DynCluster

Release 0.0.1

Yingxiao Wang

Jun 19, 2025

CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER ONE

SOURCE CODE

1.1 main.py

Listing 1: main.py

```python
"""
This project is a course project for Prof. Qinwu Xu's "Mathematical Theory of Machine Learning" (Spring 2025), School of Mathematics, Nanjing University. It mainly focuses on the problem of dynamic clustering.

The core process is as follows:
→ New data stream arrives
→ Perform clustering on the new data
→ Update cluster centers and adjust previous clustering results
→ If any cluster becomes too large, it is split
→ Adjust the clustering result accordingly
→ Wait for the next batch of data

Author: Yingxiao Wang
Email: wangbottlecap@gmail.com
"""

import logger
from constants.constants import *
from backgrounds import DataReader
from optimizer import DynamicKmeansController, StaticKmeansController
from processer import ComparisonProcessor
from plotter import ComparisonPlotter

def main():
    """
```

(continues on next page)

(continued from previous page)

```python

    """
    LOGGER = logger.setup_logging()
    LOGGER.info(f"This project is a course project for Prof. Qinwu Xu's Mathematical Theory of Machine Learning (Spring 2025), School of Mathematics, Nanjing University. It mainly focuses on the problem of dynamic clustering.")

    rd = DataReader()
    rd.read_fvecs()
    print(rd.data)

    dyn = DynamicKmeansController(data=rd.data)
    dyn.run()

    sta = StaticKmeansController(data=rd.data, k=dyn.getCentroidsNum())
    sta.run()

    processer = ComparisonProcessor(dyn, sta)
    plotter = ComparisonPlotter(processer)
    plotter.print_center_distance()
    plotter.plotAll()

    return


if __name__ == "__main__":
    main()
```

1.2 optimizer.py

Listing 2: optimizer.py

```python
import numpy as np
import time
from sklearn.cluster import KMeans

import logging
LOGGER = logging.getLogger(__name__)

```

(continues on next page)

(continued from previous page)

```python
class DynamicKmeansController():
    """
    Dynamic KMeans clustering controller for streaming data.

    Core process:
    → New data batch arrives
    → Assign new points to existing clusters (or initialize)
    → Update cluster centers
    → Split clusters exceeding max_size into two
    → Compute and print current loss
    → Repeat for next batch

    Attributes:
        data (np.ndarray): full dataset for indexing.
        batch_size (int): number of points per incoming batch.
        min_size (int): minimum cluster size before merge (future extension).
        max_size (int): maximum cluster size before split.
        centroids (list[np.ndarray]): current cluster centers.
        cluster_data_indices (dict[int, list[int]]): mapping cluster IDs to data indices.
        assignments (dict[int, int]): mapping data index to its cluster ID.
        loss_history (list[float]): recorded loss after each batch.
        time_history (list[float]): cumulative processing time after each batch.
    """

    def __init__(self, data:np.ndarray, batch_size=100, min_size=60, max_size=150, ratio = 0.01,threshold = 0.01):
        self.data = data                                  # 保存整个数据集
        self.batch_size = batch_size                      # 每次处理的批大小
        # self.threshold = threshold
        # self.ratio = ratio
        self.min_size = min_size                          # 最小簇大小（留作合并条件）
        self.max_size = max_size                          # 最大簇大小（超过需分裂）
        self.centroids = []                               # 存储当前簇中心列表
        self.cluster_data_indices = {}                    # 存储簇 ID 到样本索引的映射
        self.assignments = {}                             # 存储样本索引到簇 ID 的映射
        self.loss_history = []                            # 记录每批的损失值（inertia）
        self.time_history = []                            # 记录累计时间
        self._current_time = 0.0                          # 内部跟踪累计运行时间

```

(continues on next page)

(continued from previous page)

```
    def getCentroidsNum(self): return len(self.centroids)

    def run(self):
        """Run dynamic clustering over the entire dataset in streaming batches."""
        n_points = self.data.shape[0]                     # 数据总样本数
        # 按批次遍历数据
        for batch_start in range(0, n_points, self.batch_size):
            # 计算本批样本全局索引范围
            batch_indices = list(range(batch_start, min(batch_start + self.batch_size,
↪ n_points)))
            batch = self.data[batch_indices]              # 提取本批数据
            t0 = time.time()                              # 计时开始

            # 增量聚类、更新中心、拆分过大簇
            self._add_batch(batch, batch_indices)
            self._update_clusters()
            self._split_large_clusters()

            batch_loss = self._compute_loss()             # 计算本批后总损失
            self._current_time += time.time() - t0        # 更新累计时间

            self.loss_history.append(batch_loss)          # 记录损失
            self.time_history.append(self._current_time)  # 记录时间
            print(f"Batch {batch_start//self.batch_size + 1}: Loss = {batch_loss:.4f},
↪ Time = {self._current_time:.4f}s")

    def _add_batch(self, batch, indices):
        """Assign a new batch of points to clusters or initialize clustering."""
        if not self.centroids:                           # 如果还没有簇中心（首次调用）
            # 计算初始簇数：确保至少 1 个簇
            k_init = max(1, len(batch) // ((self.min_size + self.max_size) // 2))
            # k_init = max(1, int(self.batch_size * self.ratio))
            # 使用 KMeans 对首批数据做静态聚类初始化
            km = KMeans(n_clusters=k_init, init='k-means++', random_state=42).
↪fit(batch)
            self.centroids = list(km.cluster_centers_)    # 保存初始簇中心
            # 记录每个样本的簇标签
            for idx, lbl in zip(indices, km.labels_):
                self.assignments[idx] = lbl
            # 构建簇到样本索引的映射
```

(continues on next page)

(continued from previous page)

```
            for cid in range(k_init):
                self.cluster_data_indices[cid] = [idx for idx in indices if self.assignments[idx] == cid]
        else:
            # 对后续批次，遍历每个新点
            for idx, point in zip(indices, batch):
                # 计算该点到所有簇中心的欧氏距离
                distances = [np.linalg.norm(point - c) for c in self.centroids]
                lbl = int(np.argmin(distances))            # 选择最近的簇
                self.assignments[idx] = lbl                 # 更新样本-簇映射
                # 将样本索引追加到对应簇的列表中
                self.cluster_data_indices.setdefault(lbl, []).append(idx)

    def _update_clusters(self):
        """Recompute centroids based on current cluster memberships."""
        # 遍历所有簇
        for cid, members in self.cluster_data_indices.items():
            # 提取该簇对应的所有样本点
            pts = np.array([self.data[i] for i in members])
            if pts.size > 0:
                # 重新计算簇中心为样本均值
                self.centroids[cid] = pts.mean(axis=0)

    def _split_large_clusters(self):
        """Split any cluster larger than max_size into two subclusters."""
        # 下一个可用簇 ID
        next_cid = max(self.cluster_data_indices.keys(), default=-1) + 1
        # 找出需要分裂的簇 ID 列表
        oversized = [cid for cid, members in self.cluster_data_indices.items() if len(members) > self.max_size]
        for cid in oversized:
            members = self.cluster_data_indices.pop(cid)  # 暂时移除该簇
            pts = np.array([self.data[i] for i in members])
            # 对过大簇内样本再次做 2 簇 KMeans 拆分
            km2 = KMeans(n_clusters=2, init='k-means++', random_state=42).fit(pts)
            centers2 = km2.cluster_centers_
            labels2 = km2.labels_

            # 更新原簇中心及成员
            self.centroids[cid] = centers2[0]
```

(continues on next page)

(continued from previous page)

```
            self.cluster_data_indices[cid] = [members[i] for i, l in␣
↪enumerate(labels2) if l == 0]

            # 添加新簇中心及成员
            self.centroids.append(centers2[1])
            self.cluster_data_indices[next_cid] = [members[i] for i, l in␣
↪enumerate(labels2) if l == 1]

            # 更新分裂后所有样本的簇归属
            for idx, lbl in zip(members, labels2):
                self.assignments[idx] = cid if lbl == 0 else next_cid

            next_cid += 1                            # 更新下一个可用簇 ID

    def _compute_loss(self):
        """Compute total inertia (sum of squared distances) as loss."""
        total_loss = 0.0
        # 遍历每个簇计算簇内平方和
        for cid, members in self.cluster_data_indices.items():
            center = self.centroids[cid]                # 当前簇中心
            pts = np.array([self.data[i] for i in members])
            total_loss += np.sum((pts - center) ** 2)    # 累加平方误差
        return total_loss



class StaticKmeansController:
    """
    Static KMeans clustering controller using faiss for full-dataset clustering.

    Attributes:
        data (np.ndarray): Full dataset of shape (N, D), dtype float32.
        k (int): Number of clusters.
        niter (int): Number of iterations for faiss KMeans.
        seed (int): Random seed for reproducibility.
        centroids (np.ndarray): Final cluster centers of shape (k, D).
        assignments (dict[int, int]): Mapping from data index to assigned cluster ID.
        cluster_data_indices (dict[int, list[int]]): Mapping from cluster ID to list␣
↪of data indices.
        loss (float): Final inertia (sum of squared distances to centroids).
```

(continues on next page)

(continued from previous page)

```
        time_taken (float): Time spent on clustering (seconds).
    """

    def __init__(self, data: np.ndarray, k: int, niter: int = 100, seed: int = 42):
        self.data = data.astype('float32')
        self.k = k
        self.niter = niter
        self.seed = seed
        self.centroids = None
        self.assignments = {}
        self.cluster_data_indices = {}
        self.loss = None
        self.time_taken = None

    def run(self):
        """
        Run static KMeans clustering using faiss.Kmeans.
        Records centroids, assignments, cluster_data_indices, loss, and time_taken.
        """
        try:
            import faiss
        except ImportError:
            raise ImportError("faiss library not installed. Please install faiss to use StaticKmeansController.")

        # Initialize faiss KMeans
        d = self.data.shape[1]
        kmeans = faiss.Kmeans(d, self.k, niter=self.niter, verbose=False, seed=self.seed)

        # Train and time
        start_time = time.time()
        kmeans.train(self.data)
        self.time_taken = time.time() - start_time

        # Retrieve centroids
        self.centroids = kmeans.centroids.copy()

        # Assign each point to the nearest centroid
        distances, labels = kmeans.index.search(self.data, 1)
```

(continues on next page)

(continued from previous page)

```python
        labels = labels.flatten()

        # Build assignments and cluster-to-indices mapping
        for idx, lbl in enumerate(labels):
            self.assignments[idx] = int(lbl)
            self.cluster_data_indices.setdefault(int(lbl), []).append(idx)

        # Compute inertia (loss)
        total_loss = 0.0
        for idx, lbl in self.assignments.items():
            center = self.centroids[lbl]
            point = self.data[idx]
            total_loss += np.sum((point - center) ** 2)
        self.loss = float(total_loss)

    def report(self):
        """
        Return a summary of clustering results.
        """
        return {
            'Method': 'Static faiss.Kmeans',
            'Time (s)': self.time_taken,
            'Loss': self.loss,
            'Num Clusters': self.k,
            'Cluster Sizes': {cid: len(idxs) for cid, idxs in self.cluster_data_indices.items()}
        }
```

1.3 background.py

1.4 logger.py

Listing 3: logger.py

```python
import os
import logging

from constants.filepath_constants import RESULTS_DIR

```

(continues on next page)

(continued from previous page)

```
def setup_logging(fileName = 'app.log'):
    # set up logging file, format and location, information level

    # logFormat = "%(asctime)s - %(levelname)s - %(message)s"
    logFormat = "%(message)s"
    dateFormat = "%m/%d/%Y %H:%M:%S %p"

    # fileHandler
    path = os.path.join(RESULTS_DIR, fileName)
    file_handler = logging.FileHandler(path, mode='w')

    logging.basicConfig(
        format=logFormat,
        datefmt=dateFormat,
        level=logging.DEBUG,
        handlers= [
            file_handler,
            # stream_handler
        ]
    )

    LOGGER = logging.getLogger()
    return LOGGER
```

1.5 processer.py

Listing 4: processer.py

```
import logging
from optimizer import DynamicKmeansController, StaticKmeansController
from util import *
LOGGER = logging.getLogger(__name__)

# 2. 数据处理类
class ComparisonProcessor:
    """
    对比动态 vs 静态 聚类结果，存储以下信息:
      - center_distance: 两组簇中心平均最近距离
      - dynamic_loss, static_loss
```

(continues on next page)

(continued from previous page)

```
      - dynamic_time, static_time
      - dynamic_history: loss & time 历史
    """
    def __init__(self, dynController:DynamicKmeansController, staController:StaticKmeansController):
        self.dyn = ComparisonProcessor.summarize_dynamic(dynController)
        self.sta = ComparisonProcessor.summarize_static(staController)
        self.results = {}
        self._compute()

    def _compute_center_distance(self):
        # 对于每个动态中心，找到距离最近的静态中心，计算平均距离
        dyn_c = self.dyn['centroids']
        sta_c = self.sta['centroids']
        dists = []
        for c in dyn_c:
            diff = sta_c - c
            dist_to_sta = np.linalg.norm(diff, axis=1)
            dists.append(np.min(dist_to_sta))
        return float(np.mean(dists))

    def _compute(self):
        self.results['center_distance'] = self._compute_center_distance()
        self.results['dynamic_loss'] = float(self.dyn['loss'])
        self.results['static_loss'] = float(self.sta['loss'])
        self.results['dynamic_time'] = float(self.dyn['time'])
        self.results['static_time'] = float(self.sta['time'])
        self.results['dynamic_loss_history'] = self.dyn['loss_history']
        self.results['dynamic_time_history'] = self.dyn['time_history']


    @staticmethod
    def summarize_dynamic(controller:DynamicKmeansController):
        """
        输出动态聚类摘要信息
        Returns a dict with:
        - num_clusters: 最终簇数
        - loss: 最终 loss
        - time: 累计运行时间
        - loss_history: 每批 loss 的列表
```

(continues on next page)

(continued from previous page)

```
        - time_history: 每批累计时间的列表
        - centroids: 簇中心数组, shape=(k, D)
        """
        centroids = np.stack(controller.centroids, axis=0)
        return {
            'method': 'dynamic',
            'num_clusters': centroids.shape[0],
            'loss': controller.loss_history[-1],
            'time': controller.time_history[-1],
            'loss_history': np.array(controller.loss_history),
            'time_history': np.array(controller.time_history),
            'centroids': centroids
        }

    @staticmethod
    def summarize_static(controller:StaticKmeansController):
        """
        输出静态聚类摘要信息
        Returns a dict with:
        - num_clusters: 簇数 k
        - loss: 最终 loss
        - time: 聚类耗时
        - centroids: 簇中心数组, shape=(k, D)
        """
        return {
            'method': 'static',
            'num_clusters': controller.k,
            'loss': controller.loss,
            'time': controller.time_taken,
            'centroids': controller.centroids
        }
```

1.6 plotter.py

Listing 5: plotter.py

```python
import matplotlib.pyplot as plt

import logging
from processer import ComparisonProcessor
from constants.filepath_constants import RESULTS_DIR

LOGGER = logging.getLogger(__name__)






# 3. 画图类
class ComparisonPlotter:
    """
    接收 ComparisonProcessor, 绘制:
      - loss 对比 (bar)
      - time 对比 (bar)
      - 中心距离 (单个文本展示)
      - 动态聚类 loss & time 历史曲线
    """
    def __init__(self, comp: ComparisonProcessor):
        self.comp = comp

    def plotAll(self):
        self.plot_dynamic_history()
        self.plot_loss_comparison()
        self.plot_time_comparison()

    def plot_loss_comparison(self):
        fig = plt.figure()
        labels = ['dynamic', 'static']
        values = [self.comp.results['dynamic_loss'], self.comp.results['static_loss']]
        plt.bar(labels, values)
        plt.ylabel('Loss')
        plt.title('Dynamic vs Static Loss Comparison')
        plt.show()
```

(continues on next page)

(continued from previous page)

```python

    def plot_time_comparison(self):
        fig = plt.figure()
        labels = ['dynamic', 'static']
        values = [self.comp.results['dynamic_time'], self.comp.results['static_time']]
        plt.bar(labels, values)
        plt.ylabel('Time (s)')
        plt.title('Dynamic vs Static Time Comparison')
        plt.show()

    def print_center_distance(self):
        dist = self.comp.results['center_distance']
        print(f"Average nearest-center distance between dynamic and static: {dist:.4f}")

    def plot_dynamic_history(self):
        # Loss history
        fig1 = plt.figure()
        plt.plot(self.comp.results['dynamic_loss_history'])
        plt.xlabel('Batch')
        plt.ylabel('Loss')
        plt.title('Dynamic Clustering Loss over Batches')
        plt.show()
        # Time history
        fig2 = plt.figure()
        plt.plot(self.comp.results['dynamic_time_history'])
        plt.xlabel('Batch')
        plt.ylabel('Cumulative Time (s)')
        plt.title('Dynamic Clustering Time over Batches')
        plt.show()
```

1.7 util.py

Listing 6: util.py

```python
import numpy as np
```