Analyzing Research Conferences using Bert

Eric Chen
Harvey Mudd College
Claremont, CA
`erchen@hmc.edu`

1 Project Introduction

As machine learning and related fields grow, the amount of research output has already increased dramatically. Just this year (2023), over 58,000 research papers related to machine learning has been published according to Google Scholar. This makes finding papers related one's interest more and more difficult. In this project, I investigate the use of BERT for paper recommendation and topic modeling to help researchers identify papers relevant to their interests. To accomplish this, I apply unsupervised methods, such as K-Means, Mixture of Gaussians, and DBSCAN to cluster word embeddings generated by BERT. The resulting clusters can be used to provide recommendations related to user input as well as survey the topics of a research conference. In section 2, I will provide a brief background of recommendation models and vector embeddings. In section 3, I will describe my methods. And finally, in section 4, I provide some results and analysis of this investigation.

2 Related Works

Recommendation models often split into two categories: collaborative and content based filtering. Collaborative filtering is based on the idea that similar users have similar preferences. It creates a profile for each user based on user behavior and provides recommendations using the objects that users with similar profiles like. Popular examples of collaborative filter include Netflix (1), Spotify (2), and YouTube (3). In these cases, the user's watch histories and custom playlists are collected, and new content is recommended based on what similar users enjoy. Content based filtering finds items similar to items that the user has previously shown interest in. It relies more on finding similarity between items rather than users. Examples of content based filtering can be found in the algorithms of Spotify (2) and Amazon (4). In these cases, Costumers have the option to give specific items a like. The recommender finds new items similar to list of liked items. While there is a distinction between the two types of algorithms, modern applications often use both together. For this project, we chose to only use Content Based filtering because we have a readily access of research papers but not to user preferences. Therefore, we do not have enough data to perform a collaborative filter.

On the other hand, it is straightforward to find similarity between research papers through vectorization. Vectorizing text has been a core goal and technique of Natural Language Processing. One of the early successes was Word2vec (5) in which individual words are converted into vector embeddings. The result of Word2vec include being able to tell subtle relationship between words, which is best exemplified by "X = vector("biggest") - vector("big") + vector("small")" and finding the closest word to X is "smallest" (5). While Word2vec generates vectors from singular words, BERT can also incorporate the context of the word using its attention mechanism (6). Meaningful vectors can be used for recommendation and topic modeling. For example, Yahoo developed the prod2vec model that vectorizes receipts in user's inbox for the purpose of recommending products (7). And in the area of topic modeling, document embeddings can be clustered, and the centroid of the cluster can be used to finds words that are most relevant to the documents in the cluster (8). These applications give confidence that BERT vectors could also be used for recommendations.

3 Methods

The recommendation method can be broken into four major steps. First, we collect a set of research papers that we can use for content-based filtering. Secondly, we need to vectorize each research paper. Then, we group similar vectors into clusters based on the assumption that similar papers will have similar vector embeddings. The merits of this assumption will be discussed in the results section. To facilitate clustering, we must also perform dimensional reduction. Finally, using the clusters, we can either perform topic modeling or recommendations.

3.1 Data Collection

To recommend papers, we must first create a database of research papers. I do so by scraping the NeurIPS conference website. NeurIPS is one of the most prestigious machine learning conference, meaning the accepted papers are high quality and likely written by experienced writers. This means, the abstract will include useful information, and provide a good indication of the paper's topic. Furthermore, the NeurIPS conference is specifically for machine learning research, so I can expect the papers to be related to each other. This will make creating reasonable clusters of the papers more likely.

Since the NeurIPS website is static, I can simply use python's beautiful soup library to scrape its content. I collected every paper from the NeurIPS 2018, 2019, and 2022 sites. And for each paper, I recorded the paper title, authors, abstract, and year into a dataframe. When I proceeded to cluster the paper abstracts without any data cleaning, I found that papers with websites urls and latex symbols strongly influence the BERT embedding system and distort the clustering process. Only a few papers mentioned website urls, so I simply removed them from the dataset. In the remaining papers, I removed all latex symbols by removing all terms matching the '$.+?$' regular expression. This provided me with a dataset of 4507 papers.

3.2 Generate Abstract Embeddings

Since papers are non-numerical data, clustering algorithms cannot compare papers directly. Therefore, we use BERT to convert papers into a vector representation. These vectors will then be used as the foundation for clustering and recommendation in later steps.

BERT is an NLP model made with the goal of having a generic pretrained model that could be successfully fine tuned more many different NLP tasks (6). It was pretrained on a large vocabulary from the WordPiece dataset and Wikipedia (6). And through this process, BERT has learned to vectorize input text in a semantically meaning manner. The inputs to the BERT model is a series of text represented in tokens format. BERT maps each token to a 768 dimensional latent space, which encodes both the token's common meaning and the specific context. Since my goal is to generate one vector for each of my abstract, I tokenize my abstract, cutting out tokens exceeding the input token limits, and receive a 768 dimensional vector for each token. Then, I average all the output tokens into a single 768 dimensional vector, which serves as the vector representation of each paper's abstract.

3.3 Clustering

Having generated vector representations for each paper in the database, we are ready to perform the clustering step. There are many clustering algorithms, each with their advantages and disadvantages. For this application, I choose to use K-Means, Mixture of Gaussian, and DBSCAN.

K-means is an iterative clustering method that seeks to find the center of the clusters using the mean. K-means first makes random predictions of where the cluster centers are. On each iteration of the algorithm, K-means assigns each data point to the cluster center with the smallest euclidean distance.

At the end of each iteration, K-means updates the cluster centers using the mean of the data points in each cluster. This simple approach means that K-means is relatively faster than other clustering methods.

A similar clutsering method is Mixture of Gaussians. Mixture of Gaussians is an expectation maximization algorithm. Each cluster in a mixture of Gaussian model is seen as a Gaussian distribution. Each cluster has three parameters – the mixing coefficient, mean, and covariance. For a cluster k, the mixing coefficient π_k represents the prior probability of a data point belonging to the cluster. The mean coefficient μ_k and covariance Σ_k are simply the mean and covariance of each cluster. As an expectation maximization algorithm, there are two stages to every iteration of the Mixture of Gaussian algorithm. In the expectation stage, we calculate the probability of each data point being in a particular cluster/distribution using the Bayes rule. Formally, $w_{i,k}$, which is the probability of data point i being in cluster k, is

$$w_{i,k} = \frac{\pi_k * N(x_i|\mu_k, \Sigma_k)}{\Sigma_{j=1}^{K} \pi_j N(x_i|\mu_j, \Sigma_j)}.$$

In the Maximization step, the cluster parameters are recalculated using the new $w_{i,k}$ in the following manner,

$$\mu_k = \frac{\Sigma_{i=1}^{N} w_{i,k} x_i}{\Sigma_{i=1}^{N} w_{i,k}}$$

$$\Sigma_k = \frac{\Sigma_{i=1}^{N} w_{i,k}(x_i - \mu_k)(x_i - \mu_k)^T}{\Sigma_{i=1}^{N} w_{i,k}}$$

$$\pi_k = \frac{\Sigma_{i=1}^{N} w_{i,k}}{N}$$

. The algorithm continues in this iterative manner until the log-likelihood of the observed data is sufficiently low. The covariance parameter of Gaussian distributions allows mixture of Gaussians to cluster more varied shapes of data than K-means. This makes mixture of Gaussian a more flexible, albeit more computationally intensive, algorithm than K-means.

My final clustering algorithm is DBSCAN. DBSCAN is a density based clustering algorithm and it does not take a number of cluster parameter. In my experiments, I use a more sophisticated type of DBSCAN called HDBSCAN, which can determine density parameters itself. The only parameter I specify is that a minimum cluster must contain at least 15 data points. HDBSCAN searches through the data points to find dense regions, which it considers a cluster. The less dense regions are ignored and classified as noise. This means that unlike the previous two methods, DBSCAN will not assign every paper to a cluster. This approach is a good alternative because some research papers may not relate to other papers, particularly if it is a especially novel.

3.4 Dimensional Reduction

Clustering algorithms often suffer from the curse of dimensionality. The curse of dimensionality refers to the phenomenon where the sense of distance between two points become more distorted in higher dimensional space. All three of the clustering algorithm suffers from the curse of dimensionality and the data points we attempt to cluster have 768 dimensions, so this means we must perform dimensionality reduction first. We use the Umap algorithm for dimensional reduction. Umap is a nonlinear dimensional reduction algorithm that aims to preserve the overall structure in the data. Unlike PCA, which is linear, Umap is more suited for performing dimensionality reduction on data, whose dimensions could be related. Since we make very little assumptions about the dimensions of the BERT latent space, Umap seems to be a more appropriate dimensionality reduction algorithm.

3.5 Topic Modeling

Topic Modeling serves as a valuable application of the clusters we create as well as a good tool to check the quality of the clusters. We perform topic modeling for each clustering using the Class based Term Frequency Inverse Document Frequency method (c-TF-IDF) inspired by `https://towardsdatascience.com/topic-modeling-with-bert-779f7db187e6`. TF-IDF is a popular method in text based data mining to assign importance weights to words in a document. Words with a high TF-IDF scores are meant to differentiate their documents from other documents. The TF-IDF score for a single word in document i can be calculated as follows,

$$\text{Term Frequency} = \frac{t_i}{w_i}$$
$$\text{Inverse Document Frequency} = \log \frac{m}{\Sigma_j^n t_j}$$
$$\text{TF-IDF} = TF(i) \times IDF(i),$$

where t_i is the number of occurrence of the term in document i; w_i is the total number of all terms in document i; and m is the total number of documents. In essence, the term frequency of a word calculates how popular a term is in its document and the inverse document frequency calculates how unique the term is in the other documents. The c-TF-IDF score uses the same equations but views each cluster as a single document. Therefore, to calculate the c-TF-IDF score for each term, we first combine all abstracts in a cluster into one large document. Then, the TF-IDF score calculated using these cluster based documents is the c-TF-IDF score. Finally, we use the terms with the highest c-TF-IDF score in each cluster to represent the topic of their respective cluster.

3.6 Generate Recommendations

We generate recommendations in three step – vectorization, cluster assignment, and calculating cosine similarity scores with other abstract embeddings. The expected user input is some description of the topic of interest of length less than 512 tokens. This user input is vectorized using BERT using a process identical to the abstract vectorization process. The resulting vector is converted to the appropriate dimensions and clustered using a fitted clustering algorithm. We expect the assigned cluster to contain similar topics as the user input. The specific papers that are most related to the user input will be chosen based on cosine similarity. The cosine similarity score is a common way to measure distance between two vectors, and it is calculated using the following formula,

$$\frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \times ||\mathbf{B}||}$$

, where **A** and **B** are both vector representation of abstracts.

The cosine similarity score varies from -1 to 1. A score of -1 represents opposite topics, 0 represents no similarity, and 1 represents perfect match. Therefore, papers in the cluster with the highest cosine similarity as the user input are recommended.

4 Experiments and Results

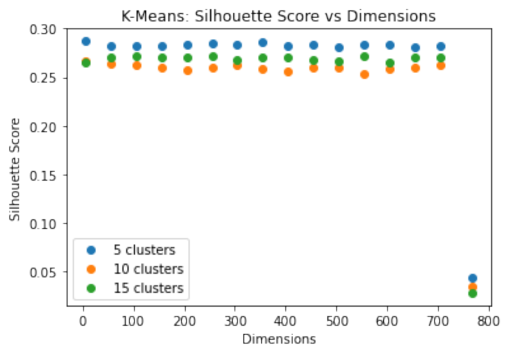
4.1 Clustering

I experimented with 16 dimensions and 3 numbers of clusters for K-Means, Mixture of Gaussian, and DBSCAN. DBSCAN does not take a number of clusters parameter, so the only variation between the runs is the dimension of the inputs. The data's original dimension is 768, and all dimensional reduction is done using the Umap algorithm. The models are evaluated using the Silhouette score at the dimension of the input data, which is represented as

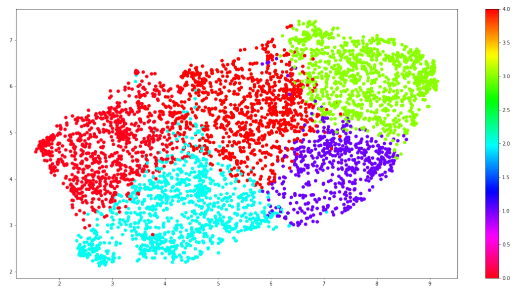
$$\frac{\Sigma_{i=1}^{N}(b_i - a_i)/max(a_i, b_i)}{N}$$

, where a_i is the average distance between the current data point and the other points in its cluster; b_i is the average distance between the current data point and other points outside of its cluster; and N is the total number of data points. Figure 1 shows the silhouette score for all the models I trained as well as the clusters produced by the best model of each algorithm.

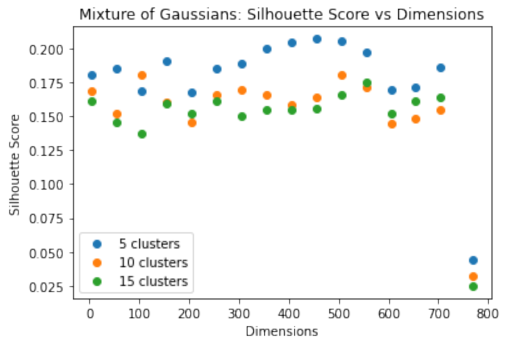
The left column of Figure 1 displays the visual results of the clustering models after dimensional reduction using U-map. The highest Silhouette score was 0.565, achieved by DBSCAN (22 cluster;



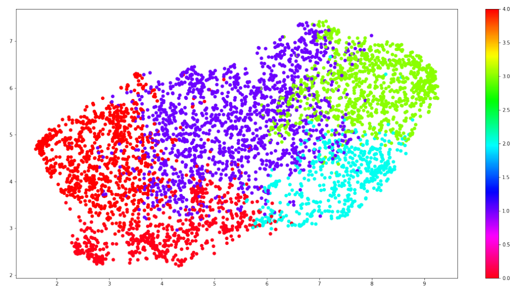
(a) K-Means Silhouette Score



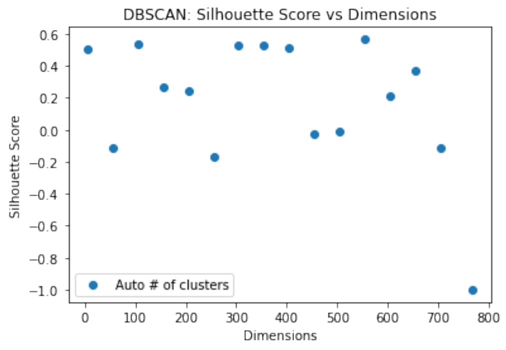
(b) K-Means Clusters



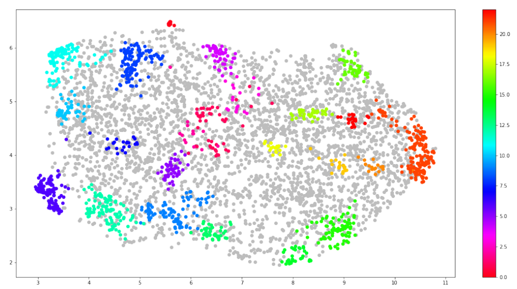
(c) Mixture of Gaussians Silhouette Score



(d) Mixture of Gaussians Clusters



(e) DBSCAN Silhouette Score



(f) DBSCAN Clusters

Figure 1: quantitative and visual results

555 dimensions). In general, the best K-means models followed second in Sillhouette scores and the best Mixture of Gaussians was third. The worst model of each of the clustering algorithm is the one fitted on data without any dimensional reduction. This is expected because all three algorithms are affected by the curse of dimensionality. The effects of the curse of dimensionality at lower dimensions are unclear. We notice a significant boost in performance after just slight dimensional reduction from 786 to 755 using U-Map. There are also no noticeable trends between silhouette scores and the dimension of the data that suggest the curse of dimensionality is at play. Therefore, I hypothesize that U-map not only performs dimensionality reduction but also highlight specific structures in the original data. If this is true, then it would explain why all three algorithms had a much easier time clustering relatively high dimensional data after U-map was applied. Regardless of how much impact U-map has on the structure of the data, it is important to note that these results were not generated solely by the clustering algorithms but by a combination of U-map and the clustering algorithms.

The best result for K-means occurs with a 5 clusters and a data dimension of 5. This combination reached a sillhouette score of 0.287. However, K-Means is a stochastic algorithm and the performance of K-Means appears similar across all dimensions, so it is possible that the best model changes with repeated runs. While the number of dimension does not affect K-Means (after preprocssing with U-map), we notice that K-Means consistently performs the best with 5 clusters. In the visual result, we see that the data points (reduced directly from 768 dimensions to 2 using U-map) are all placed relatively close to each other. If we assume this is representative of the original data structure, then there does not appear to be a clean boundary between the data points. Therefore, a lower number of clusters does appear to be the most reasonable choice for this data distribution. Note that the lack of clear boundaries in 2-dimensions does not neccessarily mean there is no difference between the data points. It is possible that the boundaries are sharper at a higher dimension. Alternatively, the lack of obvious boundaries could indicate the existence of many interdisciplinary papers (for example papers involving Reinforcement Learning as well as optimization).

The Mixture of Gaussian Results are more varied than K-Means. The best Mixture of Guassians model uses 5 clusters and a data dimension of 405. It achieved a silhouette score of 0.207. Similar to K-Means, 5 clusters appears to outperform 10 and 15 clusters. In the visual results, we notice that the clusters have a lot more overlap than in K-Means. This is because Mixture of Gaussians assigns each data point a probability of being in each cluster. Therefore, there might be points that have a high likelihood of being in multiple clusters. This characteristic of Mixture of Gaussians likely lowered its silhouette score, which rewards clusters with clearer boundaries.

Finally, we examine the results for DBSCAN. The best performing DBSCAN model achieved a silhouette score of 0.565, and it used 22 clusters with a data dimension of 555. Unlike the other two algorithms, DBSCAN appears to be heavily affected by the dimension of the data. DBSCAN is sensitive to the density of the data points. And it is possible for certain distributions of data, generated by Umap, to have different levels of structure. This dependency on Umap might explain why DBSCAN works better at certain dimensions than others. Looking at the visual results, we see that DBSCAN only clustered pockets of data it deemed most coherent. It classified most of the data points as outliers. Since DBSCAN only labeled the clearest clusters, it achieved very high silhouette scores.

5 Topic Modeling

For each clustering algorithm, we select the clusters that received the highest silhouette score. Using each cluster, we calculate the c-TF-IDF scores and rank terms by their c-TF-IDF scores. The top keywords from a few clusters are displayed in 2, and additional results for the top five largest clusters generated by each clustering method can be found in the Appendix.

We noticed that the keywords in the K-Means and Mixture of Gaussians cluster are very similar and that the DBSCAN clusters are more specific than the other two's. The similar performance between K-Means and Mixture of Gaussians is reasonable because they both separate the data into five clusters, and visually, the clusters appear to be in the same locations. On the other hand, DBSCAN benefits

Cluster 4 | 1089 Entries | K-Means
('agent', 0.006468448828225028),
('policy', 0.006113569850090786),
('reinforcement', 0.006004967972340869),
('tasks', 0.005998280456614235),
('rl', 0.005951113315221767),
('task', 0.005731366658491771),
('human', 0.005499233558366051),
('adversarial', 0.005492413901314562),
('agents', 0.005173888345253284),
('training', 0.005018854938789691)

Cluster 1 | 782 Entries | Mixture of Gaussian
('image', 0.010444141491179395),
('3d', 0.010091245947220783),
('object', 0.008817059826092766),
('images', 0.008022439152341297),
('video', 0.006923081463050266),
('semantic', 0.006456122243901056),
('language', 0.006401342538210889),
('text', 0.006288126526188164),
('art', 0.006004990866839004),
('generation', 0.00575776349690467)

Cluster 20 | 155 Entries | DBSCAN
('bounds', 0.033840245770338874),
('pac', 0.026464211112286164),
('bayes', 0.026372914344051028),
('hypothesis', 0.02491067255293611),
('kl', 0.02128755847470092),
('bound', 0.02054803879476502),
('inequality', 0.02019509049989428),
('distribution', 0.018717332747646434),
('dependent', 0.01640371056081851),
('distributions', 0.016350932459178284)

Cluster 16 | 154 Entries | DBSCAN
('convergence', 0.033921123369093246),
('stochastic', 0.02663225767708544),
('problems', 0.024576100610804025),
('gradient', 0.023545762335121408),
('optimization', 0.02147958165366273),
('decentralized', 0.02078288626661968),
('descent', 0.02070713548953359),
('nmf', 0.018644891573818427),
('monotone', 0.01834525755982332),
('rate', 0.0168362551174108)

Figure 2: Sample Topic Modeling Results. Left is the term and right is the c-TF-IDF. The top 10 most important terms in each cluster as displayed

from the ability to choose its own number of clusters. With 22 clusters, DBSCAN presents a more general overview of keywords. Terms that are overshadowed in the other clusters show up in the DBSCAN topics. For example, Cluster 20 in Figure 2, generated by DBSCAN, contains terms like "pac," "kl," "distributions" which do not show up in the top 10 keywords of any cluster generated by K-Means and Mixture of Gassuians. Not only does DBSCAN provide a greater variety of topics, it also provides more specificity in each cluster. Since it has more clusters and ignores outliers, DBSCAN generates more tight clusters. For instance, DBSCAN places "convergence" and "gradient" into one clusters and "bounds" into another cluster while K-Means and Mixutre of Gaussians group them all together. While all three terms can be related, the connection between gradient descent and convergence is stronger than "bounds" with either term in the machine learning context (This may not be true in a different subject). In these ways, DBSCAN is more flexible algorithm and provides higher quality topic modeling results than K-Means and Mixture of Gaussians. Furthermore, since all clusters appear meaningful, we can conclude that the BERT abstract embeddings do preserve semantic meaning, supporting an earlier design assumption. Otherwise, each cluster would have random keywords.

6 Paper Recommendations

Finally, we share results for paper recommendations. Overall, this recommendation system can generate reasonable recommendations related to the user input, but it does not do so reliably.

Table 1 displays some sample paper recommendations. The titles of the papers are abbreviated for space concerns. The first two rows use the exact user input. The third row uses the abstract of "Reinforcement Learning with Combinatorial Actions: An Application to Vehicle Routing" (21) as the user input.

The results for "Image Generation" can be considered successful. All paper recommendations either relate heavily to image generation or image modeling and rendering. Both K-Means and Mixture of Gaussians placed the user input in an appropriate cluster (with other image related terms). However, DBSCAN classified it as an outlier. For all other results, K-Means and Mixture of Gaussians places

Input	K-Means	Mixture of Gaussian	DBSCAN
"Image Generation"	Cluster 0 1. QC-StyleGan (9) 2. FNeVR (10) 3. Shape, Light (11)	Cluster 1 1. QC-StyleGan (9) 2. FNeVR (10) 3. MoGDE (12)	Cluster -1 1. QC-StyleGan (9) 2. FNeVR (10) 3. Coordinates (13)
"Reinforcement Learning"	Cluster 3 1. Fuzzy Learning (14) 2. Zero-Shot Transfer (15) 3. Semi-Supervised (16)	Cluster 3 1. MetaReg (17) 2. Active Learning (18) 3. An Information- (19)	Cluster -1 1. Interactive Structure (20) 2. Zero-Shot Transfer (15) 3. Semi-Supervised (16)
Reinforcement Lea... (21)	Cluster 2 1. Exponentially Wei... (22) 2. ... Linear Quadratic RL (23) 3. ... Multi-Agent (24)	Cluster 0 1. ... Multi-Agent (24) 2. Thompson Sampling (25) 3. ... Federated Learning (26)	Cluster -1 1. ... Linear Quadratic RL (2 2. Thompson Sampling (25) 3. Robust RL (27)

Table 1: Recommended papers given user input, ranked by cosine similarity. All cosine similarity scores are above 0.999.

the user input in a more relevant cluster than DBSCAN. Since we only compare the user input to embeddings from the same cluster as the user input, a cluster misclassification can dramatically impact paper recommendation. This result suggests that DBSCAN might be too strict for this paper recommendation application. The clusters that DBSCAN identifies are simply too tight to capture most user input. Because DBSCAN is always searching in the outlier pile, this either means that we cannot take advantage of a smaller more specific search space that a cluster would offer or we miss out on closely related papers that are not in the outlier cluster.

While K-Means and Mixture of Gaussians are better than DBSCAN at placing the input in a related cluster, there appears to be a bias towards one cluster. In this case, the clustering methods often place the user input in cluster 3, which contains generic deep learning terms. The broad terms in cluster 3 might contribute to many learning related terms being are placed in this cluster rather than more appropriate clusters (like ones specifically containing reinforcement learning terms). An example of this cluster confusion is the result for the user input "Reinforcement Learning." All the cluster assignments are wrong, and almost none of the recommended papers related to the user input.

Interestingly, when the user input is longer, the clustering methods are more likely to place the user input into the current cluster. This could be because having a longer user input allows BERT to create more context specific embeddings. One might imagine that for a short input like "Reinforcement Learning," BERT relies more heavily on its own understanding of the words. With a longer input, BERT's attention mechanism helps incorporate more context information into the vector. On the same topic of reinforcement learning, using the abstract of (21) helped K-Means and Mixture of Gaussian to find the appropriate cluster. As a result, the recommended papers are almost all related to reinforcement learning.

7 Conclusion

In this project, we vectorized NeurIPS papers using BERT. We used Umap to reduce the dimension of the data and used various clustering methods to group similar abstract embeddings together. From the Silhouette scores of the clustering methods at different dimensions, we found that Umap not only reduces the dimension of the data but also might highlight certain structure in the data. By creating topic models of each cluster, we found that all clusters appear to be semantically meaningful, representing different aspects of machine learning like Computer Vision and Reinforcement Learning. Furthermore, we found that DBSCAN appears to be more useful at topic modeling than K-Means and Mixture of Gaussians because it can find the appropriate number of clusters. This allows DBSCAN to identify more patches in the dataset than the other methods. Finally, we also experimented with paper recommendation using a combination of clustering and cosine similarity. We found that the BERT embeddings do provide enough semantic meaning to find related papers through this method. However, one key weakness was the system can misclassify the user embeddings, leading to poor paper recommendations. Furthermore, DBSCAN often places the user input in the outliers cluster. This observation might suggest that K-Means and Mixture of Guassians are more appropriate for paper recommendations than DBSCAN. Overall, we learned that BERT vectorization is a powerful tool to comparing text and saw the pros and cons of each clustering method.

References

[1] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Transactions on Management Information Systems (TMIS)*, vol. 6, no. 4, pp. 1–19, 2015.

[2] M. Irvine, "De-blackboxing spotify's recommendation engine." `https://blogs.commons.georgetown.edu/cctp-607-spring2019/2019/05/06/music-to-my-ears-de-blackboxing-spotifys-recommendation-algorithm/`. Accessed: 2023-05-23.

[3] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM conference on recommender systems*, pp. 191–198, 2016.

[4] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," *The adaptive web: methods and strategies of web personalization*, pp. 325–341, 2007.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[7] M. Grbovic, V. Radosavljevic, N. Djuric, N. Bhamidipati, J. Savla, V. Bhagwan, and D. Sharp, "E-commerce in your inbox: Product recommendations at scale," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1809–1818, 2015.

[8] D. Angelov, "Top2vec: Distributed representations of topics," *arXiv preprint arXiv:2008.09470*, 2020.

[9] D. V. T. Nguyen, P. T. The, T. M. Dinh, C. Pham, and A. T. Tran, "Qc-stylegan–quality controllable image generation and manipulation," *arXiv preprint arXiv:2212.00981*, 2022.

[10] B. Zeng, B. Liu, H. Li, X. Liu, J. Liu, D. Chen, W. Peng, and B. Zhang, "Fnevr: Neural volume rendering for face animation," *arXiv preprint arXiv:2209.10340*, 2022.

[11] J. Hasselgren, N. Hofmann, and J. Munkberg, "Shape, light & material decomposition from images using monte carlo rendering and denoising," *arXiv preprint arXiv:2206.03380*, 2022.

[12] Y. Zhou, Q. Liu, H. Zhu, Y. Li, S. Chang, and M. Guo, "Mogde: Boosting mobile monocular 3d object detection with ground depth estimation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 2033–2045, 2022.

[13] F. Yin, W. Liu, Z. Huang, P. Cheng, T. Chen, and G. YU, "Coordinates are not lonely–codebook prior helps implicit neural 3d representations," *arXiv preprint arXiv:2210.11170*, 2022.

[14] J. Cui and J. Liang, "Fuzzy learning machine," *Advances in Neural Information Processing Systems*, vol. 35, pp. 36693–36705, 2022.

[15] O. Marom and B. Rosman, "Zero-shot transfer with deictic object-oriented representation in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[16] H. Sun, W. W. Cohen, and L. Bing, "Semi-supervised learning with declaratively specified entropy constraints," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[17] Y. Balaji, S. Sankaranarayanan, and R. Chellappa, “Metareg: Towards domain generalization using meta-regularization,” *Advances in neural information processing systems*, vol. 31, 2018.

[18] Y. Zhu and R. Nowak, “Active learning with neural networks: Insights from nonparametric statistics,” *arXiv preprint arXiv:2210.08367*, 2022.

[19] H. J. Jeon and B. Van Roy, “An information-theoretic framework for deep learning,” in *Advances in Neural Information Processing Systems*, 2022.

[20] C. Tosh and S. Dasgupta, “Interactive structure learning with structural query-by-committee,” *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[21] A. Delarue, R. Anderson, and C. Tjandraatmadja, “Reinforcement learning with combinatorial actions: An application to vehicle routing,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 609–620, 2020.

[22] Q. Wang, J. Xiong, L. Han, H. Liu, T. Zhang, *et al.*, “Exponentially weighted imitation learning for batched historical data,” *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[23] J. Umenberger, M. Ferizbegovic, T. B. Schön, and H. Hjalmarsson, “Robust exploration in linear quadratic reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[24] M. Prajapat, M. Turchetta, M. Zeilinger, and A. Krause, “Near-optimal multi-agent learning for safe coverage control,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 14998–15012, 2022.

[25] M. K. Shirani Faradonbeh, M. S. Shirani Faradonbeh, and M. Bayati, “Thompson sampling efficiently learns to control diffusion processes,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 3871–3884, 2022.

[26] S. Lin, Y. Han, X. Li, and Z. Zhang, “Personalized federated learning towards communication efficiency, robustness and fairness,” *Advances in Neural Information Processing Systems*, 2022.

[27] K. Panaganti, Z. Xu, D. Kalathil, and M. Ghavamzadeh, “Robust reinforcement learning using offline data,” *arXiv preprint arXiv:2208.05129*, 2022.

8 Appendix

Cluster 4
('agent', 0.006468448828225028),
('policy', 0.006113569850090786),
('reinforcement', 0.006004967972340869),
('tasks', 0.005998280456614235),
('rl', 0.005951113315221767),
('task', 0.005731366658491771),
('human', 0.005499233558366051),
('adversarial', 0.005492413901314562),
('agents', 0.005173888345253284),
('training', 0.005018854938789691)

Cluster 3
('graph', 0.008353566291523163),
('inference', 0.006399689888529385),
('networks', 0.005483089671844949 5),
('variational', 0.00546394616983122),
('graphs', 0.005454282868347266),
('latent', 0.00513840812509116),
('deep', 0.005138052208766449),
('bayesian', 0.005011496740648761),
('neural', 0.004953773377445212),
('theoretical', 0.004861210570412224)

Cluster 2
('regret', 0.016775905654125355),
('setting', 0.009259993595175917),
('policy', 0.008885641497449215),
('optimal', 0.00871316779011982),
('bandit', 0.008613636587273134),
('online', 0.008120139589573444),
('algorithms', 0.0078023048890033956),
('algorithm', 0.0076468581444357085),
('reward', 0.007541807111166079),
('bound', 0.007101010192324715)

Cluster 1
('convex', 0.009297853692475645),
('convergence', 0.008966476695711125),
('gradient', 0.00860240440876727 8),
('stochastic', 0.008271534063236496),
('bounds', 0.008002219815413214),
('descent', 0.00722485486250305),
('bound', 0.007219481592048855),
('optimization', 0.007215644914121493),
('linear', 0.006938433341124399),
('non', 0.006937466157298806)

Cluster 0
('image', 0.01004836221106529),
('3d', 0.009056505596292785),
('images', 0.007699209224569082),
('object', 0.007599499720769098),
('semantic', 0.006451644614092025),
('art', 0.006283398793753842),
('video', 0.00611607486079888),
('attention', 0.005886185914395135),
('text', 0.005669352791637793),
('state', 0.005496678738778399)

Figure 3: Appendix: K-Means topics ordered by size

```
Cluster 3
('graph', 0.007719723274527709),
('networks', 0.00613862294262827),
('neural', 0.005461491287050695),
('deep', 0.005147243555511613),
('graphs', 0.005076844155725347),
('network', 0.004846901575807965),
('datasets', 0.0046188140842185),
('distribution', 0.004532897030618663),
('theoretical', 0.004480493187518499),
('framework', 0.004355545053262976)
```

```
Cluster 4
('agent', 0.007626264562417569),
('reinforcement', 0.00666374106351609),
('human', 0.006548345370828315),
('rl', 0.006519789902740199),
('policy', 0.00651630742633468),
('agents', 0.006410536491520354),
('tasks', 0.006040127636733731),
('adversarial', 0.005909530172374433),
('task', 0.00574791991683776),
('reward', 0.005460691299507448)
```

```
Cluster 0
('regret', 0.018619639497281897),
('setting', 0.009655724979667867),
('policy', 0.009559372187432224),
('bandit', 0.009522519210996233),
('optimal', 0.008942530323762988),
('algorithm', 0.008443479150846627),
('algorithms', 0.008380952495895046),
('online', 0.008320626604929054),
('games', 0.008034354644679777),
('bound', 0.007978307065442426)
```

```
Cluster 2
('convex', 0.009786323618799423),
('convergence', 0.009262001656484593),
('gradient', 0.008796903836581563),
('stochastic', 0.00824072044026778),
('optimization', 0.008190038536638604),
('bounds', 0.007264687691580342),
('descent', 0.007124847187555644),
('problems', 0.007052702091182984),
('non', 0.006959771432769223),
('rate', 0.006834021504261181)
```

```
Cluster 1
('image', 0.010444141491179395),
('3d', 0.010091245947220783),
('object', 0.008817059826092766),
('images', 0.008022439152341297),
('video', 0.006923081463050266),
('semantic', 0.006456122243901056),
('language', 0.006401342538210889),
('text', 0.006288126526188164),
('art', 0.006004990866839004),
('generation', 0.00575776349690467)
```

Figure 4: Appendix: Mixture of Guassian topics ordered by size

Cluster 20
('bounds', 0.033840245770338874),
('pac', 0.026464211112286164),
('bayes', 0.026372914344051028),
('hypothesis', 0.02491067255293611),
('kl', 0.02128755847470092),
('bound', 0.02054803879476502),
('inequality', 0.02019509049989428),
('distribution', 0.018717332747646434),
('dependent', 0.01640371056081851),
('distributions', 0.016350932459178284)

Cluster 16
('convergence', 0.033921123369093246),
('stochastic', 0.02663225767708544),
('problems', 0.024576100610804025),
('gradient', 0.023545762335121408),
('optimization', 0.02147958165366273),
('decentralized', 0.02078288626661968),
('descent', 0.02070713548953359),
('nmf', 0.018644891573818427),
('monotone', 0.01834525755982332),
('rate', 0.0168362551174108)

Cluster 5
('causal', 0.02431374369711894),
('fairness', 0.016959614219886725),
('treatment', 0.013256669552822096),
('predictions', 0.012245454167728941),
('predictive', 0.012093750118377794),
('group', 0.011817456504235236),
('observational', 0.011767223385415778),
('explanations', 0.011217815687478816),
('groups', 0.010817588127904117),
('influence', 0.010575744596740542)

Cluster 12
('policy', 0.04128950212028886),
('environment', 0.022165948506917308),
('rl', 0.018677988584288135),
('return', 0.01488035762835634),
('actor', 0.014155915486647899),
('critic', 0.013785365049739873),
('offline', 0.013706725613632599),
('reinforcement', 0.013007662214785635),
('risk', 0.012302581353111065),
('stationarity', 0.012144877605379275)

Cluster 3 ('attacks', 0.06347395363166415),
('adversarial', 0.040316528556943544),
('attack', 0.040218493744698394),
('defense', 0.031469225320391356),
('defenses', 0.029661413753448718),
('backdoor', 0.02936630408102437),
('threat', 0.020512270811769383),
('attacker', 0.01998061639920947),
('trigger', 0.018691551898700514),
('examples', 0.01808579989794549)

Cluster -1 (outliers)
('molecules', 0.05582297472090415),
('protein', 0.045972531501031755),
('drug', 0.043619227407517246),
('molecular', 0.03665974868722927),
('chemical', 0.028682710550000205),
('molecule', 0.028135541109192107),
('compounds', 0.024682985290176696),
('binding', 0.02362323350063691 8),
('proteins', 0.023045215090086016),
('conformation', 0.022391726048362384)

Figure 5: Appendix: DBSCAN topics ordered by size