2vec,y_train)
```

```
from sklearn.neighbors import KNeighborsClassifier
"""
param_grid = {'n_neighbors': list(range(1,9)),'weights' : ['uniform','distance'],'metric' : ['minkowski','euclidean','manhattan']
gs_knn = GridSearchCV(KNeighborsClassifier(),param_grid,cv=5)
gs_knn.fit(X_train_W2vec,y_train)
"""
```

```
from sklearn.ensemble import RandomForestClassifier
"""
param_grid_RF = {'n_estimators': [25, 50, 100, 150],'max_features': ['sqrt', 'log2', None],'max_depth': [3, 6, 9],'max_leaf_nodes
gs_rf = GridSearchCV(RandomForestClassifier(),param_grid_RF,cv=5)
gs_rf.fit(X_train_W2vec,y_train)
"""
```

- Đánh giá hiệu quả các mô hình học:



	Accuracy	Precision	Recall	F1
SVM	0.9284467713787086	0.8379310344827586	0.8741007194244604	0.8556338028169015
Gaussian Naive Bayes	0.7181500872600349	0.6827586206896552	0.46153846153846156	0.5507649513212797
KNN (K=2)	0.7931937172774869	0.7448275862068966	0.5699208443271768	0.6457399103139013
F1	0.8830715532286213	0.6931034482758621	0.8170731707317073	0.75

3.4.2 Một số vấn đề

- Khi sử dụng Word2vec, ta xuất những những phần tử có giá trị âm trong các vector từ. Để đưa vào mô hình Naive Bayes, ta cần phải chuẩn hóa dữ liệu trong khoảng (0,1) sử dụng MinMaxScaler() trên toàn bộ dataset.
- Ta không thể sử dụng MultinomialNB() vì đầu vào của mô hình này yêu cầu dữ liệu rời rạc, trong khi đầu vào ta có khi sử dụng Word2vec là liên tục. Do đó, ta có thể sử dụng GaussianNaiveBayes để cho ra kết quả huấn luyện.

4. Kết luận

Việc thay đổi cách trích xuất đặc trưng email mang lại kết quả rất rõ rệt, cho thấy tầm quan trọng trong lựa chọn cách trích xuất đặc trưng phù hợp với bài toán phân loại. Không những vậy, việc chọn mô hình huấn luyện cũng thể hiện vai trò quan trọng, đặc biệt là SVM cho kết quả huấn luyện tốt trong hầu hết trường hợp.

Kỹ thuật Feature Extraction cần cải tiến nhiều hơn để đối phó với sự phát triển của các kỹ thuật mới của những kẻ lừa đảo theo thời gian. Vì vậy, nhóm dự định sẽ sử dụng thêm một số công cụ khác mạnh hơn trong xử lý ngôn ngữ tự nhiên như BERT,... để trích xuất các đặc trưng mới từ các email thô mới để cải thiện độ chính xác của việc phát hiện email lừa đảo và để đối phó với việc tấn công lừa đảo.

Tài liệu tham khảo

[1] Phishing Website Detection by Machine Learning Techniques

shreyagopal/Phishing-Website-Detection-by-Machine-Learning-Techniques (github.com)

[2] Bag-of-Words Technique in Natural Language Processing: A Primer for Radiologists, *Krishna Juluru , Hao-Hsin Shih, Krishna Nand Keshava Murthy, Pierre Elnajjar ,* 2021

https://pubs.rsna.org/doi/full/10.1148/rg.2021210025

[3] Bag-of-Words Model for Beginners

https://www.kaggle.com/code/vipulgandhi/bag-of-words-model-for-beginners

[4] Basic NLP: Bag of Words, TF-IDF, Word2Vec, LSTM

https://www.kaggle.com/code/reiinakano/basic-nlp-bag-of-words-tf-idf-word2vec-lstm

[5] Word Vectors I: Introduction, SVD and Word2Vec, *CS224n: Natural Language Processing with Deep Learning*, 2019

https://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes01-wordvecs1.pdf

[6] Tutorial: Word Embeddings with SVM

https://www.kaggle.com/code/mehmetlaudatekman/tutorial-word-embeddings-with-svm

[7] Deep Learning for Phishing Detection: Taxonomy, Current Challenges and Future Directions, *Nguyet Quang Do , Ali Selamat , Ondrej Krejcar, Enrique Herrera-Viedma, Hamido Fujita*, 2022

https://core.ac.uk/download/pdf/493008115.pdf

[8] Detecting Phishing Emails Using Machine Learning Techniques , *Sa'id Abdullah Al-Saaidah* , 2017

590422b4d5dd8_1.pdf (meu.edu.jo)

[9] Phishing Email Detection By Using Machine Learning Techniques, *Tariku Yabshe Chiksa*, 2022

Tariku_Yabshe_January_2022.pdf (smuc.edu.et)

[10] Phishing Attacks Detection A Machine Learning-Based Approach, *Fatima Salahdine, Zakaria El Mrabet , Naima Kaabouch*

m74022-salahdine final.pdf (arxiv.org)

Phụ lục: Phân công và đóng góp thành viên

Họ và tên thành viên	MSSV	Phân công nhiệm vụ
Nguyễn Duy Thành	20204691	- Tiền xử lý dữ liệu - Trích xuất đặc trưng dựa trên Content-based
Nguyễn Tiến Đạt	20204640	- Cài đặt mô hình SVM, Naive Bayes
Đào Văn Thiều	20204610	- Cài đặt mô hình KNN, Randon forest
Nguyễn Trần Chung	20204520	- Tinh chỉnh siêu tham số các mô hình - Đánh giá hiệu quả
Trần Nguyễn Anh Tuấn	20200565	- Trích xuất đặc trưng dựa trên BOW, TF-IDF và Word2vec