Analytical Characterization of Instruction-Tuning Datasets Using Supervised and Unsupervised Machine Learning Approaches

Robert Horton rhorton@win-vector.com

John Mark Agosta jmagosta@win-vector.com

Alexandre Vilcek avilcek@win-vector.com

Abstract

Here we describe our general approach to characterizing a collection of instruction-tuning examples, and apply it to the open-source Databricks Dolly2 dataset. Using an iterative process of clustering, manual examination, and semi-automated curation, we have identified subcategories of instructions at a variety of granularities. We set out to characterize instruction/response pairs on three broad dimensions: the subject domain to which the instruction pertains, the syntax of the way the instruction is phrased, and the format in which the results are requested. Careful dissection of the Dolly dataset allows us to identify hierarchical groups of instructions that can be considered similar or dissimilar in a variety of aspects. This kind of characterization is necessary to enable systematic monitoring of the performance of instruction-tuned language models on narrowly defined subsets of instruction types described by multiple aspects. Such detailed monitoring will be crucial for detecting specific weaknesses in the performance of tuned models to facilitate targeted augmentation of the training sets, or other approaches to remediation of the models.

1 Introduction

The ability to follow instructions is fundamental to the current success of conversational large language models (LLMs) such as *chatGPT*. Foundational LLMs were trained on enormous unlabeled corpora, where they gained the ability to generate text effectively indistinguishable from human generated text. Fine tuning for instruction following enables these models to respond precisely to queries and commands to generate plausible and (hopefully) appropriate responses. Methods as well as datasets are being rapidly developed for these purposes.

Descriptive features of data are widely used to find weakness in model performance, and to monitor the extent to which remediation efforts can address these weaknesses (Nushi et al. [2018]). "Error Analysis" is an important and widely used approach in which an interpretable model is used to explain the errors made by a powerful deep learning model (Singla et al. [2021]). Interpretable models require interpretable features; our focus here is on developing such features, and in that context this work describes a range of both classical and novel techniques for engineering features that should be useful for explaining LLM instruction performance.

Our contribution in this paper is to demonstrate various approaches to engineering or discovering interpretable features of instruction tuning examples. These approaches range from straightforward predictions made directly from the text (e.g., complexity score), to more complicated features derived

Preprint. Under review.

using weak supervision of clustering over different kinds of embeddings. Specifically we show how an analog of conventional ROC analysis together with weak supervision gives insights not attainable from unsupervised clustering metrics alone.

1.1 The Dolly Dataset

While several well known instruction-tuning datasets are now openly available, we have chosen to study the Dolly dataset (Conover et al. [2023]) because it is relatively small, simply structured, and specifically developed to be open source so that it can be freely used for any academic or commercial purpose. However the general approaches used should be applicable to any such dataset. The dataset developers have labeled each instruction as belonging to one of seven categories. The first three categories include a context passage (e.g., the text to summarize), while the remaining categories do not have separate context passages, either because all the information is included in the instruction, or because they focus on general knowledge:

- *summarization*: instructions related to expressing the main point of the provided context passage in more concise form.
- *information_extraction*: specification of particular items of data to find in the context passage.
- *closed_qa* : questions to be answered based on the information contained in a context passage of reference text.
- *classification* : For this task, annotators were asked to make judgments about class membership (e.g. are the items in a list animals, minerals or vegetables) or to judge the properties of a short passage of text, such as the sentiment of a movie review.
- *brainstorming* : open-ended idea generation
- *creative_writing* : This task would include things like writing a poem or a love letter.
- *open_qa* (or *general_qa*): questions to be answered from general knowledge, without a specific passage of context. The use of two different labels for this category seems to be an error in the dataset.

Note that the existing category labels are extremely broad; our primary tasks here are to try to identify more granular subcategories of instructions, and to begin to characterize a set of aspects by which instructions may be considered similar or different from one another.

The present work largely focuses on analysis of the instruction and response fields, though similar characterization could be done for the context field as well.

2 Methods

Cases in the Dolly dataset consist of a category field that roughly describes the kind of query, and text fields for instruction, response, and, for certain categories, context. Here we describe a variety of approaches for discovering and engineering additional interpretable features of each case. Code used in this paper is available in our repository on GitHub [1]

2.1 Semantic embedding

Semantic embeddings were generated for each instruction, response, and context passage using the `all-MiniLM-L6-v2` SentenceTransformer model (Reimers and Gurevych [2019]). This model generates relatively small embedding vectors (384 dimensions) which are convenient for logistical and performance reasons while we develop techniques, but the approaches we discuss here should all work with longer vectors as well.

2.2 Latent semantic analysis focused on long n-grams

N-gram vectors were computed as follows, using scikit learn functions from `feature_extraction.text`: Term counts were computed using ngram lengths between 5

[1] `https://github.com/rmhorton/instruction_tuning`

and 12, retaining only terms with a count above a threshold of 25. These counts were used to compute term frequency - inverse document frequency (TF/IDF) vectors, which were then reduced to 100 dimensions using PCA. Similarly, ***POS n-gram vectors*** were computed by repeating this process after first replacing the words in the text with their corresponding parts of speech using spaCy (Honnibal et al. [2020]) [2], so "Do aliens exist?" becomes "aux noun verb punct".

2.3 Clustering

For a given embedding, we compute a matrix of pairwise distances using `scipy.spatial.distance.pdist` with the 'cosine' metric, and use that to construct a dendrogram using `scipy.cluster.hierarchy.ward`. This dendrogram was successively sliced using a series of log 2 spaced distance thresholds. The resulting cluster partitions are named using a capital letter 'A' to denote the clusters using the largest distance threshold (giving a small number of large clusters), and sequential capital letters of the alphabet to indicate successively smaller thresholds. The hierarchical nature of the dendrogram is retained in these sliced partitions such that every 'B' level cluster is entirely contained within an 'A' level cluster, and each 'A' level cluster is partitioned completely into a set of 'B' level clusters, and so forth. To put similar items close together on a list, we simply sort them by 'A' cluster, then 'B', cluster, then 'C' cluster, etc. Examples of clustered and sorted instructions are shown in Figures 1 and 2.

2.4 Text patterns

For each case we create a collection of binary labels using regular expressions that identify word patterns for particular aspects, such as the subject domain the question is about, or the form of the question revealed by its syntax. These labels are 'soft' in the sense that they are fairly crude approximations of the underlying aspects of interest. Examples are shown in Figure 5.

2.5 Co-occurrence between cases, clusters, and labels

Co-occurrence between two attributes is a tabulation of how often the possible values of one attribute are found together in the same record with each possible value of the other attribute. The simplest way to compute this in Python is by cross tabulation, which generates a matrix we can plot as a clustermap. This implementation is so concise that we provide the essential code used to generate Figure 3 here:

```
import pandas as pd
# Cross-tabulate clusters and labels
xtab = pd.crosstab(dd['category'], dd[cluster_col])
# Divide the cols by their sums, so each column shows
# the distribution of categories with a cluster.
xtab_norm = xtab.div(xtab.sum(axis=0), axis=1)
# Plot clustermap
import seaborn as sns
sns.clustermap(xtab_norm, figsize=(20, 4), dendrogram_ratio=(.05, .2), cbar_pos=None)
```

While cross-tabulations are simple to compute and display, they do not scale well to large numbers of categories, and the matrix visualization cannot relate more than two kinds of things at a time. To study more complex and more fine-grained relationships we turn to graphs.

To quantify co-occurrence frequency, we borrow the term *confidence* from market basket analysis; this is the frequency with which instructions that have a given feature also have a second feature. Features in this case can be any categorical attribute of the instruction/response pair, such as the category labels already present in the dataset, cluster IDs we have assigned through various clustering approaches, flags from our regex patterns, etc. Note that confidence is a directional metric; the probability that an instruction in the classification category will also belong to a particular cluster is not the same as the probability that an instruction in the cluster will also be in the classification category. These directional edges are indicated by arrows in the visualization (see Figure 4).

[2] `https://spacy.io/`

We have implemented the computation of these statistics in SQL so that it will be portable to common database systems (including Spark). For local implementation in Python, the function simply wraps the underlying SQL query in a call to a sqlite in-memory database.

Computation and visualization of the graphs is done with our ThoughtGraph library [3], which formats the data for use with the visjs network visualization library [4] and adds a slider to interactively set a filtering threshold for edge weights, so that weaker edges can be removed to reveal the stronger connections.

2.6 Analysis of label distribution across clusters

To characterize the clusters we use the soft-labelings provided by the text patterns. Clusterings that capture a soft-label well will concentrate instances of that aspect in a few clusters compared to clusters that are more randomly distributed.

The quality of the clusters is measured by their purity. Purity is a function of the fraction f of labelled cases in the cluster, defined as $p(f) = \min(p, 1-p)$. As the fraction of labels reaches an extreme the purity increases. Entropy, $h(p) = -2f(p)\log(f(p))$, is a differentiable function of p that properly emphasizes differences at extremes over the mid-range. We use entropy to quantify the degree of concentration of the aspect pattern across the clusters by comparing the cluster size-weighted entropy of the clustering with the entropy over all cases considered as a single group. Any clustering will decrease entropy. The decrease in entropy of the distribution of a binary label (a positive number; more is better) is a supervised measure of the quality of the clustering.

We have derived a novel evaluation method for visualizing cluster quality, based on classical Receiver Operating Characteristic (ROC) curves that extends our entropy measure. For each cluster the fraction of positive and negative items out of all positives and negatives for that labelling are computed. Then TPR (positives) and FPR (negatives) are computed for each cluster as a running sum of these fractions, and plotted to form the characteristic concave ROC curve for the clusters. Furthermore the AUC for this curve is a monotonic, concave function of the clustering entropy. More importantly, the location of a cluster along the curve indicates that cluster's effect on the AUC, in addition to its ranking by purity. Note that the clustering method is unaware of the labels; we are not clustering on the labels, only on the embedding vectors. However the labels' distribution across different clusterings suggests the semantic meaning of the clusters.

An ROC curve typically shows how well positive and negative cases are separated by the scores of an ML model. We adapt this approach to show how well clustering separates the positive and negative cases by using the mean frequency of positive labels in each cluster as the predicted score for each case in the cluster. Typically an ROC curve has one diagonal segment for each unique score in the test set; we modify the plotting slightly so that each cluster receives its own segment, even when multiple clusters have the same score, and add marks on the plot to show the individual segments. This makes it easy to see the difference between a large cluster and multiple small clusters with the same score.

2.7 Estimating text complexity

Alongside the composite features introduced by the clusterings, we can consider other features derived directly from the text. Text complexity can be assessed from various angles, including readability and reading comprehension. Our focus here leans towards reading comprehension, which gauges the difficulty of an entity (e.g., humans or LLMs) to comprehend, reason, and answer questions based on text passages.

To compute text complexity metrics, we used the py-readability-metrics package, selecting the Gunning Fog index (Wikipedia authors [2023]). This index was found to have the highest correlation with reading comprehension levels in our curated RACE dataset (Lai et al. [2017], Liang et al. [2019]). We then computed Gunning Fog readability scores for the Dolly dataset, specifically emphasizing the `closed_qa` instruction category.

[3] https://github.com/rmhorton/ThoughtGraph

[4] https://github.com/visjs/vis-network

instruction	category	inst_A	inst_B	inst_C	inst_D	inst_E
Who are U2	open_qa	A01	B01	C002	D0006	E0015
Who does what in U2?	information_extraction	A01	B01	C002	D0006	E0015
Where do U2 come from?	information_extraction	A01	B01	C002	D0006	E0015
What type of music do U2 make?	information_extraction	A01	B01	C002	D0006	E0015
What city in Ireland is the band U2 from?	closed_qa	A01	B01	C002	D0006	E0016
Tell me which of these are great Irish Bands: The Hot House Flowers, The Waterboys,The Proclaimers, Deacon Blue, U2, Westlife,	classification	A01	B01	C002	D0006	E0016

Figure 1: Examples of instructions clustered by semantic similarity and sorted to put similar items close together. These instructions are all about the band U2, but they have some variety in structure and intent. The last two sentences are in a different E-level cluster; note that these sentences mention Ireland/Irish where the others do not.

instruction	category	inst_A	inst_B	inst_C	inst_D	inst_E
Identify which instrument is string or woodwind: Agiarut, Piccolo	classification	A19	B89	C408	D1819	E6884
Identify which instrument is string or woodwind: Diplica, Kontra	classification	A19	B89	C408	D1819	E6884
Identify which instrument is string or woodwind: Giga, Panflute, Chuniri, Dizi	classification	A19	B89	C408	D1819	E6884
Identify which instrument is string or percussion: Chime bar, Pinaka vina	classification	A19	B89	C409	D1820	E6885
Identify which instrument is string or percussion: Den-den daiko, Luc huyen cam	classification	A19	B89	C409	D1820	E6885
Identify which instrument is string or percussion: Conga, Ninera	classification	A19	B89	C409	D1820	E6885

Figure 2: These sentences are very similar in structure as well as content, and they cluster very tightly together. Examples like these illustrate why we were motivated to focus on longer n-grams, or n-grams of parts of speech, to try to find representations that emphasize syntactic structure.

3 Results

3.1 Clustering

Clustering on semantic embeddings and sorting by cluster names over a hierarchy of granularities let us read through the instructions with similar examples being put close together on a list. Figures 1 and 2 show examples.

Examining these clustered examples led us to propose three general dimensions by which instances may be considered similar or different:

- *domain*: The domain dimension is the topic or knowledge domain an instruction applies to (like cars or musical instruments).
- *framework*: The degree to which instructions share syntactic structure (these are things that could be put into a template).
- *output format*: Parts of the instruction specifying how the results should be presented.

Examples of regular expressions designed to soft-label cases related to these dimensions are given in Figure 5. These dimensions are by no means exhaustive (for example, another common way to characterize instructions is by the NLP task required to be solved), but we found them useful as a rough framework.

3.2 Co-occurrence

Figure 3 was generated by cross-tabulating clusters with categories for two different kinds of clusters (semantic embedding based clusters in panel A, and POS n-gram based clusters n panel B).

The left panel of Figure 4 is a high-level view of a co-occurrence graph showing all the common clusters of entries in the instruction and response fields and how they relate to the categories. Because

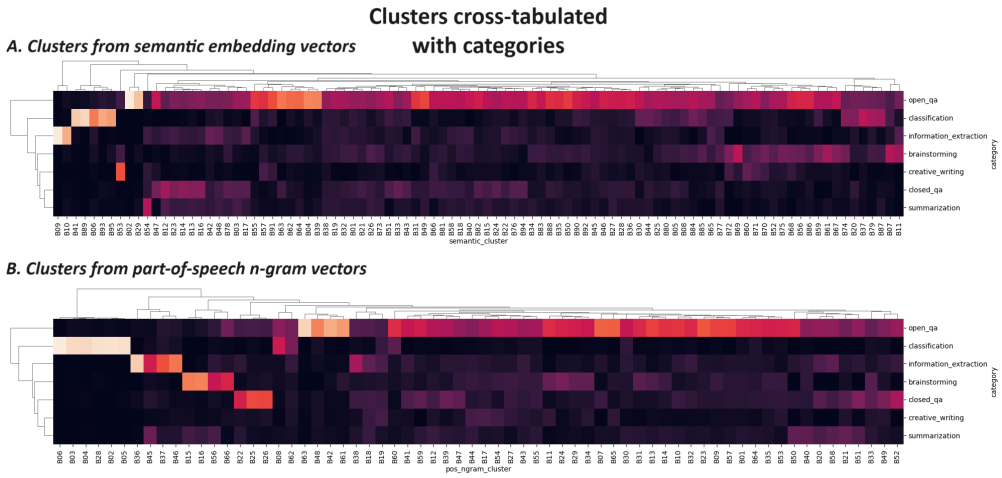


Figure 3: Clustermaps of cross-tabulations between clusters and category labels. Panel ***A*** shows results with clusters generated from *semantic embeddings*, and panel ***B*** shows clusters generated from *POS n-gram vectors* computed from part-of-speech n-grams. Counts were normalized by dividing by the column sum, so each column shows how the items in a cluster are distributed across categories. Note that though these two ways of clustering instructions use similar naming conventions ('B' indicates the level of granularity, so these clusters are smaller and more focused than 'A' level clusters, but more general than 'C' level), but the clusters in the two panels were computed from completely different representations and the column names (x-axis labels) do not correspond between panels. It is clear from these visualizations that some of the POS n-gram clusters show even stronger relationships to particular categories (eg, classification) than the clusters derived from semantic embeddings.

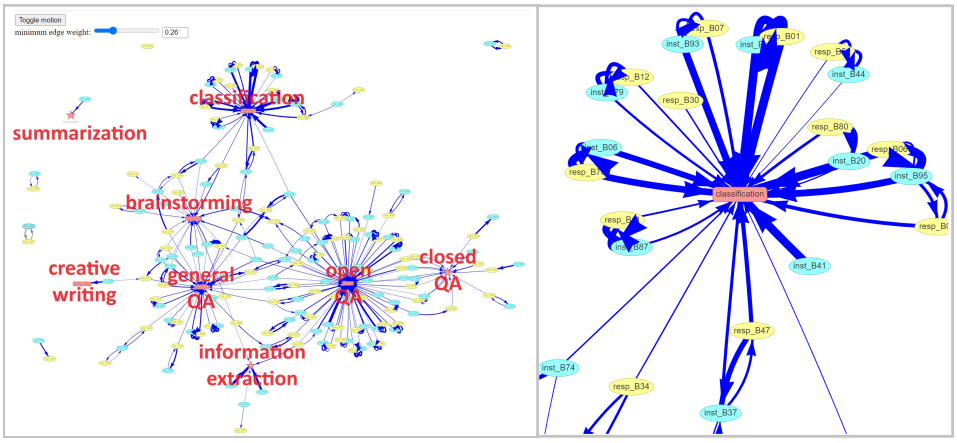


Figure 4: **Co-occurrence graph between category labels, instruction clusters, and response clusters.** *Left Panel:* Wide-angle view. Nodes representing category labels are in red, with those associated with context passages (`closed_qa`, `summarization`, and `information_extraction`) using star shapes and the other categories using rectangles. Instruction clusters are shown in light blue, and response clusters are in yellow. *Right panel:* A zoomed-in view of the co-occurrence graph showing clusters related to `classification`, showing clear subcategories characterized by both instruction and response clusters.

name	regex	dimension
animals	\b(animal|cat|dog|pet)	domain
sports	\baseball(?!: bat)|basketball|badminton|tennis(?!: shoe)|soccer|futbol|football (?!: stadium)	domain
science_fiction	science fiction	domain
alternatives	(,|\b(or)\b)	framework
output_format	(display|the results|as a (bulleted)?list|list them|format this|format them|in the format|each record|new line|comma separated|separated by|separate them with a comma|JSON|XML|markdown)	output format

Figure 5: Examples of regular expression patterns used for identifying soft labels. Each pattern has a name that is suitable for use as a Pandas column header. The patterns may include positive or negative lookaround assertions so we can specify, for example, that we want it to match 'baseball' but not 'baseball bat' because we noticed cases where baseball bats were used for purposes unrelated to baseball. Each pattern is labeled with a 'dimension' indicating whether it is intended to identify characteristics of syntactic frameworks, specifications of output format, or the topical domain of the instruction.

the text is too small to read at this magnification, we have indicated the positions of the nodes representing category labels. Controls are visible in the upper left corner; the button can be used to turn off the physics engine driving force-directed layout, and the slider can be used to select a threshold for filtering out weaker edges. At the edge threshold shown, the nodes for the *summarization* and *creative_writing* labels each have only one edge connecting them to a cluster node. However, most of the labels form centers of communities, with edges coming in from multiple instruction and response clusters.

The strongest edge on this graph (from 'inst_B02' to 'open_qa') has a confidence just over 0.98; that is, more than 98% of the instructions in that cluster have that category label. The threshold slider is set to 0.26 so the thinnest lines in the figure are just above that strength. There are multiple edges with confidence above 95% pointing into the *classification* node. Note that the stronger arrows typically go from the less common node to the more common one; since there are only 8 different category labels and every instruction/response pair has a label, the label nodes cover much larger numbers of examples than any of the clusters do.

In the right hand panel of Figure 4 we zoom in on the clusters associated with the **classification** label. The clustering process was performed independently on the **instruction** (blue nodes) and **response** fields (yellow nodes). Labels are shown in red. Note that most of the relationships between instruction and response nodes have edges pointing in both directions, meaning that a high fraction of the instruction/response pairs where the instruction falls into that particular 'inst' cluster also have a response that falls into the indicated 'resp' cluster, and vice versa. A pattern particularly prominent in this view is the association of both elements of instruction/response node pairs with the same category. These indicate clear subcategories. These kinds of patterns are recognizable in other categories, but they are most clear for classification.

3.3 Examples of patterns found in clusters

Figure 5 shows examples of regular expressions that attempt to capture patterns we noticed when examining clusters.

The 'alternatives' pattern provides an example of how we used the clustering results, examination of examples, and some experimentation with patterns to figure out a signal that helps to explain the clustering.

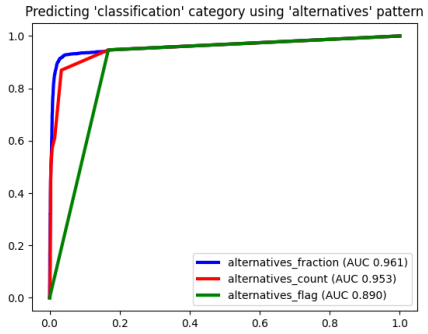


Figure 6: *Using the `alternatives` pattern in instruction text to predict the `classification` label.* The regex pattern we have named 'alternatives' recognizes a comma or the word 'or'. If we use a binary flag indicating the presence of this pattern as a predictor for the `classification` label we get the two-segment ROC curve shown in a the green line. If we instead use the count of the number of times the pattern appears in the text as a predictor, we get the ROC curve shown by the red line. And if we take the ratio of this count to the length of the instruction text, we get even better predictions, as shown by the blue curve.

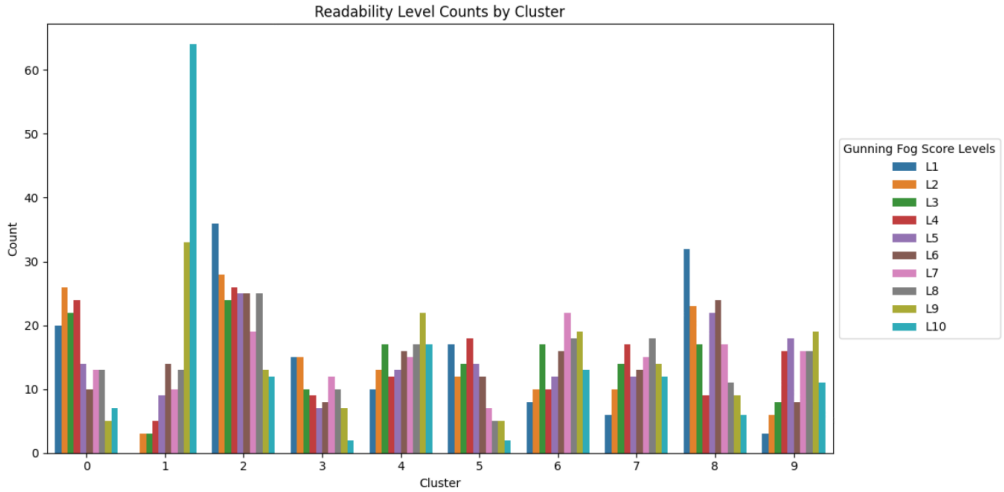


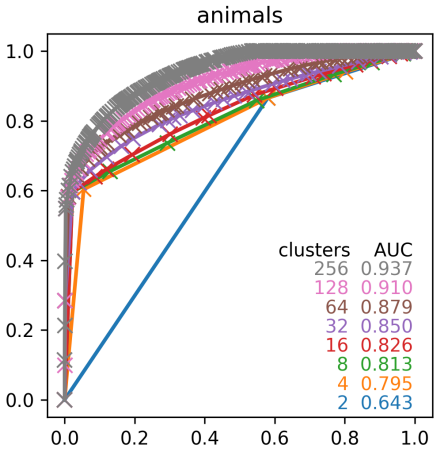
Figure 7: *Distribution of readability levels across clusters*. Several of these clusters show obvious skew for different readability levels. Cluster 8 focuses on texts detailing geographical facts written in straightforward language, many of which have the lowest readability scores. Cluster 2 primarily covers biographical information about individuals, similarly presented in simple language with many also ranking at the lower readability spectrum. In contrast, Cluster 1 features texts on science and technology, composed in a sophisticated style, and often delves into intricate technical and scientific concepts, resulting in strong skewing toward the highest readability scores.

As we saw in panel B of Figure 3, several of the POS n-gram clusters are composed almost entirely of classification instructions. To explore this pattern further we used the raw POS n-gram features to predict the classification category with a single decision tree. Most of the signal was found by a simple tree of depth two that only looked at two n-gram patterns; 'punct propn punct propn' and 'punct noun punct noun'. These are either two nouns or two proper nouns followed by punctuation marks. This led us to suspect that the relevant signal is a list of items, so the alternatives pattern looks for markers used to separate items in such a list. Figure 6 shows how well simple scores based on this pattern serves to predict the classification category.

3.4 Text complexity

We examined the distribution of readability scores across different sentence embedding clusters within the *closed_qa* instruction category (see Figure 7), observing that clusters within this dataset can differ widely on this metric.

Figure 8: A "Cluster ROC curve" showing how the frequency of the label for the 'animals' pattern is distributed across clusters at various granularities, Semantic embeddings were used to generate a hierarchical clustering dendrogram, which was sliced at a series of levels to produce numbers of clusters from 2 to 256 on a log2 scale. An ROC curve and associated AUC are computed for each slicing, as indicated by colors. Since each cluster is represented by a distinct segment of the curve, we can see that even in the gray curve (256 clusters) a small number of clusters contain the majority of the cases matching the 'animals' pattern, and a very large fraction of the cases in those clusters match the pattern.



3.5 Label distribution across clusters

Figure 8 shows a "cluster ROC curve", which is a new twist on an old type of plot, showing the distribution of a binary label across clusters.

4 Discussion

We have demonstrated in an instruction tuning dataset a variety of approaches for discovering and engineering interpretable features that can support error analysis of models trained or tested with this data.

Once you teach an LLM to follow instructions, you can instruct it to generate more instruction-tuning data (Wang et al. [2023]) which can in turn be used to improve the model, analogous to a RepRap (Jones et al. [2011]) for language models. Careful measurement of performance across a wide range of granular instruction categories will certainly be important for monitoring, and most likely for guiding this kind of process.

The label-based ROC analysis described in this paper is not limited to use with clusterings; it could be applied to any process that results in a partition of examples (such as multiclass classification). True positive and false positive rates can be calculated describing the distribution of labels within each partition, and each partition will be represented by a segment of the resulting ROC curve.

References

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. Free dolly: Introducing the world's first truly open instruction-tuned llm, 2023. URL `https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm`.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python. 2020. doi: 10.5281/zenodo.1212303.

Rhys Jones, Patrick Haufe, Edward Sells, Pejman Iravani, Vik Olliver, Chris Palmer, and Adrian Bowyer. Reprap – the replicating rapid prototyper. *Robotica*, 29(1):177–191, 2011. doi: 10.1017/S026357471000069X.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations, 2017.

Yichan Liang, Jianheng Li, and Jian Yin. A new multi-choice reading comprehension dataset for curriculum learning. In Wee Sun Lee and Taiji Suzuki, editors, *Proceedings of The Eleventh Asian Conference on Machine Learning*, volume 101 of *Proceedings of Machine Learning Research*,

pages 742–757. PMLR, 17–19 Nov 2019. URL https://proceedings.mlr.press/v101/liang19a.html.

Besmira Nushi, Ece Kamar, and Eric Horvitz. Towards accountable ai: Hybrid human-machine analyses for characterizing system failure, 2018.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

Sahil Singla, Besmira Nushi, Shital Shah, Ece Kamar, and Eric Horvitz. Understanding failures of deep networks via robust feature extraction, 2021.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions, 2023.

Wikipedia authors. Gunning fog index — Wikipedia, the free encyclopedia, 2023. URL https://en.wikipedia.org/wiki/Gunning_fog_index. [Online; accessed 4-October-2023].