Online Clustering & Semantic Enrichment of Textual Data with Apache Beam

By Konstantin Buschmeier
Machine Learning Engineer @ ML6
Machine Learning services company.

We help our clients build machine learning applications using technologies such as Apache Beam.
Motivation

Semantic Enrichment
Add semantic information to text documents.

Online Clustering
Arrange documents into not yet defined groups as they come in.
Semantic Enrichment

- Count word occurrences
- Add geo location
- Categorise: Add predefined labels
- Sentiment Analysis
- Filter profanity
- Extract keywords
- Named-Entity Recognition/Linking
- Summarize
- Word/sentence/document embeddings
- OCR correction
- Translation
- Coreference Resolution
Semantic Enrichment

- Count word occurrences
- Add geo location
- Categorise: Add predefined labels
- Sentiment Analysis
- Filter profanity
- Extract keywords
- Named-Entity Recognition/Linking
- Summarize
- Word/sentence/document embeddings
- OCR correction
- Translation
- Coreference Resolution
Semantic Enrichment

- Count word occurrences
- Add geo location
- Categorise: Add predefined labels
- Sentiment Analysis
- Filter profanity
- Extract keywords
- Named-Entity Recognition/Linking
- Summarize
- Word/sentence/document embeddings
- OCR correction
- Translation
- Coreference Resolution
Semantic Enrichment

- Count word occurrences
- Add geo location
- Categorise: Add predefined labels
- Sentiment Analysis
- Filter profanity
- Extract keywords
- Named-Entity Recognition/Linking
- Summarize
- Word/sentence/document embeddings
- OCR correction
- Translation
- Coreference Resolution
Semantic Enrichment

- Count word occurrences
- Add geo location
- Categorise: Add predefined labels
- Sentiment Analysis
- Filter profanity
- Extract keywords
- Named-Entity Recognition/Linking
- Summarize
- Word/sentence/document embeddings
- OCR correction
- Translation
- Coreference Resolution

```python
class TextEmbedding(beam.DoFn):
    """Get the text embedding using the Universal Sentence Encoder."""
    model_url = 'https://example.com/models/universal-sentence-encoder/4'
    headers = {'Content-Type': 'application/json'}

    def embed(self, text):
        payload = {'text': text}
        response = requests.post(model_url, json=payload, headers=headers)
        return response['embedding']

    def process(self, element, *args, **kwargs):
        text = element.get('text', '')
        if text:
            yield {
                **element,
                'text_embedding': self.embed(text)
            }
```
Semantic Enrichment

- Count word occurrences
- Add geo location
- Categorise: Add predefined labels
- Sentiment Analysis
- Filter profanity
- Extract keywords
- Named-Entity Recognition/Linking
- Summarize
- Word/sentence/document embeddings
- OCR correction
- Translation
- Coreference Resolution

```python
class TextEmbedding(beam.DoFn):
    """Get the text embedding using the Universal Sentence Encoder."""

    model_url = "https://example.com/models/universal-sentence-encoder/4"
    headers = {"Content-Type": "application/json"}

    def setup(self):
        self.session = requests.Session()

    def embed(self, texts):
        payload = {'text': text}
        response = self.session.post(model_url, json=payload, headers=headers)
        return response['embedding']

    def process(self, element, *args, **kwargs):
        text = element.get('text', '')

        if text:
            yield {
                **deepcopy(element),
                'text_embedding': self.embed(text)
            }
```
Semantic Enrichment

- Count word occurrences
- Add geo location
- Categorise: Add predefined labels
- Sentiment Analysis
- Filter profanity
- Extract keywords
- Named-Entity Recognition/Linking
- Summarize
- Word/sentence/document embeddings
- OCR correction
- Translation
- Coreference Resolution

```python
from apache_beam.ml.inference.base import RunInference
from apache_beam.ml.inference.base import PytorchModelHandlerKeyedTensor

model_config=AutoConfig.from_pretrained('sentence-transformers/paraphrase-MiniLM-L6-v2', return_dict=True)
model_handler = PytorchNoBatchModelHandler(
    state_dict_path='sentence_transformer.pth', model_class=BertModel,
    model_params = {'config': model_config})

with pipeline as p:
    text = (p | 'Create Examples' >> beam.Create([example_docs]))

    text_and_tokenized_text_tuple = (text
                                      | 'Tokenize Sentence' >> beam.Map(tokenize_with_sentence_transformer))

    embedding = (text_and_tokenized_text_tuple
                 | 'Run Inference' >> RunInference(KeyedModelHandler(model_handler))
                 | 'Postprocess' >> beam.ParDo(SentenceBertPostProcessor()))
```
Semantic Enrichment

→ Beam provides great tools:

- Batch/Streaming
- Filtering
- Grouping
- Windowing
- ...
Examples
Online Clustering

Arrange documents into not yet defined groups as they come in.
Online Clustering
Online Clustering
Online Clustering
Online Clustering

Agglomerative Clustering

Distance Matrix

Cluster
Online Clustering

Agglomerative Clustering

5 clusters
4 clusters
3 clusters
2 clusters
1 cluster
Online Clustering

Clustering is usually a **batch operation**.
Online Clustering

Clustering is usually a **batch operation**.

What do we need?
Online Clustering

Clustering is usually a **batch operation**.

What do we need?

A **clustering** algorithm that works **iteratively**.
Online Clustering

Clustering is usually a **batch operation**.

What do we need?

A **clustering** algorithm that works **iteratively**.

A mechanism to access the **previous state**.
BIRCH
BIRCH
BIRCH
BIRCH

Add documents iteratively.

Build a tree structure that contains summaries of subclusters that are sufficient for cluster decisions.

_Tight, local subclusters_ are summarised.

Very fast, input data only needs to be read once, O(n).

Resulting summaries can be used as input to other clustering algorithms.
Online Clustering

What do we need?

A *clustering* algorithm that works *iteratively*.

A mechanism to access the *previous state*. 
Stateful Processing
Stateful Processing

def run():
    """Main function that defines the pipeline and runs it."""
    pipeline = beam.Pipeline()

    # Input text documents
    docs = (pipeline
        | "Load documents" >> beam.Create(example_docs)
    )

    # Enrichment
    enriched_docs = (docs
        | "Count words" >> beam.ParDo(CountWords())
    )

    # Difference to running average
    differences = (enriched_docs
        | "The state is partitioned by key: Use a single key" >> beam.Map(lambda e: (1, e))
        | "Difference to Running Average" >> beam.ParDo(StatefulAverageDifference())
    )

    # Print
    printout = (differences
        | "Print" >> beam.Map(print)
    )

    pipeline.run().wait_until_finish()
class StatefulOnlineClustering(beam.DoFn):
    """Group documents using online clustering of embeddings."""

    DOCS_SPEC = ReadModifyWriteStateSpec('documents', PickleCoder())
    EMBEDDINGS_SPEC = ReadModifyWriteStateSpec('embeddings', PickleCoder())
    BIRCH_MODEL_SPEC = ReadModifyWriteStateSpec('clustering_model', PickleCoder())
    LABEL_MAP_SPEC = ReadModifyWriteStateSpec('label_map', PickleCoder())
    PREVIOUS_ASSIGNMENT_SPEC = ReadModifyWriteStateSpec('previous_cluster_assignment', PickleCoder())
    UPDATE_COUNTER_SPEC = ReadModifyWriteStateSpec('update_counter', PickleCoder())

    def process(self,
                 model_state=beam.DoFn.StateParam(BIRCH_MODEL_SPEC),
                 label_map_state=beam.DoFn.StateParam(LABEL_MAP_SPEC),
                 collected_docs_state=beam.DoFn.StateParam(DOCS_SPEC),
                 collected_embeddings_state=beam.DoFn.StateParam(EMBEDDINGS_SPEC),
                 previous_assignments_state=beam.DoFn.StateParam(PREVIOUS_ASSIGNMENT_SPEC),
                 update_counter_state=beam.DoFn.StateParam(UPDATE_COUNTER_SPEC),
                 *args,
                 **kwargs,
                 *):
        # 1. Initialise or load states
        clustering = model_state.read() or Birch(n_clusters=None, threshold=0.7)
        label_map = label_map_state.read() or dict()
        collected_documents = collected_docs_state.read() or dict()
        collected_embeddings = collected_embeddings_state.read() or dict()
        previous_assignments = previous_assignments_state.read() or dict()
        update_counter = update_counter_state.read() or Counter()

        # 2. Collect documents
        collected_documents = dict(np.array(list(collected_embeddings.values())
                                             if collected_embeddings
                                             else []))

        # 3. Cluster documents
        ed_text_embedding = ed_text_embedding
        embedding_vector = embedding_vector
        t_2d = t_2d(embedding_vector)
        new_assignments = clustering.predict(collected_documents)

        # 4. Update labels and cluster model
        new_labels = update_labels(label_map, new_assignments, update_counter)

        # 5. Write labels and embeddings
        label_map.write(new_labels)
        collected_embeddings.write(collected_documents)

        # 6. Yield cluster results
        yield {
            'cluster_id': cluster_id,
            'updates': update_counter[cluster_id],
            'documents': items,
        }
Online Clustering

BIRCH:
A clustering algorithm that works iteratively.

Stateful processing:
A mechanism to access the previous state.
Example

Semantic Enrichment & Online Clustering of textual data using Apache Beam
July could be the first month with no measurable rain in Austin since 2015.

I like turtles.

Star Trek is an awesome series.

Dry conditions, low humidity, and breezy winds will allow any fires to spread rapidly.

Star Wars is my favourite movie!

I reject the later edits. Clearly, Han Solo shot first!
New cluster: 10940495-f79e-436a
Documents:
  Movies (Star Wars 1)
New cluster: 10940495-f79e-436a
Documents:
  Movies (Star Wars 1)

Update: 10940495-f79e-436a
Documents:
  Movies (Star Wars 1)
  Movies (Star Wars 2)
New cluster: 10940495-f79e-436a
Documents:
  Movies (Star Wars 1)

Update: 10940495-f79e-436a
Documents:
  Movies (Star Wars 1)
  Movies (Star Wars 2)

New cluster: 072a9e0d-4763-4afa
Documents:
  Turtles
New cluster: 10940495-f79e-436a
Documents:
  Movies (Star Wars 1)

Update: 10940495-f79e-436a
Documents:
  Movies (Star Wars 1)
  Movies (Star Wars 2)

New cluster: 072a9e0d-4763-4afa
Documents:
  Turtles

New cluster: fbdae2ec-9cc6-48b3
Documents:
  Weather 1
New cluster: 10940495-f79e-436a
Documents:
Movies (Star Wars 1)

Update: 10940495-f79e-436a
Documents:
Movies (Star Wars 1)
Movies (Star Wars 2)

New cluster: 072a9e0d-4763-4afa
Documents:
Turtles

New cluster: fbdaee2ec-9cc6-48b3
Documents:
Weather 1

Update: 10940495-f79e-436a
Documents:
Movies (Star Wars 1)
Movies (Star Wars 2)
Movies (Star Trek)
New cluster: 10940495-f79e-436a
Documents:
  Movies (Star Wars 1)

Update: 10940495-f79e-436a
Documents:
  Movies (Star Wars 1)
  Movies (Star Wars 2)

New cluster: 072a9e0d-4763-4afa
Documents:
  Turtles

New cluster: fbdae2ec-9cc6-48b3
Documents:
  Weather 1

Update: 10940495-f79e-436a
Documents:
  Movies (Star Wars 1)
  Movies (Star Wars 2)
  Movies (Star Trek)

Update: fbdae2ec-9cc6-48b3
Documents:
  Weather 1
  Weather 2
Summary

Semantic enrichment adds information from the content to the documents. This often involves machine learning which are expensive operations.

Online clustering allows the grouping of text documents into groups that are unknown up-front in real-time. Stateful processing enables iterative cluster model building.

BIRCH is an iterative clustering algorithm that can handle very large amounts of data.
Summary

Semantic enrichment adds information from the content to the documents. This often involves machine learning which are expensive operations.

Online clustering allows the grouping of text documents into groups that are unknown up-front in real-time. Stateful processing enables iterative cluster model building.

BIRCH is an iterative clustering algorithm that can handle very large amounts of data.

Real-time Productionizing: Streaming pipeline

Enrichment

Serve ML models using microservices or RunInference. Initialise connection in the setup of the DoFn & use time-batched requests.

Clustering: Tidy up the state once in a while by pruning outdated elements.
Q & A

@ml6team
linkedin.com/company/ml6team

ML6 is hiring
https://www.ml6.eu/join-us
Further Reading

- Stateful Processing with Apache Beam
- Timely (and Stateful) Processing with Apache Beam
- BIRCH: An Efficient Data Clustering Method for Very Large Databases (paper)
- RunInference examples, RunInference (Beam Summit 2022) by Andy Ye