Parallel k-means

with joblib and numba

Christian Mancini

Firenze, Settembre 2024

Parallel k-means

L’obbiettivo del progetto è quello di utilizzare la libreria Joblib per parallelizzare il calcolo del k-means e superare la limitazione del GIL. Oltre a joblib è stata usata anche la libreia Numba che è un compilatore Just in Time per eseguire codice su CPU o GPU. (Nel nostro caso solo CPU). La differenza è che JobLib crea processi che utilizzano l’interprete, Numba compila e poi esegue senza dover usare l’interprete.

Joblib implementazione

```
def compute_label_joblib(point, centroids):
    k = centroids.shape[0]
    distances = np.empty(k)
    for i in range(k):
        distances[i] = np.linalg.norm(point - centroids[i])
    return np.argmin(distances)
```

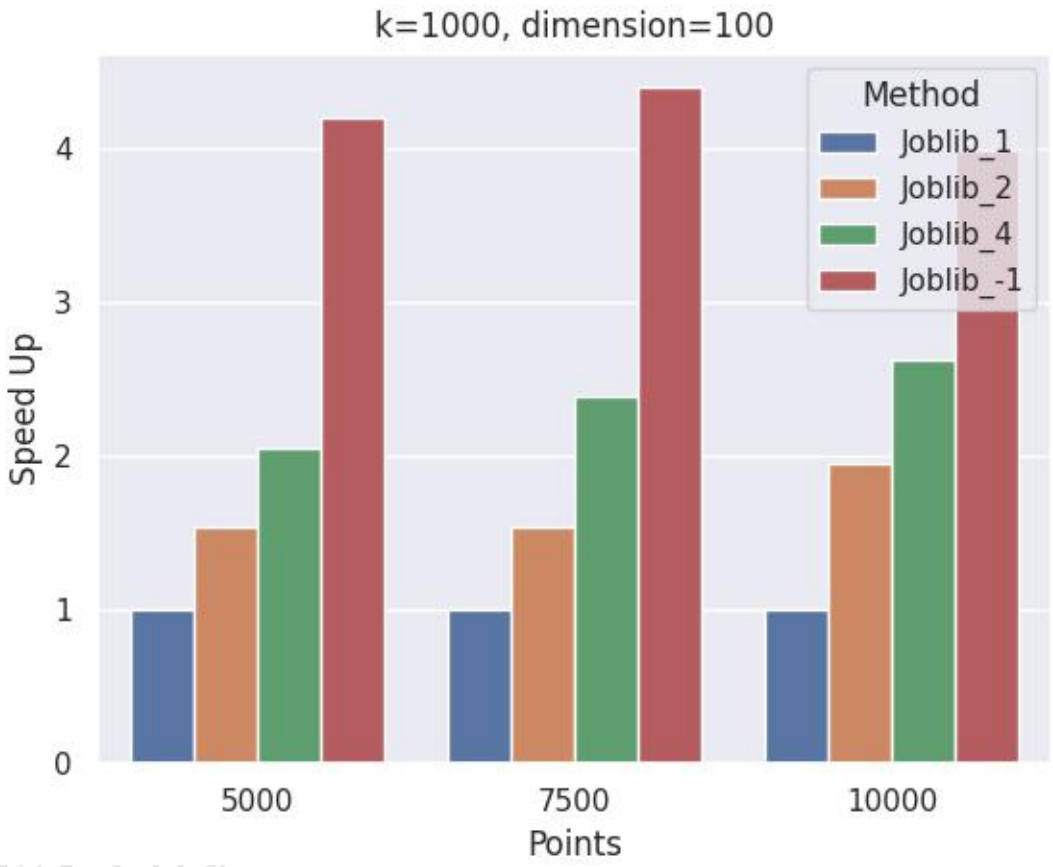
L'implementazione è la stessa di quella sequenziale. Con Joblib ciò che cambia è la chiamata della funzione.

Joblib implementazione

```
def fit(self, n_jobs = 1):
    for _ in range(100):
        if self.use_numba:
            new_labels = np.empty(self.points.shape[0], dtype=np.int32)
            for i in range(self.points.shape[0]):
                new_labels[i] = compute_label_numba(self.points[i], self.centroids)
        else:
            new_labels = Parallel(n_jobs=n_jobs)(delayed(compute_label_joblib)(self.points[i], self.centroids) for i in range(self.points.shape[0]))
            new_labels = np.array(new_labels)
```

Nel metodo fit della classe kMeans viene chiamata la funzione con joblib specificando il numero di jobs.

Joblib SpeedUp



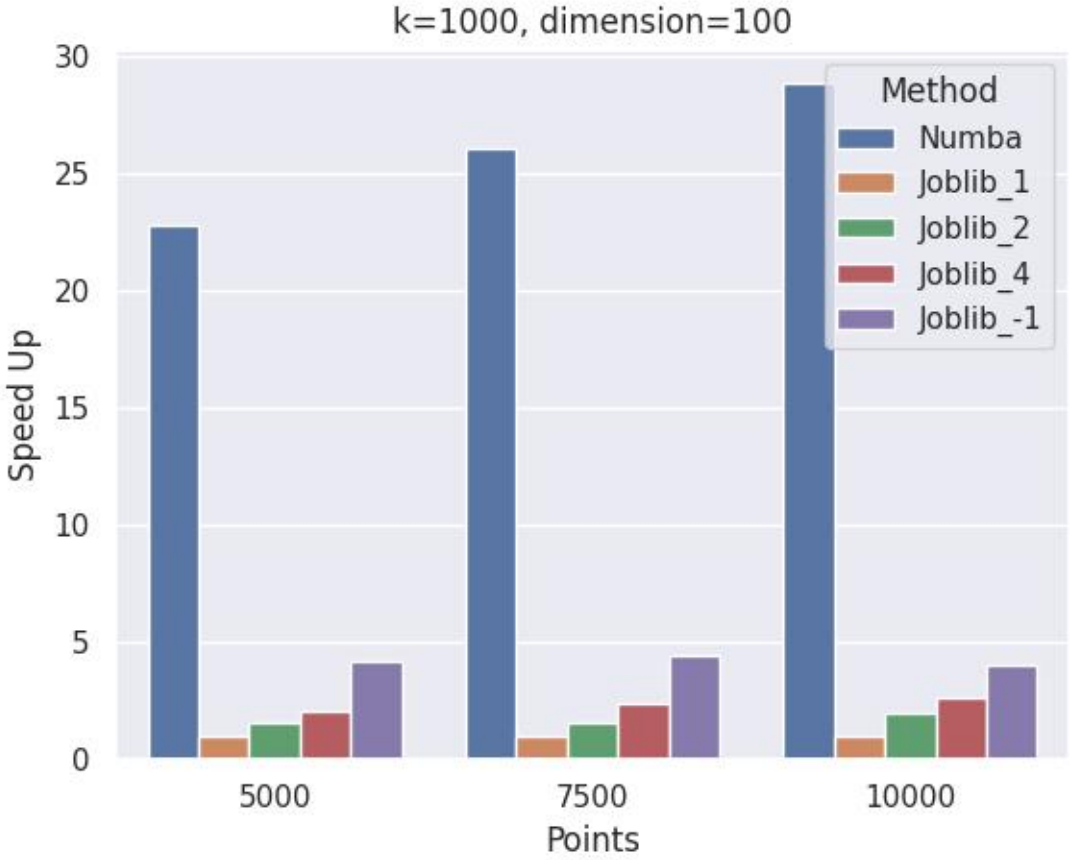
Usando delle dimensioni che giustifichino l'overhead della gestione della parallelizzazione, possiamo ottenere uno speedup di 4 volte utilizzando tutti i core disponibili (in questo caso 16).
Non otteniamo però uno speedup lineare nel numero di core.

Numba implementazione

```
@njit(parallel=True)
def compute_label_numba(point, centroids):
    k = centroids.shape[0]
    distances = np.empty(k)
    for i in prange(k):
        distances[i] = np.linalg.norm(point - centroids[i])
    return np.argmin(distances)
```

Parallelizzare con numba è molto semplice, basta decorare la funzione dichiarando che deve essere compilata e parallelizzata. La n prima del decoratore jit indica di restituire un errore se numba non riesce a compilare la funzione e di non fare il fallback su python.
Per i for inoltre si deve usare prange al posto di range.

Numba SpeedUp



Con Numba otteniamo uno speed up decisamente maggiore, superiore a 20 volte.