Supplementary Material
cblearn: Comparison-based Machine Learning in Python

David-Elias Künstle and Ulrike von Luxburg
University of Tübingen and Tübingen AI Center, Germany

22 September 2023

Empirical evaluation

We generated embeddings of comparison-based datasets to measure runtime and triplet accuracy as a small empirical evaluation of our ordinal embedding implementations. We compared various CPU and GPU implementations in `cblearn` with third-party implementations in R (`loe` Terada and Luxburg 2014), and *MATLAB* (Maaten and Weinberger 2012). In contrast to synthetic benchmarks (e.g., Vankadara et al. 2021), we used the real-world datasets that can be accessed through *cblearn*, converted to triplets. Every algorithm runs once per dataset on a compute node (8-core of Intel®Xeon ®Gold 6240; 96GB RAM; NVIDIA 2080ti); just a few runs did not yield results due to resource demanding implementations or bugs: our *FORTE* implementation exceeded the memory limit on the *imagenet-v2* dataset, the third-party implementation of *tSTE* timed out on *things* and *imagenet-v2* datasets. The third-party *SOE* implementation reached a limit on dataset size on *imagenet-v2*. Probably due to numerical issues, our *CKL-GPU* implementation did not crash but returned non-numerical values on the *musician* dataset.

The benchmarking scripts and results are publicly available in a separate repository[1].

Is there a "best" estimator?

Comparing all ordinal embedding estimators in `cblearn`, *SOE* shows accurate and fast results; *CKL*, *GNMDS*, and *tSTE* were performing about equally well (Figure 1). The GPU implementations are slower on the tested datasets, presumably because they had some initial computational overhead.

[1] https://github.com/cblearn/cblearn-benchmark

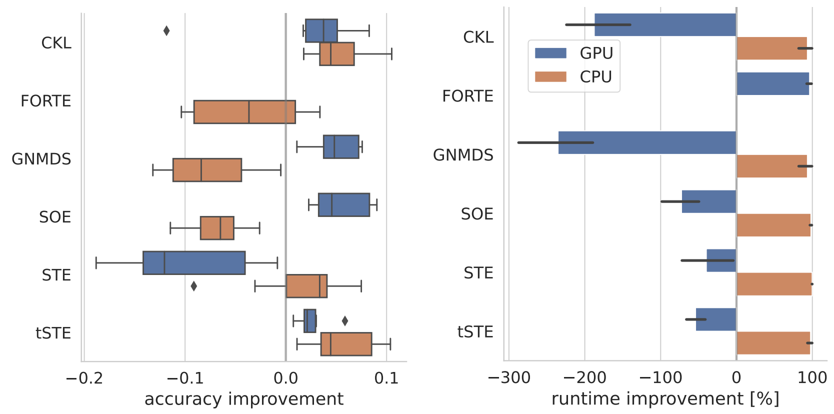


Figure 1: The triplet error and runtime improvement per estimator. The improvement is relative to the average accuracy or runtime (at 0), calculated over all shown estimators. Compared to this average, a positive accuracy improvement means the algorithm can represent more triplets, while a positive runtime improvement means the algorithm is faster. Likewise, negative improvement indicates fewer triplets and lower speed, respectively.

When should GPU implementations be preferred?

In terms of accuracy and runtime, our GPU (or `pytorch`) implementations could not outperform the GPU (or `scipy`) pendants on the tested datasets. However, Figure 1 shows the GPU runtime grows slower with the number of triplets, such that they potentially outperform CPU implementations with large datasets of 10^7 triplets and more. In some cases, the GPU implementations show the overall best accuracy. An additional advantage of GPU implementations is that they require no explicit gradient definition, which allows easier tweaking of algorithms. However, the stochastic optimization process of these GPU implementations might be more sensitive to hyperparameter choices.

How does `cblearn` compare to other implementations?

In a small comparison, our implementations were $50 - 100\%$ faster with approximately the same accuracy as reference implementations (Figure 3). We compared our CPU implementations of *SOE*, *STE* and *tSTE* with the corresponding reference implementations in R , `loe` (Terada and Luxburg 2014), and *MATLAB* (Maaten and Weinberger 2012). Additionally, the latter offers an *CKL* implementation to compare with. Please note that despite our care, runtime comparisons of different interpreters offer room for some confounding effects. The results shown should nevertheless indicate a trend.

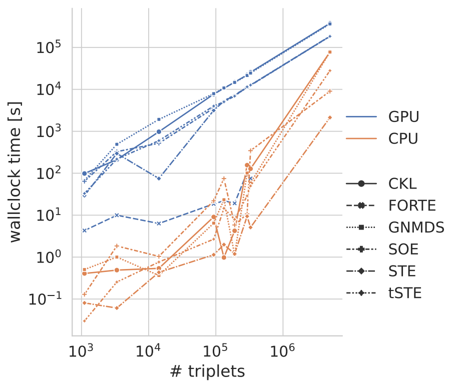


Figure 2: The runtime increases almost linearly with the number of triplets. However, GPU implementations have a flatter slope and thus can compensate for the initial time overhead on large datasets.

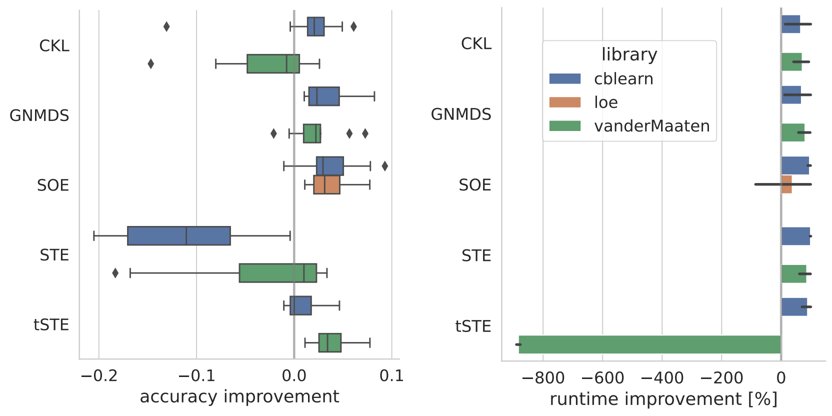


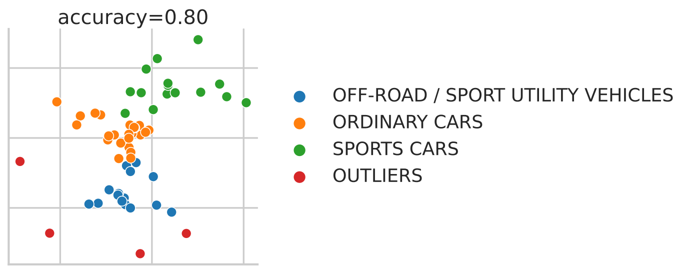
Figure 3: The triplet error and runtime improvement per estimator. Improvement is calculated against the average performance per dataset of all estimators in the plot.

Code example

```
from cblearn import datasets, preprocessing, embedding
from sklearn.model_selection import cross_val_score
import seaborn as sns; sns.set_theme("poster", "whitegrid")

cars = datasets.fetch_car_similarity()
triplets = preprocessing.triplets_from_mostcentral(cars.triplet, cars.response)
accuracy = cross_val_score(embedding.SOE(n_components=2), triplets, cv=5).mean()
```

```
embedding = embedding.SOE(n_components=2).fit_transform(triplets)
fg = sns.relplot(x=embedding[:, 0], y=embedding[:, 1],
        hue=cars.class_name[cars.class_id])
fg.set(title=f"accuracy={accuracy:.2f}", xticklabels=[], yticklabels=[])
fg.tight_layout(); fg.savefig("images/car_example.pdf")
```



References

Maaten, Laurens van der, and Kilian Weinberger. 2012. "Stochastic Triplet Embedding." In *International Workshop on Machine Learning for Signal Processing*, 1–6.

Terada, Yoshikazu, and Ulrike Luxburg. 2014. "Local Ordinal Embedding." In *International Conference on Machine Learning (Icml)*.

Vankadara, Leena Chennuru, Siavash Haghiri, Michael Lohaus, Faiz Ul Wahab, and Ulrike von Luxburg. 2021. "Insights into Ordinal Embedding Algorithms: A Systematic Evaluation." *arXiv:1912.01666 [Cs, Stat]*.