Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Clustering Social Network Texts

Egor Sarana

Supervisor: Ing. Jan Drchal, Ph.D.
Field of study: Open Informatics
Subfield: Artificial Intelligence and Computer Science
January 2024

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Sarana Egor** Personal ID number: **499025**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Clustering Social Network Texts

Bachelor’s thesis title in Czech:

Shlukování textů ze sociálních sítí

Guidelines:

The task is to experiment with various clustering methods and semantic text embeddings and their applicability to cluster texts from social networks such as Twitter (X) or Telegram:
1) Research state-of-the-art text clustering systems and methods of text embedding extraction.
2) Choose a set of diverse clustering and embedding methods most-likely appropriate for social network texts.
3) Find public clustering datasets or create ones (with the help of the supervisor).
4) Research and design evaluation methodology.
5) Perform experiments measuring contribution of the embedding and clustering method choice to the quality of the overall clustering.
6) Show and analyse results on Twitter and Telegram data supplied by the supervisor.

Bibliography / sources:

[1] Diana, Korladinova. Extracting Keywords from Textual Data Clusters. Bachelor thesis. České vysoké učení technické v Praze., 2023.
[2] Ibrahim, R., S. Zeebaree, and K. Jacksi. "Survey on semantic similarity based on document clustering." Adv. sci. technol. eng. syst. j 4.5 (2019): 115-122.
[3] Kharlamov, Alexander A., et al. "Social network sentiment analysis and message clustering." International Conference on Internet Science. Cham: Springer International Publishing, 2019.
[4] Curiskis, Stephan A., et al. "An evaluation of document clustering and topic modelling in two online social networks: Twitter and Reddit." Information Processing & Management 57.2 (2020): 102034.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jan Drchal, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **25.09.2023** Deadline for bachelor thesis submission: **09.01.2024**

Assignment valid until: **22.09.2024**

Ing. Jan Drchal, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student's signature

Acknowledgements

I would like to express my deepest thanks to my supervisor Ing. Jan Drchal, Ph.D., he advised me whenever I ran into trouble.

I am also grateful to the CTU for all the knowledge I acquired during my study.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, January 9, 2024

Abstract

This work aims to create a diverse dataset for the evaluation of social network text clustering and explores different combinations of text embedding methods, dimensionality reduction techniques, and clustering methods. We choose a wide range of the most appropriate evaluation metrics, build an evaluation pipeline, and test the most interesting models.

Keywords: nlp, text embeddings, dimensionality reduction, clustering, social networks

Supervisor: Ing. Jan Drchal, Ph.D.

Abstrakt

Cílem této práce je vytvořit rozmanitou datovou sadu pro hodnocení shlukování textů ze sociálních sítí a vyzkoušet různé kombinace metod vnoření textu, technik redukce dimenzionality a metod shlukování. Vybíráme širokou škálu nejvhodnějších metrik pro hodnocení, vytváříme vhodné potrubí a testujeme nejzajímavější modely.

Klíčová slova: nlp, vnoření textu, redukce dimenzionality, shlukování, sociální sítě

Překlad názvu: Shlukování textů ze sociálních sítí

Contents

Figures

Tables

Chapter 1

Introduction

Document clustering is a key task in machine learning. Clustering can be used to extract topics from a group of documents and characterize the corpus as a whole. Clustering is used in document group summarization, information retrieval, noise document detection (filtering), content recommendation, news aggregation, and many other applications.

Short text clustering (STC) is a separate topic. It is used within social applications for sentiment analysis, spam detection, recommendations, and so on. In the era of social media, short text data is very abundant and it has its own specific features (Stieglitz et al. (2018)). It is the sparsity of representation, the maximum concentration of information in short text which is difficult to extract, and a large number of errors and misspellings per word.

In this work, we will create a dataset, select metrics, and evaluate which models do the best job of dealing with the challenges of short text clustering. The paper (Ahmed et al. (2023)) can serve as a very good starting point. It describes and provides an overview of the main models from TF-IDF to transformers, from LDA to DBSCAN, and in general how an end-to-end pipeline should be constructed.

The work has associated code with repository[1].

- In **generate_dataset.ipynb** we are dealing with the problem of qualitative dataset generation and noise tweet detection.
- **clustering.ipynb** contains the main pipeline from text tokenization and pre-processing up to clusterization, basic evaluation, and saving of results.
- Within **analysis.ipynb** you can find evaluation table analysis and basic visualization.
- **evaluation.py** contains evaluation metrics, a class for storing evaluation table, embeddings, and clustering of every model.
- And finally **data** folder has all source datasets and the final one.

[1] https://github.com/anabolicobsession/tweet-clustering

Chapter 2

Text Embeddings

Computers are not good at working directly with symbols compared to humans. To use all the computing capabilities, you need to be able to represent words, sentences, and entire texts (depending on the task) in numerical form. Depending on the method of mapping, there may be many additional benefits of numerical representation. Word embeddings can often extract semantic relationships. The word embedding space, despite its limited dimension, is capable of representing a potentially unimaginable amount of textual data. In this chapter, we will go through the basic methods of representing text in computer memory.

2.1 TF-IDF

TF-IDF (short for **term frequency-inverse document frequency**) is a measure that assigns importance to each word in a collection of documents based on statistics. In the context of text embeddings, it has some similarities to simple word counting, but it penalizes words that appear too often in the corpus. It assigns less weight to them. Since a word that appears in most documents does little to characterize one particular document.

TF-IDF, as the name implies, is calculated based on two terms:

- **TF**: represents the frequency of a term t within a particular document d.

$$TF(t,d) = \frac{\text{number of occurrences of } t \text{ in } d}{\text{number of terms in } d}$$

- **IDF**: inverse document frequency, or how often the term t is present in documents.

$$IDF(t) = \log \frac{\text{number of documents } d \text{ that contain term } t}{\text{number of documents}}$$

Using this technique it is possible not only to estimate how important a term is in a document but also to represent an entire document as a vector.

We will use *scikit-learn*[1] TF-IDF implementation that differs slightly from the original, such as adding one to the inverse document frequency to "smooth

[1] Python module for machine learning.

out" the weights. This is the same if each term was in at least one document by default. This helps to avoid division by zero. There are also some other modifications; all words with a maximum frequency greater than 0.5 are excluded from the dictionary because they do little to distinguish one particular document from the rest. The specific implementation is contained within *clustering.ipynb* jupyter notebook.

2.2 Word2Vec

Word2Vec is an influential word embedding technique introduced by a team of researchers led by Tomas Mikolov at 2013 (Mikolov et al. (2013a) and (Mikolov et al. (2013b)). Word2Vec aims to capture semantic relationships between words. Basically, Word2Vec is a shallow, two-layer neural network that is trying to learn word representations that maximize the prediction of other context words on a large corpus of text.

We have short documents, so it is a good idea to focus on the skip-gram architecture. In the skip-gram architecture, the model predicts the context words (surrounding words) given the target word (center word). The input is a single target word, and the output is a probability distribution over the context words. The objective is:

$$\prod_{i \in C} Pr(w_j : j \in N + i | w_j)$$

where C is a corpus (tokens) and N represents sliding window. In other words, we are trying to maximize the probability of the whole corpus. It is not computationally efficient to evaluate the objective through the whole process, so the hierarchical softmax and/or negative sampling.

The sentence (tweets) embeddings can be obtained by simple average word embeddings of the whole tweet. One can use TF-IDF weights for better averaging, but the former is a more general choice, so we will use it.

Word2Vec has significantly influenced the field of NLP and has paved the way for more advanced models, such as GloVe (Pennington et al. (2014)), FastText (Bojanowski et al. (2017)), and contextualized embeddings like ELMo and BERT. It remains a foundational technique for capturing distributed representations of words. We include it as the baseline model.

2.3 DistilRoBERTa

DistilRoBERTa (short for **Distilled RoBERTa**, Sanh et al. (2020)) is a compressed and lighter version of the RoBERTa (Robustly optimized BERT approach, Liu et al. (2019)) model. RoBERTa is itself a version of BERT (Bidirectional Encoder Representations from Transformers), which has been pre-trained on a huge corpus and has demonstrated state-of-the-art performance on various tasks. So we need to start the explanation from the beginning.

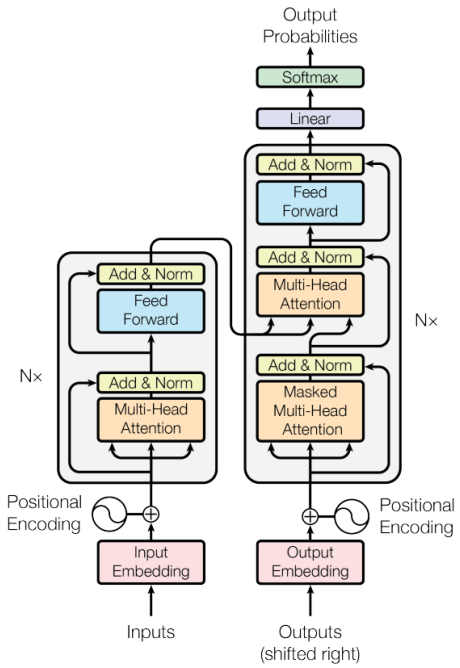

Figure 2.1: Transformer model architecture (Vaswani et al. (2017)).

The objective of transformers is to produce language models that can create numerical representations from text using pretraining, where each token could only rely on context information from previous tokens, and then fine-tuned for some downstream tasks like text classification, text summarization and so on.

The architecture of the base BERT follows follows:

- 12 transformer blocks.
- 12 self-attention heads within each attention layer for base
- Embeddings of hidden size of 768 for base.

Benefit of BERT over more traditional approaches is that it learns to compute text representations in context. This means that the representations computed for a word in a specific sentence would be different from the representations for the same word in a different sentence. This context also comprises stopwords, which can very much change the meaning of a sentence. The same goes for punctuation: a question mark can certainly change the overall meaning of a sentence. Therefore, removing stopwords and punctuation would just imply removing context which BERT could have used to get better results.

Now, RoBERTa is actually based on BERT, with a few changes:

- Training the model longer, with bigger batches, over more data.
- Removing the next sentence prediction objective.

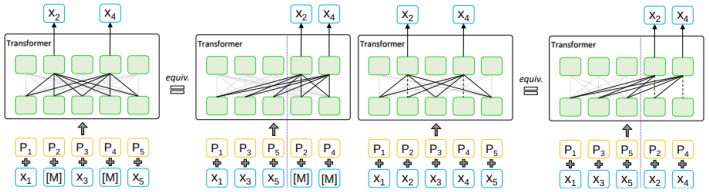

Figure 2.2: A unified view of MLM and PLM, where x_i and p_i represent token and position embeddings (Song et al. (2020)).

- Training on longer sequences.
- Dynamically changing the masking pattern applied to the training data.

DistilRoBERTa, as the name implies, follows the same training procedure as DistilBERT. The model has 6 layers, 768 dimensions, and 12 heads, totalizing 82M parameters (compared to 125M parameters for RoBERTa-base). On average DistilRoBERTa is twice as fast as RoBERTa-base.

2.4 MPNet

MPNet (Song et al. (2020)) is an attempt to combine the best achievements of BERT and XLNet (Yang et al. (2019)) and at the same time avoid their limitations. MPNet leverages the dependency among predicted tokens through permuted language modeling (vs. MLM in BERT), and takes auxiliary position information as input to make the model see a full sentence and thus reducing the position discrepancy (vs. PLM in XLNet) (the figure 2.2).

MPNet leverages the dependency among the predicted tokens through permuted language modeling and makes the model to see auxiliary position information to reduce the discrepancy between pre-training and fine-tuning. Experiments on various tasks demonstrate that MPNet outperforms MLM and PLM, as well as previous strong pre-trained models such as BERT, XLNet, RoBERTa by a large margin.

The training objective of MPNet is:

$$\mathbb{E}_{z \in Z_n} \sum_{t=c+1}^{n} \log P(x_{z_t}) | x_{z<t}, M_z > C; \theta)$$

So the main difference is that MPNet is conditioned on $x_{z<t}$ (the tokens before x_{z_t} rather than only the non-predicted tokens $x_{z_{\leq c}}$ in MLM). MLM can see the position information of the full sentence, but cannot model the dependency among the predicted tokens, which cannot learn the complicated semantic relationships well; 2) PLM can model the dependency among the predicted tokens with autoregressive prediction, but it cannot see the position information of the full sentence, which will cause mismatches between the pretraining and fine-tuning since the position information of the full sentence can be seen in the downstream tasks. In the end, MPNet carries more information.

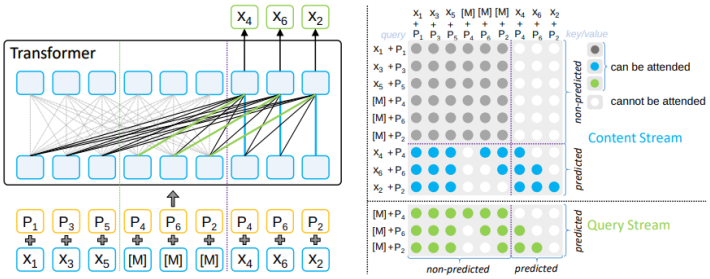

Figure 2.3: Structure and attention mask of MPNet (Song et al. (2020)).

2.5 Universal Sentence Encoder

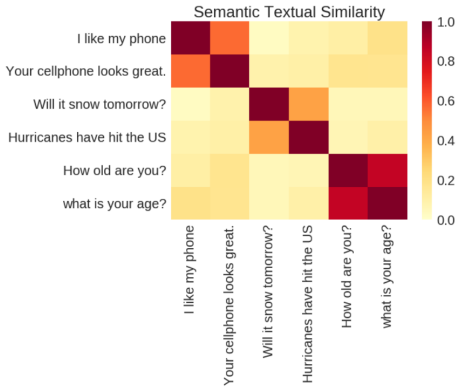

Figure 2.4: Sentence similarity scores using embeddings from the universal sentence encoder (Cer et al. (2018))).

The main concept of USE (Cer et al. (2018)) is to be able to summarize any sentence into a vector of 512 dimensions. This shared embedding is employed for various tasks, and based on the errors encountered during these tasks, the sentence embedding is updated. Since the same embedding is utilized across multiple generic tasks, it selectively captures the most informative features while discarding noise. The intuition is that this results in a universal embedding transferable to a diverse range of NLP tasks, including relatedness, clustering, paraphrase detection, and text classification.

In a simplified version, the encoder is constructed based on the architecture proposed by Iyyer et al. (2015). Initially, the embeddings for words and bi-grams present in a sentence are averaged together. Subsequently, they pass through a 4-layer feed-forward deep neural network (DNN), yielding a 512-dimensional sentence embedding as the output. The embeddings for words and bi-grams are learned during the training process.

The speed of the USE is an order of magnitude faster than some modern transformers, even at the expense of lower accuracy. Its main advantage, is ease and speed of use.

Chapter 3

Dimensionality Reduction

Dimensionality reduction is a technique that reduces the dimensionality of data while preserving maximum information and mutual relationships. There are many reasons to include this technique in your pipeline:

- **Computational efficiency**: high-dimensional word embeddings can be computationally expensive and memory-intensive, especially when working with large datasets. Dimensionality reduction helps make computation more efficient by speeding up the training and inference processes.
- **Curse of dimensionality**: the curse of dimensionality (Köppen (2000)) refers to the challenges and issues that arise when working with high-dimensional data: sparsity of data, computational instability, all distances approximating the same value and so on.
- **Memory usage**: storing and processing high-dimensional vectors requires more memory. By reducing the dimensionality of word embeddings, memory requirements are reduced, making it more feasible to work with large vocabularies and models.
- **Noise reduction**: high-dimensional word embeddings may capture noise or irrelevant information, especially when trained on large and diverse datasets. Dimensionality reduction can help in removing less informative dimensions and emphasizing the most relevant features.

3.1 Truncated SVD

Truncated singular value decomposition (truncated SVD) is a dimensionality reduction technique that is particularly useful for sparse or large matrices. SVD decomposes a matrix into three other matrices, and truncated SVD is a variant of this decomposition that retains only a specified number of the most significant singular values and their corresponding vectors. In other words, SVD decomposition allows us to find the hidden axes along which the variance is maximized. It is very similar to PCA, except that truncated SVD does not center data, and decomposition is not done on the covariance matrix. SVD decomposition is such a decomposition of matrix $X \in \mathbb{R}^{m \times n}$ that:

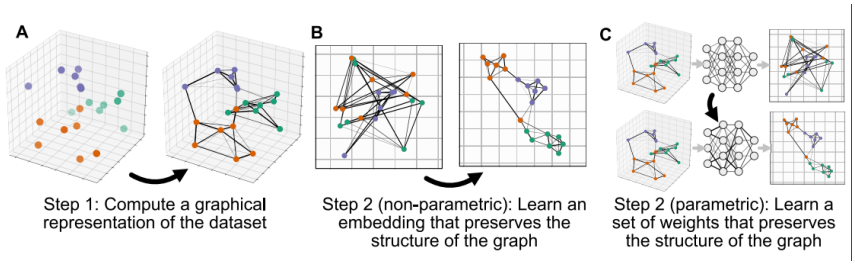

Figure 3.1: Overview of UMAP ($A \rightarrow B$) and Parametric UMAP ($A \rightarrow C$).

$$X = U \cdot S \cdot V^T$$

where U is an $m \times m$ orthogonal matrix, S is an $m \times n$ diagonal matrix with singular values on the diagonal, and V^T is an $n \times n$ orthogonal matrix. The truncated matrix is obtained by taking the most significant singular values and their corresponding axes.

Even though truncated PCA is fast, it also has its drawbacks. Truncated SVD can not learn any non-trivial dependencies, it is limited to linear projections.

3.2 UMAP

UMAP (Uniform Manifold Approximation and Projection, McInnes et al. (2020), the figure 3.1) is a novel manifold learning technique for dimension reduction. UMAP is constructed from a theoretical framework based in Riemannian geometry and algebraic topology. In contrast to PCA or truncated SVD, the projection is non-linear. The algorithm can be viewed as an alternative to t-SNE (t-Distributed Stochastic Neighbor Embedding, van der Maaten and Hinton (2008)). It is an order of magnitude faster, scalable to large real data, needs no much parameter tuning (as compared to t-SNE), and has minimal output variance during multiple runs. It is a truly state-of-the-art dimensionality reduction algorithm.

The main tuning parameter is *n_neighbors*. In simple terms, it has a similar role to perplexity in t-SNE. It allows for regulation of the balance between preserving the local and global structure of original data. As a rule of thumb, we will use *n_neighbors=15* for dimensionality reduction and output embedding size in range (5, 15).

Chapter 4

Clustering Methods

Clustering is a technique and unsupervised learning task that aims to group similar samples based on their features, forming so-called clusters. The main goal is to find patterns and hidden structures without using ground truth.

4.1 LDA

The first technique is exceptional in the sense that it does not require converting words into a numerical representation. Only tokenization (and standard preprocessing: lowercasing, lemmatization, and so on). **Latent Dirichlet Sllocation** (Blei et al. (2001)) is a probabilistic generative model based on the statistical properties of documents.

LDA is based on the assumption that each document in a collection can be represented as a mixture of a fixed number of topics, and each word in a document is attributable to one of the document's topics. The "latent" part of it refers to the hidden or unobservable variables in the model. These latent variables include the topics associated with each document and the distribution of words within each topic. It utilizes the Dirichlet distribution to model the distribution of topics in documents and the distribution of words in topics.

LDA utilizes the Dirichlet distribution to model the distribution of topics in documents and the distribution of words in topics. The model outputs two distributions.

- **Document-topic distribution:** the proportion of topics present in each document.
- **Topic-word distribution:** the distribution of words for each topic.

The goal of LDA is to infer the underlying topic structure of the documents. Given a set of documents, LDA estimates the parameters (document-topic distribution and topic-word distribution) that best explain the observed data (words in the documents).

Even though this is basically a topic modeling field, we can still use it for clustering as well, because topics are the same clusters. We will use the

gensim[1] implementation with the number of passes (over a corpus) of 10.

4.2 K-Means

K-Means is one of the most popular, if not the most popular clustering algorithm. And it is not surprising, that it is fast enough, easy to implement, easy to interpret, time-tested, and scalable to large data. K-Means can be considered as a special case of expectation–maximization[2] algorithm. After initialization of centroids consists of two iterative steps:

- **Assignment step:** assign all the points to the closest cluster centroid.
- **Update step:** recalculate centroids for observations assigned to each cluster.

The algorithm has converged when the assignments no longer change. The algorithm is not guaranteed to find the optimum.

The main goal of K-Means is to minimize inertia (loss, WCSS) defined as:

$$\min_{S} \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2$$

where $(x_1, x_2, \ldots, x_n)$ are observation in d-dimensional space, $S = S_1, S_2, \ldots, S_k$ is space clustering and μ_i is the mean (centroid) of points in S_i.

There are also smarter ways to initialize first centroids. For example, K-Means++ (Arthur and Vassilvitskii (2007)) initialized centroids not randomly, but using a specific distribution. There are also modifications to process big amounts of data, such as mini-batch K-Means. We will utilize all of that. As both K-Means and mini-batch K-Means optimize a non-convex objective function, their clustering is not guaranteed to be optimal for a given random seed. So we will run the algorithm more times (specifically 5) and choose one final clusterization based on the lowest inertia (loss).

Before any experiments we must be aware of curse dimensionality (Köppen (2000)) as K-Means is especially affected by this phenomenon. K-Means based on the notion of distance, most often Euclidean, and the larger space we deal with, the more sparse it is, the more similar distances have any two points in the space. There are many more details that are described in the chapter 3 and in the associated paper.

4.3 CLARA

CLARA (Clustering for Large Applications) is based on a very similar to K-Means algorithm K-Medoids. The main difference is that K-Medoids uses

[1] https://radimrehurek.com/gensim/

[2] Iterative method to find (local) maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables

Figure 4.1: Example of clusters produced by K-Means and CLARA. Source: Clustering Pokemon.

medoids instead of centroids (analogy median and mean). A medoid is a data point within a cluster whose average dissimilarity to all the other points in the cluster is minimal. Medoids must be part of the dataset, so the main implication is that K-Medoids better deal with noise (outliers). But K-Medoids are too slow for large applications, so CLARA solves this problem by utilizing random subsets of datasets.

Otherwise, it has very similar properties to K-Means and share similar drawbacks (such as the curse of dimensionality), so we will replicate the evaluation technique for CLARA (five runs with the lowest inertia).

4.4 HDBSCAN

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that extends traditional density-based clustering methods (DBSCAN). HDBSCAN is particularly designed to handle datasets with varying densities and is capable of identifying clusters of different shapes and sizes.

As the name implies, it identifies clusters based on the density of data points in the feature space. HDBSCAN builds a hierarchy of clusters by considering different levels of density. It uses a condensed tree representation called a Minimum Spanning Tree (MST) to encode the hierarchy of clusters. The key features are:

- Unlike traditional density-based methods, HDBSCAN can identify **clusters of varying densities.** It adapts to the local density of the data, allowing it to find clusters in regions with different levels of density.
- HDBSCAN is **robust to noise** and is capable of distinguishing noise

from actual clusters. It identifies points that do not belong to any cluster as noise or outliers.

- It is capable of **auto-detecting the most appropriate number of clusters.**

Even though it does not handle high dimensionality well. In most cases, a dimensionality reduction to less than 100 dimensions of input is a must-have. It is also not the easiest clustering algorithm for evaluation with ground truth, as it throws noise labels along with normal labels and it is up to you how to interpret those labels. We will use the simple approach. We will assign the label of the nearest non-noise sample to every noise sample.

Chapter 5

Metrics

A metric refers to a quantitative measure used to evaluate a model's performance on a specific task. Metrics allow you to evaluate the effectiveness of a model and can be used to compare different models. However, it is difficult to characterize an entire model with a single number, so we will look at a variety of metrics that attempt to evaluate most aspects of clustering

5.1 Intrinsic Metrics

Intrinsic metrics are the most common for evaluating clustering because they do not require ground truth. As the name implies, they try to estimate the final clustering based on the intrinsic properties of the clusters. This is also their limitation, as they often assume the simplest, most probable form of clusters (convex or isotropic). They often fail to take into account that clusters may overlap. In higher dimensions, intrinsic metrics may get worsen for comparing different models because of the curse of dimensionality[1]. And most importantly, they are very dependent on the choice of model and parameters. In our case, on embedding space. As we will see later, often intrinsic in different spaces are incomparable. That is why we will limit ourselves to one, but the most popular intrinsic metric.

5.1.1 Silhouette Score

Silhouette score (Rousseeuw (1987)) is probably the most popular intrinsic evaluation metric (Naik et al. (2015), Lossio-Ventura et al. (2021)). It is computed as follows:

$$a(i) = \frac{\sum_{j \in C_I, i \neq j} d(i,j)}{|C_I| - 1}$$

In other words, it is the mean distance between sample i and all other samples between clusters. Then the mean dissimilarity of sample i to some cluster C_J (where $C_I \neq C_J$) is computed as:

[1] Curse of dimensionality refers to the challenges and issues that arise when working with high-dimensional data (sparsity of data, computational instability, all distances approximating the same value and so on).

$$b(i) = \min_{J \neq I} \frac{\sum_{j \in C_J} d(i,j)}{|C_J|}$$

Then we can finally define a silhouette score of one sample i:

$$s(i) = \begin{cases} \frac{b(i)-a(i)}{\max a(i), b(i)}, & \text{if } |C_I| > 1 \\ 0, & \text{if } |C_I| = 1 \end{cases} \tag{5.1}$$

To get the silhouette score of a dataset, we average the silhouette score over all samples. Values range from -1 to 1. Higher values often indicate good clustering.

5.2 Extrinsic Metrics

Extrinsic metrics allow for a much better analysis of the resulting clusters. They can be used freely to compare a wide variety of models (clustering in different embedding spaces).

5.2.1 Adjusted Rand Index

Adjusted rand index (ARI) is probably the most popular extrinsic clustering evaluation metric (Lossio-Ventura et al. (2021), Ahmed et al. (2023)). ARI is adjusted for the chance rand index (RI). The latter can be computed as:

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

in terms of true positives and false negatives pairs. Adjusted rand index:

$$ARI = \frac{RI - E(RI)}{\max(RI) - E(RI)}$$

Even from the formula above we can see similarity with accuracy. In fact, this is what it is, the main difference is that with ARI for computing pairs are used and it is not necessary to know the mapping between ground truth clusters and the model's clusters.

5.2.2 Adjusted Mutual Information

Adjusted mutual information (AMI) is an adjustment of the mutual information (MI) score to account for chance. It accounts for the fact that the MI is generally higher for two clusterings with a larger number of clusters, regardless of whether there is actually more information shared. It can be computed in a similar fashion as ARI:

$$ARI = \frac{MI(U,V) - E(MI(U,V))}{\max(H(U), H(V)) - E(MI(U,V)}$$

AMI ranges from 0 to 1 (the higher is better).

5.2.3 V-Measure, Homogeneity and Completeness

V-measure, homogeneity, and completeness in clustering are in some sense similar to F-measure, recall, and precision in classification.

A clustering result satisfies homogeneity if all of its clusters contain only samples that are members of a single class. The metric is independent of the absolute values of the labels (as well as other extrinsic clustering metrics): a permutation of the cluster labels will not change the score value in any way.

A clustering result satisfies completeness if all the samples that are members of a given class are elements of the same cluster. The metric can also be computed by swapping inputs for homogeneity metric.

V-measure is then defined as their harmonic mean:

$$\text{V-measure} = \frac{2 * \text{homogeneity} * \text{completeness}}{\text{homogeneity} + \text{completeness}}$$

that can also be biased into one of two metrics using additional parameter *beta*, but we will not use it, because there are no reasons to prefer one metric over another.

5.3 Classification Metrics

5.3.1 Label Matching

In general, there is no point in applying classification metrics to clustering, because clustering labels do not carry any meaning. These are just designations that if you mix them up, nothing will change. But if we visualize assignments, for example using a contingency matrix[2], we can notice a pattern. When clusters are initially highly separated and have few common samples, it is relatively easy to map clustering labels to ground truth labels.

In the figure 5.1 we can clearly see that *cluster 1* label corresponds to the topic *airline support*. Any other assignments make no sense due to minimum overlap. There is no need to start from scratch, as the assignment problem[3] is already solved. Hungarian algorithm solves the problem in polynomial time, which we utilize in *evaluation.match_cluster_labels* using *scipy*[4] solver.

```
def match_cluster_labels(clusters0, clusters1):
    mapping = linear_sum_assignment(
        -sklearn.metrics.cluster.contingency_matrix(clusters0, clusters1)
    )[1].tolist()  # mapping from classes to clusters
    reversed_mapping = [0] * len(mapping)  # mapping from clusters to classes
    for i, v in enumerate(mapping): reversed_mapping[v] = i
    return [reversed_mapping[c] for c in clusters1]
```

[2]Table that displays the multivariate frequency distribution of the variable.

[3]Special type of linear programming problem where the objective is to minimize the cost or time of completing a number of jobs by a number of persons.

[4]https://scipy.org

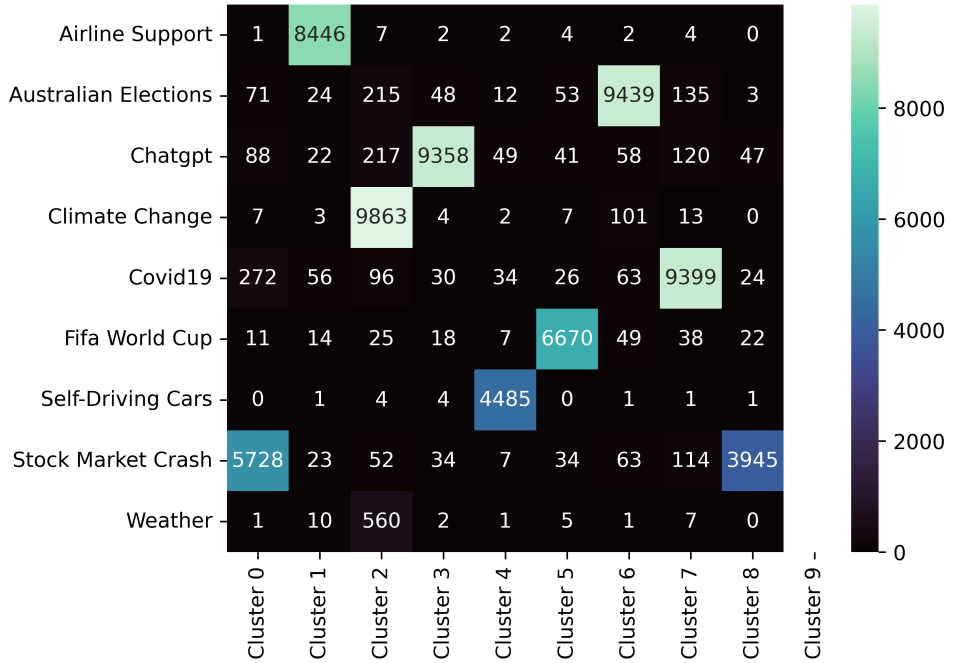

Figure 5.1: Contingency matrix based on distilroberta+umap10+kmeans pipeline clusterization. Columns correspond to ground truth, rows to the assigned cluster label.

5.3.2 Accuracy

Accuracy is a fairly simple metric that shows how accurate the classifier is. It is one of the most common metrics in classification tasks. In multiclass classification accuracy is defined as follows:

$$\text{Accuracy} = \frac{\text{number of correct classifications}}{\text{number of all classifications}}$$

However, accuracy is far from being the best metric, especially when the classes are not balanced. After all, a class with more samples will have a much higher weight. We are talking about the so-called accuracy paradox[5]. A classifier can have a very high accuracy in general, but a terrible precision in a poorly represented class. That is why we do not stop at accuracy and go further.

5.3.3 F1 Score, Precision and Recall

Precision is a measure of the accuracy of the positive predictions made by a model. In terms of true-positives and false-negatives counts it is defined as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

[5]Paradox that higher accuracy does not necessarily result in higher predictive performance.

High precision indicates that the model has a low rate of false positives. In other words, when the model predicts a positive outcome, it is more likely to be correct.

Recall accordingly is defined as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

A high recall indicates that the model is good at identifying positive samples among all the actual positives.

Precision and recall are often in tension with each other; improving one may come at the cost of the other. So here comes the F1 score, a harmonic mean of precision and recall:

$$F1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Those metrics are computed per class. So we need to aggregate them to one number. There are multiple choices. For example, weighted by cardinality of classes aggregation of recall will result in accuracy. Simple averaging seems to be the most appropriate aggregation approach for this work. Because we want all classes (topics) to be equally represented by the metric. Otherwise, in the case of weighted aggregation, larger classes will dominate smaller ones.

5.3.4 Minimum Precision and Minimum Recall

Previous metrics may miss one case. When one small class is almost completely misclassified. On some metrics, this will appear more, but overall it will still not be very noticeable. That is why we will use the worst (minimum) recall and the worst (minimum) precision in the class in addition to other metrics. They will immediately show if some class was underrepresented in previous metrics.

5.4 Time

Some models may have the best performance, but will it make sense if they take an order of magnitude longer to run? To handle this, the running time will be measured for each model. Since some models are pre-trained and others require no training, it makes sense to call this time running time rather than training time. It will help you evaluate models from a completely different perspective.

5.5 Notes

It is worth noting that all metrics (except time) range from 0 to 1 and higher values indicate better performance. It is no coincidence. During the testing phase of the whole project, many more metrics were selected. For example, such as CHI (Calinski–Harabasz index), DBI (Davies–Bouldin

index), FMI (Fowlkes-Mallows index), PMF (pair-counting F-measure), and so on. But they were all excluded for different reasons. Some are simply not very informative, others are too correlated with existing metrics.

Chapter 6

Data

6.1 Source Datasets

Individual datasets have been collected using freely available sources.

- **airline support:** Airline Twitter Sentiment (data.world, 2015, 14640 tweets)

  Tweets mentioning different airline companies. Mainly to get help or give feedback.

  Example: "@United Wonder why people hate dealing with airlines? Ridiculous and inflexible "policies". I need a phone number and a resolution. Now."

- **australian elections** Australian Election 2019 Tweets (Kaggle, 2019, 182748 tweets)

  Example: "I did not vote for the liberal party ... I never normally share who I vote for ... I care about Australia and our planet and our future... climate change was the biggest concern I have and liberals will not deliver on that. I'm very concerned for our country #ripaustralia"

- **chatgpt:** 500k ChatGPT-related Tweets Jan-Mar 2023 (Kaggle, 2023, 493745 tweets)

  Tweets that contain hashtags and mentions about ChatGPT.

  Example: "@famous_dyl Love how everyone is doing these GPT challenges. I've actually modified pinescript trading scripts using chatGPT that made great trades."

- **climate change:** Twitter Climate Change Sentiment Dataset (Kaggle, 2019, 43943 tweets)

  This dataset aggregates tweets pertaining to climate change collected between Apr 27, 2015 and Feb 21, 2018. Each tweet is labeled independently by 3 reviewers. This dataset only contains tweets that all 3 reviewers agreed on (the rest were discarded)

Example: "@bakerlarry84 @tedcruz There are enough evidence to proof global warming exist. The clowns in the republican party don't believe in global"

- **covid19:** Coronavirus tweets NLP - Text Classification (Kaggle, 2020, 44955 tweets)

  Example: "@JohnnyKirsch Wells Fargo is committed to helping customers experiencing hardships due to COVID-19. Customers can call 1-800-219-9739 to speak with a trained specialist about options available for their consumer lending, small business and deposit product"

- **fifa world cup:** FIFA World Cup 2022 Tweets (Kaggle, 2022, 22360 tweets).

  The dataset includes tweets in English containing the hashtag #WorldCup2022.

  Example: "Qatar lied and cheated their way to host a World Cup. There is nothing for them to be proud of #Worldcup2022 #WorldCup"

- **self-driving cars:** Sentiment Self-driving Cars (data.world, 2015, 7156 tweets).

  Example: "Driverless cars are not worth the risk. Don't want to be on the highway when the server crashes #SadMacFace #BlueScreenofDeath".

- **stock market crash:** Huge crash in the Stock market 2022 (Kaggle, 2022, 33164 tweets).

  Example: "Hopefully this #stockmarketcrash ain't too bad. At least it's a good discount on some stonks like #GME. I can't wait to add to my portfolio".

- **weather:** Weather sentiment evaluated (data.world, 2015, 1000 tweets).

  Example: "Love this Iowa weather that goes from warm and sunny to rainy and lightening in less than 2 seconds:):):)".

The topics were not chosen at random. For the most part, they are disjunctive, have little in common, and it is not so easy to find a tweet that would belong to two topics at once. This is important during evaluation, as it removes ambiguity when using ground truth metrics. That is, it is possible to uniquely associate each tweet with a specific topic in most cases.

6.2 Noise Removal

Some of the original datasets contain a huge number of tweets. They were collected using just hashtags, without manual verification. As a result, you can often find languages other than English, encoding errors, tweets that do not carry any meaning or do not belong to any topic. This will significantly worsen the evaluation, so cleaning up noise tweets is necessary.

```
a = trim_dataset_by(
    df,
    ascending=True,

    # sort_by='length',
    length=(70, None),

    # sort_by='word fraction',
    word_fraction=(0.7, None),

    # sort_by='token length',
    token_length=(3.5, 8),

    sort_by='english score',
    english_score=(1000, None),
)
a[:60]
```

```
samples: 21289, reduction: 87.10%
```

	id	text	length	word fraction	token length	english score
118115	1128160964199608320	Liberal candidates thinks only heterosexual couples should be able to adopt, that carbon emissions aren't pollution and that marriage stops domestic violence.\n\nThese deranged candidate are here to do one job. Kill the working class.\n#auspol #ausvotes\nhttps://t.co/I49LmqVcqp	274	0.700730	6.210526	1000.0
45521	1129923506010038272	Morrison is on record as saying labour laws need reform so teenagers can get workplace experience. He's also pro-coal. We'll see kids working in the mines, just like Dickensian England, within 12 months. Mark my words. #auspol	226	0.761062	5.135135	1000.0

Figure 6.1: Dataset pruning.

Outlier removal, as well as generating one large dataset, is contained inside *generate_dataset.ipynb*. To remove noise the following approaches were used (the figure 6.1):

- **Pruning by tweet length:** tweets from each topic were sorted by length and tweets with extreme values were removed. These values are different for different topics; in some places, tweets that are 30 characters long are already noise, but in others they are only starting from 70 characters. Too long ones also sometimes became outliers, but this was less common.
- **Pruning by word fraction:** or each tweet, the proportion of words (tokens that begin with letters of the alphabet) was calculated and outliers were removed. The approach was very effective for pruning, it was good at finding tweets that contained few words or information.
- **Pruning by token length:** the average token length was calculated and, as a rule, tweets with values outside the interval (3, 10) were noise.
- **Pruning by English confidence:** using Google's Compact Language Detector 2 (Ooms (2023)) was calculated confidence if a tweet was written in English. Tweets with scores below 600 were mostly noise and tweets with scores below 800 were quite common. CL2 is one of the fastest language detectors with enough good accuracy.
- **Similar tweet detection:** many of the original datasets contain duplicates. Complete matches are very easy to remove using *the pandas*[1] *drop function*. But there are also many tweets that differ by one character, one mention, or one hashtag. To detect such similar tweets, *the RapidFuzz*[2] *library* function was used. Among all the libraries, one of the few was

[1] https://pandas.pydata.org/
[2] https://github.com/rapidfuzz/RapidFuzz

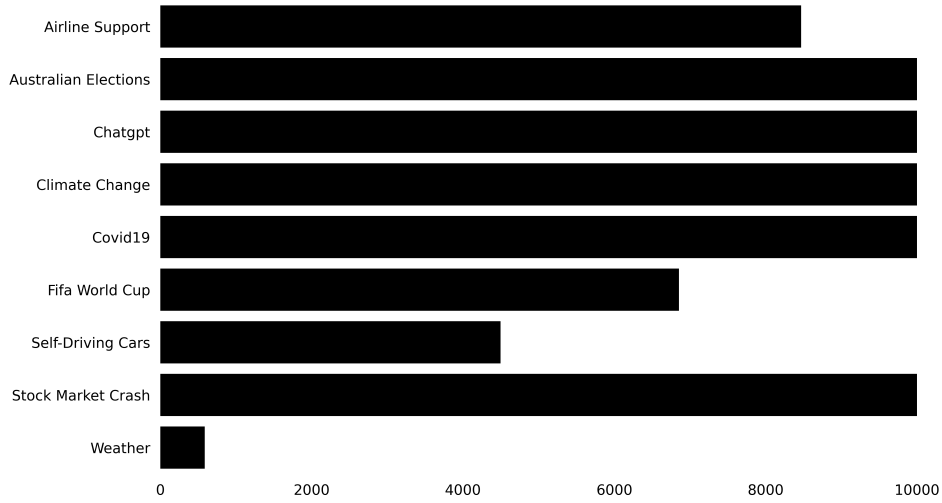

Figure 6.2: Topic distribution in the dataset.

fast enough to run on a large dataset. For each tweet, the similarity with each subsequent (not previous tweet) was calculated and the similarity (directed acyclic graph) was obtained. Tweets with a similarity score higher than 0.8 were removed. This is not the most accurate method, but one of the fastest, because computing similarity is not a cheap operation.

6.3 Dataset Generation

Basic preprocessing was done. All invalid characters were removed and incorrectly decoded characters were replaced, if possible. In datasets where there were too many hashtags of the same type, such as in **stock crash market** or **australian elections**, some hashtags were removed, since otherwise it would be too easy to determine topic. There were a lot of tweets of the same type that began with *rt @* and were actually retweets. If there were too many such tweets (retweets) within topic, the first part containing retweet was removed.

When a significant part of the noise has been removed, the encodings have been corrected, the invalid characters have been fixed - the final dataset can be generated. The size of the dataset is controlled by constants defined in *constants.py*. For evaluation, a dataset with 70406 tweets was generated. The generation can be completely reproduced; one should use seed 0. The distribution of topics was deliberately made unbalanced (the figure 6.2). **weather** topic only contains 587 tweets (compared to **australian election's** 10,000). This brings clustering closer to real-world applications, where not all topics may have thousands of samples.

The final dataset is quite diverse. Some tweets contain almost no hashtags or mentions (**weather**). Some are the complete opposite of this (**stock market crash**, **fifa world cap**). In some topics, mentions do not carry any

information at all, while in others, due to frequent mentions, it is possible to make a forecast based on mentions alone (**airline support**). The final dataset is saved as *data/datased70000.csv* and can be explored in any of all three jupyter notebooks.

Chapter 7

Evaluation

7.1 Setup

Since the use of any qualitative metrics other than intrinsic requires the same number of classes (topics) and clusters (produced by the model), it was decided not to use automatic detecting of the number of clusters where this is allowed (HDBSCAN). Otherwise, the evaluation would be too subjective. It would be necessary to evaluate the clusters using intrinsic metrics that are not comparable along different embedding spaces, visualization, and judge by keywords, and although there are metrics for this, such as topic keyword Coherence score or UMass index, the analysis would still be quite subjective. Therefore, for massive evaluation, the models utilize the number of clusters in advance.

The dataset was pre-processed in *clustering.ipynb*. Spacey's tokenizer and lemmatizer "en_core_web_md" were used to produce the tokenized corpus. All links were replaced with *%link*, numbers with *%number*, words with a document frequency of less than 4 and above 0.5 were removed from the vocabulary. Transformers use their own tokenizers.

All methods were run with minimal parameter fine-tuning to avoid overfitting (usually 1 or 2 parameter tuning except embedding size). The results can be approximately reproduced with *constants.SEED = 0*. Hardware used: A4000 (45 GiB RAM, 8 CPU, 16 GiB GPU).

To save space, abbreviations were used, i.e use = Universal Sentence Encoder, GMM = Gaussian Mixture Model, mrec = Minimum Recall, time = Time in seconds and so on.

It is hard to fit all metrics to PDF width, so some less important metrics were pruned, but the full table can still be found in *evaluation/<name of dataset>*.

7.2 Results

The best performance (the figure 7.1) was shown by **distilroberta+umap5+hdbscan**, which is surprising because MPNet always outperformed DistilRoBERTa on smaller data (10000 tweets) and UMAP is the current state-of-the-art trans-

former.

UMAP has proven to be one of the best state-of-the-art reduction dimensionality techniques. in contrast to SVD, it creates embeddings of a very small size, but of excellent quality, preserving most of the semantic properties and relationships between tweets. It outperforms K-Means most of the time. UMAP with embedding size 5 is a good default choice. Also, UMAP + HBDCSAN often work well together.

use+umap5+kmeans combines excellent performance and high speed (the figure 7.2). If you need speed, Universal Sentence Autoencoder is a good default choice. On the contrary, MPNet is the slowest.

Word22Vec can be a good choice if you do not want to use pre-trained transformers and do the training from scratch.

LDA and Doc2Vec perform just horribly on short texts. No wonder, LDA is based on counting; it requires large numbers. For Doc2Vec to be trained, the corpus and document length are too small.

The figure 7.1 can give a hint which metrics (AMI, homogeneity, completeness and so on) can be excluded without affecting the quality of evaluation.

7.3 Future work

There's a lot of room to continue. Next, one can explore methods with automatic detection of the number of clusters. Even though it is harder to evaluate, but one can use other intrinsic metrics[1] or analyze clusters using topic keywords extraction (KeyBART Kulkarni et al. (2022)) and calculate Coherence or UMass score.

Some other methods can be added for comparison: Deep Amortized Clustering (Lee et al. (2019)) or Tweet2Vec (Vosoughi et al. (2016)).

The *deep_embedded_clustering.py* file contains code for Deep Embeddings Clustering (Xie et al. (2016)), which is really interesting approach to clustering. I have never managed to choose the right architecture and train it with high enough ARI, but this may be due to the fact that there is an error in the code. One might try to fix it, as this method looks very promising.

[1] Calinski–Harabasz index and Davies–Bouldin index

	ari	acc	f1	rec	pre	mpre	ss	h	c	time
ground truth assignment	1.000	1.000	1.000	1.000	1.000	1.000	NaN	1.000	1.000	NaN
distilroberta+umap5+hdbscan	0.914	0.963	0.958	0.959	0.958	0.901	0.775	0.899	0.898	132
distilroberta+umap5+gmm	0.908	0.958	0.937	0.957	0.925	0.605	0.762	0.895	0.890	112
word2vec300+umap5+kmeans	0.903	0.958	0.952	0.951	0.953	0.886	0.795	0.885	0.886	134
word2vec300+umap5+hdbscan	0.903	0.957	0.952	0.951	0.953	0.888	0.793	0.885	0.885	146
word2vec100+umap5+kmeans	0.896	0.954	0.949	0.944	0.954	0.921	0.786	0.876	0.878	104
word2vec100+umap5+gmm	0.896	0.954	0.948	0.944	0.954	0.922	0.786	0.876	0.878	104
mpnet+umap5+hdbscan	0.893	0.946	0.852	0.852	0.852	0.003	0.748	0.882	0.881	186
word2vec50+umap5+kmeans	0.882	0.948	0.938	0.934	0.944	0.872	0.759	0.862	0.863	78
mpnet+umap5+kmeans	0.876	0.926	0.874	0.936	0.877	0.165	0.748	0.887	0.860	171
mpnet+umap5+clara	0.875	0.925	0.872	0.935	0.877	0.160	0.743	0.887	0.859	166
distilroberta+umap10+clara	0.871	0.912	0.836	0.826	0.854	0.000	0.758	0.887	0.863	114
mpnet+umap10+kmeans	0.869	0.923	0.842	0.834	0.854	0.001	0.769	0.878	0.857	172
distilroberta+umap5+clara	0.863	0.901	0.829	0.817	0.850	0.000	0.766	0.885	0.859	112
distilroberta+umap10+kmeans	0.862	0.896	0.826	0.813	0.849	0.000	0.770	0.886	0.858	114
distilroberta+umap5+kmeans	0.862	0.897	0.826	0.813	0.850	0.000	0.765	0.886	0.858	111
mpnet+svd50+kmeans	0.835	0.895	0.828	0.818	0.850	0.010	0.206	0.850	0.823	113
use+svd30+kmeans	0.828	0.923	0.921	0.922	0.920	0.850	0.739	0.819	0.819	13
use+umap5+kmeans	0.828	0.923	0.921	0.922	0.920	0.850	0.739	0.819	0.819	56
tfidf+umap5+kmeans	0.826	0.907	0.856	0.917	0.859	0.157	0.556	0.821	0.795	76
fasttext100+umap5+kmeans	0.825	0.876	0.812	0.796	0.839	0.002	0.677	0.843	0.814	79
tfidf+umap5+clara	0.818	0.901	0.833	0.842	0.843	0.052	0.545	0.813	0.791	78
word2vec20+umap5+hdbscan	0.802	0.876	0.810	0.795	0.835	0.001	0.677	0.816	0.791	139
word2vec20+umap5+kmeans	0.802	0.876	0.810	0.795	0.835	0.001	0.677	0.816	0.791	71
tfidf+svd50+kmeans	0.749	0.867	0.795	0.786	0.814	0.007	0.159	0.782	0.769	3
lda	0.237	0.482	0.438	0.479	0.492	0.039	NaN	0.294	0.297	84
doc2vec100+umap5+kmeans	0.034	0.235	0.216	0.211	0.242	0.001	0.238	0.062	0.062	143
random clustering	0.000	0.110	0.104	0.112	0.110	0.009	NaN	0.000	0.000	NaN

Table 7.1: Pipeline ARI comparison.

	ari	acc	f1	rec	pre	mpre	ss	h	c	time
tfidf+svd50+kmeans	0.749	0.867	0.795	0.786	0.814	0.007	0.159	0.782	0.769	3
use+svd30+kmeans	0.828	0.923	0.921	0.922	0.920	0.850	0.739	0.819	0.819	13
use+umap5+kmeans	0.828	0.923	0.921	0.922	0.920	0.850	0.739	0.819	0.819	56
word2vec20+umap5+kmeans	0.802	0.876	0.810	0.795	0.835	0.001	0.677	0.816	0.791	71
tfidf+umap5+kmeans	0.826	0.907	0.856	0.917	0.859	0.157	0.556	0.821	0.795	76
tfidf+umap5+clara	0.818	0.901	0.833	0.842	0.843	0.052	0.545	0.813	0.791	78
word2vec50+umap5+kmeans	0.882	0.948	0.938	0.934	0.944	0.872	0.759	0.862	0.863	78
fasttext100+umap5+kmeans	0.825	0.876	0.812	0.796	0.839	0.002	0.677	0.843	0.814	79
lda	0.237	0.482	0.438	0.479	0.492	0.039	NaN	0.294	0.297	84
word2vec100+umap5+kmeans	0.896	0.954	0.949	0.944	0.954	0.921	0.786	0.876	0.878	104
word2vec100+umap5+gmm	0.896	0.954	0.948	0.944	0.954	0.922	0.786	0.876	0.878	104
distilroberta+umap5+kmeans	0.862	0.897	0.826	0.813	0.850	0.000	0.765	0.886	0.858	111
distilroberta+umap5+gmm	0.908	0.958	0.937	0.957	0.925	0.605	0.762	0.895	0.890	112
distilroberta+umap5+clara	0.863	0.901	0.829	0.817	0.850	0.000	0.766	0.885	0.859	112
mpnet+svd50+kmeans	0.835	0.895	0.828	0.818	0.850	0.010	0.206	0.850	0.823	113
distilroberta+umap10+clara	0.871	0.912	0.836	0.826	0.854	0.000	0.758	0.887	0.863	114
distilroberta+umap10+kmeans	0.862	0.896	0.826	0.813	0.849	0.000	0.770	0.886	0.858	114
distilroberta+umap5+hdbscan	0.914	0.963	0.958	0.959	0.958	0.901	0.775	0.899	0.898	132
word2vec300+umap5+kmeans	0.903	0.958	0.952	0.951	0.953	0.886	0.795	0.885	0.886	134
word2vec20+umap5+hdbscan	0.802	0.876	0.810	0.795	0.835	0.001	0.677	0.816	0.791	139
doc2vec100+umap5+kmeans	0.034	0.235	0.216	0.211	0.242	0.001	0.238	0.062	0.062	143
word2vec300+umap5+hdbscan	0.903	0.957	0.952	0.951	0.953	0.888	0.793	0.885	0.885	146
mpnet+umap5+clara	0.875	0.925	0.872	0.935	0.877	0.160	0.743	0.887	0.859	166
mpnet+umap5+kmeans	0.876	0.926	0.874	0.936	0.877	0.165	0.748	0.887	0.860	171
mpnet+umap10+kmeans	0.869	0.923	0.842	0.834	0.854	0.001	0.769	0.878	0.857	172
mpnet+umap5+hdbscan	0.893	0.946	0.852	0.852	0.852	0.003	0.748	0.882	0.881	186
ground truth assignment	1.000	1.000	1.000	1.000	1.000	1.000	NaN	1.000	1.000	NaN
random clustering	-0.000	0.111	0.104	0.110	0.111	0.008	NaN	0.000	0.000	NaN

Table 7.2: Pipeline speed comparison.

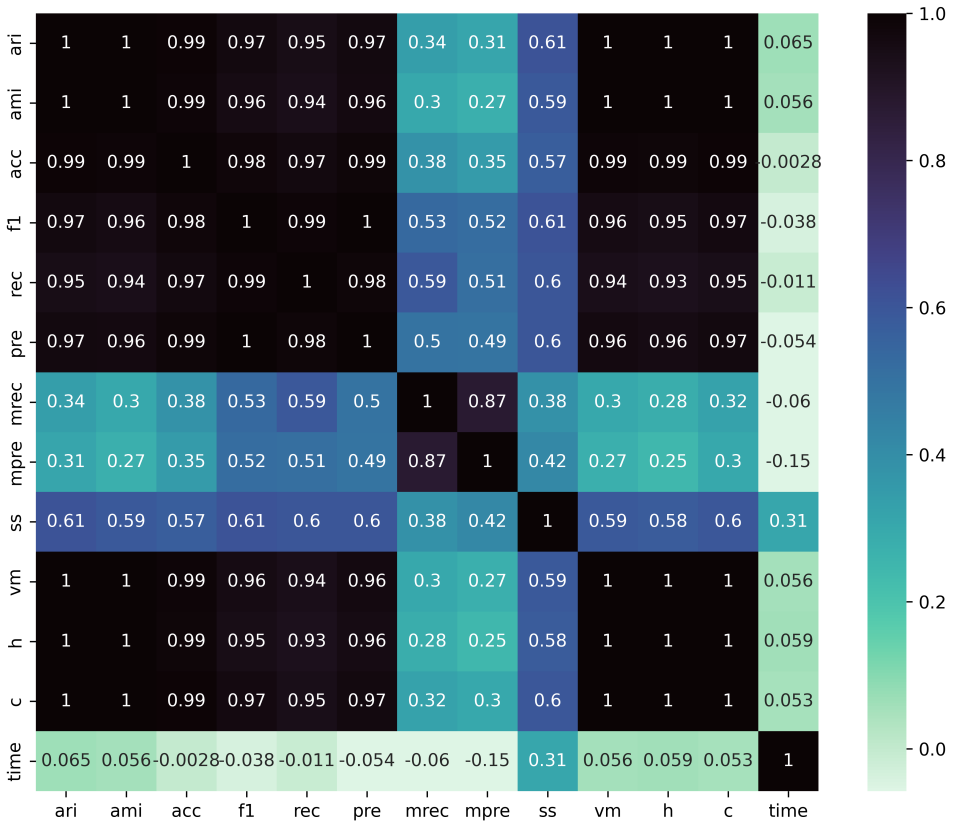

Figure 7.1: Metric correlation heatmap.

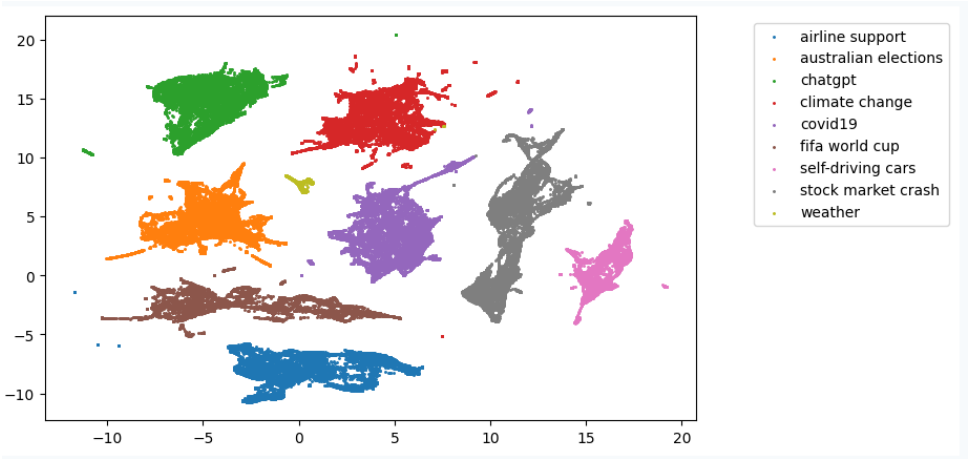

Figure 7.2: Clusters of the best model (distilroberta+umap5+hdbscan) projected into 2 dimensions with UMAP.

Bibliography

Ahmed, M. H., Tiun, S., Omar, N., and Sani, N. S. (2023). Short text clustering algorithms, application and challenges: A survey. *Applied Sciences*, 13(1).

Arthur, D. and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA. Society for Industrial and Applied Mathematics.

Blei, D., Ng, A., and Jordan, M. (2001). Latent dirichlet allocation. volume 3, pages 601–608.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information.

Cer, D., Yang, Y., yi Kong, S., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y.-H., Strope, B., and Kurzweil, R. (2018). Universal sentence encoder.

Iyyer, M., Manjunatha, V., Boyd-Graber, J., and Daumé III, H. (2015). Deep unordered composition rivals syntactic methods for text classification. In Zong, C. and Strube, M., editors, *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1681–1691, Beijing, China. Association for Computational Linguistics.

Köppen, M. (2000). The curse of dimensionality. In *5th online world conference on soft computing in industrial applications (WSC5)*, volume 1, pages 4–8.

Kulkarni, M., Mahata, D., Arora, R., and Bhowmik, R. (2022). Learning rich representation of keyphrases from text.

Lee, J., Lee, Y., and Teh, Y. W. (2019). Deep amortized clustering.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Lossio-Ventura, J. A., Gonzales, S., Morzan, J., Alatrista-Salas, H., Hernandez-Boussard, T., and Bian, J. (2021). Evaluation of clustering and topic modeling methods over health-related tweets and emails. *Artificial Intelligence in Medicine*, 117:102096.

McInnes, L., Healy, J., and Melville, J. (2020). Umap: Uniform manifold approximation and projection for dimension reduction.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality.

Naik, M. P., Prajapati, H. B., and Dabhi, V. K. (2015). A survey on semantic document clustering. In *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–10.

Ooms, J. (2023). cld2: Google's compact language detector 2. R package version 1.2.4.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. volume 14, pages 1532–1543.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. (2020). Mpnet: Masked and permuted pre-training for language understanding.

Stieglitz, S., Mirbabaie, M., Ross, B., and Neuberger, C. (2018). Social media analytics – challenges in topic discovery, data collection, and data preparation. *International Journal of Information Management*, 39:156–168.

van der Maaten, L. and Hinton, G. (2008). Viualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

Vosoughi, S., Vijayaraghavan, P., and Roy, D. (2016). Tweet2vec: Learning tweet embeddings using character-level cnn-lstm encoder-decoder. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, SIGIR '16. ACM.

Xie, J., Girshick, R., and Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.