セキュリティ関連文書のレコメンデーション

2.1 セキュリティ対応で取り扱う情報

SOCやCSIRTなどのセキュリティ対応を行う組織を運用していると、いやおうなく日々多くの情報と向き合うことになります。「情報」と一言で書きましたが、そこには性質も扱い方も様々なものが含まれています。多くの組織で、セキュリティ対応を行っている各人が、それぞれの仕事に必要な情報を収集、編集、作成していることでしょう。

システム開発の視点から見ると、これらの情報は大きく2つに分けられます。機械的な処理がしやすいように定型化されたものと、非定型なものです。

セキュリティ対応組織で使用することがある定型化された情報の例としては、IPアドレスのブラックリストやTCP/IPのポートデータベース、セキュリティ対策を自動化するために策定されたSCAPなどが挙げられます。意味付けをして定型化された情報は機械的な処理がしやすいため、セキュリティ対応を支援するようなシステムでは、これらの情報が広く活用されています。

一方、セキュリティ対応で同じく必要となる定型化されていない情報、例えば自然言語で書かれた文書や画像などについては機械的な処理がしづらく、対応支援システムでも参考情報などで文書として表示する以外の扱いが難しいことがありました。非定型の文書や画像は、そのときに作成したかったレポートなどに記述することで単に蓄積するだけになるなど、その後の活用が人的努力に任されていることが多いのではないでしょうか。

では、非定型の情報を必要なときに取り出す、例えばセキュリティ対応中に関連情報として参照するにはどうするのが良いのでしょうか。

2.2 非定型な情報の取り扱い

このような問題を解決しようとしている技術はいくつかあり、そのうち最初に思い付くのが全文検索システムです。作成した文書を集めて全文検索システムに保管して、必要なときにキーワード検索して探し出すのがとても便利であることは誰もが実感しているでしょう。

一方で、ユーザが何かを探していなくても重要と思われるものを提示するシステムがあります。ショッピングサイトでオススメされる商品や、ニュースサイトなどで表示される関連記事がその一例で、このようなものをレコメンドシステムと呼びます。

便利で利用実績も多いこのような技術を手元にある自然言語で記述された文書に対しても使いたいと考えたとき、全文検索システムの導入は容易でしたが、レコメンドシステムなどのいわゆる集合知を活用するシステムの導入は困難でした。社内にあるセキュリティ対応に必要な情報には関係者外秘のものが多く、それを使用するのも組織内の限られたメンバーのみであるため、そもそものユーザ数が少なく集合知を活用できる程のデータが蓄積できません。

このような前提を踏まえて、ユーザの行動履歴を使わずに非定型の文書そのものの情報だけをもとにお勧めを決める手法を試してみたいと思います。この手法の利点として、ユーザの行動や文書そのものを外部に出すことなく扱えることも挙げられます。ユーザがある文書を選んだとき、それと関連のある文書をお勧めすることを考えてみましょう。

2.3 自然言語処理とトピックモデル

自然言語で書かれた非定型の文書を扱うための技術はどのようなものでしょうか。このような技術は自然言語処理と呼ばれる分野で長く研究されてきており、様々な要素技術が存在しています。その1つに、機械学習の技術で文書データを解析するトピックモデルという手法があります(図-1)。

2.3.1 トピックモデル

文書には様々な種類のものがあります。普段私たちが読む文書に限っても、技術文書とニュース記事は違う種類の文書だと思えます。ニュース記事にも国際情勢を報じたもの、スポーツの結果を報じたものなど、いくつもの種類があります。そして文書の種類によって、使われる言葉の種類や頻度に違いがあると考えられます。また、1つの文書が属する種類は1つであるとは限りません。例えば国際紛争に起因して発生したインターネット上の攻撃のことを書いた文書は、国際情勢と情報セキュリティ、どちらにも属することになるでしょう。

トピックモデルでは文書の属する種類のことをトピックと呼び、文書は次のようにしてできあがると仮定しています。

まず、ある確率分布に従って文書のトピックの混ざり具合が決まります。そこからトピックごとの言葉の出現確率を使って文書が言葉で埋まっていき、最終的な文書ができあがります。実際の文書データから、これらトピックや単語に関する確率分布を求めておけば、今注目されているトピックが何か調べたり、トピックに基づいて文書を分類したりといったことに応用できます。

このような考え方に基づく手法を総称してトピックモデルと呼んでいます。潜在的ディリクレ配分法(Latent Dirichlet Allocation、LDA)という手法を代表として、様々なバリエーションが考え出され研究されています。

関連度の高い文書同士ではトピックの分布も近くなると期待できるので、これを利用して、ユーザに選ばれた文書とトピック分布の近い文書をお勧めとして提示する方法を取ることにします。本稿ではトピック分布の計算にはLDAを使います。

2.3.2 自然言語処理技術による前処理

LDAのアルゴリズムは文書のリストを渡すだけで処理してくれるわけではなく、テキスト内の単語とその出現回数を入力するものです。つまり、これに合うよう文書に様々な前処理を施す必要があります。同時に、LDAのモデル生成の際の精度向上や必要なデータ量の低減を狙い、不要と考えられる情報を削減します。

必要とされる前処理は以下のようなものです。

1. 文書データから本文を抽出
   文書データから本文以外の部分を取り除きます。
2. 単語に分割
   英語の文章であれば、基本的にはスペースや改行で分割すれば概ね実現できます。ただし、例えば一行の文字数制限でハイフネーション後に改行された場合など、いくつか対応が必要なポイントがあります。活用形を同じ単語として扱いたければ[*1]、stemmingやlemmatizingといった自然言語処理の技術を使います。日本語の文章

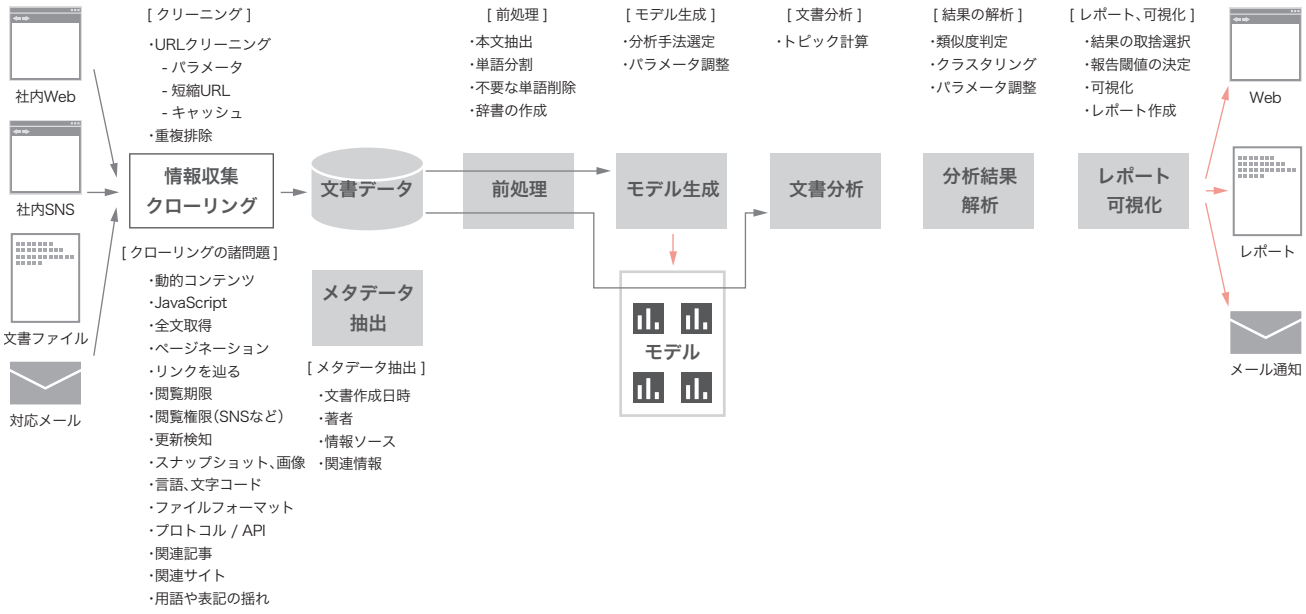

図-1 トピックモデルを利用した自然言語処理の全体像

*1 例:word - wordsやwrite - wrote - written。

では単語を分ける明確な区切りがないので、形態素解析と呼ばれる処理を使うことになるでしょう。

3. 「不要」な単語の削除
   例えばどの文書にも頻繁に登場する「です／ます」などは文章の内容にはあまり関係ないと考えられます。こうした、目的に対して役立ちそうにもない単語でデータが大きく膨らんだところで分析精度が向上するとは考えにくいので、前処理の段階で削除しておくのが常套手段になっています。
   削除対象とする単語をあらかじめ決めておくストップワードと呼ばれる方式のほか、入力テキスト中で登場回数の多い単語を削除する方式、ごく稀にしか使われない単語も削除する方式などが使われています。どのような単語を削除すると効果的なのかは、先例にならいながら目的に応じて試行錯誤することになります。
4. 文書ごとの単語別頻度表を作成
   文書内の単語の登場回数を数え上げます。このような単語の出現頻度表をBag of Wordsと呼び、それぞれの文書に1つずつ頻度表を作成します。LDAに実際に渡すのはこれらの頻度表のリストです。

この手順でお気づきのとおり、Bag of Wordsに単語の登場順序は反映されないので、前後関係や文脈といった情報はLDAでは考慮されないことが分かります。

2.4 プロトタイプ作成

これらの技術を検証するために、実際にプロトタイプを作成してみました。本稿では米国の非営利団体MITRE社[*2]が公開している脆弱性情報データベースCVE[*3]のsummaryを使用することにします。英文の自然言語処理に関する技術は多くがライブラリ化されているので、それを使用するだけで多くのことができます。

CVE-2018-5383

Bluetooth firmware or operating system software drivers in macOS versions before 10.13, High Sierra and iOS versions before 11.4, and Android versions before the 2018-06-05 patch may not sufficiently validate elliptic curve parameters used to generate public keys during a Diffie-Hellman key exchange, which may allow a remote attacker to obtain the encryption key used by the device.

図-2 CVE IDごとにサマリ文書を保存したテキストファイルの例

まずは元となるデータを作成しましょう。今回は2018年にリリースされたCVE 7,692件を処理対象にします。米NISTが提供しているNVD Data Feeds[*4]のページから、CVEのデータをダウンロードします。JSON形式のものもベータ版として提供されていましたが、今回は以前に扱ったことがあるXML形式のデータを取得しました。このデータにはCVE IDや公開日時、最終更新日時、関連情報の参照先などが、計算機でも扱いやすい形に構造化されて含まれていますが、今回はこの中からあえて非定型の自然言語記述（vuln:summary）だけを取り出して、CVE IDごとに別のファイルに保存します（図-2）。これで元となるテキストファイルが準備できました。

2.4.1 文章の前処理

LDAモデル作成に必要な一連の処理ではPythonのgensim[*5]やnltk[*6]というライブラリを使用します。

まずは元となる文書にLDAモデルを作成するために必要な前処理を施します。今回ターゲットとしたCVEのデータには含まれていませんでしたが、例えばWebサイトに掲載された英文に対する定番の処理として、

- HTMLタグの除去やHTML特殊文字の処理
- 行末でハイフネーションされた単語の接続

などがあります。

次に単語分割（tokenize）、lemmatize、ストップワード除去と呼ばれる一連の処理をして文書から単語データを生成します。lemmatizeは関数に文字列を渡すだけですが、内部では名詞、動詞、形容詞、副詞だけを抽出し活用形を原型に揃えるなど、非常に多くの複雑な処理をしています。

＊2　MITRE社（https://www.mitre.org/）。
＊3　CVE（https://cve.mitre.org/）。
＊4　NVD Data Feeds（https://nvd.nist.gov/vuln/data-feeds）。
＊5　gensim（https://radimrehurek.com/gensim/）。
＊6　nltk（http://www.nltk.org/）。

ここまでの処理を対象とするそれぞれの文書に適用して、取り出した単語データのリストを作成します。

```
def normalize(txt):
    # De-hyphenation of words across a line-break(行末のハイフンを除去)
    txt = re.sub(r'-\n', '', txt)
    # Concatenate lines(改行を除去)
    txt = re.sub(r'\n', ' ', txt)
    # Tokenization and lemmatization (単語分割と lemmatize)
    tokens = [ re.sub(r'/[A-Z]+$', '', x.decode('utf-8'))
                for x in gensim.utils.lemmatize(txt) ]
    # Remove stop-words(ストップワードを除去)
    stopwords = nltk.corpus.stopwords.words('english')
    tokens = [ token for token in tokens if token not in stopwords ]
    return tokens

docs = []
for path in files:
    with open(path, encoding='utf-8') as f:
        txt = ''.join(f.readlines())
    tokens = normalize(txt)
    docs.append(tokens)
```

normalize関数中で使用しているgensim.utils.lemmatize()関数は、単語の後ろに品詞の情報をつけて返すという仕様のため、正規表現でそれを取り除いています。

```
[b'cve/VB',
 b'high/JJ',
 b'rate/NN',
 b'vlan/NN',
 ...
```

2.4.2 辞書とBag of Wordsを作成

今回使用したgensimライブラリでは、単語にIDを振って扱います。作成した単語データのリストを元に単語にIDを割り振り、それぞれの文書に含まれている単語を数え上げます。このデータ構造のことを辞書と呼びます。

その後、その辞書の内容を解析して整理します。具体的には、

- 共通して出現することが多い単語
  (例:20%以上の文書に含まれている単語)
- 滅多に出現しない単語
  (例:1つの文書にしか出現しない単語)

などを取り除くのが定式となっているようです。

この処理を省いたりパラメータを変更する実験をしてみたところ、共通して出現する単語を除去しないで作ったモデルで、分類の精度が落ちる傾向が見てとれました。滅多に出現しない単語を取り除く処理に関しては、実験した範囲では目に見える程の効果は確認できませんでした。

整理した辞書を使用して、文書ごとにBag of Words(BoW)ベクトルと呼ばれるものを作成します。これは辞書に含まれる単語が文書中に出現する回数をベクトルで表したものです。

```
dic = gensim.corpora.Dictionary(docs)
dic.filter_extremes(no_above=0.2, no_below=1)
bow = [ dic.doc2bow(doc) for doc in docs ]
```

2.4.3 LDAモデルを生成

辞書とBag of WordsからLDAモデルを生成して、モデルとここまで作った関連データをファイルに保存します。

LDAモデルの生成時にトピック数を指定する必要があります。このトピック数ですが、元となる文書や出現する単語の量、偏り方などによって適正な値が変わるようで、その決め方にもいろいろな流儀があるようです。対象とする文書や数を変更して実験を繰り返してみたところ、手元では30～50の値を指定したときに比較的良い結果が見えるモデルが生成できたことが多かったため、ここでは50を指定しました。

```
lda = gensim.models.ldamodel.LdaModel(bow, id2word=dic, num_topics=50)

lda.save(filename_model)
dic.save(filename_dic)
gensim.corpora.MmCorpus.serialize(filename_corpus, bow)
```

2.5 生成したモデルを使用して文書を分析

このモデルを使って、さっそく対象となる文書を分析してみます。それぞれの文書がどのようなトピックを含んでいるか、結果を得ることができました。

```
results = []
for doc in docs:
    bow = lda.id2word.doc2bow(doc)
    doc_topics = lda.get_document_topics(bow)
    results.append(doc_topics)
```

2.5.1 類似した文書をピックアップする

文書ごとに持つトピックの分析結果を使って、ベクトル間のコサイン類似度を計算することで、すべての文書の中からトピック成分の近い文書を選び出すことができます。そこで、最近

ニュースなどで話題になったCVEをいくつか選んで、それと類似したものを選び出すことができるか検証してみました。

例えば8月のMicrosoft月例パッチで修正されたCVE-2018-8373（図-3）を調べると、同様の「Scripting Engine Memory Corruption Vulnerability」に関するCVE（CVE-2018-0955、CVE-2018-0996、CVE-2018-1001、CVE-2018-8267など）がずらっとリストアップされました。

```
[('CVE-2018-0955', 1.0),
 ('CVE-2018-0988', 1.0),
 ('CVE-2018-1001', 1.0),
 ('CVE-2018-8267', 1.0),
 ('CVE-2018-8353', 1.0),
 ('CVE-2018-8371', 1.0),
 ('CVE-2018-8373', 1.0),
 ('CVE-2018-8389', 1.0),
 ('CVE-2018-0996', 0.9999999),
 ('CVE-2018-8242', 0.9999997),
 ('CVE-2018-0839', 0.9968462),
 ...
 ('CVE-2018-8385', 0.9579928),
 ...
 ('CVE-2018-8372', 0.8273082),
 ('CVE-2018-8355', 0.8272891),
 ...
 ('CVE-2018-8359', 0.7355896),
```

類似判定された文書をピックアップして読んでみると、確かにほぼ定型文と言っていいほどよく似ているものが多く見られます。

今回のCVE-2018-8373サマリの中に出現している関連CVEがどの程度類似しているかを調べると、0.73 〜1.0の範囲に出現していました。しかし、この範囲のコサイン類似度を持つCVEは178件あります。類似した文書をピックアップすること自体はできているとも言えるかもしれませんが、ユーザが見たいものを提示するためにはこの類似度をどう扱うべきか、もうひと工夫必要そうです。

続いて、Foreshadow-NGなどとして知られるCVE-2018-3620（図-4）「CPUの投機的実行機能に対するサイドチャネル攻撃」と類似したCVEを計算してみると、関連するCVE-2018-3646、CVE-2018-3615などを含むリストを作成できました。しかし、それらと似たようなコサイン類似度でPHP製のチャットプログラムやWebアプリケーションフレームワークの脆弱性もリストに入っています。トピックのコサイン類似度だけを閾値として判断するとあまりよい結果が得られない可能性があります。

```
[('CVE-2018-3620', 0.9999924),
 ('CVE-2018-3646', 0.9803927),
 ('CVE-2018-3640', 0.88989854),
 ('CVE-2018-3693', 0.8448397),
 ('CVE-2018-3615', 0.8389072),
 ('CVE-2018-5954', 0.8318751),
 ('CVE-2018-1000181', 0.80068177),
 ...
```

このようにして、トピックモデルとコサイン類似度を用いてCVEのsummaryからそれと類似するCVEを選び出した場合、似た傾向のものをある程度ピックアップできることが確かめられました。

一方で、あまり似ていなさそうな内容の文書までまとめて選び出してしまうケースが多く見られました。また、期待する文書が選び出されていないケースもありました。

CVE-2018-5390（図-5）はLinux kernelのTCP実装にDoS脆弱性があったものですが、上位に類似判定された文書にはそのような内容の文書は選び出されていませんでした。例えば、Linux kernelの脆弱性は対象としたCVEに105件含まれていましたが、それらの類似度は高くありませんでした。

```
[('CVE-2018-5390', 0.99944544),
 ('CVE-2018-1237', 0.81856203),
 ('CVE-2018-1240', 0.8044424),
 ('CVE-2018-1217', 0.80138516),
 ...
```

A remote code execution vulnerability exists in the way that the scripting engine handles objects in memory in Internet Explorer, aka "Scripting Engine Memory Corruption Vulnerability." This affects Internet Explorer 9, Internet Explorer 11, Internet Explorer 10. This CVE ID is unique from CVE-2018-8353, CVE-2018-8355, CVE-2018-8359, CVE-2018-8371, CVE-2018-8372, CVE-2018-8385, CVE-2018-8389, CVE-2018-8390.

図-3 CVE-2018-8373 のサマリ

Systems with microprocessors utilizing speculative execution and address translations may allow unauthorized disclosure of information residing in the L1 data cache to an attacker with local user access via a terminal page fault and a side-channel analysis.

図-4 CVE-2018-3620 のサマリ

Linux kernel versions 4.9+ can be forced to make very expensive calls to tcp_collapse_ofo_queue() and tcp_prune_ofo_queue() for every incoming packet which can lead to a denial of service.

図-5 CVE-2018-5390 のサマリ

これらの結果に影響を与える要素としては、コサイン類似度の閾値をどれくらいに置くかということ以外に、モデル生成時に各段階で与えることができるパラメータがあります。本稿で例示した以外にも実験を繰り返した結果、これらのさじ加減によって結果が大きく変わることがありました。それぞれのパラメータ調整をすることが、結果をつぶさに見る前には難しいという、トピックモデルの扱いづらさも垣間見ることができました。

2.5.2 周辺情報の活用などによる精度向上

トピックモデルは、非定型の自然言語で記述された文書群から類似したものを手早く選び出す手法として有力なものの1つです。一方で、読み手が期待する文書を適切にピックアップできるかで出力結果を評価するならば、その精度を上げていくためには用途に応じた工夫が必要です。なぜなら、読み手が期待する「類似文書」は場面によって異なると考えられるからです。

例えば、日々のニュースにおいて類似の記事をまとめたい場合は直近の情報以外は邪魔になると考えられます。しかし、発生の稀な問題に対応するため関係する文書を調べているときは、読み手は古い情報まで含めて類似の文書を探し出したいのではないでしょうか。

トピックモデルの学術研究ではこうしたニーズの多様性に応えようと、モデルの構造を工夫して、特定用途での精度向上を狙った研究も進められています。例えばトピックの時系列変化を分析に採り入れたモデルや著者名を分析に採り入れたモデルなどがあります。

今回は、CVEやセキュリティ関連の文書をターゲットに、単純に周辺情報を活用したフィルタリングを実験してみました。例えば、社内の内部向けメモのようなものは、作成日時の前後に作られたものを優先することで、ユーザが読みたいものを採り上げる精度を上げることができました。社内の文書をターゲットにした場合、プロジェクト名や文書の置かれているパス名に含まれているキーワードでの絞り込みも有効です。

もっとシチュエーションに依存しない絞込み方法で精度向上できないかを試してみたところ、DBSCANなどのPythonのscikit-learn[*7]ライブラリに含まれるクラスタリングアルゴリズムを使い、トピックのクラスタリングを行った結果をもとにリストをフィルタリングすることで良好な振り分けができる場合があることが確認できました。しかし、この手法もクラスタリングを行う際のパラメータ設定で大きくクラスタの様子が変わるためチューニングが必須で、シチュエーションに依存せず安定的に使うことは難しそうです。

今回、非定型の情報をうまく扱う手段を探して、自然言語処理とトピックモデルを中心にいろいろと実験してみました。1つの手法だけで満足のいく結果を得られるものはありませんでしたが、目的に応じて複数の手段を組み合わせることで、望みの出力に近づける工夫はできそうなことが分かりました。これらの知見から、一定の条件下で非定型の文書を取り扱う場面への適用を考えていきます。

執筆者:
永尾 禎啓（ながお ただあき）
IIJ セキュリティ本部 セキュリティ情報統括室 シニアエンジニア。
1998年4月入社。セキュリティサービス開発やSDN開発などを経て、現在は理論的視点から情報セキュリティ全般の調査活動に従事。
IIJグループの緊急対応チーム、IIJ-SECT メンバー。

執筆者:
桃井 康成（ももい やすなり）
IIJ セキュリティ本部 セキュリティ情報統括室 リードエンジニア。
1999年1月入社。セキュリティサービスや無線ICタグ関連システムなど各種サービスの研究開発を経て、現在は情報セキュリティ全般に関わる調査研究活動に従事。IIJ-SECTメンバーとして、日本セキュリティオペレーション事業者協議会(ISOG-J)、ICT-ISACなどの活動や運営に参加。

*7　scikit-learn(http://scikit-learn.org/)。