StackExchange Prediction and Analysis

ADM stackexchange project

Mateusz Mazur, Tomasz Makowski, Tomasz Kawiak

November 18, 2025

Table of contents

EDA

Overview

- As of now, we have collected a dataset containing 100,000 questions from StackExchange (specifically StackOverflow), along with their associated metadata.

- As stated earlier, the dataset has the following features:

#	Column	Dtype
0	title	object
1	has_accepted_answer	bool
2	accepted_answer_score	float64
3	time_to_accepted_answer_hours	float64
4	question_score	int64
5	question_text	object
6	num_tags	int64
7	tags	object
8	accepted_answer_id	float64
9	accepted_answer_length_chars	float64
10	accepted_answer_length_tokens	float64

Deduplication

- Out of these questions, 8 were dropped for being duplicates.

Acceptance Analysis

- From those remaining, 39938 questions have an accepted answer, while 60054 do not.
- Interestingly, out of those accepted answers, only 12000 have `time_to_accepted_answer_hours` defined (i.e., non-null values).

Time to Accepted

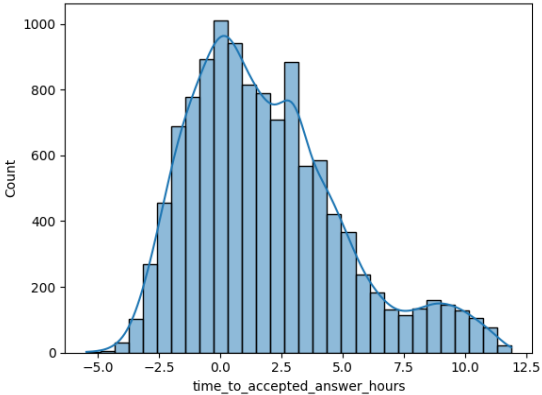

Figure 1: Histogram of log of time to accepted answer

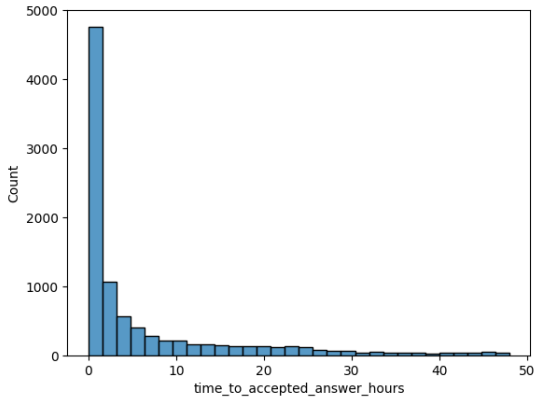

Figure 2: Histogram of time to accepted answer with time < 48

Tags

- The dataset contains a total of 7684 unique tags. The most common being:

Tag	Count
python	1528
c#	746
javascript	703
c++	689
java	592

Extreme Multi-Label Classification

- Since we are dealing with such a vast number of different tags, we propose the following approaches to investigate:

1. Frequency-Based Filtering
   - Aside from the top `N` most common tags, one approach could limit the scope to a subset of tags relevant to our analysis – e.g. **top N programming languages**
2. "Semantic Clustering":
   - Use pre-trained embeddings to represent question descriptions in a vector space.
   - Apply clustering algorithms to group similar questions together based on their embeddings.

Tags EDA

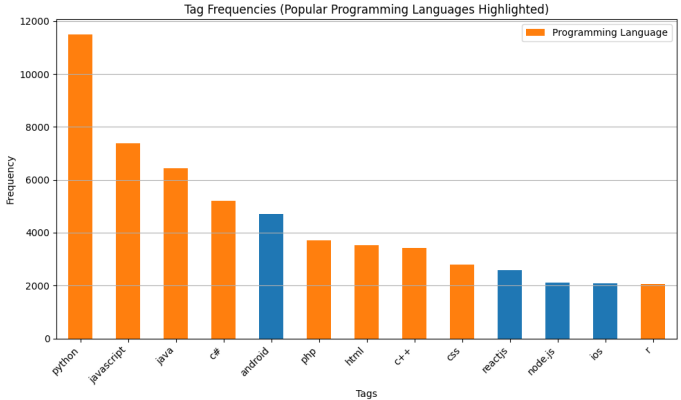

Figure 3: Tag Frequency

We filtered the tags using the list of programming languages (from Wikipedia) and selected the most frequent ones:

```
['python', 'javascript', 'java', 'c#', 'php', 'html', 'c++', 'css', 'r']
```

Key observations about the resulting subset:

- **Subset size:** 42,037 questions
- **Class balance:** The distribution across these languages is relatively balanced (see last plot), and can be further balanced if needed.
- **Multi-label cases:** Some questions are tagged with multiple programming languages.

Distribution of programming language tags per question:

# Tags	# Questions
1	38,547
2	3,015
3	464
4	11

Score

- Finally, we analyzed the distribution of the scores of the questions in the dataset (already of the selected programming-language-tagged subset).

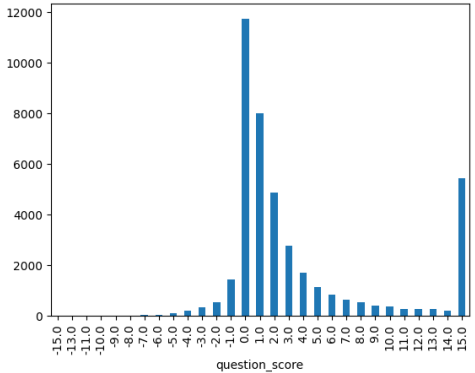

Figure 4: Question Score Histogram

Based on the hisogram, we proposed the following score classes:

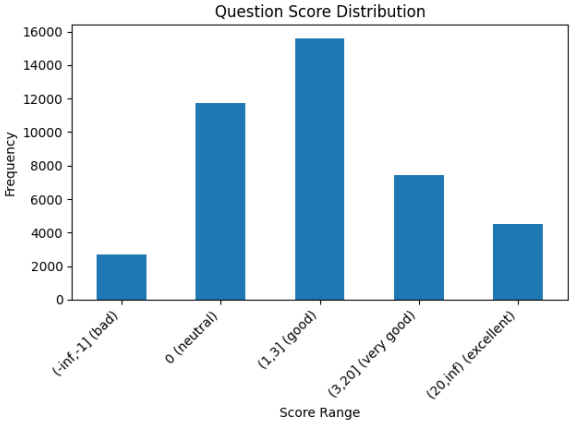

Figure 5: Question Score Distribution

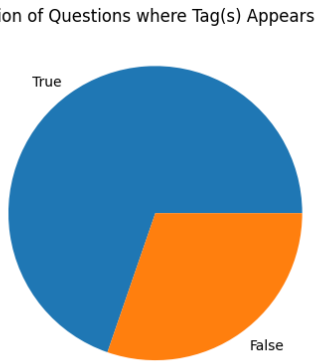

Figure 6: Proportion of Questions where Tag(s) appears in text

Conclusions

- the data about the accepted answer is provided for only small fraction of the data.
- there are too many tags to perform the classification using all of them
- however, we can choose a certain subset (e.g. most popular programming languages) and focus on it
- we can also try to predict the class of the score of the answer (integer score mapped to classes of uneven frequencies)

TAG Embeddings

Initial problem

- Having 22753 unique tags poses a significant problem for our dataset,
- each questions can have multiple tags (for example `python`,`pandas`,...)

Embeddings

- We've decided to embed tags, as well as question texts using `qwen3-embedding:8b` model, translating each tag into a 4096 vector.
- This resulted in a dataframe of shape 22753x4096 vector, which introduced more issues:
  - Lack of memory to process this data (the dtype is `f64`)
  - Too large dimensions

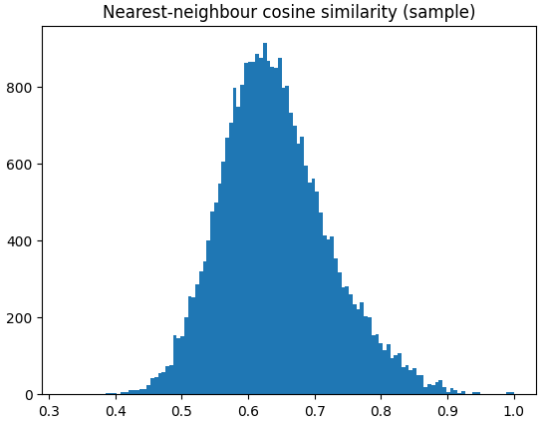

Figure 7: Question embeddings cosine similarity.

Dimensionality reduction

- Initial approach considered `HDBSCAN` (Hierarchical Density-Based Spatial Clustering of Applications with Noise), but due to memory required to perform this operation (over 64GB of RAM)
- Later on we've moved to a `Birch` algorithm, with `HDBSCAN` as well as `AgglomerativeClustering` which got unsatisfactory results

UMAP

- After reading forums, we've decided to try `UMAP` (Uniform Manifold Approximation and Projection) which sounded ideal for our problem, since it can be used for general non-linear dimension reduction.

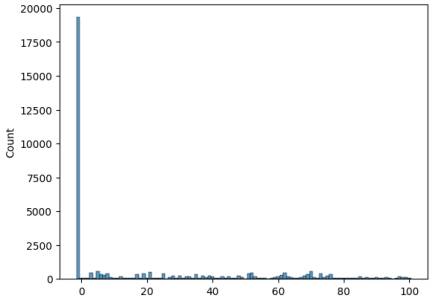

Figure 8: Recurring problem during UMAP and other clustering algorithms

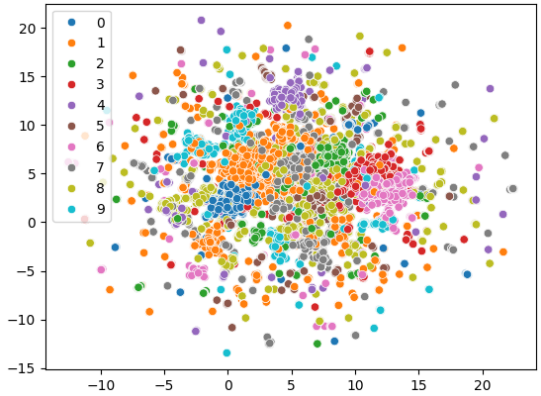

Figure 9: Naiive UMAP reducing tag embeddings to 2 components.

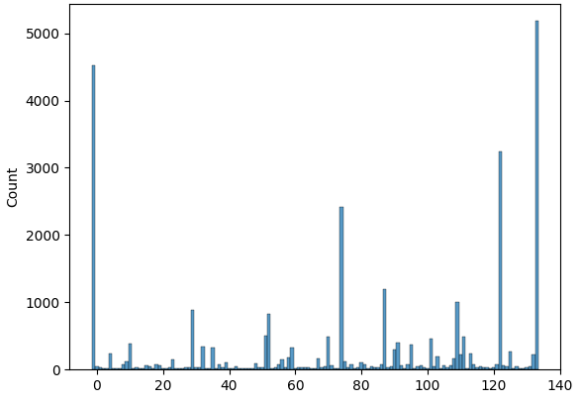

Figure 10: Histogram of labels resulting from UMAP reduction coupled with HDBSCAN

Optuna

- A hyperparameter optimization framework which we hoped would find optimal parameters for our dimensionality reduction task.
- We need a score which we can maximize/minimize during dimension reduction:
  - Caliński-Harabasz index
    - ratio of the between-cluster separation to the within-cluster dispersion, normalized by their number of degrees of freedom.
  - Silhouette score
  - Cluster persistence
    - stability of each cluster, indicating well-defined grouping

UMAP Optimization

- UMAP coupled with HDBSCAN
- Optimized hyperparameters:
  - `umap_n_components`
  - `umap_n_neighbors`
  - `umap_min_dist`
  - `umap_metric`
  - `hdb_min_cluster_size`
  - `hdb_min_samples`
  - `hdb_cluster_selection_method`
  - `hdb_cluster_selection_epsilon`

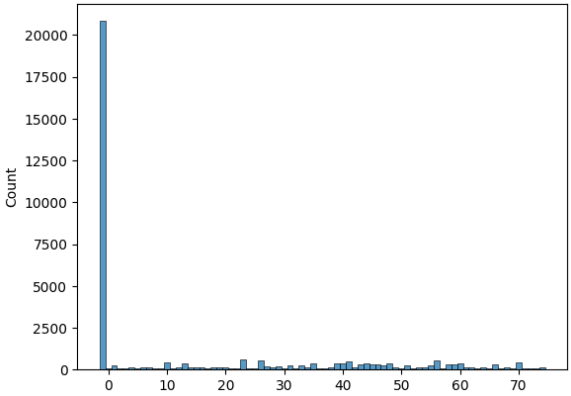

Figure 11: Results of optuna training

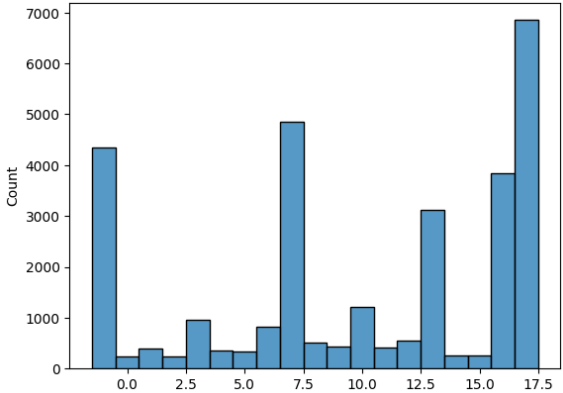

Figure 12: Results of Optuna optimization of only HDBSCAN hyperparameters search ran on UMAP reduction to 5 components

Conclusions

- We have to find better approach
- Possible culprits:
  - For UMAP to work as intended ‘The data must be uniformly distributed on Riemannian manifold’
  - additionally ‘The Riemannian metric is locally constant (or can be approximated as such)’
    - embeddings generally create spaces with varying local geometry depending on the semantic density of the training data.
  - Too few optimization runs.

Recursive Embedding and Clustering

- Based on Spotify's blog.

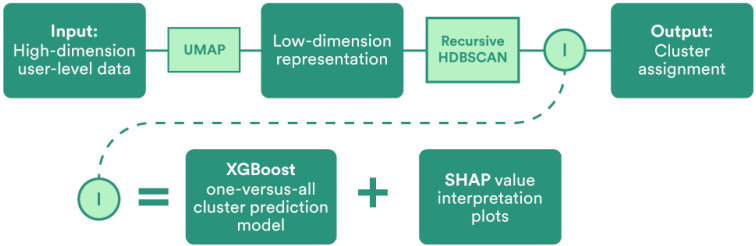

Figure 13: Diagram representing the complete recursive embedding and clustering process. Source

Recursive KMEANS clustering

- Since properly implementing HDBSCAN in the context of our problem is still a standing question, we've decided to use `KMEANS`.
- Initially, we've specified the clustering hierarchy structure `100,100`.
- After finding those centroids, we've taken their mean embeddings and selected 'most descriptive' tags for each cluster (based on cosine/euclidean metric).
- euclidean metric worked 'better' in the case of KMEANS.

	cluster_mean	top_5_popular_tags
facebook	[0.0052028694, -0.002167758, -0.017349796, -0....	[instagram-api, instagram, instagram-graph-api...
pyobject	[0.004231446, 0.0026400657, -0.0142704435, -0....	[python, pandas, jupyter-notebook, jupyter, py...
google-workspace-add-ons	[0.022996485, 0.0034478805, -0.014011028, -0.0...	[google-merchant-center, google-signin, google...
.net	[0.008871326, 0.00042595304, -0.003047627, -0....	[c#, wpf, .net, blazor, asp.net-core]
coverflow	[0.018253814, 0.0024098884, 0.0059977337, -0.0...	[background-color, css, tailwind-css, css-posi...
azure	[0.02082736, 0.0026177948, 0.0014911512, -0.00...	[azure-pipelines, azure, azureservicebus, azur...
importerror	[0.017333947, 0.00064369285, 0.0025193274, -0....	[segmentation-fault, try-catch, connection-ref...

Figure 14: Initial results

Recursive Spherical KMEANS clustering

- Since spherical KMEANS is better suited for cosine similarity and for text data in general, we've decided to try it as well. Based on a Accelerating Spherical k-Means papaer [1].

	cluster_mean	top_5_popular_tags
facebook	[0.004879954, -0.0019050128, -0.016777342, -0....	[instagram-api, instagram, instagram-graph-api...
integer	[0.027635492, 0.0065359636, -0.0029253308, -0....	[intervals, differential-equations, math, maxi...
storage	[0.008282, -0.016636355, -0.0092419945, -0.004...	[dicomweb, remotestorage, storage, multipartfo...
tf-idf	[0.027263727, -0.007913517, 0.0023698097, -0.0...	[wpml, locale, dictionary, spacy, microsoft-tr...
qt	[0.01614129, 0.005696066, 0.016760174, -0.0084...	[qt, qml, pyqt, qasync, qt6]

Figure 15: Sample results

Results

- To verify the quality of new clusters, we've used a very basic predictor based on the dot product of normalized question embeddings and tag cluster embeddings.
  - We've taken 2 most similar clusters for each question.

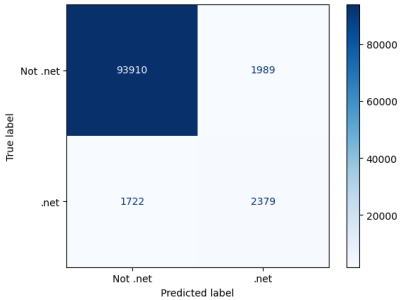

Figure 16: Results of top 2 results for .net

	tag1	tag2
count	780	780
unique	780	780
top	(mongodb, 0.5270306638492696)	(hyperfilesql, 0.4293950990405074)
freq	1	1

Figure 17: Most popular results for mongodb

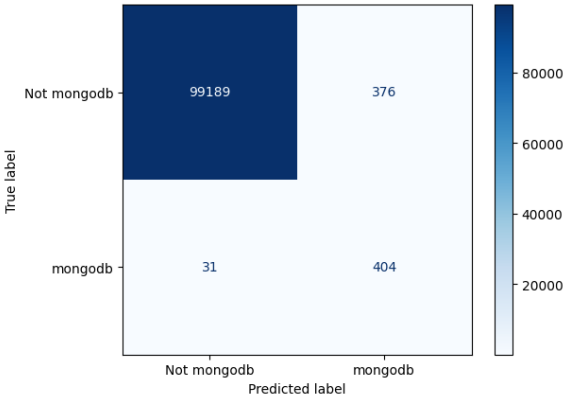

Figure 18: Results of top 2 results for mongodb

Thank you for your attention

Questions

References

References

[1] Schubert, E., Lang, A. and Feher, G. 2021. Accelerating spherical k-means. *Similarity search and applications*. Springer International Publishing. 217–231.