




[Blog](#) [Solutions](#) [Stack](#) [News](#) [Customers](#) [Generative AI](#) [Culture](#) 
+
[Cloud](#)

Use a Japanese language NLP model in Elasticsearch to enable semantic searches

Prerequisites

Process of executing a semantic search

Eland installation

Importing the NLP model

Implementing semantic search using vector embeddings

Text classification (sentiment analysis)

Feedback

Conclusion

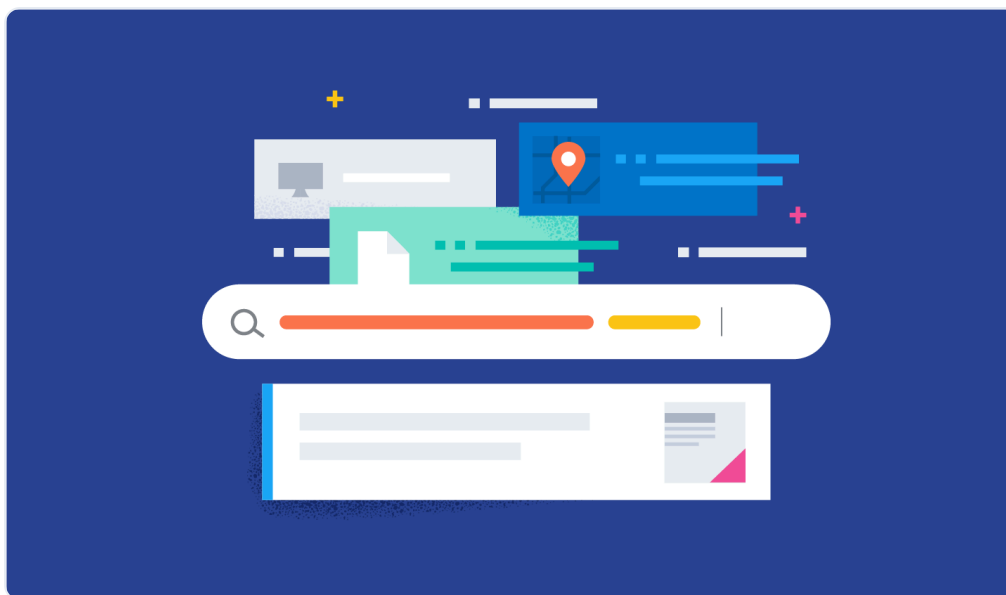
[Close](#)

Use a Japanese language NLP model in Elasticsearch to enable semantic searches

By **Dai Sugimori**

31 August 2023





Quickly finding necessary documents from among the large volume of internal documents and product information generated every day is an extremely important task in both work and daily life. However, if there is a high volume of documents to search through, it can be a time-consuming process even for computers to re-read all of the documents in real time and find the target file. That is what led to the appearance of Elasticsearch® and other search engine software. When a search engine is used, search index data is first created so that key search terms included in documents can be used to quickly find those documents.

However, even if the user has a general idea of what type of information they are searching for, they may not be able to recall a suitable keyword or they may search for another expression that has the same meaning. Elasticsearch enables synonyms and similar terms to be defined to handle such situations, but in some cases it can be difficult to simply use a correspondence table to convert a search query into a more suitable one.

To address this need, Elasticsearch 8.0 released the vector search feature, which searches by the semantic content of a phrase. Alongside that, we also have a [blog series](#) on how to use Elasticsearch to perform vector searches and other NLP tasks. However, up through the 8.8 release, it was not able to correctly analyze text in languages other than English.



With the 8.9 release, Elastic added functionality for properly analyzing Japanese in text analysis processing. This functionality enables Elasticsearch to perform semantic searches like vector search on Japanese text, as well as natural language processing tasks such as sentiment analysis in Japanese. In this article, we will provide specific step-by-step instructions on how to use these features.

Prerequisites

Before implementing semantic search, confirm the prerequisites for using this feature. In Elasticsearch clusters, individual roles are assigned [node roles](#). Meanwhile, Elasticsearch [machine learning nodes](#) are what drive the machine learning model. To use this feature, there must be a machine learning node active in the Elasticsearch cluster, so be sure to confirm that this is the case in advance. You must also have a Platinum license or higher to use machine learning nodes. However, a trial license can be used if you only want to test the features out and ensure they work. To verify operation functionality in a development environment or similar instance, activate the trial on the Kibana® screen or via the [API](#).

Process of executing a semantic search

The following steps are required to execute a semantic search in Elasticsearch.

1. (Preparation) Install Eland and related libraries on the workstation.
2. Import the machine learning model to enable natural language processing tasks.
3. Index text analysis results in the imported machine learning model.
4. Perform a kNN search using the machine learning model.

Semantic search is not the only thing enabled by natural language processing. In the second half of this blog entry, we will present examples of how to use the machine learning model, which enables text classification tasks, to perform sentiment analysis of text (positive and negative categories).



Let's proceed to the in-depth explanations of how to perform the following tasks.

Eland installation

Elasticsearch is now capable of behaving like a natural language processing platform. However, the reality is that no in-depth natural language processing is actually implemented in Elasticsearch. Any necessary natural language processing must be imported into Elasticsearch by the user as a machine learning model. This import process is performed using Eland. Since users can freely import an external model in this way, they are able to add the machine learning functionality they need whenever they need it.

[Eland](#) is a Python library provided by Elastic that enables users to link Elasticsearch data with comprehensive Python machine learning libraries such as PyTorch and scikit-learn. The `eland_import_hub_model` command line tool bundled within Eland can be used to import into Elasticsearch NLP models that have been published to Hugging Face. All command line tasks covered below in this article assume the use of a Python notebook such as Google Colaboratory. (Naturally, other types of terminals can be used, such as a Mac or Linux machine. In such cases, ignore the ! at the beginning of the commands below.)

First, install dependent libraries.

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

 [Copy](#)

Fugashi, ipadic, and unidic_lite will be required to use a Japanese model.



After these libraries have been installed, Eland can be installed as well. Eland 8.9.0 or later will be required to use a Japanese model, so be sure to note the version number.

```
!pip install eland
```

 [Copy](#)

Once installation is complete, use the command below to confirm that Eland can be used.

```
!eland_import_hub_model -h
```

 [Copy](#)

Importing the NLP model

The primary method of enabling vector search is the same as the method used in English in [this article](#). We will briefly cover the same procedure here to review.

As explained above, the appropriate machine learning model must be imported into Elasticsearch in order to enable NLP processing in Elasticsearch. It is possible to implement a machine learning model yourself using PyTorch, but this also requires sufficient expertise with machine learning and natural language processing as well as the computing power that machine learning requires. However, there is now an online repository, Hugging Face, that is heavily used by researchers and developers in the machine learning and natural language processing spaces, and many models are published to this repository. In this example, we will use a model published to Hugging Face to implement semantic search functionality.

To start, let's pick a model on Hugging Face that will embed (vectorize) Japanese sentences into numerical vectors. In this article, we will use the model linked below.

- [cl-tohoku/bert-base-japanese-v2](#)



Let's cover some points to note when selecting a Japanese model in 8.9.

First, only the [BERT](#) model algorithm is supported. Check the tags on Hugging Face and other information to confirm that the desired NLP model is a BERT trained model.

Additionally, for BERT and other NLP tasks, text that is input is then "pre-tokenized," which means the text is divided into units at the word level. In this case, a Japanese language morphological analysis engine is used to pre-tokenize our Japanese text. Elasticsearch 8.9 supports morphological analysis using MeCab. On Hugging Face's models page, open the "Files and versions" tab and view the contents of the tokenizer_config.json file. Confirm that the value for word_tokenizer_type is mecab.

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

 [Copy](#)

Unfortunately, if the model you want to use has a value other than mecab for word_tokenizer_type, that model is not currently supported by Elasticsearch. We welcome feedback regarding any specific word tokenizer types (word_tokenizer_type) for which support is required.

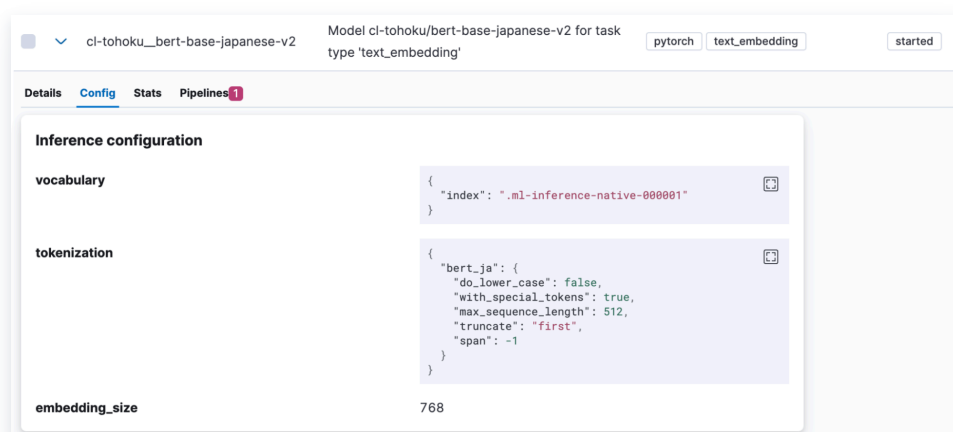
Once you have decided on a model to import, the steps required to import are the same as for an English model. First, use eland_import_hub_model to import the model into Elasticsearch. Refer to [this page](#) for how to use eland_import_hub_model.



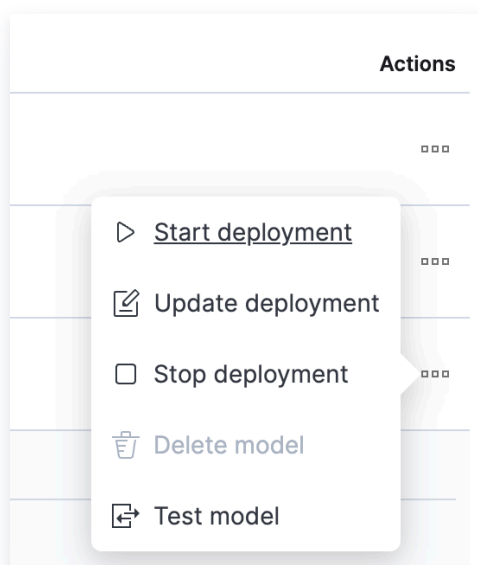
```
!eland_import_hub_model \  
--url "https://your.elasticserach" \  
--es-api-key "your_api_key" \  
--hub-model-id cl-tohoku/bert-base-japanese-v2 \  
--task-type text_embedding \  
--start
```

[Copy](#)

Once the model has been imported successfully, it will be displayed under Machine Learning > Model Management > Trained Models in Kibana. Open the model "Config" tab to see that "bert_ja" is used for tokenization and the model is set up correctly to handle Japanese.



Once the model upload is complete, let's test it out. Click a button in the Actions column to open the menu.



Select "Test model", then enter any Japanese phrase under "Input text" and click the "Test" button.

You will then be able to see that the model vectorized the Japanese text you input into a numerical string, as shown here. It seems to be working correctly.

Implementing semantic search using vector embeddings

Now that the model has been uploaded, we can implement semantic search (vector search) functionality in Elasticsearch.

First, in order to perform a vector search, it is necessary to index the vector values in which the original Japanese text was embedded. To do this, we will create a pipeline that includes an [inference processor](#) to vectorize the Japanese text before it is input into the index using the model uploaded earlier.

```
PUT _ingest/pipeline/japanese-text-embeddings
{
  "description": "Text embedding pipeline",
  "processors": [
    {
```




```
"inference": {
  "model_id": "cl-tohoku__bert-base-japanese",
  "target_field": "text_embedding",
  "field_map": {
    "title": "text_field"
  }
}
```

[Copy](#)

When the inference processor is used, the model specified in `model_id` is applied to the text saved in the target field (in this case, `title`) and the output thereof is stored in `target_field`. Additionally, each model expects a different field (in this case, `text_field`) for the input value for the process. For that reason, `field_map` is used to specify the correspondence between the actual input field for the target of the process and the field name expected by the ML model.

Once the pipeline is set up, it can be used to create indices. Since fields will be needed in those indices to store vectors, we will define the appropriate mapping. In the example below, the `text_embedding.predicted_value` field is set up to hold 768-dimensional [dense_vector](#) data. Note that the number of dimensions varies by model. Check the model's page on Hugging Face (`hidden_size` value in the model's `config.json`) or elsewhere and set an appropriate number.

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





If there is already an index that includes the Japanese text data targeted for searching, then the reindex API can be used. In this example, the original text data is in the `japanese-text` index. A document containing the vectorization of that text is registered to the `japanese-text-embeddings` index.

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



Alternatively, a document can be registered directly for testing purposes by specifying a pipeline created as shown below and storing it in the index.

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



Once the vectorized document has been registered, it is finally possible to execute a search. A [kNN](#) (k-nearest neighbor) search is one available method that makes use of vectors. We will now execute a kNN vector search using the `query_vector_builder` option in the standard `_search` API. When `query_vector_builder` is used, the model specified in `model_id` can be used to convert the text in



model_text into a query containing a vector in which that text is embedded.

```
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-japane
        "model_text": "日本語でElasticsearchを検索
      }
    }
  }
}
```

[Copy](#)

When this search query is executed, the following type of response is received.

```
"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
      "text_embedding": {
        "predicted_value": [
          -0.13586345314979553,
          -0.6291824579238892,
          0.32779985666275024,
          0.36690405011177063,
```

[Copy](#)

Search successful! The search also contains fields in which the Japanese has been embedded. In most actual use cases, it is not necessary for this text to be included in the response. In such cases, use the [_source parameter](#) or some other method to exclude that information from the response (or take other such action).

For fine-tuning search rankings, the [Reciprocal rank fusion \(RRF\)](#) feature, which adeptly blends the results of vector searches and standard keyword searches, has been released. Be sure to take a look at this as well.

This concludes our discussion of how to enable semantic search using vector searching. Although setting this functionality up may require more work than a standard search and you may encounter some unique machine learning vocabulary, you will be able to execute searches in almost the same way as normal once the setup phase is complete, so please give it a try.

Text classification (sentiment analysis)

Since we have seen that we can use vector searching using kNN in Japanese, let's look at using other NLP tasks in the same way.

Text classification is a task in which text input is placed into categories of some kind. For this example, we will use a sentiment analysis model found on Hugging Face that judges whether Japanese text input is accompanied by positive sentiment or by negative sentiment ([koheiduck/bert-japanese-finetuned-sentiment](#)). Looking at its tokenizer_config.json, it can be seen that this model also uses mecab as its word_tokenizer_type, so it can be used with Elasticsearch's bert_ja.

As we did before, use Eland to import the model into Elasticsearch.

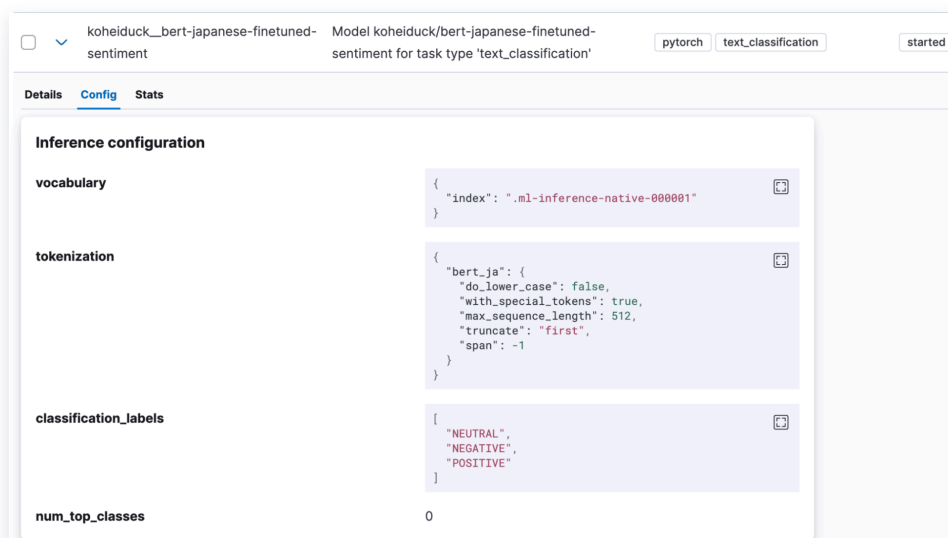
```
!eland_import_hub_model \  
--url "https://your.elasticserach" \  
--es-api-key "your_api_key" \  
--hub-model-id koheiduck/bert-japanese-finetuned
```



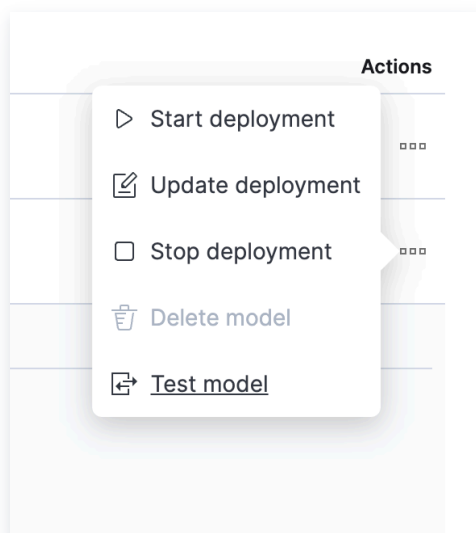
```
--task-type text_classification \  
--start
```

[Copy](#)

Once the model has been imported successfully, it will be displayed under Machine Learning > Model Management > Trained Models in Kibana.



In this case as well, click "Test model" in the Actions menu.



As before, this will display a dialog box for testing. Enter the text to be classified here, and the text will be classified into POSITIVE, NEUTRAL, or NEGATIVE. As a test, let's enter: "I'm happy that I'm now able to execute NLP tasks with Elasticsearch." As shown below, this generated a result of 99.2% positive.



×

Test trained model

koheiduck__bert-japanese-finetuned-sentiment

Test using text

Test using existing index

Text classification

Test how well the model classifies your input text.

Input text

ElasticsearchでNLPのタスクが実行できるようになって嬉しい。

Test

Output

Raw output

ElasticsearchでNLPのタスクが実行できるようになって嬉しい。

POSITIVE

0.992

NEUTRAL

0.00668

NEGATIVE

0.00115

Below is the same process executed via API.

```
POST _ml/trained_models/koheiduck__bert-japanese
{
  "docs": [{"text_field": "ElasticsearchでNLPのタ
  "inference_config": {
    "text_classification": {
      "num_top_classes": 3
    }
  }
}
```

 [Copy](#)

The response is as follows:

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



```
"class_probability": 0.992165109012463
"class_score": 0.9921651090124636
},
{
  "class_name": "NEUTRAL",
```

[Copy](#)

Since this process can also be executed with the inference processor, it is possible to attach these analysis results before the Japanese text is indexed. For example, applying this process to the text of comments for specific products could help convert user ratings for those products into numerical values.

Feedback

As of Elasticsearch 8.9, Japanese language NLP model support is still in the technical preview stage. Please contact Elastic® if you find any bugs or need support for non-BERT algorithms or non-MeCab tokenizers, etc.

GitHub Issues is the best way to send feedback to Elastic. [Under "Issues" in the elastic/elasticsearch repository](#), add the `:ml` tag and raise your request. The appropriate team will investigate.

As a consulting architect for Elastic (and therefore not on the development team), I was able to achieve the addition of support for the Japanese language by sending a [pull request](#) for a modification via GitHub as an outside contributor. If you are a developer and have a feature that you would like to request be added with a specific use case, please give it a try just as I did.

Conclusion

At present, Elastic is investing a lot of resources in implementing NLP functionality using machine learning into its search features, and there are increasingly more such functions that can be executed in Elasticsearch. However, most features are released first with English support and limited support for other languages.

However, we are very happy that it was decided to offer support for Japanese. We hope that these new Elasticsearch features help to



make your searches more meaningful.

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.

Elastic, Elasticsearch, ESRE, Elasticsearch Relevance Engine and associated marks are trademarks, logos or registered trademarks of Elasticsearch N.V. in the United States and other countries. All other company and product names are trademarks, logos or registered trademarks of their respective owners.

SHARE



Sign up for Elastic Cloud free trial

Spin up a fully loaded deployment on the cloud provider you choose. As the company behind **Elasticsearch**, we bring our features and support to your Elastic clusters in the cloud.



Start free trial

FOLLOW US



ABOUT US

About Elastic
Leadership
DE&I
Blog
Newsroom

PARTNERS

Find a partner
Partner login
Request access
Become a partner

INVESTOR RELATIONS

Investor resources
Governance
Financials
Stock

TRUST & SECURITY

Trust center
EthicsPoint portal
ECCN report
Ethics email

EXCELLENCE AWARDS

Previous winners
ElasticON Tour
Become a sponsor
All events

JOIN US

Careers
Career portal
How we hire

[Trademarks](#) [Terms of Use](#) [Privacy](#) [Sitemap](#)

© 2024. Elasticsearch B.V. All Rights Reserved

Elastic, Elasticsearch and other related marks are trademarks, logos or registered trademarks of Elasticsearch B.V. in the United States and other countries.
Apache, Apache Lucene, Apache Hadoop, Hadoop, HDFS and the yellow elephant logo are trademarks of the [Apache Software Foundation](#) in the United States and/or other



countries. All other brand names, product names, or trademarks belong to their respective owners.

