Elasticsearchで日本語NLPモデルを利用してセマンティック検索を実現する

前提条件

セマンティック検索を実行するまでの流れ

Elandのインストール

NLPモデルのインポート

ベクトル埋め込みを利用したセマンティック検索の実装

テキスト分類（感情分析）

フィードバック

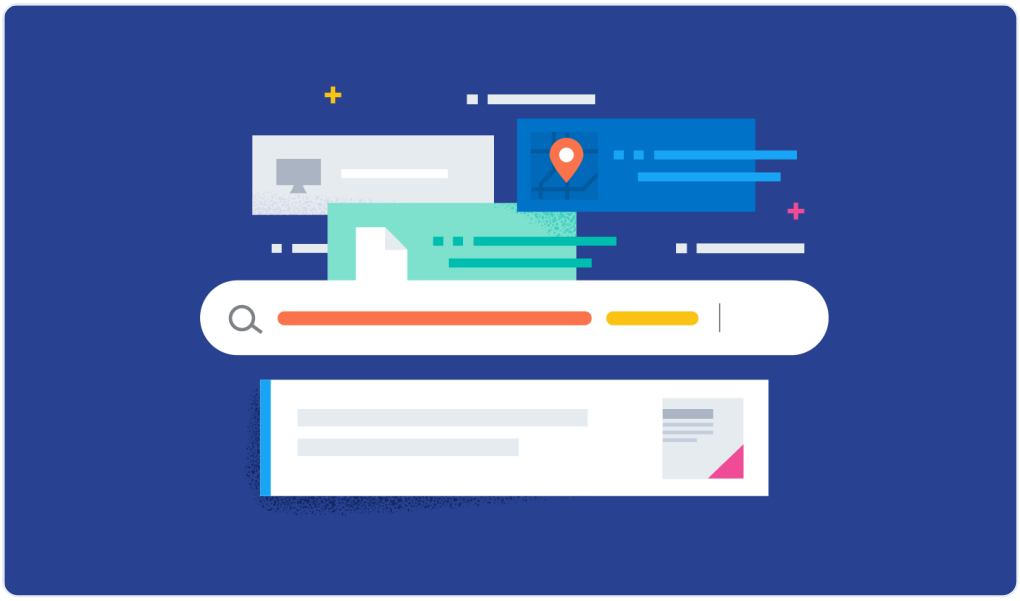
おわりに

終了

Elasticsearchで日本語NLPモデルを利用してセマンティック検索を実現する

著者 **杉森 大輔**

2023年8月31日



組織内のドキュメントや製品情報など日々大量に生成される文書の中から必要なものを素早く探し出すことは、日々の業務、あるいは日常生活の中でも非常に重要な作業です。しかし検索対象になるドキュメントが大量に存在する場合、例えコンピューターを使っていたとしても、その場で全てのドキュメントを読み込み直して目的の文書を見つけ出すことは非常に時間がかかる処理になります。そこでElasticsearchのような検索エンジンと呼ばれるソフトウェアが登場しました。検索エンジンを使うと事前に検索インデックスと呼ばれる索引データが作成され、見つけたいドキュメントに含まれるキーワードを使って検索するだけで素早くその目的を達成できます。

しかし、中には探したい情報のイメージはあっても、適切なキーワードを思い出せなかったり、同じ意味でも別の表現で検索してしまったりすることがあります。Elasticsearchでは類義語、同義語（シノニム）を定義してこれらを適切に処理する機能もありますが、単純な対応表を使うだけで適切なクエリーに変換することは難しいケースもあります。

そこでElasticsearchではv8.0で文章の意味（セマンティック）を使って検索をするベクトル検索の機能をリリースしました。それに合わせ、ベクトル検索やそのほかのNLPタスクをElasticsearch上で処理する方法について**ブログシリーズ**でも取り上げています。しかしv8.8リリース時点までは英語以外のテキストを正しく分析することができていませんでした。

そこでElasticではテキスト分析処理において日本語を適切に分析する機能を実装し、v8.9でリリースしました。この機能によって、日本語のテキストもベクトル検索のようなセマンティック検索や、日本語の感情分析のような自然言語処理タスクをElasticsearchで利用できるようになっています。この記事ではこれらの機能の具体的な利用方法をステップバイステップで紹介します。

前提条件

実際のセマンティック検索を実装する前に、この機能を利用するための前提条件について確認しておきましょう。Elasticsearchのクラスターにおいて、それぞれのノードは**ノードロール**と呼ばれる役割を割り当てられています。そして機械学習モデルを動作させるのはElasticsearchの**MLノード**になります。従ってこの機能を利用する際には、Elasticsearchのクラスター内にMLノードが起動している必要がありますので、事前に確認してください。また、MLノードの利用はPlutinumライセンス以上が必要となります。ただし動作確認をしたい場合にはトライアルライセンスでも可能です。開発環境等での動作検証の場合はKibanaの画面、または**API**経由でトライアルを有効化してください。

セマンティック検索を実行するまでの流れ

Elasticsearchでセマンティック検索を実行する場合、以下のようなステップを踏む必要があります。

1. （事前準備）作業端末へのElandおよび関連ライブラリのインストール
2. 自然言語処理タスクを実現するための機械学習モデルのインポート
3. インポートした機械学習モデルでのテキスト分析結果のインデキシング
4. 機械学習モデルを利用したkNN検索

また自然言語処理はセマンティック検索を実現するためだけのものではありません。このブログの後半ではテキスト分類のタスクを実現する機械学習モデルを利用して、テキストの感情分析（ポジティブ・ネガティブの分類）を実現する例も紹介します。

では、それぞれの具体的な方法について以下で詳しく解説します。

Elandのインストール

現在Elasticsearchは自然言語処理をするためのプラットフォームとして振る舞うことができるようになりました。しかし、実は具体的な自然言語処理がElasticsearch上に実装されているわけではありません。必要な自然言語処理は、機械学習モデルとしてユーザーがElasticsearchにインポートする必要があります。このインポート処理は、Elandを使って実現できます。このように外部のモデルを自由にインポートできるようにすることで、ユーザーが必要とする機械学習の機能をオンデマンドで追加することができるようになっています。

ElandはElasticが提供するPythonライブラリで、ElasticsearchのデータとPyTorchやscikit-learn などのPythonの充実した機械学習ライブラリを連携させるための機能を提供しています。このElandにバンドルされる eland_import_hub_model というコマンドラインツールを利用すると、Hugging Faceで公開されているNLPモデルをElasticsearchにインポートすることができます。以下、この記事ではコマンドラインベースの作業はGoogle Colaboratory等、PythonのNotebookを使って実行することを前提とします（もちろんMacやLinux等それ以外のターミナルを利用することもできます。その場合は先頭の!を無視してください）。

まずは依存しているライブラリをインストールします。

```
!pip install torch==1.13
!pip install transformers
!pip install sentence_transformers
!pip install fugashi
!pip install ipadic
!pip install unidic_lite
```

Copy

ここで、fugashi, ipadic, unidic_liteは日本語モデルを利用する際に必要となります。

これらのライブラリがインストールできたらElandをインストールします。日本語のモデルを利用するためにはElandの8.9.0以降が必要になりますので、バージョンには注意してください。

```
!pip install eland
```

Copy

インストールできたら以下のコマンドで利用できる様になっているかを確認しましょう。

```
!eland_import_hub_model -h
```

Copy

NLPモデルのインポート

ベクトル検索の大きな実現方法については、**こちらの記事**にある英語における実現方法と同様です。ここでは復習も兼ねて一通り同様の手順を紹介したいと思います。

前説で説明した通りElasticsearchでNLP処理を実現するためには、そのNLPを実現するための機械学習モデルをElasticsearchにインポートしないといけません。機械学習モデルはPyTorchを利用して自分で実装することも可能ですが、そのためには機械学習と自然言語処理についての十分な知識と、機械学習に必要となるマシンパワーが必要です。しかし現在はHugging Faceという機械学習・自然言語処理の研究者や開発者が活発に利用しているウェブ上のリポジトリがあり、ここで多くのモデルが公開されています。今回はこのHugging Face上で公開されているモデルを使ってセマンティック検索を実装していきます。

それでは、日本語の文章を数値列にエンベッド（ベクトル化）するためのモデルをHugging Faceで選定しましょう。今回の記事では以下のモデルを利用することにします。

- **cl-tohoku/bert-base-japanese-v2**

ここで日本語のモデルを選定する際のv8.9における注意を列挙します。

まず、モデルのアルゴリズムとしては**Bert**のみをサポートしています。Hugging Face上のタグなどで対象のNLPモデルがBertで学習されたモデルであることを確認してください。

また、BertなどのNLPタスクでは、入力されたテキストを単語レベルで分割するpre-tokenizeという処理が行われます。この際に日本語のpre-tokenizeに利用されるのが、日本語形態素解析エンジンです。Elasticsearch v8.9では、MeCabによる形態素解析をサポートしています。Hugging Faceのモデルページから Files and versions タブを開き、tokenizer_config.jsonファイルの内容を確認してください。ここでword_tokenizer_typeの値がmecabとなっていることを確認してください。

```
{
  "do_lower_case": false,
  "word_tokenizer_type": "mecab",
  "subword_tokenizer_type": "wordpiece",
  "mecab_kwargs": {
    "mecab_dic": "unidic_lite"
  }
}
```

Copy

もしも利用したいモデルのword_tokenizer_typeがmecab以外の場合、残念ながら現時点ではそのモデルはElasticsearchで取り扱うことはできません。もし具体的に別のword_tokenizer_typeのサポートが必要なケースがありましたら、ぜひフィードバックをお寄せください。

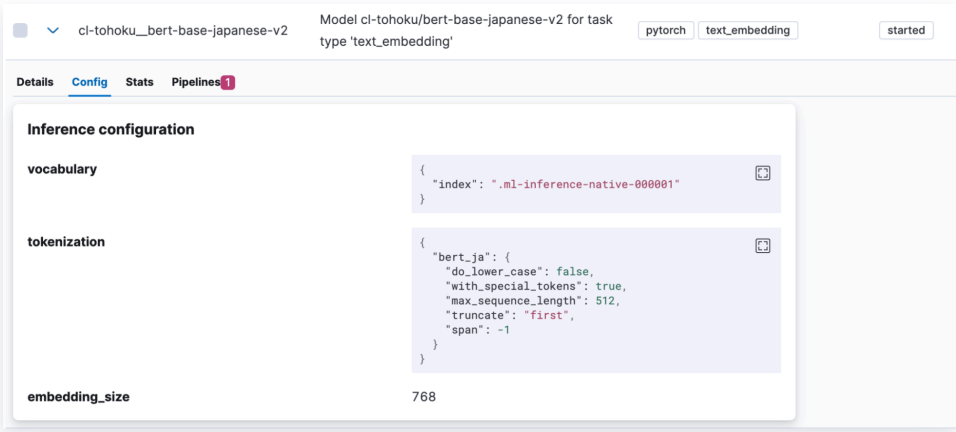
さて、インポートするモデルが決まったらあとは英語のモデルをインポートする時と手順は同じです。まずeland_import_hub_modelを使ってモデルをElasticsearchにインポートします。eland_import_hub_modelの使い方については**こちらのページ**を参照してください。

```
!eland_import_hub_model \
--url "https://your.elasticserach" \
--es-api-key "your_api_key" \
--hub-model-id cl-tohoku/bert-base-japanese-v2 \
```

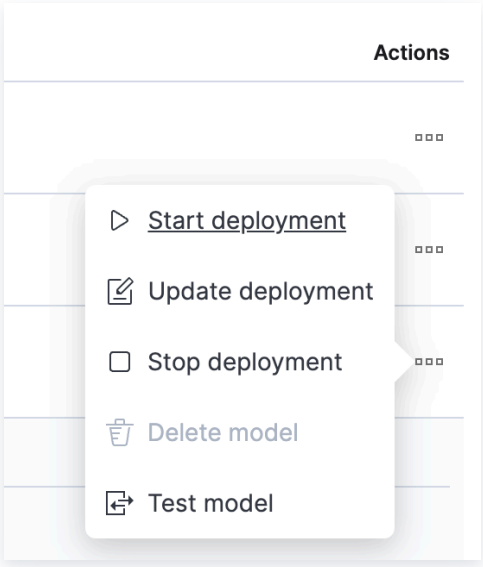
```
--task-type text_embedding \
--start
```

Copy

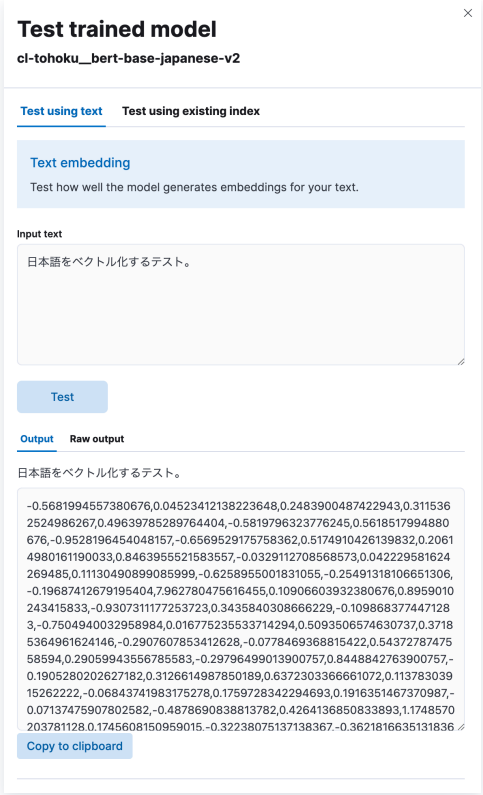
正常にインポートが完了すると、KibanaのMachine Learning > Model Management >Trained Modelsにインポートしたモデルが表示されます。ここでモデルのConfigタブを開くと、tokenizationとして”bert_ja”が使われており、日本語を扱うモデルとして正しく登録されていることがわかります。



モデルがアップロードできたらテストしてみましょう。Actions列に表示されているボタンをクリックしてメニューを開きます。



ここでTest modelを選択し、Input textに任意の日本語の文章を入力してTextボタンをクリックします。



するとこのように、このモデルを使って入力された日本語のテキストが数値列にベクトル化できていることが分かります。正しく動作しているようですね。

ベクトル埋め込みを利用したセマンティック検索の実装

モデルがアップロードできたので、いよいよElasticsearchにセマンティック検索（ベクトル検索）の機能を実装していきます。

まず、ベクトル検索をするためには、インデックスに元の日本語テキストをエンベッドしたベクトル値がインデックスされている必要があります。そこで、先ほどアップロードしたモデルを使ってインデックスに投入される前に日本語テキストをベクトル化する**inferenceプロセッサー**を含むパイプラインを作成します。

```
PUT _ingest/pipeline/japanese-text-embeddings
{
 "description": "Text embedding pipeline",
 "processors": [
  {
   "inference": {
    "model_id": "cl-tohoku__bert-base-japanese-v2",
```

```
  "target_field": "text_embedding",
  "field_map": {
   "title": "text_field"
  }
 }
```

Copy

Inferenceプロセッサーを利用すると、対象のフィールド（ここではtitle）に保存されているテキストに対してmodel_idで指定したモデルの処理を適用し、結果をtarget_fieldの値に格納します。また各モデルはそれぞれ個別のフィールド（ここではtext_field）を処理の入力値として期待しています。そこで処理の対象となる実際の入力フィールドとMLモデルが期待するフィールド名を合わせるためにfield_mapにその対応を指定します。

パイプラインが登録できたら、これを利用してインデックスを作成します。作成するインデックスにはベクトルを保存するフィールドが必要なため、適切にmappingを定義しておきます。以下の例では `text_embedding.predicted_value` というフィールドに768次元の **dense_vector**（密ベクトル）型のデータを保持できるように設定しています。注意点として、この「768次元」という数値はモデルによって異なります。Hugging Faceのページなど（モデルのconfig.jsonにあるhidden_sizeの値）を確認して適切な数値を設定してください。

```
PUT japanese-text-with-embeddings
{
 "mappings": {
  "properties": {
   "text_embedding.predicted_value": {
    "type": "dense_vector",
    "dims": 768,
    "index": true,
    "similarity": "cosine"
   }
  }
 }
}
```

Copy

すでに検索対象の日本語テキストデータを含むインデックスがある場合は、reindex APIを利用できます。ここではjapanese-textというインデックスに元のテキストデータがあるものとし、それを元にjapanese-text-embeddingsというインデックスにテキストをベクトル化したドキュメントを登録しています。

```
POST _reindex?wait_for_completion=false
{
 "source":{
  "index": "japanese-text"
 },
 "dest":{
  "index": "japanese-text-with-embeddings",
  "pipeline": "japanese-text-embeddings"
 }
}
```

Copy

あるいはテストのために直接ドキュメントを登録する場合、以下のように作成したpipelineを指定してインデックスに書き込みます。

```
POST japanese-text-with-embeddings/_doc?pipeline=japanes
{
 "title": "日本語のドキュメントをベクトル化してインデックスに
}
```

Copy

ベクトル化されたドキュメントの登録が完了したらいよいよ検索が可能です。ベクトルを使った検索としては**knn**（K最近傍検索）という手法が利用可能です。ここで通常の_search APIでquery_vector_builderというオプションを使ってknnのベクトル検索をしたいと思います。query_vector_builderを使うと、model_idで指定したモデルを使い、model_textで指定したテキストをエンベッドしたベクトルをクエリーに変換することができます。

```
GET japanese-text-with-embeddings/_search
{
 "knn": {
  "field": "text_embedding.predicted_value",
  "k": 10,
  "num_candidates": 100,
  "query_vector_builder": {
   "text_embedding": {
    "model_id": "cl-tohoku__bert-base-japanese-v2",
    "model_text": "日本語でElasticsearchを検索したい"
   }
  }
 }
```

Copy

この検索クエリーを実行すると、以下の様なレスポンスが得られます。

```
 "hits": [
  {
   "_index": "japanese-text-with-embeddings",
   "_id": "vOD6MIoBdRdLZd7EKaBy",
   "_score": 0.82438844,
   "_source": {
    "title": "日本語のドキュメントをベクトル化してインデック
    "text_embedding": {
     "predicted_value": [
      -0.13586345314979553,
      -0.6291824579238892,
      0.32779985666275024,
      0.36690405011177063
```

Copy

検索できました！　また検索内容には日本語をエンベッドしたフィールドが含まれています。実際のユースケースではレスポンスに含まれる必要がないことが多いと思われます。その場合は_sourceパラメーターなどを利用してレスポンスから除外するなどの対応をしてください。

また検索ランキングについてチューニングする際は、ベクトル検索と通常のキーワード検索の結果をうまくブレンドする**Reciprocal rank fusion (RRF)**という機能もリリースされています。こちらも合わせて確認してください。

ベクトル検索を使ったセマンティック検索を実現する流れについては以上になります。通常の検索に比べ、若干手間が必要だったり機械学習特有の用語が出てきますが、一度設定してしまえば検索自体は通常とほとんど変わらない方法で実現できますので、ぜひ一度試してみてください。

テキスト分類（感情分析）

日本語でkNNを用いたベクトル検索が利用できることが確認できたので、他のNLPタスクも同様に利用できることを見ていきたいと思います。

Text classificationは入力されたテキストを、何らかのカテゴリーに分類する処理を行うタスクです。ここでは入力された日本語のテキストがポジティブな感情を伴うものか、逆にネガティブなものなのかを判定する感情分析のモデル(**koheiduck/bert-japanese-finetuned-sentiment**) がHugging Faceにありましたので、そちらを利用してみたいと思います。tokenizer_config.jsonを確認するとこのモデルもword_tokenizer_typeとしてmecabを利用していることがわかりますので、Elasticsearchのbert_jaで利用可能です。

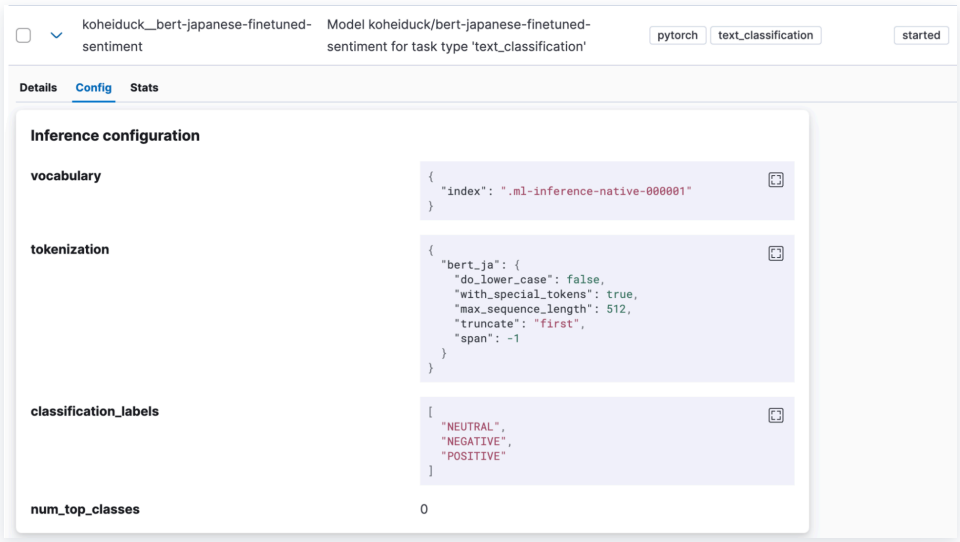
先ほどと同様にこのモデルをElandを使ってElasticsearchにインポートします。

```
!eland_import_hub_model \
--url "https://your.elasticserach" \
--es-api-key "your_api_key" \
--hub-model-id koheiduck/bert-japanese-finetuned-sentiment
--task-type text_classification \
--start
```

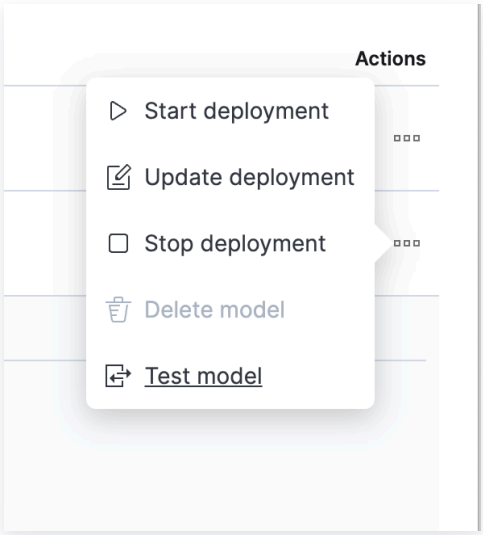
Copy

正常にインポートが完了すると、KibanaのMachine Learning > Model Management >Trained Modelsにインポートしたモデルが表示されま

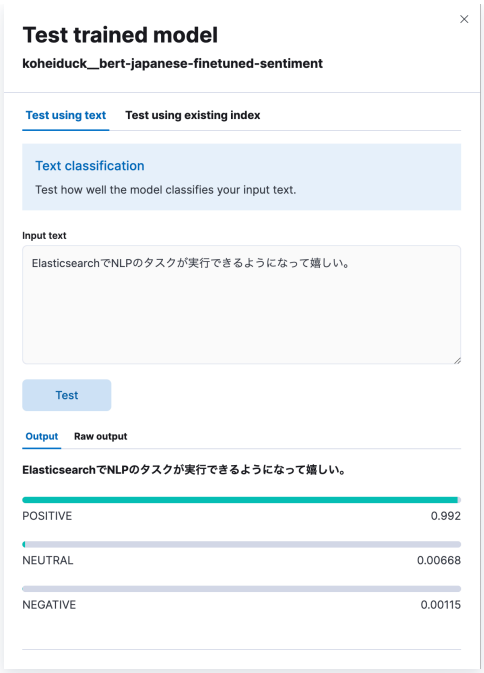
す。



ここでもActionsのメニューから「Test model」をクリックしましょう。



するとこれも先ほどと同様にテストようのダイアログが表示されます。ここに分類するテキストを入力すると、そのテキストが POSITIVE / NEUTRAL / NEGATIVE に分類されます。試しに「ElasticsearchでNLPのタスクが実行できるようになって嬉しい。」と入力すると、以下のように99.2%ポジティブという結果になりました。



また、同様の処理をAPIで実行する場合は以下のようになります。

```
POST _ml/trained_models/koheiduck__bert-japanese-finetun
{
 "docs": [{"text_field": "ElasticsearchでNLPのタスクが実行でき
 "inference_config": {
  "text_classification": {
   "num_top_classes": 3
  }
 }
}
```

Copy

レスポンスは以下の通り。

```
{
 "inference_results": [
  {
   "predicted_value": "POSITIVE",
   "top_classes": [
    {
     "class_name": "POSITIVE",
```

```
      "class_probability": 0.9921651090124636,
      "class_score": 0.9921651090124636
    },
    {
      "class_name": "NEUTRAL",
```

Copy

この処理も当然inferenceプロセッサーで実行できるため、日本語のテキストがインデックスされる際、事前にこの分析結果を付加しておくことができます。たとえば特定の製品などへのコメントのテキストに対してこの処理を適用することで、その製品のユーザーからの評価などを数値化することに利用することも可能ではないでしょうか。

フィードバック

Elasticsearch 8.9の時点で日本語のNLPモデルサポートはテクニカルプレビューの状態です。もしも何らかの不具合を発見した場合や、あるいはBert以外のアルゴリズムやMeCab以外のトークナイザーへの対応などの要望がある場合は、ぜひElasticにお知らせください。

Elasticにフィードバックを送るには、Github Issueが最も適切な方法です。 elastic/elasticsearchリポジトリのIssuesに、`:ml` のタグをつけて要望を上げていただければ、適切なチームが精査して対応を検討します。

ところで、この日本語対応へのサポート追加はElasticのコンサルティングアーキテクトである（つまり開発部門所属ではない）筆者が外部コントリビューターとして修正のプルリクエストをGithubに送ることで実現しました。もしこれを読んでいるあなたが開発者で、今後具体的なユースケースで機能追加要望等がある場合はぜひ同じように挑戦してみてください。

おわりに

現在Elasticでは検索機能に機械学習を用いたNLPの機能を実装することに多くのリソースを投入しており、様々な機能をElasticsearchの上で動作させることができるようになってきています。しかし多くの機能はまず英語のサポートを優先してリリースされており、その他の言語へのサポートは限定的でした。

しかしまず日本語にて英語以外の言語へのサポートを提供できることになったことは、大変嬉しく感じています。ぜひこのElasticsearchの新機能を活用して、よりユーザーにとって意味のある検索体験を実現し、素晴らしいソフトウェアを構築してください。

The release and timing of any features or functionality described in this post remain at Elastic's sole discretion. Any features or functionality not currently available may not be delivered on time or at all.

In this blog post, we may have used or referred to third party generative AI tools, which are owned and operated by their respective owners. Elastic does not have any control over the third party tools and we have no responsibility or liability for their content, operation or use, nor for any loss or damage that may arise from your use of such tools. Please exercise caution when using AI tools with personal, sensitive or confidential information. Any data you submit may be used for AI training or other purposes. There is no guarantee that information you provide will be kept secure or confidential. You should familiarize yourself with the privacy practices and terms of use of any generative AI tools prior to use.

株式

EXCELLENCE AWARDS

過去の受賞者

ElasticONツアー

イベントに協賛する

すべてのイベント