FELIX: Automatic and Interpretable Feature Engineering Using LLMs

Simon Malberg$^{[0009-0000-3781-4700]}$ (✉), Edoardo Mosca$^{[0000-0003-4045-5328]}$, and Georg Groh

School of Computation, Information and Technology, Technical University of Munich, 80333 Munich, Germany `{simon.malberg, edoardo.mosca}@tum.de, grohg@in.tum.de`

Appendix

This document includes only the Appendix with the supplementary material.

A Model Design Validity

This experiment investigates the validity of the feature selection and value assignment procedures used by FELIX. It is to be shown that both, clustering and value assignments via LLMs are valid and reliable means to achieve their respective goals.

A.1 Clustering Validity

We hypothesize that not all features generated by FELIX during the feature generation phase are useful. In particular, we expect two cases where features are dispensable or even harmful: (A) Some features may be duplicates, i.e., they have the same or a highly similar name and meaning. This is because features are generated in multiple independent iterations over pairs of examples with some examples appearing in multiple pairs. It is likely that some features are proposed in multiple iterations. (B) Some features may simply not be insightful with regards to the classification task as they do not generalize beyond a single example pair. While FELIX has no mechanism to discover dispensable features of the second type, the goal of the consolidation phase is exactly to remove the duplicate features of the first type. To show the validity of utilizing clustering methods for feature consolidation, we compare different clustering methods against a lower and an upper baseline.

Setup. We perform the feature generation phase of FELIX to learn a set of m_{cand} candidate features. We then skip the feature selection phase and jump directly to the value assignment phase to assign values for all m_{cand} features on a training dataset with 100 examples. We then evaluate three clustering methods in finding a consolidated feature set with $1 \leq m_{sel} \leq m_{cand}$ features solely based on text embeddings of the feature names and descriptions:

1. **Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN)**[1] with the hyperparameters described in Appendix Section B. HDBSCAN uses an internal heuristic to determine the optimal number of clusters, which is the number of selected features m_{sel} in our case. Hence, we only observe a single value for m_{sel} for HDBSCAN.
2. **Hierarchical Agglomerative Clustering**, which determines m_{sel} based on a distance threshold hyperparameter d_{thres}. We vary d_{thres} to get clusterings for the full range of possible feature set sizes $\{1, 2, \dots, m_{cand}\}$.
3. **K-means**, where $k = m_{sel}$ is a hyperparameter. We vary k to get clusterings for the full range of possible feature set sizes $k \in \{1, 2, \dots, m_{cand}\}$.

We compare the clustering methods against two baselines that also generate feature sets for each $m_{sel} \in \{1, 2, \dots, m_{cand}\}$:

1. **A supervised forward feature selection** that starts with an empty feature set and iteratively adds the feature to the feature set that yields the best classification performance until all features are included—this baseline is thus capable of discarding unhelpful features of the second type as F1 scores are known for the feature selection (supervised). This should theoretically indicate the upper bound of classification performance[2].
2. **A random feature selection** that starts with an empty feature set and randomly adds another feature until all features are included. To smooth out results, we run random feature selection five times for each dataset and report the average performance.

For each clustering/feature set, we measure end-to-end performance of the classifiers on only those features that are part of the clustering. Performance is measured as the average F1 score achieved by a Logistic Regression and a Random Forest classifier, each evaluated using 5-fold cross validation on a test dataset with 100 examples. We repeat this experiment for all five datasets introduced in Section 4.1 of the main paper and for both, the numerical and the categorical variant of FELIX.

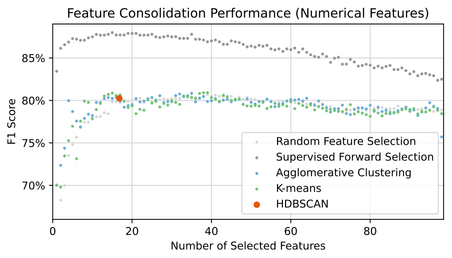
Results. Figure 1 shows the results of the clustering validation experiment averaged across all five datasets. We observe that feature consolidation with all three methods is generally able to identify feature sets that yield a better classification performance than (A) a purely random feature selection and (B) the full feature set. However, the performance advantage is often small and there is still a visible performance gap to a theoretically optimal supervised feature selection. Future research may focus on smarter feature selection techniques to identify the most informative features. For categorical features, the performance advantage over random feature selection is noticeably larger than for numerical features.

[1] https://hdbscan.readthedocs.io/en/latest/

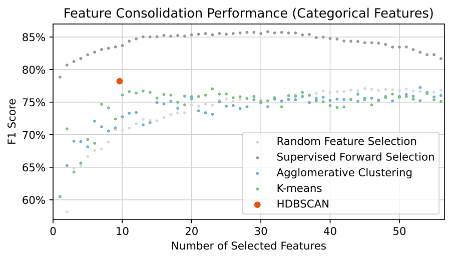
[2] An exhaustive feature selection that tries all different combinations of features would likely perform even better but is computationally infeasible in our case.

Interestingly, performance of the consolidation methods based on clustering seems to fall below performance of a random feature selection when feature sets grow larger than approximately 20 to 40 features. We suppose that this happens because all unique *and* informative features have already been selected at this point. When feature sets grow even further, clustering methods prefer to add non-duplicate but possibly uninformative features whereas random feature selection has a slightly higher likelihood of including seemingly duplicate features that still add some degree of new information.

While there is no significant performance difference between the three clustering methods on numerical features, differences are more visible on categorical features. HDBSCAN results in the highest classification performance on average, followed by K-means and lastly Agglomerative Clustering. On average, HDBSCAN also seems to identify a close-to-optimal feature set size as it often selects a number of clusters close to where the other methods show an "elbow" in clustering performance. These results vary from dataset to dataset (especially when using numerical features) and different methods may be superior for other datasets not evaluated in this paper. In our case, we chose to use HDBSCAN as the consolidation method in all experiments presented in the main paper as it delivers the best average performance and also provides a reliable way to determine the size of the consolidated feature set.



(a) Numerical features



(b) Categorical features

Fig. 1: Results of the clustering validity experiment. Figure 1a shows classification performance based on different feature set sizes and consolidation methods for numerical features while Figure 1b shows the same for categorical features. Results shown are averages across all five datasets. For HDBSCAN, the shown datapoint is the average F1 score and average number of selected features across all five datasets.

A.2 Value Assignment Validity

As LLMs are autoregressive, values generated for features during value assignment are not entirely independent from each other but influenced by features and values seen and generated previously in the value assignment prompt. To ensure that the value assignment approach produces reliable data representations for the downstream classifier, we experimented with different algorithm design choices.

First, we discovered that randomly re-ordering features in the value assignment prompt for each example harms classification performance when compared against a stable feature order (e.g., F1 scores on the *Papers* dataset dropping by more than 23% when using FELIX GPT-3.5 Numerical). This is because feature values assigned after a random re-ordering are significantly different from feature values assigned without such a re-ordering (e.g., 72% of feature values were significantly different at the 1% level when using FELIX GPT-3.5 Numerical on the *Papers* dataset). Hence, we decided to introduce the feature set in the same order in each value assignment prompt to the LLM.

When examples are long texts and a learned feature set comprises a large number of features, value assignment prompts sometimes exceed the context window of the LLM. In a second step, we therefore assessed whether splitting value assignment of an example into multiple prompts affects assigned feature values. When assigning values to the first half of a feature set in one prompt and to the second half in a different prompt, we found that assigned values for the second half of the feature set were significantly different from values assigned during just a single prompt with the entire feature set (e.g., 79% of features in the second half receiving values that were significantly different at the 1% level when using FELIX GPT-3.5 Numerical on the *Papers* dataset). Consequently, we decided to not split value assignment into multiple prompts but instead use an LLM variant with a larger context window (e.g., `gpt-3.5-turbo-16k` with a 16,000 token context window instead of `gpt-3.5-turbo` with a 4,000 token context window) whenever a prompt exceeds the context window. When repeatedly assigning feature values to the same example with both LLM variants, we found no significant difference in assigned values between the smaller- and the larger-context variant.

Lastly, we found that assigning feature values for the same example multiple times with `temperature=0.0` results in near-zero variance of the assigned feature values while assigning feature values for different examples results in significantly different feature values, hence sufficiently discriminating different examples from each another. Based on these insights, we conducted all experiments presented in the main paper with `temperature=0.0` for value assignment prompts and a stable feature order, as well as assigning values to all features in a single prompt while using a larger-context LLM variant when prompts exceeded the allowed context window. We used smaller-context variants whenever possible to save API costs.

B HDBSCAN Hyperparameter Optimization

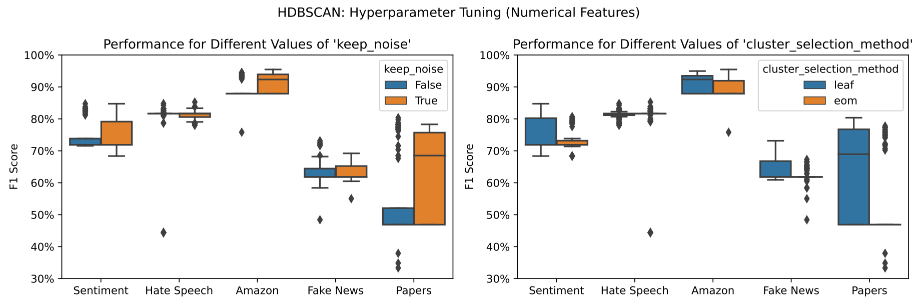
Selecting a suitable feature set during FELIX's consolidation phase is critical for the end-to-end classification performance. While K-means and Hierarchical Agglomerative Clustering (evaluated in Appendix Section A.1) are flexible to identify any number of clusters/features, HDBSCAN runs a heuristic for finding the ideal number of clusters/features. This heuristic is potentially sensitive to the hyperparameters defined for HDBSCAN. Hence, we evaluate different combinations of hyperparameters to find a combination that yields the best results. Figure 2 shows the hyperparameter optimization results for numerical features and Figure 3 shows the results for categorical features.

We investigate different values for the following hyperparameters:

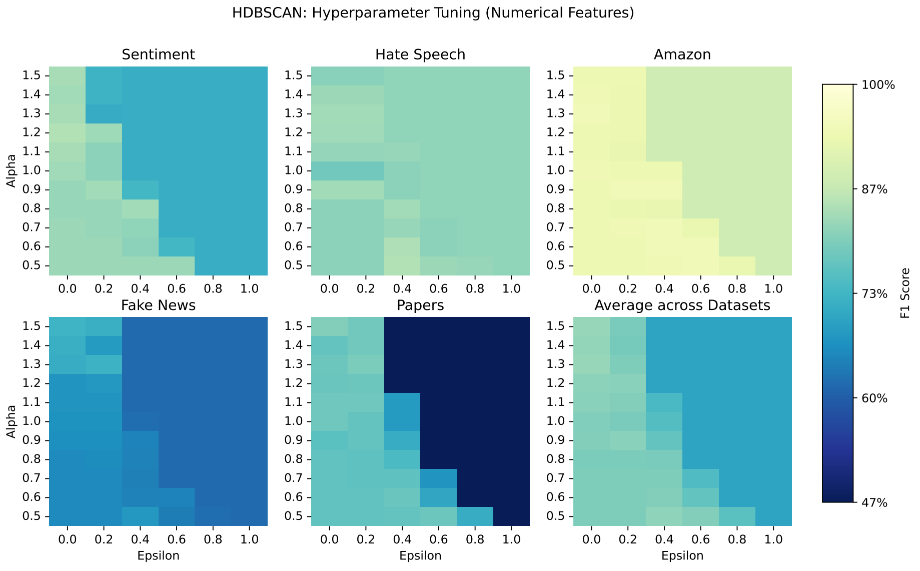
- `keep_noise`: This binary hyperparameter determines if features that HDBSCAN identifies as noise (typically features that are the result of only one iteration of the generation phase) will be kept (`True`) or discarded (`False`).
- `cluster_selection_method`: This multinomial hyperparameter determines how HDBSCAN's heuristic selects flat clusters from the cluster tree hierarchy. `eom` ("excess of mass") tends to select a few large cluster whereas `leaf` selects leaf nodes from the tree, producing a larger number of small and homogeneous clusters.
- `alpha`: This numerical hyperparameter influences the distance scaling in robust single linkage. Its default value is `1.0`. We vary it between `0.5` and `1.5`.
- `epsilon`: This numerical hyperparameter sets the distance threshold below which clusters are not split up any further. The default value is `0.0`. We vary it between `0.0` and `1.0`.

We can observe that discarding features identified as noise is generally beneficial to classification performance and results in significantly smaller feature sets. In contrast to that, setting `cluster_selection_method` to `leaf` selects larger feature sets resulting in better top performance. Hence, we decide to choose a balanced combination of discarding noisy features and selecting clusters from `leaf` nodes.

For the numeric hyperparameters, we observe more diverse results. When features are categorical, the default values of `alpha` and `epsilon` seem to deliver the best results on average. When features are numerical, there is no clearly dominant value combination across datasets. The official HDBSCAN documentation recommends avoiding altering values of `alpha`. Therefore, we decide to stay with default values for `alpha` and `epsilon` throughout all experiments.

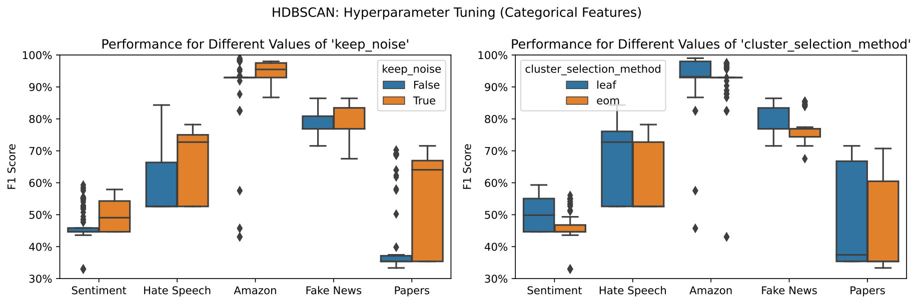


(a) Discrete hyperparameters

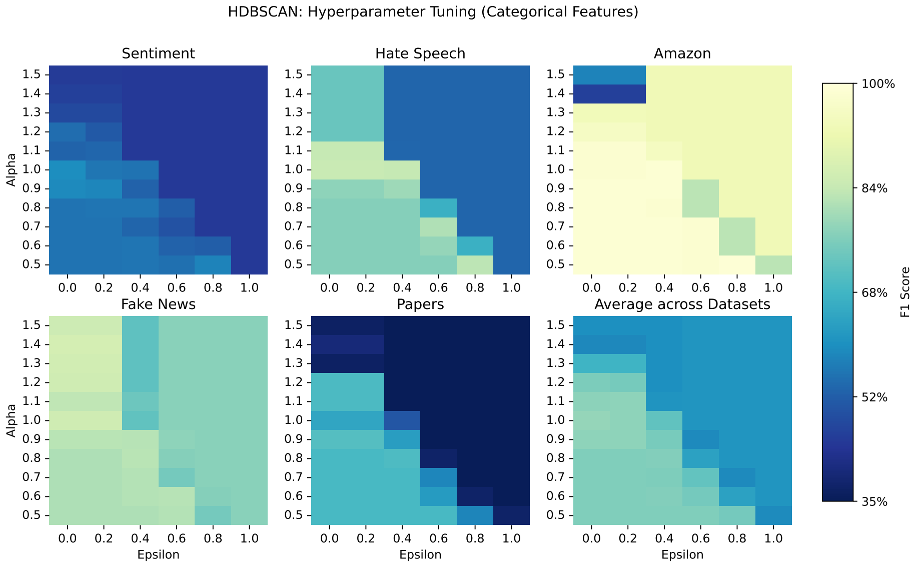


(b) Continuous hyperparameters

Fig. 2: End-to-end classification performance for different hyperparameter configurations of HDBSCAN for numerical features. Figure 2a shows variations of the discrete hyperparameters whereas Figure 2b shows variations of the numeric hyperparameters while the discrete hyperparameters are set to their optimal values.



(a) Discrete hyperparameters



(b) Continuous hyperparameters

Fig. 3: End-to-end classification performance for different hyperparameter configurations of HDBSCAN for categorical features. Figure 3a shows variations of the discrete hyperparameters whereas Figure 3b shows variations of the numeric hyperparameters while the discrete hyperparameters are set to their optimal values.

C Prompts

We use *Langchain*[3] to define prompt templates and parse LLM outputs. The formatting instructions are generated by Langchain based on *Pydantic*[4] objects. Prompts 1 and 2 state the feature generation and value assignment prompts, respectively, used by FELIX Categorical. Prompts 3 and 4 state the feature generation and value assignment prompts, respectively, used by FELIX Numerical. The stated prompts are concrete examples for the Amazon dataset. We truncated some parts for readability.

Prompt 1: Feature generation prompt for FELIX Categorical. Blue parts are constant, red parts vary with the dataset and example.

```
System:
You are a data scientist. Your goal is to engineer a set of features suitable
    to distinguish examples from different classes. The concrete context is
    the following: Examples are product reviews from Amazon that are either
    'NEGATIVE' (rating with 1 or 2 stars) or 'POSITIVE' (rating with 4 or 5
    stars). The downstream machine learning task is learning a classifier
    (e.g., Random Forest) that learns to distinguish 'NEGATIVE' and
    'POSITIVE' reviews.

User:
##### The following example is from class 'POSITIVE' #####

Title: Spaetzle Noodles

Content: This type of spaetzle maker is easier to manuveur than the old press
    kind and much easier on the hands. The difference is ...


##### The following example is from class 'NEGATIVE' #####

Title: Anyone who likes this better than the Pekinpah is a moron.

Content: All the pretty people in this film. Even the Rudy character played
    by Michael Madsen. This is ...


##### Instructions #####

Based on the above examples and their classes, create a list of features that
    would best distinguish examples of class 'POSITIVE' from examples of
    class 'NEGATIVE'. Features can be anything from very specific (e.g.,
    occurrences of a specific word) to very abstract (e.g., describing the
    logical flow of text). However, features should generalize even to new
    and previously unseen examples that may come from a slightly different
```

[3] https://www.langchain.com/

[4] https://docs.pydantic.dev/

```
domain. Each feature should be categorical with at least 2 but no more than 5 different possible values. Possible values should be words but not numbers. For any given example, each feature should unambiguously take exactly one value out of the list of possible values. Thus, avoid features that could take two or more values for any example and ensure that there is always exactly one value that fits an example (if necessary, include fallback values such as 'Other', 'Neutral', or 'Not applicable').

The output should be formatted as a JSON instance that conforms to the JSON schema below.

As an example, for the schema "properties": "foo": "title": "Foo", "description": "a list of strings", "type": "array", "items": "type": "string", "required": ["foo"] the object "foo": ["bar", "baz"] is a well-formatted instance of the schema. The object "properties": "foo": ["bar", "baz"] is not well-formatted.

Here is the output schema:
"'
{"properties": {"features": {"title": "Features", "description": "list of categorical features", "type": "array", "items": {"$ref": "#/definitions/CategoricalFeature"}}}, "required": ["features"], "definitions": {"CategoricalFeature": {"title": "CategoricalFeature", "type": "object", "properties": {"name": {"title": "Name", "description": "concise name of the feature", "type": "string"}, "possible_values": {"title": "Possible Values", "description": "list of 2-5 different possible values allowed for this feature; the feature can take exactly one of these values at once", "type": "array", "items": {"type": "string"}}, "description": {"title": "Description", "description": "short description of the meaning of this feature", "type": "string"}}, "required": ["name", "possible_values", "description"]}}}
"'
```

Prompt 2: Value assignment prompt for FELIX Categorical. Blue parts are constant, red parts vary with the dataset and example.

```
System:
You are a data annotator. Your task is to annotate a given example by scoring it with regards to several criteria. The concrete context is the following: Examples are product reviews from Amazon that are either 'NEGATIVE' (rating with 1 or 2 stars) or 'POSITIVE' (rating with 4 or 5 stars). The downstream machine learning task is learning a classifier (e.g., Random Forest) that learns to distinguish 'NEGATIVE' and 'POSITIVE' reviews.

User:
##### Here is an example that you should annotate #####

Title: Spaetzle Noodles

```

```
Content: This type of spaetzle maker is easier to manuveur than the old press
    kind and much easier on the hands. The difference is ...


##### Instructions #####

Score the above example along the following list of criteria. For each
    criteria, assign one of the possible values that most accurately
    describes the example.

The output should be a markdown code snippet formatted in the following
    schema, including the leading and trailing "“‘json" and "“‘":

“‘json
{
    "constructiveness": string // 'Whether the review provides constructive
    feedback or is solely critical' (value can be any of ['Constructive',
    'Critical', 'Neutral', 'NotApplicable'])
    "useofsuperlatives": string // 'Frequency of superlative adjectives and
    adverbs' (value can be any of ['Many', 'Few', 'None'])
    "exclamationusage": string // 'Usage of exclamation marks' (value can be
    any of ['Present', 'Absent'])
    ...
}
“‘

Do not add any comments (starting with //) inside the JSON!
```

Prompt 3: Feature generation prompt for FELIX Numerical. Blue parts are constant, red parts vary with the dataset and example.

```
System:
You are a data scientist. Your goal is to engineer a set of features suitable
    to distinguish examples from different classes. The concrete context is
    the following: Examples are product reviews from Amazon that are either
    'NEGATIVE' (rating with 1 or 2 stars) or 'POSITIVE' (rating with 4 or 5
    stars). The downstream machine learning task is learning a classifier
    (e.g., Random Forest) that learns to distinguish 'NEGATIVE' and
    'POSITIVE' reviews.

User:
##### The following example is from class 'POSITIVE' #####

Title: Spaetzle Noodles

Content: This type of spaetzle maker is easier to manuveur than the old press
    kind and much easier on the hands. The difference is ...


##### The following example is from class 'NEGATIVE' #####
```

```

Title: Anyone who likes this better than the Pekinpah is a moron.

Content: All the pretty people in this film. Even the Rudy character played
    by Michael Madsen. This is ...


##### Instructions #####

Based on the above examples and their classes, create a list of features that
    would best distinguish examples of class 'POSITIVE' from examples of
    class 'NEGATIVE'. Features can be anything from very specific (e.g.,
    occurrences of a specific word) to very abstract (e.g., describing the
    logical flow of text). However, features should generalize even to new
    and previously unseen examples that may come from a slightly different
    domain. Each feature should be numeric and measured on a scale of 0 to
    10. Therefore, indicate for each feature what 0 and 10 mean, respectively
    (e.g., 0 = "few", 10 = "many").

The output should be formatted as a JSON instance that conforms to the JSON
    schema below.

As an example, for the schema "properties": "foo": "title": "Foo",
    "description": "a list of strings", "type": "array", "items": "type":
    "string", "required": ["foo"] the object "foo": ["bar", "baz"] is a
    well-formatted instance of the schema. The object "properties": "foo":
    ["bar", "baz"] is not well-formatted.

Here is the output schema:
"'
{"properties": {"features": {"title": "Features", "description": "list of
    numeric features", "type": "array", "items": {"$ref":
    "#/definitions/NumericalFeature"}}}, "required": ["features"],
    "definitions": {"NumericalFeature": {"title": "NumericalFeature",
    "type": "object", "properties": {"name": {"title": "Name", "description":
    "concise name of the feature", "type": "string"}, "zero": {"title":
    "Zero", "description": "meaning of feature value of 0", "type":
    "string"}, "ten": {"title": "Ten", "description": "meaning of feature
    value of 10", "type": "string"}, "description": {"title": "Description",
    "description": "short description of the meaning of this feature",
    "type": "string"}}, "required": ["name", "zero", "ten", "description"]}}}
"'
```

Prompt 4: Value assignment prompt for FELIX Numerical. Blue parts are constant, red parts vary with the dataset and example.

```
System:
You are a data annotator. Your task is to annotate a given example by scoring
    it with regards to several criteria. The concrete context is the
    following: Examples are product reviews from Amazon that are either
    'NEGATIVE' (rating with 1 or 2 stars) or 'POSITIVE' (rating with 4 or 5
    stars). The downstream machine learning task is learning a classifier
    (e.g., Random Forest) that learns to distinguish 'NEGATIVE' and
    'POSITIVE' reviews.

User:
##### Here is an example that you should annotate #####

Title: Spaetzle Noodles

Content: This type of spaetzle maker is easier to manuveur than the old press
    kind and much easier on the hands. The difference is ...


##### Instructions #####

Score the above example along the following list of criteria. For each
    criteria, assign an integer value from 0 to 10 that most accurately
    describes the example.

The output should be a markdown code snippet formatted in the following
    schema, including the leading and trailing "“‘json" and "“‘":

“‘json
{
    "useofsuperlatives": int // 'useofsuperlatives' (value from 0 to 10, 0
    means 'no superlatives', 10 means 'frequent use of superlatives')
    "personalpronounscount": int // 'personalpronounscount' (value from 0 to
    10, 0 means 'no personal pronouns', 10 means 'high frequency of personal
    pronouns')
    "exclamationpointusage": int // 'exclamationpointusage' (value from 0 to
    10, 0 means 'no exclamation points', 10 means 'excessive use of
    exclamation points')
    ...
}
“‘

Do not add any comments (starting with //) inside the JSON!
```

D Context/Task Descriptions

Table 1 shows the context/task descriptions provided to FELIX for each of the six datasets.

Dataset	**Context/task description**
Sentiment	Examples are tweets in multiple languages which express either a 'NEGATIVE' or 'POSITIVE' sentiment. The downstream machine learning task is learning a classifier (e.g., Random Forest) that learns to distinguish 'NEGATIVE' and 'POSITIVE' tweets, independent of the language they are written in.
Hate Speech	Examples are posts sampled from a white supremacist forum and are either 'HATE SPEECH' (when the posts contain hate speech) or 'NO HATE SPEECH' (when the posts do not contain hate speech). The downstream machine learning task is learning a classifier (e.g., Random Forest) that learns to distinguish 'HATE SPEECH' and 'NO HATE SPEECH' forum posts.
Amazon	Examples are product reviews from Amazon that are either 'NEGATIVE' (rating with 1 or 2 stars) or 'POSITIVE' (rating with 4 or 5 stars). The downstream machine learning task is learning a classifier (e.g., Random Forest) that learns to distinguish 'NEGATIVE' and 'POSITIVE' reviews.
Yelp	Examples are reviews from Yelp that are either 'NEGATIVE' (rating with 1 or 2 stars) or 'POSITIVE' (rating with 4 or 5 stars). The downstream machine learning task is learning a classifier (e.g., Random Forest) that learns to distinguish 'NEGATIVE' and 'POSITIVE' reviews.
Fake News	Examples are coming from a dataset with news articles that are either 'FAKE NEWS' or 'REAL NEWS'. The downstream machine learning task is learning a classifier (e.g., Random Forest) that learns to distinguish 'FAKE NEWS' and 'REAL NEWS' articles.
Papers	Examples are coming from a dataset with scientific papers that are either 'HUMAN-WRITTEN' (i.e., written by a real person) or 'MACHINE-GENERATED' (i.e., generated by a machine learning model). The downstream machine learning task is learning a classifier (e.g., Random Forest) that learns to distinguish 'HUMAN-WRITTEN' and 'MACHINE-GENERATED' scientific papers.

Table 1: Overview of the context/task descriptions used for each dataset.

E Runtime and Cost

A summary of the runtime and occurred API costs of the evaluated methods can be found in Table 2.

		API cost				**Transformation**
Data representation	**Dataset**	**Generation** (USD)	**Value assignment** (USD)	**Total** (USD)	**Per example** (Cents)	**Runtime** (seconds)
FELIX GPT-4 Categorical	Sentiment	0.49	3.36	3.85	2.57	3,843.8
	Hate Speech	0.43	2.75	3.18	2.12	3,078.0
	Amazon	0.53	3.08	3.61	2.40	3,236.2
	Fake News	0.75	4.56	5.31	3.54	4,280.1
	Papers	1.05	5.19	6.24	4.16	4,069.7
	Average	**0.65**	**3.79**	**4.44**	**2.96**	**3,701.5**
FELIX GPT-4 Numerical	Sentiment	0.64	4.65	5.29	3.53	4,417.1
	Hate Speech	0.61	4.59	5.20	3.46	5,149.0
	Amazon	0.63	4.43	5.06	3.37	4,382.5
	Fake News	0.84	4.99	5.83	3.89	3,729.0
	Papers	1.08	5.83	6.91	4.61	3,849.0
	Average	**0.76**	**4.90**	**5.66**	**3.77**	**4,305.3**
FELIX GPT-3.5 Categorical	Sentiment	0.04	0.21	0.25	0.17	955.6
	Hate Speech	0.04	0.16	0.20	0.13	562.3
	Amazon	0.05	0.21	0.26	0.17	653.6
	Fake News	0.07	0.36	0.42	0.28	970.9
	Papers	0.11	0.42	0.53	0.35	1,142.7
	Average	**0.06**	**0.27**	**0.33**	**0.22**	**857.0**
FELIX GPT-3.5 Numerical	Sentiment	0.04	0.39	0.43	0.29	1,538.6
	Hate Speech	0.04	0.36	0.41	0.27	1,324.9
	Amazon	0.05	0.35	0.39	0.26	1,111.3
	Fake News	0.08	0.63	0.71	0.47	1,624.5
	Papers	0.11	0.58	0.69	0.46	1,311.0
	Average	**0.06**	**0.46**	**0.53**	**0.35**	**1,382.1**
Zero-Shot GPT-4	Sentiment	N/A	N/A	0.11	0.08	N/A
	Hate Speech	N/A	N/A	0.12	0.08	N/A
	Amazon	N/A	N/A	0.18	0.12	N/A
	Fake News	N/A	N/A	0.62	0.41	N/A
	Papers	N/A	N/A	1.02	0.68	N/A
	Average	**N/A**	**N/A**	**0.41**	**0.27**	**N/A**
Zero-Shot GPT-3.5	Sentiment	N/A	N/A	0.02	0.01	N/A
	Hate Speech	N/A	N/A	0.02	0.01	N/A
	Amazon	N/A	N/A	0.03	0.02	N/A
	Fake News	N/A	N/A	0.09	0.06	N/A
	Papers	N/A	N/A	0.15	0.10	N/A
	Average	**N/A**	**N/A**	**0.06**	**0.04**	**N/A**
Text Embeddings	Sentiment	N/A	N/A	N/A	N/A	2.9
	Hate Speech	N/A	N/A	N/A	N/A	2.5
	Amazon	N/A	N/A	N/A	N/A	2.1
	Fake News	N/A	N/A	N/A	N/A	4.8
	Papers	N/A	N/A	N/A	N/A	4.2
	Average	**N/A**	**N/A**	**N/A**	**N/A**	**3.3**
TF-IDF	Sentiment	N/A	N/A	N/A	N/A	0.2
	Hate Speech	N/A	N/A	N/A	N/A	0.2
	Amazon	N/A	N/A	N/A	N/A	0.3
	Fake News	N/A	N/A	N/A	N/A	1.4
	Papers	N/A	N/A	N/A	N/A	1.5
	Average	**N/A**	**N/A**	**N/A**	**N/A**	**0.7**

Table 2: Summary of runtime and API cost of FELIX compared to other data representation methods.