Draco-LLM chart recommendation system

Mateusz Wojciechowski

December 2024

This report refers to my github repository: draco-LLM-char-recommendation-system

1 Task 1, part 1: A quick introduction to the system

In this section, I will briefly describe how the system is built. I will not go into detail about the implementation itself. More details on this matter will be provided in the README.md file attached to the repository.

1.1 Loading the dataset and preparing the schema

I began by loading a dataset from Vega-datasets, in this case, the Seattle weather dataset. Using the Draco library, I converted this dataframe to a facts schema.

1.2 Choosing columns for visualization

To obtain the preferred pair of columns for visualization, I created a prompt to a GPT model from the OpenAI API. I also created all possible remaining pairs of columns, to test their visualizations against the ones chosen by the LLM.

1.3 Generating chart recommendations

For all possible pairs of columns I generated top: 1, 3 and 5 visualizations according to Draco.

2 Task 1, part 2: Assigning scores to Draco recommendations

This section covers the approach I took to scoring obtained recommendations. For this purpose I decided to use both Draco constraint scores and evaluation scores **from Lida library I forked and slightly modified**. I observed that many generated visualizations had very similar or identical Draco constraint costs and were judged only by relying on formal visualization principles, which is why I decided to use another source of evaluation to get a bigger picture. In the cases with multiple visualizations generated I took mean scores for all referring to a single column pair.

2.1 Modified Lida vizevaluator

Lida is an open-source visualization recommendation tool published recently by Microsoft and described in the following paper: [Dib23]. It has multiple functionalities such as dataset summarization, goal generation and natural language based refinement, but for the purpose of this task I only utilized its modified chart evaluator. I forked the library and changed its vizevaluator.py file so that it was **compatible with Vega-Lite chart specification**. All the changes are described in the docstring located inside this file. You can find my Lida fork here: lida-for-vega-lite. The evaluator assigned score from 1 to 10 to each of the following dimensions: bugs, visualization type, data encoding, aesthetics. I took the mean of all those scores as the **Lida score**.

2.2 Draco cost

As described previously along with Lida evaluator I used Draco constraint costs as they provide a valuable insight into the quality of the charts. In the case of Draco cost, the lower it is, the better the visualization according to Draco.

2.3 Combining both scores

I combined both scores to create a final score described by the following formula:

$$\text{final\_score} = \frac{\frac{\text{lida\_score}}{10} + \frac{1}{1+\text{draco\_cost}}}{2}$$

3 Results

The visualizations in this section are only a part of all generated. You can find the rest of them in the *results_visualization.ipynb* notebook attached to the repository.

3.1 Lida score, Draco cost and Final score

These particular plots refer to the case where I generated the top 1 visualization according to Draco for each column pair. The pair chosen by the LLM is highlighted.

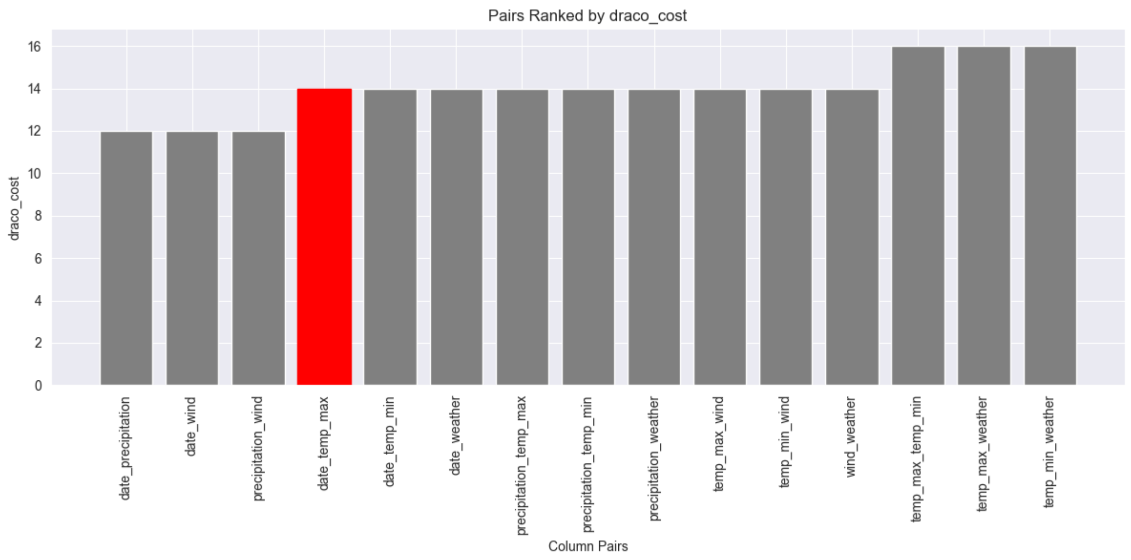

Figure 1: Draco costs for all column pairs

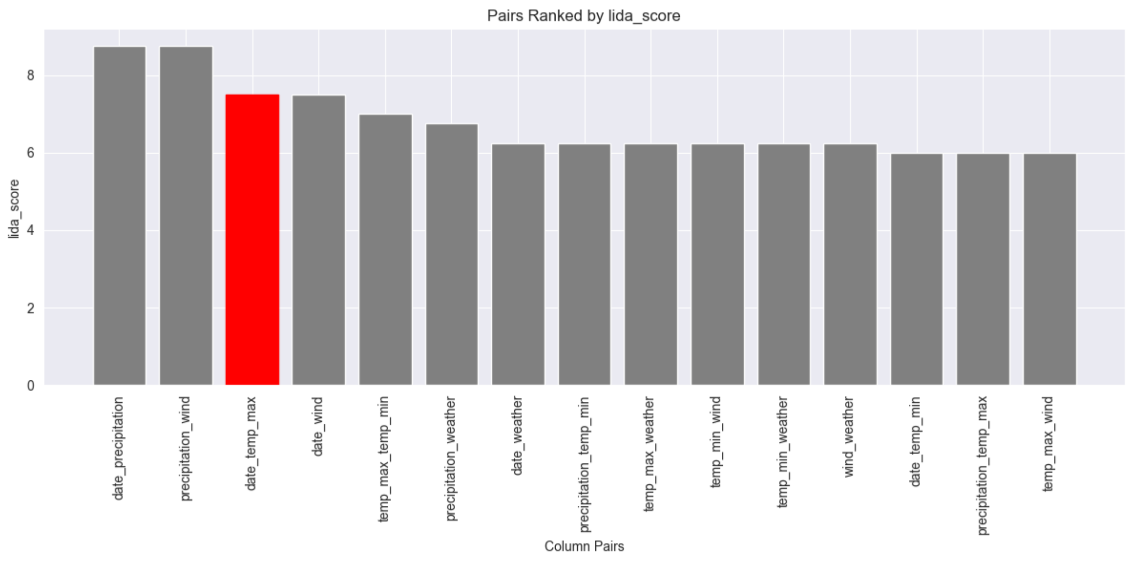

Figure 2: Lida score for all column pairs

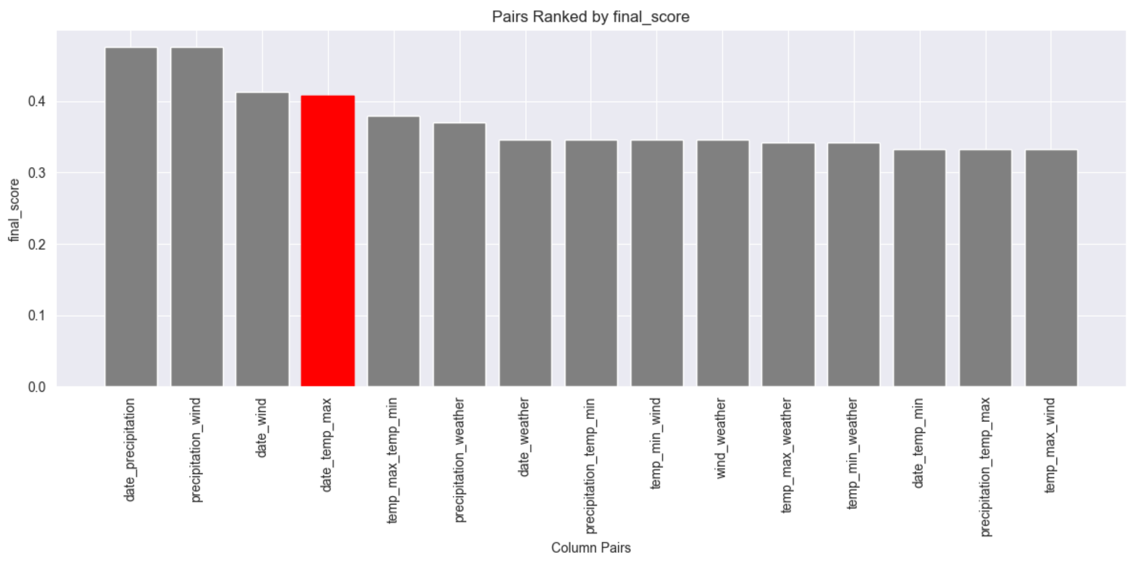

Figure 3: Final score for all column pairs

3.2 The relationship between Draco cost and Lida score

The plots below present the relationship between Lida score and Draco cost for the case of generating the top 1 recommendation. Plots for other cases can also be found in the notebook mentioned above.

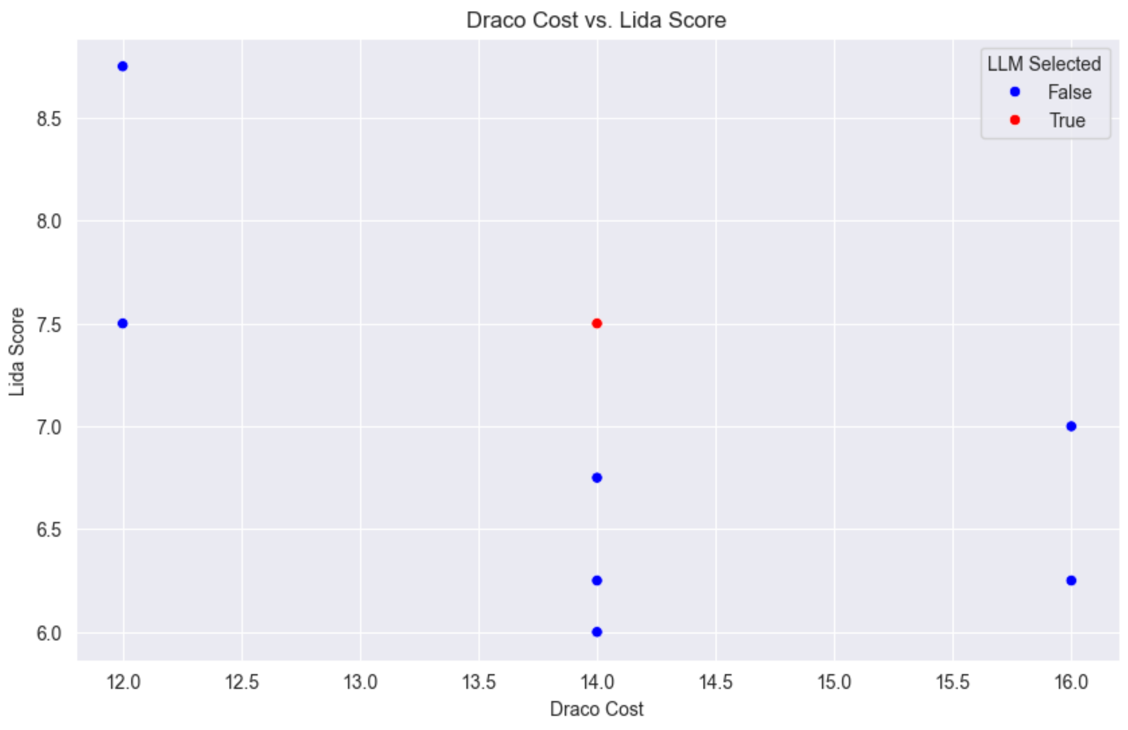

Figure 4: Relationship between Draco cost and Lida score

3.3 Result Table

The LLM chose **date and temp_max** columns for visualization. Results of the top final scores and the column pairs for which they were achieved are presented below. In conclusion, LLM-picked columns did not lead to the best visualization recommendations according to the scoring I used.

Table 1: Final scores result Table

Chart count	**Final score for a selected pair**	**highest scoring column pair**	**Highest final score**
1	0.4083	date, precipitation	0.4760
3	0.3958	precipitation, wind	0.4760
5	0.3879	precipitation, wind	0.4273

4 Task 2.1

One of the most important elements of machine learning projects is evaluating solutions and being able to compare them. How would you evaluate your chart recommendation system from Task 1? You may research existing approaches or provide your own ideas.

4.1 How to evaluate the system from Task 1

Evaluating such a system would require seeking a source of visualization ground truth. As in the initial part of developing such a system we would not be able to collect feedback from a wide and diverse group of users, we need to approximate what users of such a system would desire. That is why, in my opinion, we should divide the evaluation process into two parts: **pre-deployment evaluation and post-deployment evaluation**.

4.2 Pre-deployment evaluation

Even though Draco is built upon formal rule sets and is able to generate correct visualizations, it **lacks a human factor**. During my thought process and while reading multiple scientific papers on this matter, I came to the conclusion that this system requires feedback from a different type of visual recommendation model in order to assess whether the generated visualization is valid.

Some of the papers I read share my opinion on the matter. For instance, the VizML paper [Hu+19] :-

> Most recommender systems encode these visualization guidelines as collection of "if-then" statements, or rules, to automatically generate visualizations. In contrast, machine learning (ML)-based systems directly learn the relationship between data and visualizations by training models on analyst interaction. While recent systems (rule based such as Draco) are exciting, **they do not learn to make visualization design choices as an analyst would**

There are multiple chart recommendation tools described in research papers achieving standout results in the field, which we could use to evaluate the output of our system. In some cases, it would require tuning them to our preference as I did with Lida. These tools would allow us to modify Draco's knowledge base and also learn weights as it is possible according to Draco's documentation. I think combining both Draco and some other (likely ML-based) tools for evaluation is likely to yield promising results in the field of visualization recommendation, comprising both visualization correctness and the human factor. In my opinion, generating evaluation based on multiple sources of visual recommendation knowledge combined would prove to be the most effective, though the effectiveness of those tools should be verified.

Some of the tools that we could use to evaluate the quality of recommended charts:

4.2.1 Lida

Lida, as described earlier in this report, relies almost entirely on LLM-based mechanisms. Since it is open-source, we can tailor it to our specific needs, as demonstrated above. For evaluation purposes, its `vizevaluator` and `vizrepairer` modules might be particularly useful for both scoring recommended charts and suggesting improvements. Lida can not only generate numerical scores that capture best-practice criteria, but it can also provide natural language explanations for why certain aspects of a visualization may or may not be optimal.

4.2.2 DeepEye

Although DeepEye [Yuy+18] is not the newest tool, its functionalities still suit our evaluation goals well. We can employ its Visualization Ranking module to decide which of two (or more) plots is more appropriate. This results in a numeric ranking score or classification probability reflecting chart suitability. If multiple charts are produced for the same dataset or column pair, DeepEye effectively performs pairwise comparisons like a mini "tournament", to find the best option.

Moreover, by leveraging DeepEye's scoring, it is possible to construct a **dataset suitable for learning Draco's constraint weights** by systematically comparing pairs of visualizations. This approach could provide Draco with detailed, data-driven insights into which design choices tend to work better in practice.

4.2.3 VizML

VizML, as described in its paper, has been shown to be highly effective when recommending charts, thanks to training on a large corpus of datasets and associated visualizations. The main obstacle lies in its recommendation format, which is expressed as a set of design choices rather than a straightforward specification like Vega-Lite. If we bridge that gap, for example, by implementing a translator that converts VizML's design-choice output into a Vega-Lite specification, VizML's predictions can help evaluate Draco's design decisions. Essentially, if Draco's final choices deviate significantly from what VizML would recommend based on its large-scale training corpus, we could assign a lower confidence rating.

4.2.4 Combining These Evaluation Methods

All three tools: Lida, DeepEye, and VizML offer distinct perspectives on visualization quality. Lida scores charts from an LLM-oriented perspective and provides human-readable feedback, DeepEye ranks visualizations based on a learned "good vs. bad" approach, and VizML offers corpus-driven design-choice probabilities. By incorporating all three we could: Assign Lida quality scores, compare suggestions using DeepEye and check how the recommended design choices match VizML's predictions. Lastly combining it all into an aggregated score we should get a versatile evaluation metric.

4.2.5 Evaluating LLM's column choice

Along the evaluation process we should also monitor and if needed, tune LLM's column choice, for example through prompt engineering similarly to Lida.

4.3 Post-deployment evaluation

Post-deployment evaluation of our system can benefit from a combination of real-world usage data, direct user feedback and iterative refinement. Once the system is live, collecting logs of user interactions, such as, which recommended charts are accepted, edited, or rejected, becomes crucial, as it provides insights into how well the suggestions resonate with actual needs. Monitoring patterns of usage over time, for example, how frequently certain recommendations are chosen or how often users dismiss them, can reveal system strengths and weaknesses. In addition to quantitative metrics like **click-through rates**, it is essential to gather qualitative feedback in the form of surveys or interviews. These conversations can identify usability issues, highlight valuable edge cases, and uncover aspects of the recommendation pipeline that might cause confusion or frustration. Over the longer term, it may be necessary to conduct controlled studies in which different versions of the recommendation engine, or different tuning parameters, are compared for performance on real tasks. Post-deployment tuning can also involve adjusting constraint weights or refining the LLM to better reflect trends in the data or user tasks.

5 Task 2.2

In Task 1, you found out how constraint solvers can help us in recommending visualizations. Can you find other examples or provide your own ideas of how constraint solvers or other logic programming techniques can be combined with LLMs?

5.1 Combining logic programming and LLMs

To answer this question, I once again researched and read a couple of scientific papers. I managed to find a few interesting examples, which I will describe below.

In the first paper I came across, titled "Combining Constraint Programming Reasoning With Large Language Model Prediction" [RDB24], the authors merge the strengths of Constraint Programming (CP) and Large Language Models (LLMs) to tackle text generation tasks where sentences must satisfy precise structural constraints (for example: fixed length, forbidden words). Their approach, called GenCP, allows a constraint solver to handle logical and numerical constraints while utilizing an LLM to provide the "meaning" side. As the CP solver enforces all hard rules (like exact word counts or mandatory keywords), the LLM dynamically proposes contextually relevant words. Together, they produce texts that satisfy the requirements while still being in an understandable format, demonstrating how LLMs and constraint solvers can effectively work together for constrained text generation.

Another unconventional way of combining logic programming and LLMs is described in the paper "Coupling Large Language Models with Logic Programming for Robust and General Reasoning from Text" [YIL23]. The authors combine Large Language Models (LLMs) with Logic Programming, specifically Answer Set Programming (ASP), to tackle general reasoning tasks without retraining for each new benchmark. The LLM parses natural language statements into logical facts, while an ASP solver provides declarative, explainable reasoning over these parsed facts. By separating semantic parsing from logic-based inference, the method achieves state-of-the-art results on multiple tasks such as bAbI, StepGame, CLUTRR, and gSCAN.

References

[Dib23] Victor Dibia. “LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 113–126. DOI: `10.18653/v1/2023.acl-demo.11`. URL: `https://aclanthology.org/2023.acl-demo.11`.

[Hu+19] Kevin Hu et al. “VizML: A Machine Learning Approach to Visualization Recommendation”. In: *Proceedings of the 2019 Conference on Human Factors in Computing Systems (CHI)*. ACM, 2019.

[RDB24] Florian Régin, Elisabetta De Maria, and Alexandre Bonlarron. “Combining Constraint Programming Reasoning with Large Language Model Predictions”. en. In: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. DOI: `10.4230/LIPICS.CP.2024.25`. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2024.25`.

[YIL23] Zhun Yang, Adam Ishay, and Joohyung Lee. *Coupling Large Language Models with Logic Programming for Robust and General Reasoning from Text*. 2023. arXiv: `2307.07696 [cs.CL]`. URL: `https://arxiv.org/abs/2307.07696`.

[Yuy+18] Yuyu et al. “DeepEye: Towards Automatic Data Visualization”. In: *2018 IEEE 34th International Conference on Data Engineering*. 2018.