Online Clustering & Semantic Enrichment of Textual Data with Apache Beam

By Konstantin Buschmeier
Machine Learning Engineer @ ML6

BEAM SUMMIT Austin, 2022

ML6

Machine Learning services company.

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

BEAM SUMMIT

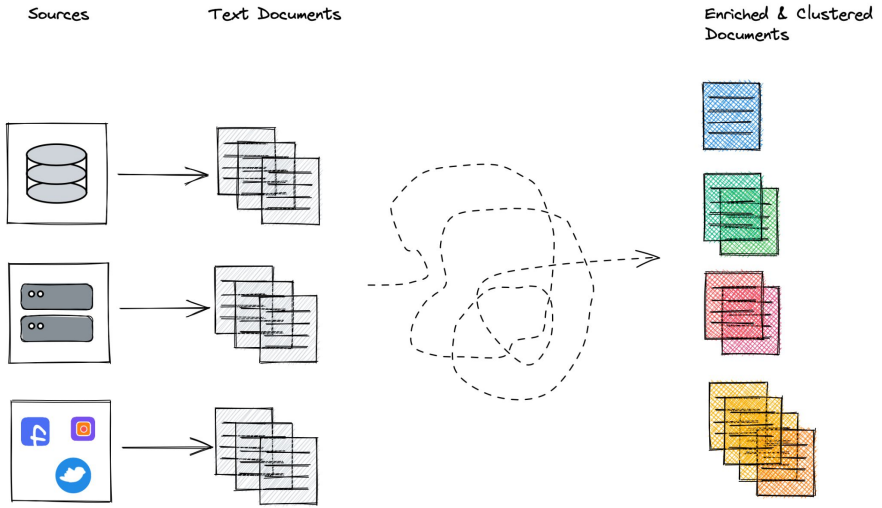
Motivation

Semantic Enrichment

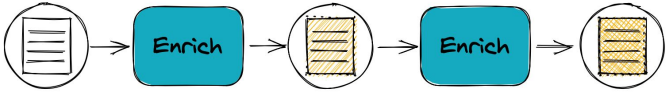
Add semantic information to text documents.

Online Clustering

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

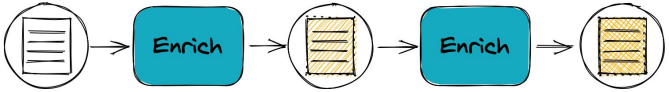


Semantic Enrichment



- 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

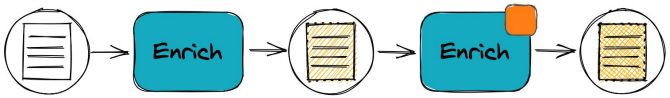
Semantic Enrichment



- 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

```
class CountWords(beam.DoFn):
    def process(self, element, *args, **kwargs):
        text = element.get('text', "")
        words = text.split(' ')
        yield {
            **element,
            'word_count': len(words)
        }
```

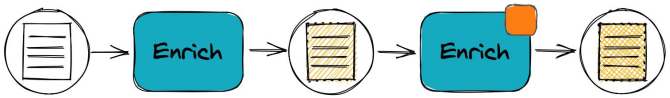
Semantic Enrichment



- 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

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

    def embed(self, texts):
        module_url = "https://tfhub.dev/google/universal-sentence-encoder/4"
        model = hub.load(module_url)

        if isinstance(texts, list):
            return np.array(model(texts))
        else:
            return np.array(model([texts]))

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

        if text:
            yield {
                **copy.deepcopy(element),
                'text_embedding': np.squeeze(self.embed(text))
            }
```

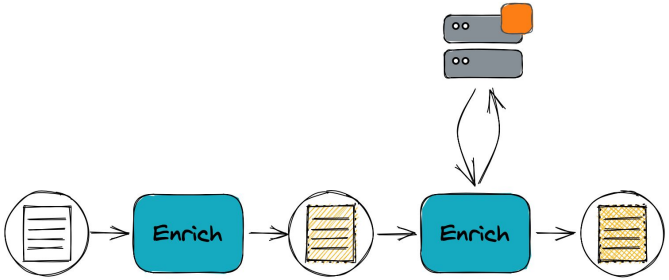
Semantic Enrichment



- 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

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

    module_url = "https://tfhub.dev/google/universal-sentence-encoder/4"

    def setup(self):
        self.model = hub.load(self.module_url)

    def embed(self, texts):
        if isinstance(texts, list):
            return np.array(self.model(texts))
        else:
            return np.array(self.model([texts]))

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

        if text:
            yield {
                **copy.deepcopy(element),
                'text_embedding': np.squeeze(self.embed(text))
            }
```

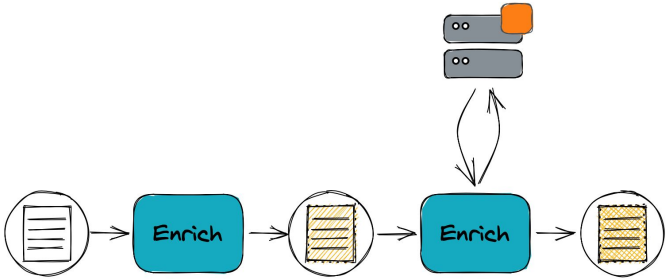
Semantic Enrichment



- 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

```
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, texts):
        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)
            }
```

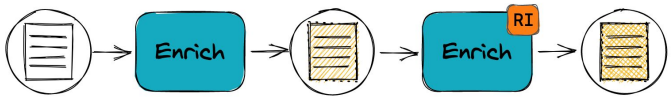
Semantic Enrichment



- 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

```
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 {
                **copy.deepcopy(element),
                'text_embedding': self.embed(text)
            }
```

Semantic Enrichment



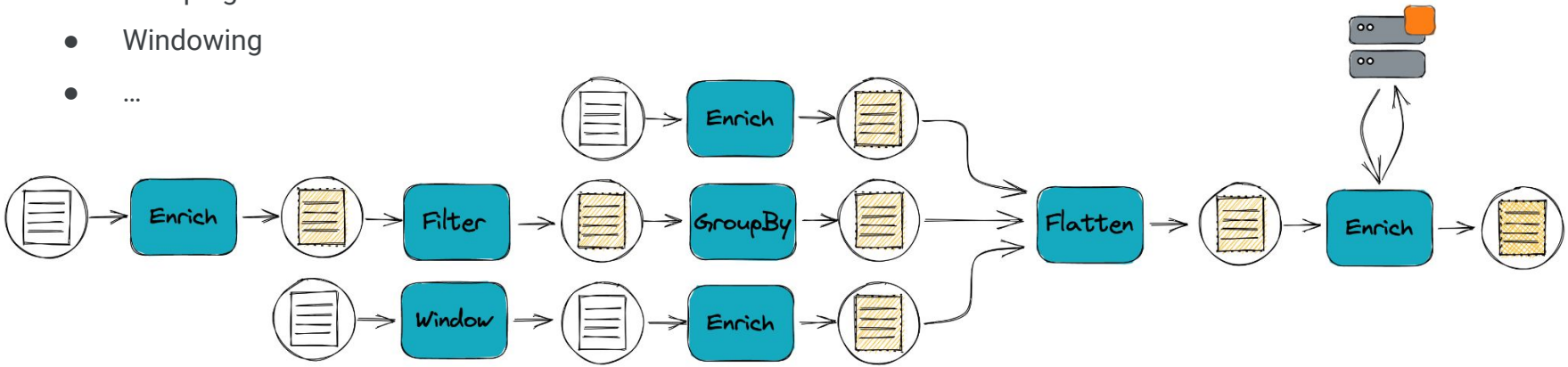
- Count word occurrences
- Add geo location
- Categorise: Add predefined labels
- Sentiment Analysis
- Filter profanity
- Extract keywords
- Named-Entity Recognition/Linking
- Summarize
- Word/sentence/document embeddings
- OCR correction
- Translation
- Coreference Resolution

```
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())
    )
```

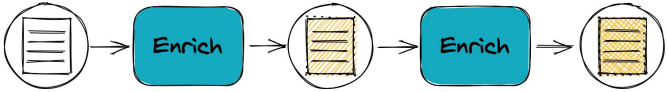
Semantic Enrichment

→ Beam provides great tools:

- Batch/Streaming
- Filtering
- Grouping
- Windowing
- …



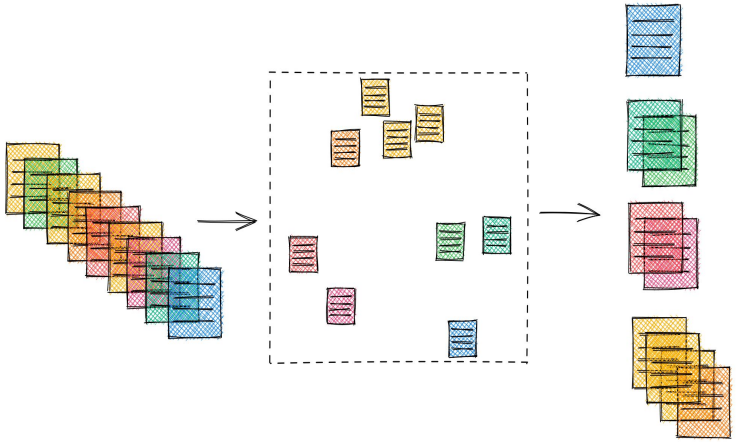
Examples



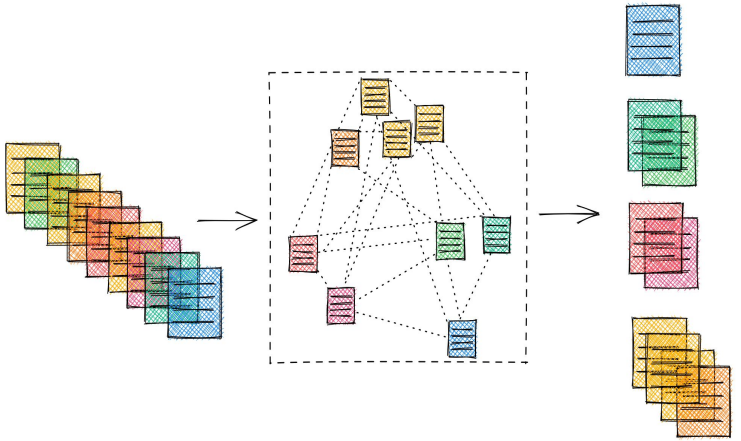
Online Clustering

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

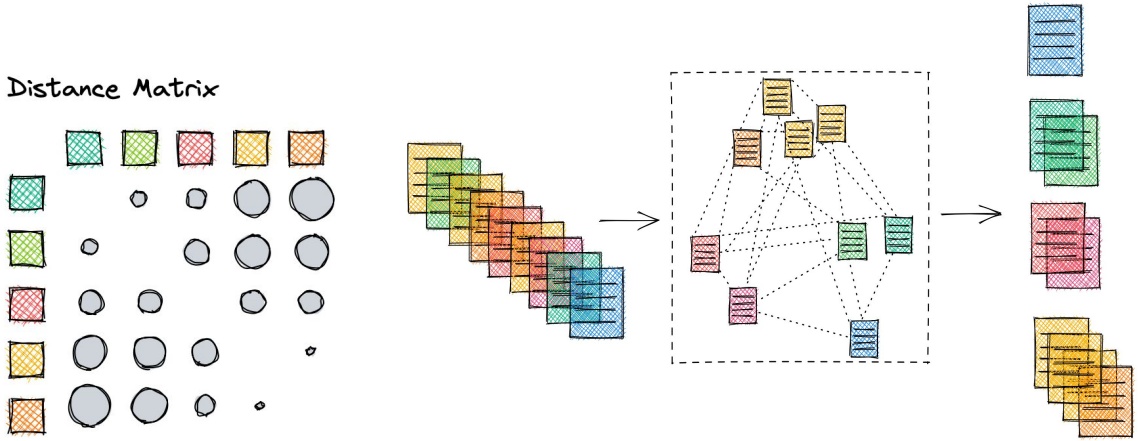
Online Clustering



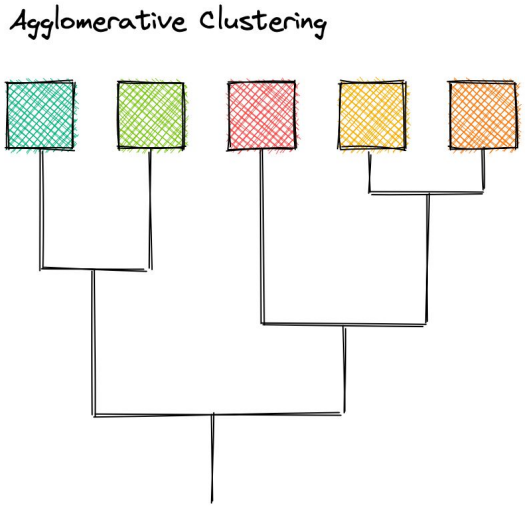
Online Clustering

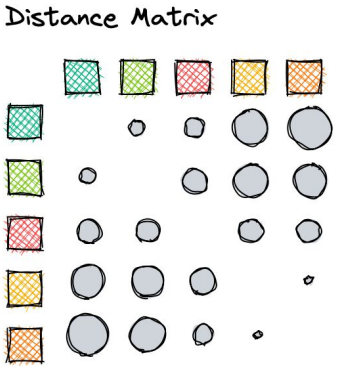


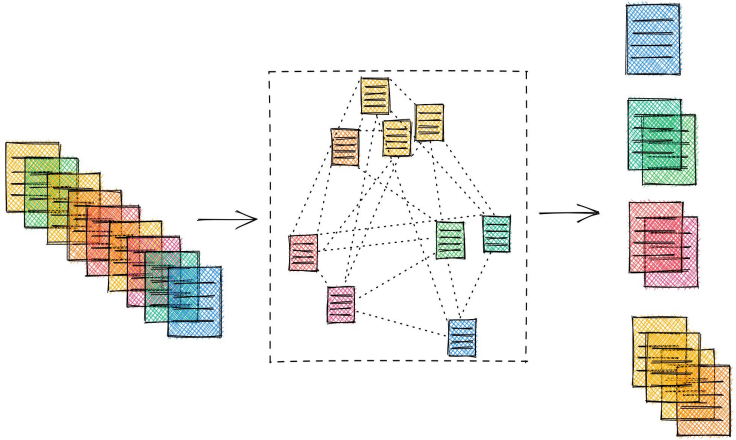
Online Clustering



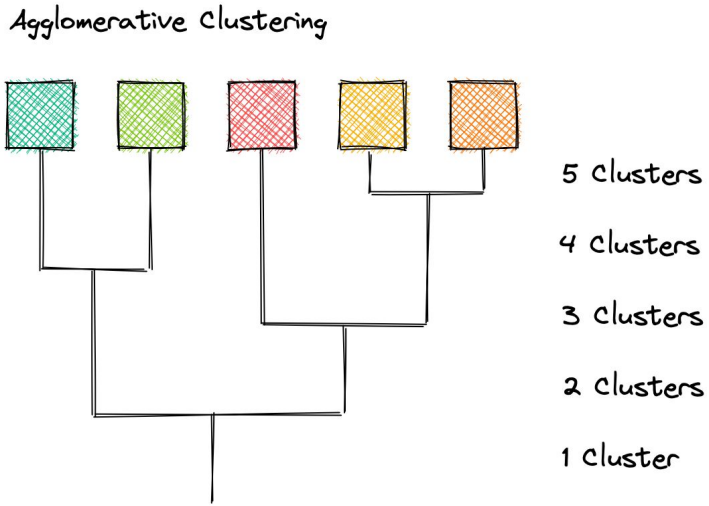
Online Clustering

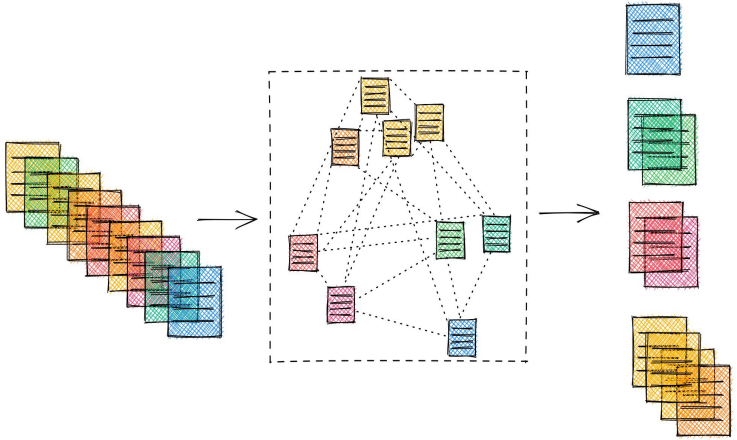






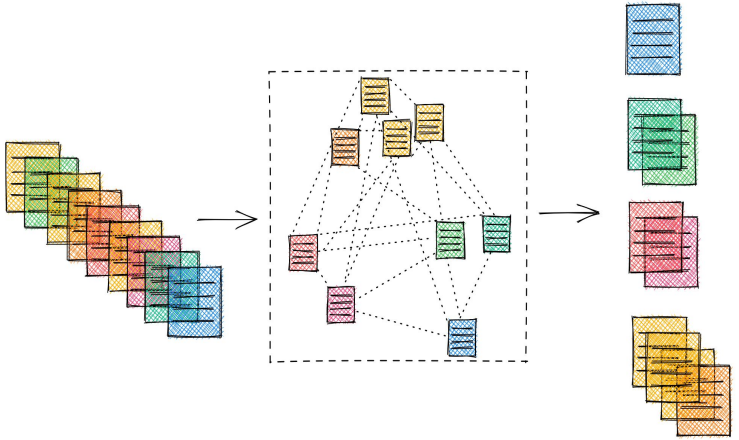
Online Clustering





Online Clustering

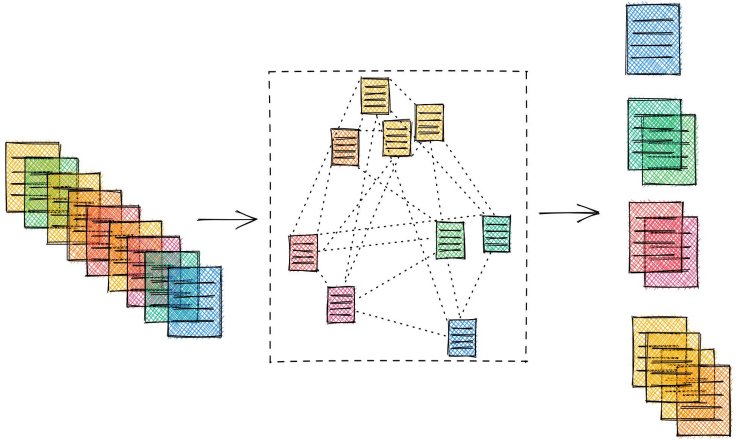
Clustering is usually a **batch operation**.



Online Clustering

Clustering is usually a **batch operation**.

What do we need?

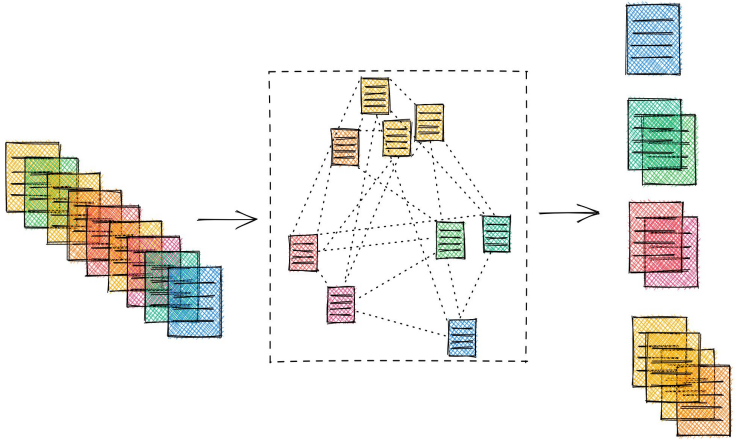


Online Clustering

Clustering is usually a **batch operation**.

What do we need?

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



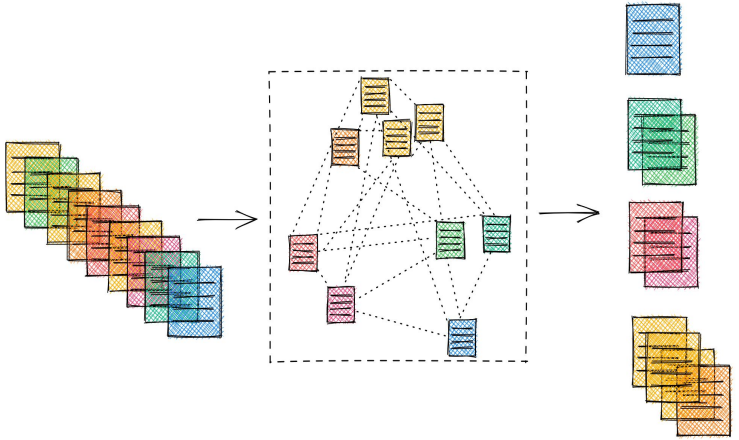
Online Clustering

Clustering is usually a **batch operation**.

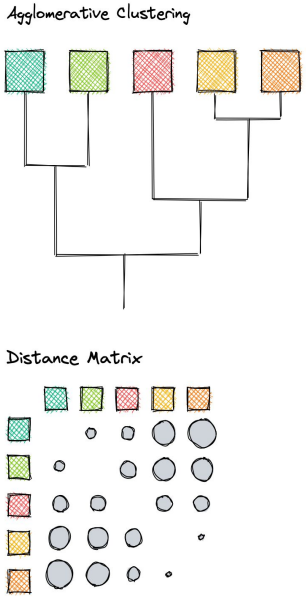
What do we need?

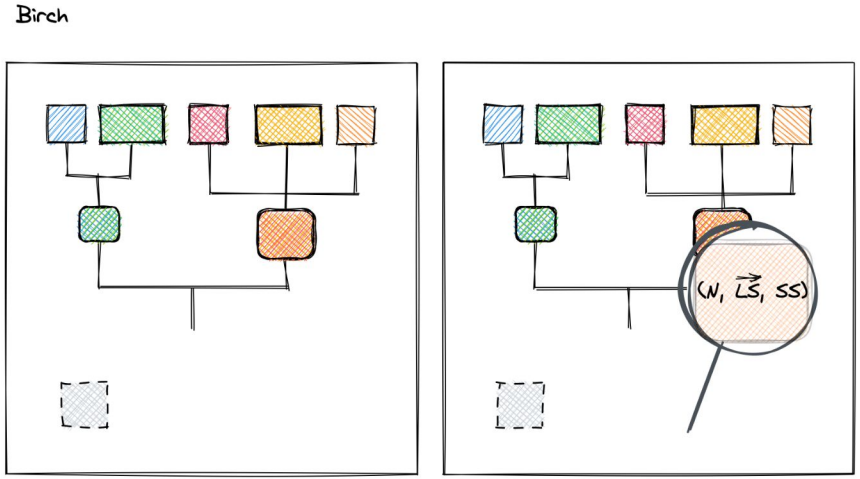
A **clustering** algorithm that works **iteratively**.

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



BIRCH





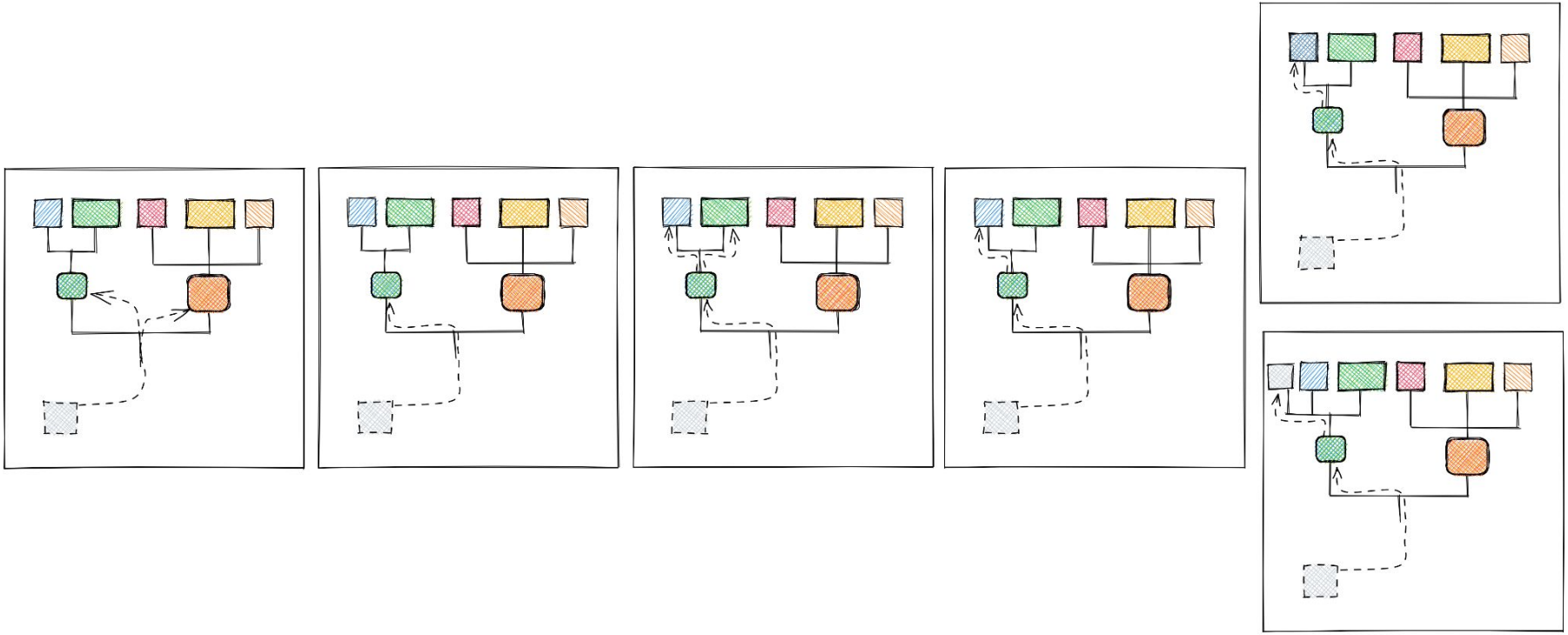
Centroid
Radius
Diameter

D0, D1, D2, D3, D4

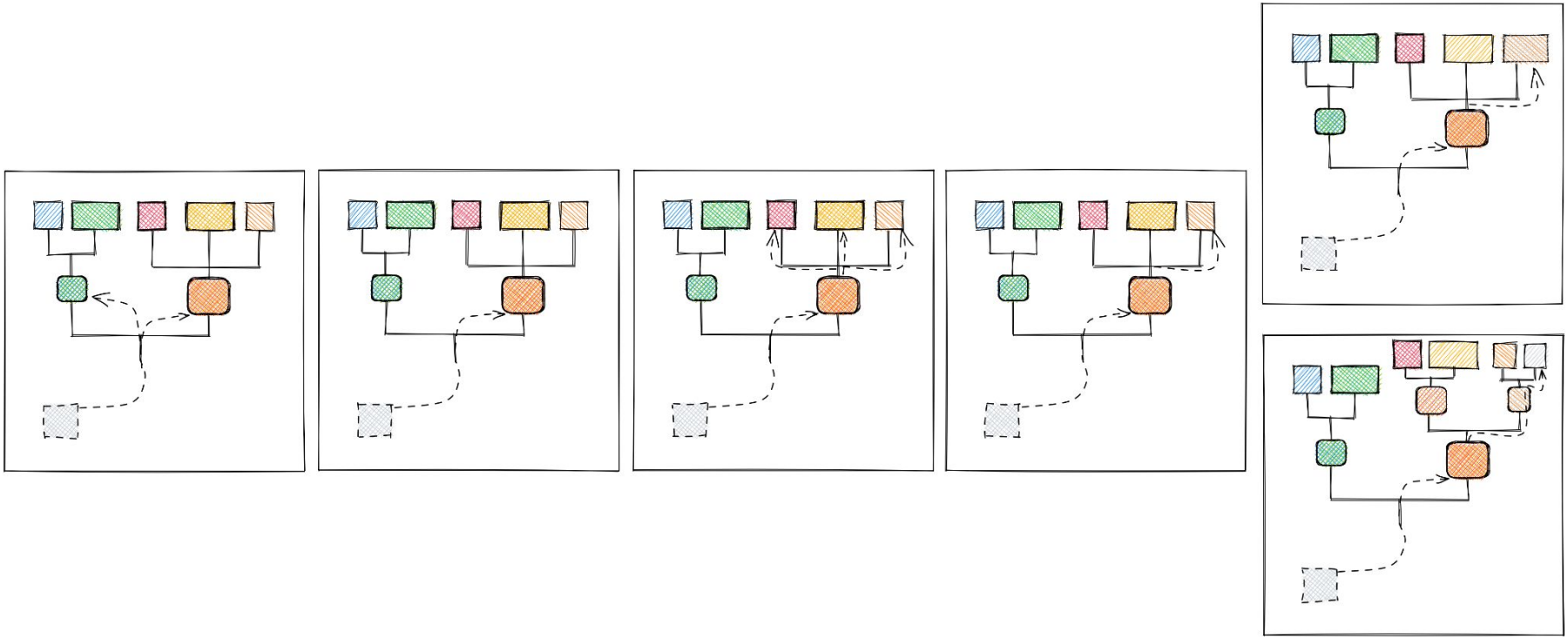
Additive Theorem:

N1+N2, $\vec{LS}$1+$\vec{LS}$2, SS+SS

BIRCH



BIRCH



BIRCH

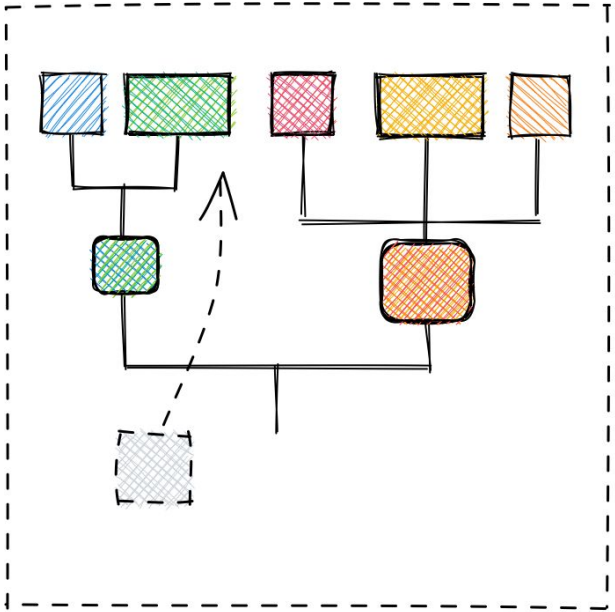
Add documents **iteratively**.

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.

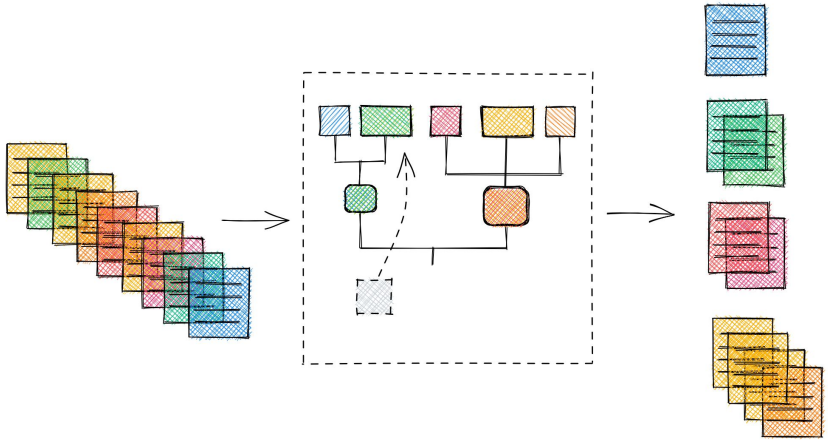


Online Clustering

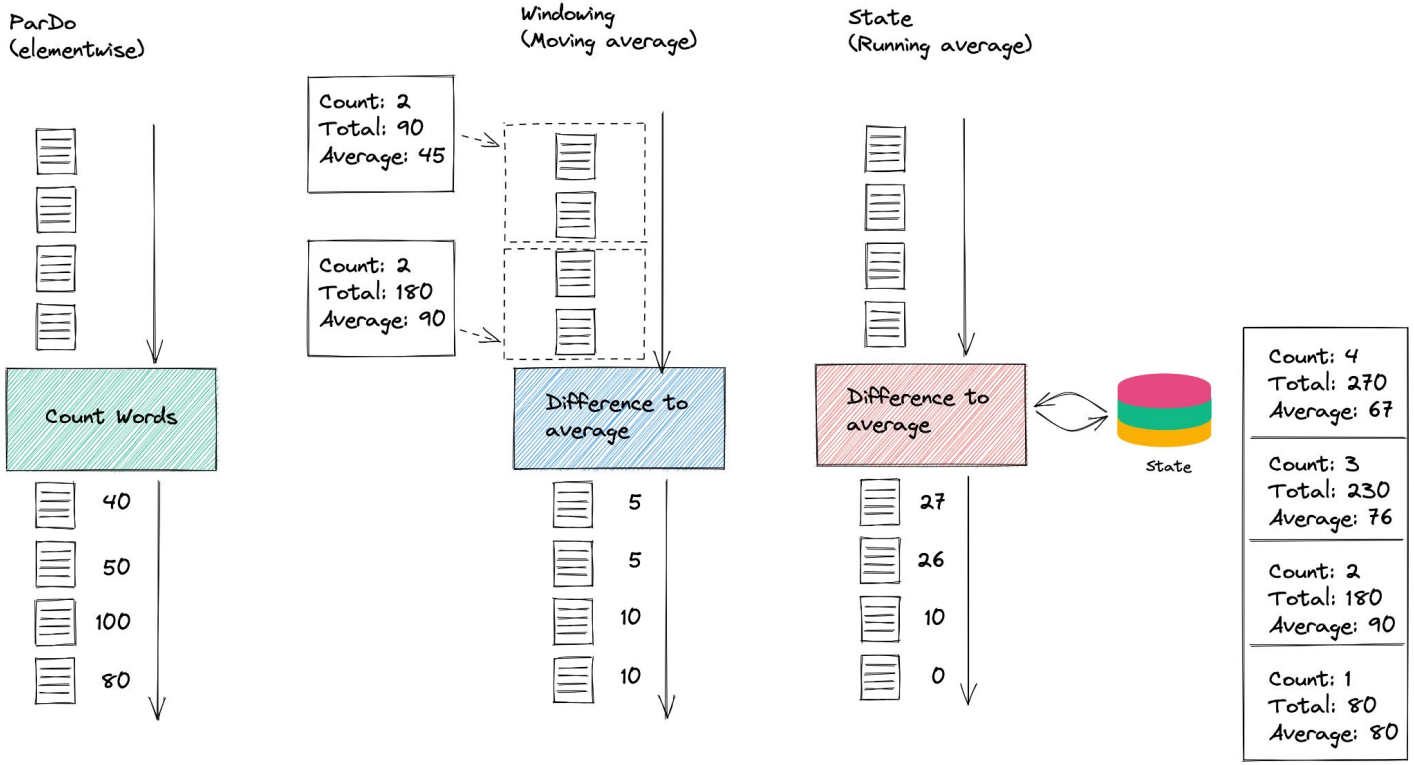
What do we need?

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

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



Stateful Processing



Stateful Processing

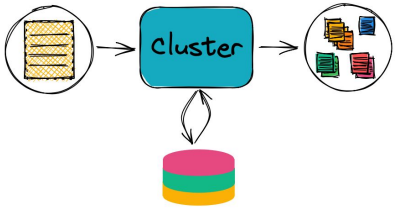
```python
class CountWords(beam.DoFn):
    def process(self, element, *args, **kwargs):
        text = element.get('text', "")
        words = text.split(' ')
        yield {
            **copy.deepcopy(element),
            'word_count': len(words)
        }
```

```python
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
        | "Make key-value pair" >> beam.Map(lambda e: (1, e))
        | "Difference to Running Average" >> beam.ParDo(StatefulAverageDifference())
    )

    # Print
    _ = (
        differences
        | "Print" >> beam.Map(pprint)
    )

    pipeline.run().wait_until_finish()
```

```python
class StatefulAverageDifference(beam.DoFn):

    DOCUMENT_COUNT_SPEC = ReadModifyWriteStateSpec('document_count', PickleCoder())
    WORD_TOTAL_SPEC = ReadModifyWriteStateSpec('word_total', PickleCoder())

    def process(self, element,
                document_count_state=beam.DoFn.StateParam(DOCUMENT_COUNT_SPEC),
                word_total_state=beam.DoFn.StateParam(WORD_TOTAL_SPEC),
                *args, **kwargs):

        # 1. Initialise or load states
        document_count = document_count_state.read() or int()
        word_total = word_total_state.read() or int()

        # 2. Extract document, update state, and calculate average
        _, doc = element # The state is partitioned by key
        document_count = document_count + 1
        word_total = word_total + doc['word_count']
        average = word_total / document_count
        difference = abs(doc['word_count'] - average)

        # 3. Write states
        document_count_state.write(document_count)
        word_total_state.write(word_total)

        # 4. Yield element
        yield {
            'uuid': doc['uuid'],
            'word_count': doc['word_count'],
            'difference': difference,
        }
```

```python
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,
        element,
        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()

ate, and add to clustering model

ed_text_embedding']
 embedding_vector
{'uuid': doc_uuid}
t_2d(embedding_vector))

ollected documents
ict(np.array(list(collected_embeddings.values()
map, new_assignments, update_counter = (
StatefulOnlineClustering.collect_updated_clusters(
abels, label_map, collected_documents,
_embeddings, previous_assignments, update_counter))

(clustering)
rite(label_map)
ate.write(collected_documents)
ngs_state.write(collected_embeddings)
nts_state.write(new_assignments)
ate.write(update_counter)

d clusters
tems in clusters_awaiting_update.items():
        yield {
            'cluster_id': cluster_id,
            'updates': update_counter[cluster_id],
            'documents': items,
        }
```

Online Clustering

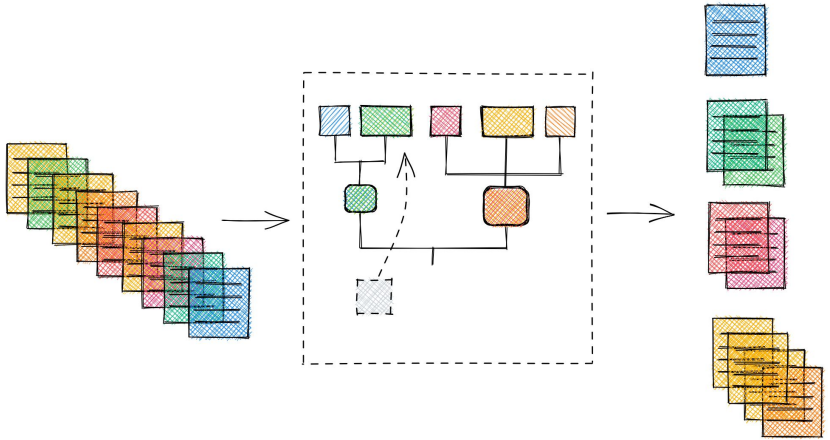


BIRCH:

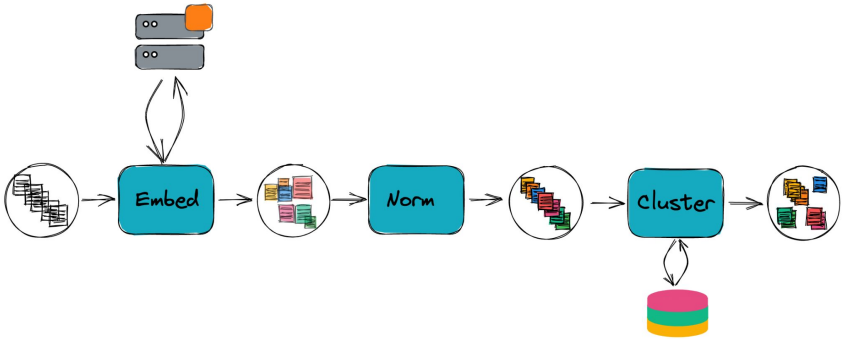
A **clustering** algorithm that works **iteratively**.

Stateful processing:

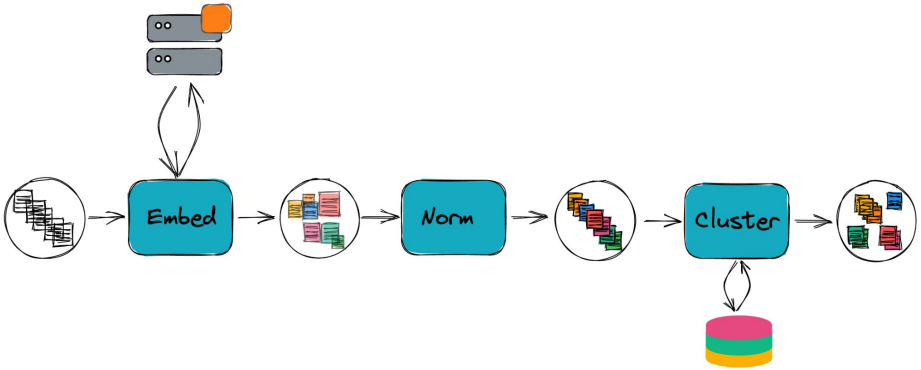
A mechanism to access the **previous state**.



Example



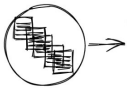
Semantic Enrichment &
Online Clustering
of textual data using Apache Beam


Embed
Norm
Cluster

"Star Wars is my favourite movie!"

"I reject the later edits. Clearly, Han Solo shot first!"

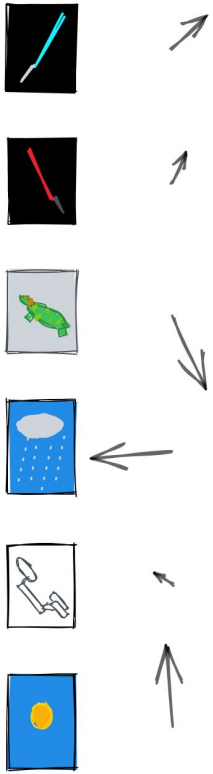
"I like turtles."

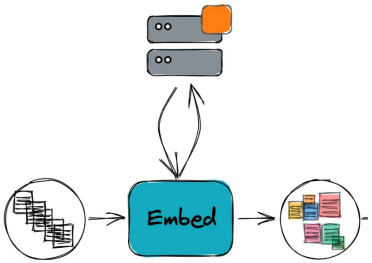


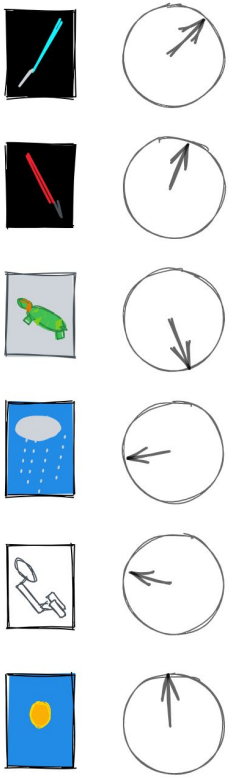
“July could be the first month with no measurable rain in Austin since 2015."

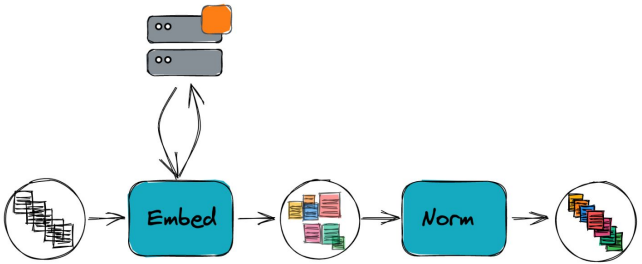
"Star Trek is an awesome series."

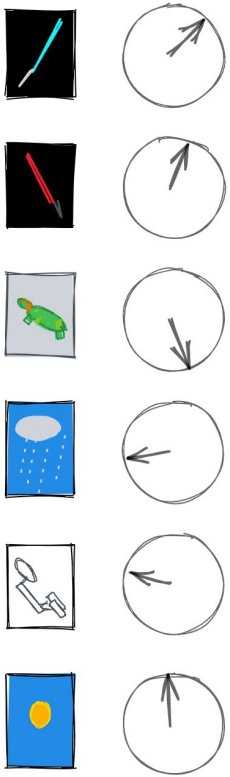
"Dry conditions, low humidity, and breezy winds will allow any fires to spread rapidly."

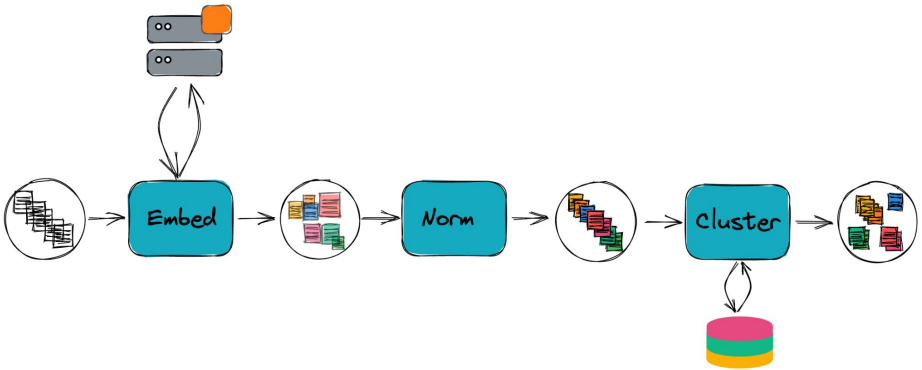


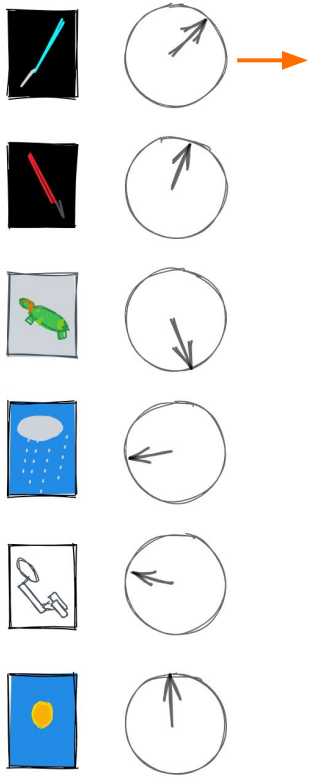

Embed



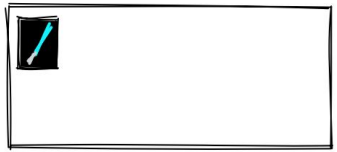

Embed
Norm

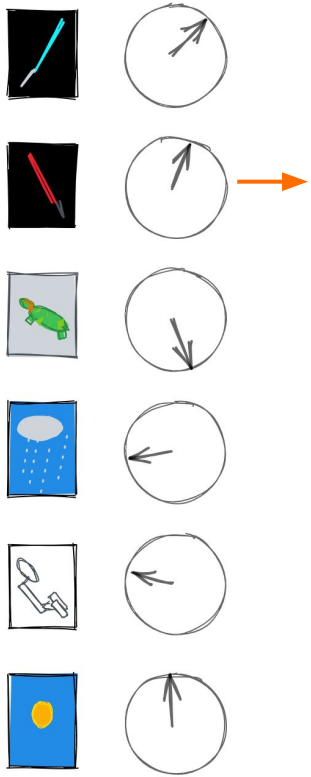


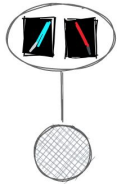

Embed
Norm
Cluster



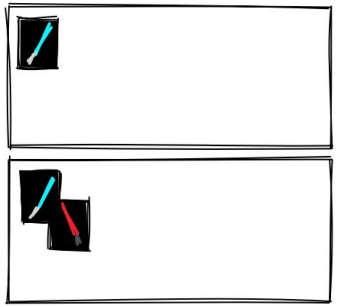
```
New cluster: 10940495-f79e-436a
Documents:
        Movies (Star Wars 1)
```

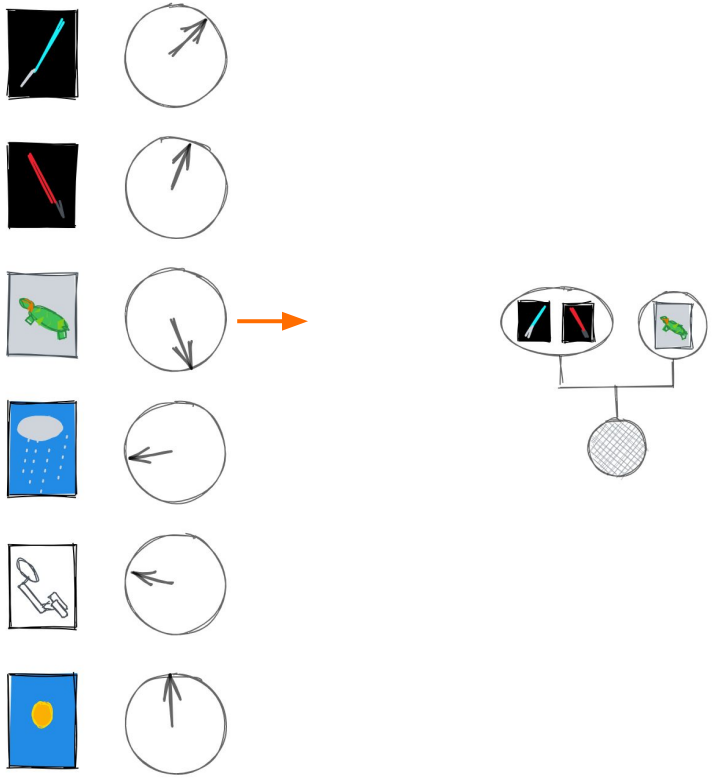




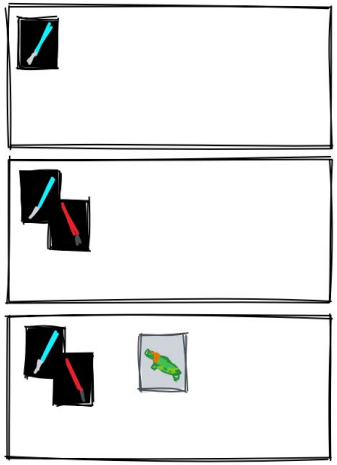


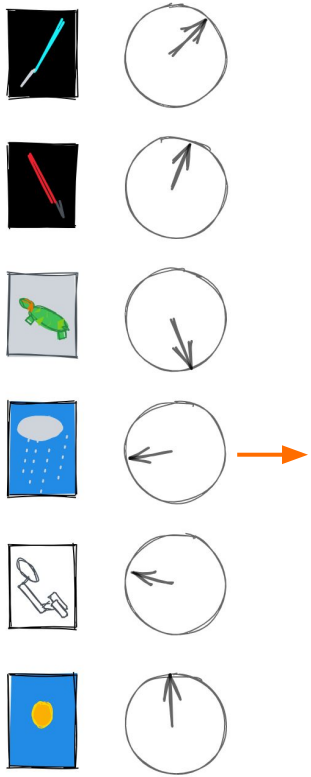
```
New cluster: 10940495-f79e-436a
Documents:
        Movies (Star Wars 1)

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

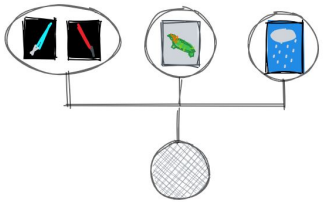




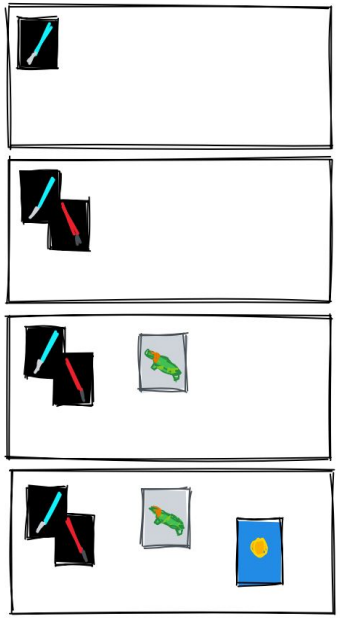
```
New cluster: 10940495-f79e-436a
Documents:
        Movies (Star Wars 1)

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

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

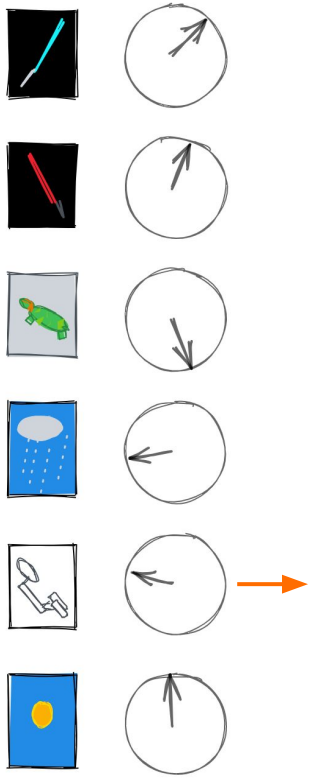


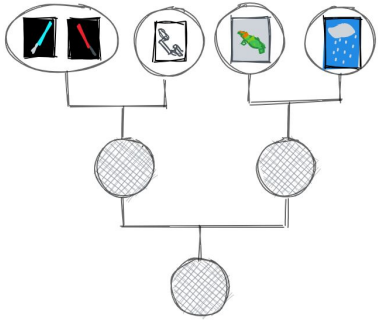




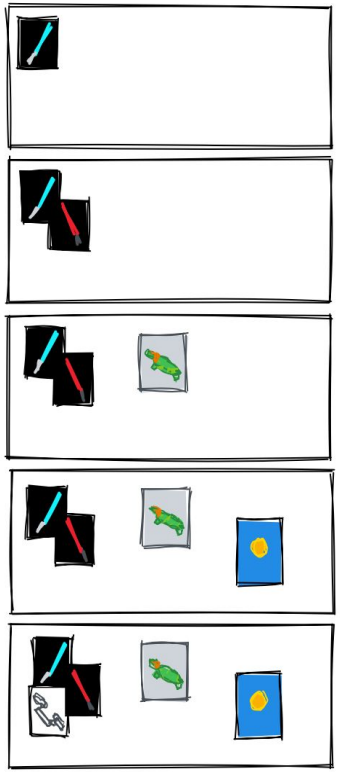
```
New cluster: 10940495-f79e-436a
Documents:
        Movies (Star Wars 1)

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
```

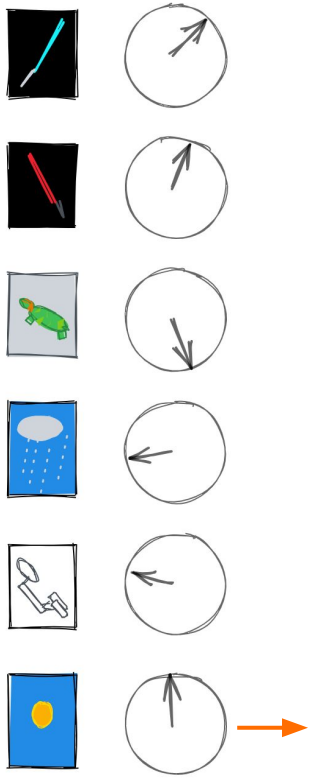


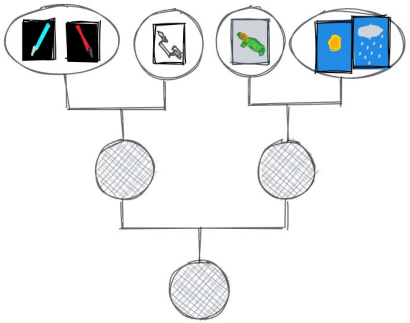




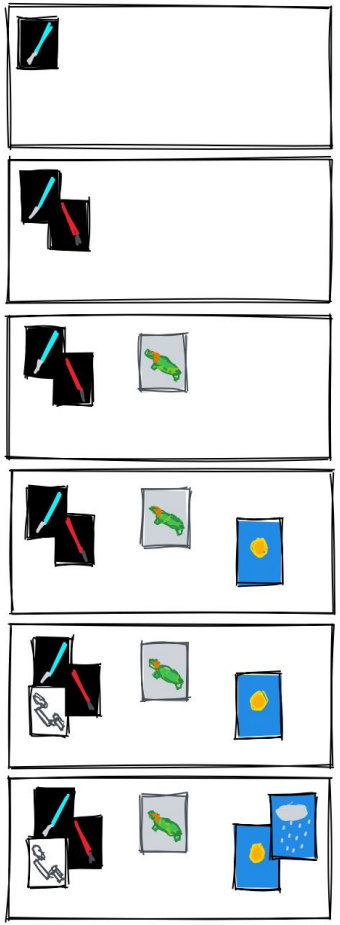
```
New cluster: 10940495-f79e-436a
Documents:
        Movies (Star Wars 1)

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

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

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

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



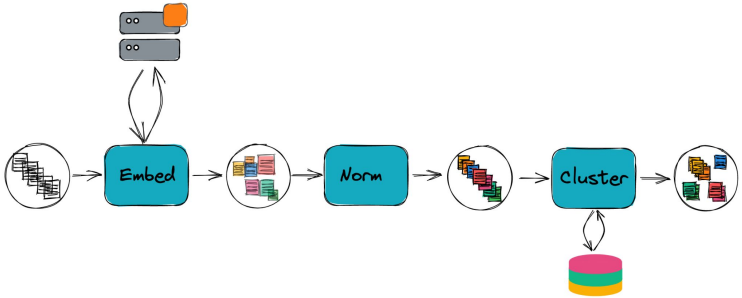




```
New cluster: 10940495-f79e-436a
Documents:
        Movies (Star Wars 1)

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

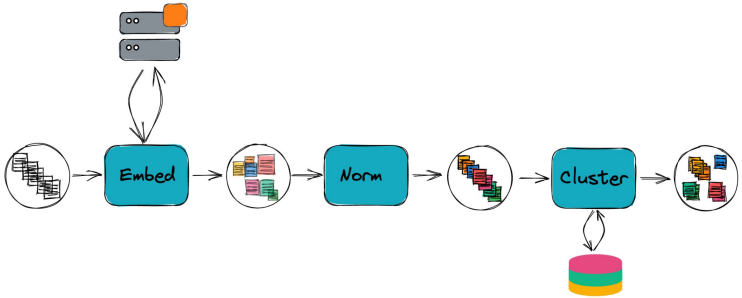


Semantic enrichment adds information from the content to the documents. This often involves **machine learning** which are **expensive operations**.

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

BEAM SUMMIT Austin, 2022

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