Refining Pretrained Text Embeddings: Explorations using Manifold Learning Techniques

William Das
Department of Computer Science
Columbia University
New York, NY 10027
whd2108@columbia.edu

Albert Jan
Department of Computer Science
Columbia University
New York, NY 10027
aj3051@columbia.edu

Alan Ma
Department of Mathematics
Columbia University
New York, NY 10027
ajm2337@columbia.edu

Abstract

Pretrained text embeddings are fundamental representations used in natural language processing, capturing semantic and syntactic properties of words and phrases. Current techniques assume embeddings reside in a Euclidean space, and previous studies argue that semantic information is represented linearly in the embedding space. We explore the possibility that the intrinsic geometry of the embedding space is better captured by a nonlinear manifold, and examine techniques to better refine these embeddings inspired by various manifold learning methods. We explore this hypothesis in two contexts: (1) zero-shot classification and (2) supervised classification. We refine pretrained text embeddings using manifold learning techniques to aid performance in these two contexts, and apply our methods on a downstream text classification task of classifying groups of text into different categories, using their embedddings extracted from a pretrained transformer model based off of BERT. We compare the performance of linear methods like PCA with nonlinear approaches such as Isomap and locally linear embeddings (LLE). We also implement neighborhood preserving embeddings (NPE) and their variants on downstream tasks that rely on the embeddings' geometry. We further propose novel supervised refinement methods that leverage class labels to learn more semantically meaningful manifolds. We leverage label centroids and embeddings for supervised embeddings refinement, and further introduce "attractors" and "repellers" to model the push and pull forces that cluster semantically similar embeddings, while separating dissimilar ones. Experiments on text classification using kNN demonstrate that the refined embeddings achieve improved accuracy compared to the original pretrained embeddings. Further, we find that nonlinear methods tend to underperform PCA, supporting the conclusion that text embeddings lie on a linear subspace.

1 Introduction

A key challenge in natural language processing (NLP) is deciding how to represent text. This is most commonly done through text embeddings, which represent text as real-valued vectors in some

Preprint. Under review.

vector space. The geometry of these embeddings is of upmost importance: not only do they inform applications like classification and text generation, but many tasks, such as topic modelling, interact directly with the geometry of the embeddings. Modern language models, whether trained to predict the next token in a sequence or to simply classify it, learn a transformation from the input space to an embedding space which captures the semantic essence of a text sequence. Some models are even explicitly trained to embed an input sequence well for downstream tasks, and do only that. Regardless, previous studies heavily suggest that the embedding spaces of large language models (LLMs) enjoy remarkable geometric properties. Notably, a dominant hypothesis is that high-level semantic concepts are represented linearly in the representation space of a model [10] [11]. For example, in the context of word embeddings, prior experiments demonstrate empirically that `embedding("man")` $-$ `embedding("woman")`, `embedding("king")` $-$ `embedding("queen")`, and other similar pairs all belong to a shared low-dimensional linear subspace [8]. In this sense, the subspace captures the concept of `Male/Female`. Other studies also motivate the existence of linear concept representations by demonstrating the plausibility of "intervention"—the idea that the value of a certain concept captured by an embedding can be changed (without changing other concepts) by adding a "steering" vector to the embedding. For instance, given a generative language model's embedding of an English text sequence, we can change its output to French by simply adding an `English/French` vector to the embedding [16]. So, the current literature appears to take a stance that linear subspaces are, at minimum, a *good approximation* of the space on which semantic concepts reside in embedding space—suggesting that it is reasonable to leverage geometric notions like cosine similarity, linear projections, etc. directly in embedding space. Our work, then, probes the plausibility that text embeddings actually take a somewhat non-linear structure in embedding space—which can potentially be better approximated as a lower-dimensional manifold. To assess this hypothesis, we take a dataset of labelled text, retrieve their embeddings via a pre-trained encoder, and attempt to "refine" them by further projecting them onto a low-dimensional space. We compare the performance of applying principal components analysis (PCA), a standard linear dimension reduction method, with a number of nonlinear methods on downstream tasks that rely heavily on the geometry of our embeddings. We also investigate the idea of refining text embeddings for task-specific settings in a supervised environment—e.g. for text classification—by extending fully unsupervised manifold learning algorithms to incorporate label information in a novel way.

2 Related Work

2.1 Language Model Representations

As mentioned, language models learn some internal representation for text, regardless of their training objective—text generation, classification, etc. First, we formalize this notion for clarity. A language model takes in context vector x and maps it to a vector $\lambda(x)$ in a representation space $\Lambda \simeq \mathbb{R}^d$. For some model architectures, $\lambda(x)$ is simply the output of the model. We call these models "encoders" as they are specifically trained to embed x well, and leave it to other downstream models/tasks to make use of these embeddings. On the other hand, a model capable of generating text may "unembed" this embedding after producing it, i.e. map $\lambda(x)$ to a probability distribution over the next word, from which we can sample. We call such a model a "decoder." Although encoder and decoder architectures may contain fundamental differences, we refer to $\lambda(x)$ as any language model's embedding of a vector x. Of course, x being a vector implies that even before feeding text into the language model, we have somehow converted it into a sequence of numbers. This process is referred to as tokenization, which breaks a text sequence down into smaller "tokens" (which could be parts of words, combinations of words, punctuation, etc.). Typically, a token is mapped to a unique integer.

A hallmark encoder architecture is BERT [4], a pre-trained transformer network that, at the time of its release, set state-of-the-art results for tasks such as sentence classification and question answering. However, BERT does not compute *independent* sentence embeddings—in order to compare the semantic similarity of two sentences, both sentences must be fed into the network together. Sentence-Bert (BERT) [12] proposes a modification to BERT which produces semantically meaningful sentence embeddings. Sentences with similar semantics are represented by vectors that appear to be closer together in Euclidean space, which allows us to leverage notions like cosine similarity. Studies also suggest that semantic information is well-approximated by linear subspaces of the embedding space, as described previously. However, other experiments suggest that these observations do not reveal the entire story of embedding space geometry; it has been shown that SBERT embeddings can be re-embedded using manifold learning techniques, yielding promising results on tasks such as document clustering and text classification. For example, Neighborhood Preserving Embedding (NPE) and other variants of Locally Linear Embedding (LLE), which we describe below, have been applied on text embeddings to improve performance on said tasks [2] [3].

2.2 Isomap

Isomap is a nonlinear dimensionality reduction algorithm that can be considered a refinement of multidimensional scaling (MDS), introduced in [15]. Classical MDS is a dimensionality reduction technique that projects data from a high-dimensional input space onto a lower-dimensional linear subspace, preserving pairwise distances. Isomap alters MDS in order to learn nonlinear manifolds by computing the geodesic distance between any two data points.

First, we define the geodesic distance. Let M be a compact Riemannian manifold. Then for any p, q on M, define the distance between the two as

$$\inf_{\gamma} \int_p^q |\dot{\gamma}(t)| \, dt$$

Where γ is any curve on M between p and q. In our case, where data is a finite sample from a compact manifold M, we can estimate the curves from a point p to a point q as a sequence of "hops" between data points.

More concretely, Isomap interpolates the underlying manifold of a finite sample by "connecting" points which are sufficiently close (determined with a ε threshold or k-nearest neighbors). From here, an adjacency matrix can be constructed by setting the distance between two points to the Euclidean distance if they are connected, or infinity if they are not. Then, the curves between two data points can be approximated by all paths between those two points through connected data points. The geodesic distance can then be approximated by the shortest path. It is clear that as the size of the sample approaches infinity, the geodesic distance estimate approaches the true geodesic distance.

With geodesic distances, Isomap can then construct a distance matrix, and then apply classical MDS to reduce to the target dimension.

2.3 Locally Linear Embedding

Locally Linear Embedding (LLE) is another nonlinear dimensionality reduction algorithm, first introduced in [13]. While Isomap leverages more global features of manifolds, LLE takes advantage of the fact that manifolds are locally Euclidean (Each p on a manifold M has a neighborhood U that is homeomorphic to a subset of Euclidean space).

This means that if a finite sample X of a manifold is sufficiently large, then for any data point x_0, on a sufficiently small neighborhood of x_0, x_0 and its neighbors $x_1, ..., x_k$ form a linear subspace. Then, x_0 can be written as a linear combination of its neighbors $x_1, ..., x_k$. We find this linear combination for all $X_i \in X$ by taking the k nearest neighbors of each X_i, and minimizing the following cost function:

$$\mathcal{E}(W) = \sum_i \left| X_i - \sum_j W_{ij} X_j \right|^2$$

Such that $W_{ij} = 0$ if X_j not a k-nearest neighbor of X_i, and $\sum_j W_{ij} = 1$. This problem reduces to solving a linear system of equations and thus is computationally tractable.

Given a target dimension d, we would like the embed X in $\mathbb{R}^d$ in some way that preserves the structure of X. This can be done by trying to reconstruct the locally Euclidean neighborhoods considered in the above step. Let $Y \subseteq \mathbb{R}^d$ be an embedding of X into $\mathbb{R}^d$. Then we would like to minimize the following cost function:

$$\Phi(Y) = \sum_i \left| Y_i - \sum_j W_{ij} Y_j \right|^2$$

By Rayleitz-Ritz, this reduces to solving an eigenvector problem, which is once again computationally tractable.

2.4 Neighborhood Preserving Embedding and Variants

2.4.1 Neighborhood Preserving Embedding

Neighborhood preserving embedding (NPE) [5] can be thought of as a linear approximation of LLE. NPE attempts to address two issues with manifold learning techniques such as Isomap and LPE. First, Isomap and LLE are computationally expensive because of their nonlinear nature. By restricting to linear transformations, NPE is much cheaper. Second, nonlinear methods output the embedded data, and not the transformation. This means that if you use nonlinear dimensionality reduction methods on training data, it is unclear how to apply the same nonlinear transform to the testing data. NPE solves both of these problems.

The first two steps are the same as LLE. In an attempt to find a local neighborhood of a data point $x \in X$ where the shape of the data is locally Euclidean, k-nearest neighbors or some other method is used to identify points close to x. Then, the same minimization problem as LLE is solved to find the linear reconstruction weights of any point x:

$$\mathcal{E}(W) = \sum_i \left| X_i - \sum_j W_{ij} X_j \right|^2$$

Such that $W_{ij} = 0$ if X_j not a k-nearest neighbor of X_i, and $\sum_j W_{ij} = 1$. However, the next step is different. Let X be the matrix with the data points x as the columns, $M = (I - W)^T (I - W)$. Then the optimal linear projection which minimizes the cost function $\Phi(Y) = \sum_i \left| Y_i - \sum_j W_{ij} Y_j \right|^2$ is given by $B = (XX^T)^{-1}(XMX^T)$. In the rare case where X is not full rank, we can use SVD to consider a singular projection of X. Then, setting B to be the matrix with the first d eigenvectors of B as the columns, we can project X to some d dimensional linear subspace Y with $Y = A^T X$. One can think of this as minimizing $\Phi(Y)$ using only linear transformations.

Generally, we can think of NPE as LLE restricted to linear transformations. NPE attempts to preserve the local neighborhood structure of the data, in the hope of preserving the global manifold structure, but only using linear transformations. NPE also has an important variant, Kernel NPE and Supervised NPE, which we discuss further.

2.4.2 Kernel NPE

Kernel NPE [14] allows us to introduce nonlinear transformations to NPE through the kernel trick. This is quite simple: simply replace the matrix X in the cost functions above with the kernel matrix $K = [k(x_i, x_j)]$ for a kernel of your choice.

A more computationally efficient away to do this is to first perform kernel PCA, then run normal NPE on the output of kPCA.

2.4.3 Supervised NPE

Supervised NPE [1] allows us to use the labels to inform the projection. Let X, M, and Y be as before. Given C classes, if training sample x_i belongs in class c, construct a one-hot encoded vector h_i with the cth entry of h_i one and all others zero. Let H be the matrix of these vectors, h. Then, find the reconstruction weight matrix W similar to regular NPE. For the reconstruction step, our new modified cost function to minimize is:

$$\Phi(Y) = \frac{1}{2} \sum_{i=1}^{n} \left| Y_i - \sum_j j = 1^k W_{ij} Y_j \right| + \beta \sum_{i=1}^{n} |Y_i - H_i|$$

Where β is a balance parameter used to determine the importance of label information. From here, we find that the optimal projection matrix A, with $Y = A^T X$, is given by

$$\beta(XMX^T + \beta XX^T)^{-T} XH^T$$

Intuitively, supervised NPE creates attraction points with H, and drags samples towards their correct attraction point. This approach helps use class labels to inform manifold structure, in the hopes of it leading to better embeddings. We explore both this algorithm, and a simple variant of it to aid in constructing the weight matrix for NPE.

2.5 Refining Embeddings with Manifold Learning

3 Method

3.1 Dataset and Pretrained Model

We chose to use the 20 Newsgroups dataset [9]. This dataset is a mainstay of natural language processing and is commonly used to evaluate the classification performance of different language models. The dataset is composed of 20,000 text documents distributed across 20 different newsgroup labels.

Note that the labels do not come as natural language. An example of a label is `talk.politics.misc`. Some basic preprocessing had to be done to turn the labels into natural language processable by a language model. We changed all periods (".") into spaces (" ") and expanded all words ("misc" $\rightarrow$ "miscellaneous").

For our pretrained model, we used the Alibaba `gte-base-en-v1.5` model[7]. This model is an encoder which is able to return sentence-level text embeddings in 768 dimensional Euclidean space. It is built off of the BERT model discussed in section 2.1 and is able to achieve state-of-the-art results on multiple benchmarks.

In our supervised problem formulation, we have as inputs text embeddings, and as corresponding outputs, the respective classes that they belong to. For evaluating performance, we primarily use a simple k-NN algorithm to assess how the cosine similarity between transformed embeddings is impacted. We do all our embeddings refinement on a designated training set, consisting of train embeddings and labels, and do our evaluations on a designated test set consisting of test embeddings and labels.

3.2 Refining Embeddings and Zero-Shot Classification

3.2.1 Zero-Shot Classification

Zero-shot classification is a classification problem where the model is asked to predict classes it has never seen before. In other words, the model is asked to classify new categories with zero training data.

In natural language processing, given a language model, a piece of text, and candidate labels, zero-shot classification is typically done as explained in algorithm 1. For each piece of text, pairwise cosine similarity is calculated between the text embedding and each of the label embeddings, and the text is assigned to the label with the highest cosine similarity.

Algorithm 1 Zero-shot classification

```
1: procedure ZEROSHOTCLASSIFICATION(model, text, candidatelabel)
2:     textemb ← model(text)
3:     labelemb ← []
4:     for label in candidatelabel do
5:         labelemb ← model(label)
6:     end for
7:     similarity ← pairwiseCosineSimilarity(textemb, labelemb)
8:     return argmax(similarity, axis=1)
9: end procedure
```

Zero-shot classification is an interesting and relevant downstream task that is reliant on embedding quality. As can be seen in the algorithm, the geometric properties of the embedding directly impact the resulting classification since classification is done by comparing cosine similarities. As such, we believe that zero-shot classification is a good benchmark for understanding how our methods may refine embedding quality. Further, zero-shot classification is an active use case of word embeddings, so any improvement we may find will have an immediate impact.

3.2.2 Refining Embeddings

We use a procedure similar to [12] to refine embeddings from the Alibaba model. Given a corpus of text S, we use the model to return embeddings X. We then use our manifold learning task on the model to return refined embeddings, Y.

Since zero-shot classification is an unsupervised task, we could not use many of the supervised manifold learning techniques discussed in this paper. So the techniques were limited to PCA, NPE, Isomap, and LLE.

3.3 Geodesic Distance as a Similarity Measure

To do zero-shot-classification, we also experiment with keeping the embeddings from the Alibaba model (i.e. not further refining them), but instead using a non-cosine similarity measure. In particular, we take the distance between two embeddings to be their geodesic distance, which we approximate by constructing the graph used by Isomap to interpolate a manifold (see 2.2), and calculating the shortest path between the embeddings via Dijkstra's algorithm. Then, we simply classify a text embedding by choosing the label whose embedding is geodesically closest.

3.4 Supervised Embeddings Refinement Methods

The underlying goal in this setting is to, given a set of train and test embeddings and labels, refine the quality of the train embeddings so that semantically similar words are closer together. In the context of text embeddings, we formulate this problem as maximizing the average pairwise cosine similarity of label-wise embeddings, or increasing intra-class cosine similarity of embeddings.

To start with, we devised and tested two basic embedding refinement methods using the train labels given to serve as a baseline method.

Our first method refines embeddings by moving them towards the centroid of their respective class. The centroid is calculated as the median of all embeddings within the class, which is less sensitive to outliers than the mean. This approach helps in minimizing intra-class variance and thus maximizing the cosine similarity among class members.

The steps for the centroid-based refinement method are as follows:

1. Compute the centroid for each class.
2. Adjust each embedding by blending it with its class centroid.
3. Normalize the modified embeddings.

Algorithm 2 Modify embeddings with class centroids

```
1: procedure ModifyEmbeddingsWithLabels(embeddings, labels, alpha)
2:     centroids ← empty dictionary
3:     for each unique label in labels do
4:         centroids[label] ← median(embeddings[where labels = label])
5:     end for
6:     modified_embeddings ← copy of embeddings
7:     for i ← 1 to length(embeddings) do
8:         label ← labels[i]
9:         centroid ← centroids[label]
10:        modified_embeddings[i] ← $\alpha$ · centroid + (1 - $\alpha$) · embeddings[i]
11:    end for
12:    return normalize(modified_embeddings)
13: end procedure
```

The second method involves refining embeddings by aligning them towards a predefined label embedding, which may be derived from another model or a different set of data representing ideal label characteristics. In essence, we embed the text labels using our pretrained embeddings model, and use those label embeddings to guide the embeddings refinement.

This method is similar to the first but uses an external label embedding instead of the class centroid to move embeddings towards. The steps are as follows:

1. Each class label has an associated predefined embedding.
2. Each text embedding is adjusted towards its corresponding label embedding.
3. The embeddings are then normalized.

Algorithm 3 Modify embeddings with label embeddings

```
1: procedure ModifyEmbeddingsWithLabelEmbeddings( embeddings, labels, label_embeddings, $\alpha$)
2:     modified_embeddings $\leftarrow$ copy of embeddings
3:     for $i \leftarrow 1$ to length(embeddings) do
4:         label $\leftarrow$ labels[$i$]
5:         label_embedding $\leftarrow$ label_embeddings[label]
6:         modified_embeddings[i] $\leftarrow \alpha \cdot$ label_embedding $+ (1-\alpha) \cdot$ embeddings[$i$]
7:     end for
8:     return normalize(modified_embeddings)
9: end procedure
```

Both methods aim to refine embeddings by reducing the variability within classes, thus enhancing the model's ability to categorize and process similar data points effectively. The choice of method depends on whether class centroids or predefined label embeddings are more representative of class characteristics in a particular application.

For both methods we set the α to 0.5 to evenly balance out the embeddings and clusters towards which they are pulled towards. We note that more experimentation can be done with exploring different α thresholds.

3.5 Refining Embeddings Using "Attractors" and "Repellers"

We devised and experimented with another approach that refines embeddings by not just pulling them towards "attractors" as above, but also pushing them away from "repellers" in the latent embedding space, analogous to theories in dynamical systems and chaos theory. We regard attractors as typically central points (medians) for embeddings—such as centroids of clusters in K-Means—that share the same label. As with our basic baseline methods, we posit that these points should attract embeddings of the same class towards them. We regard repellers as points that should repel embeddings of different classes, calculated as positions opposite to the centroid of embeddings from different classes, adjusted to exaggerate the distance from the attractor.

3.5.1 Manifold Learning Inspirations

Manifold learning involves the assumption that the data (in this case, embeddings) lie on a low-dimensional manifold within a higher-dimensional space. Our refinement algorithms above assume:

- **Local Similarity:** Embeddings that are close in the high-dimensional space (sharing the same label) should remain close in the refined space, guided by attractors.
- **Global Dissimilarity:** Embeddings from different categories should be distinguishable, enforced by repellers.
- **Geometric Structure Preservation:** The refined embeddings should preserve the geometric structures of the data manifold, such as clusters or groups indicated by labels.

Attractors and Repellers

The algorithm computes attractors and repellers for each class label. Attractors are defined as the centroid of the embeddings belonging to a specific class, representing the semantic center of that class. Repellers, on the other hand, are calculated as the difference between twice the centroid of embeddings from other classes and the corresponding attractor. This positioning of repellers encourages the embeddings to move away from the semantic space of other classes. We note that this definition for repellers may benefit from additional exploration.

Attractors are defined as the median of embeddings for each unique label, essentially acting as centroids that pull similar embeddings closer:

$$\text{attractors}[label] = \frac{1}{n_{label}} \sum_{i \in I_{label}} \mathbf{e}_i$$

where I_{label} is the set of indices of embeddings with the corresponding label and $\mathbf{e}_i$ are the embedding vectors. We also experiment, however, with using the Wasserstein barycenter as the attracting distribution, as well.

Repellers are calculated to push embeddings away from the mean of other class embeddings, aiming to increase class separability:

$$\text{repellers}[label] = 2 \times \left(\frac{1}{N - n_{label}} \sum_{j \notin I_{label}} \mathbf{e}_j \right) - \text{attractors}[label]$$

where N is the total number of embeddings in the train set.

Given these attractors and repellers, we define a refinement process that iteratively updates the embeddings using an update rule based on the distances to the attractors and repellers. The update rule incorporates a learning rate and momentum to control the magnitude and direction of the updates. The distance metric used in this algorithm is cosine distance, which measures the dissimilarity between two vectors in terms of their angle. We note that an extension of this worth exploring is applying gradient descent.

The refinement of embeddings is achieved through an iterative process, where each embedding is updated by moving towards its class attractor and away from its class repeller:

$$\mathbf{e}_i^{(t+1)} = \mathbf{e}_i^{(t)} + \alpha \mathbf{m}_i^{(t)} + \eta \left(\left(\text{attractors}[\text{label}_i] - \mathbf{e}_i^{(t)} \right) d_a - \left(\mathbf{e}_i^{(t)} - \text{repellers}[\text{label}_i] \right) d_r \right)$$

where d_a and d_r are the distances to the attractor and repeller, respectively, η is the learning rate, and α is the momentum coefficient.

Algorithm 4 Refine Embeddings using Attractors and Repellers

```
1: procedure RefineEmbeddings(embeddings, labels, attractors, repellers, $\eta$, $\alpha$, iterations)
2:     refined_embeddings ← embeddings
3:     momentum ← zeros_like(embeddings)
4:     for iter ← 1 to iterations do
5:         for $i$ ← 1 to length(embeddings) do
6:             label ← labels[$i$]
7:             attractor ← attractors[label]
8:             repeller ← repellers[label]
9:             distance_to_attractor ← pairwise_distances(embeddings[$i$], attractor)
10:            distance_to_repeller ← pairwise_distances(embeddings[$i$], repeller)
11:            attraction_force ← (attractor − embeddings[$i$])
12:            × distance_to_attractor
13:            repulsion_force ← (embeddings[$i$] − repeller) × distance_to_repeller
14:            update ← $\eta$ (attraction_force + repulsion_force)
15:            momentum[$i$] ← $\alpha$ × momentum[$i$] + update
16:            refined_embeddings[$i$] ← refined_embeddings[$i$] + momentum[$i$]
17:        end for
18:    end for
19:    return normalize(refined_embeddings, axis = 1)
20: end procedure
```

The algorithm seeks to refine the representation of these embeddings by actively manipulating their positions in the space to better reflect intrinsic manifold structures using both local and global semantic information.

By reinforcing the proximity of semantically similar embeddings and ensuring the dispersion of dissimilar embeddings, we posit that these algorithms could create a more intuitive and useful embedding space, where distance more accurately reflects semantic similarity and distinction.

3.6 Kernel NPE

As described above, we also employ a kernel based NPE algorithm that applies NPE on kernel-PCA transformed data. This method starts by applying Kernel PCA, which uses a kernel function to project the original data into a higher-dimensional feature space where linear relationships are more discernible. After transforming the data, NPE is applied to the kernel-transformed space to preserve local neighborhoods. The process involves constructing neighborhoods, computing a weight matrix that minimizes the reconstruction error from these neighborhoods, and finally calculating the embedding by solving a generalized eigenvalue problem.

3.7 Modified Supervised NPE

We also employ a modified version of Supervised Neighborhood Preserving Embedding (SNPE) that incorporates class label information during the construction of the weight matrix. In this modified approach, neighbors with the same class label as the target point are assigned higher weights compared to neighbors from different classes. This enhancement aims to prioritize the preservation of local neighborhood structure within each class, leading to a more discriminative embedding space. We do this both for NPE and kernel NPE.

4 Experiments

4.1 Preliminary Exploration and Manifold Learning Justification

Given that our data does not arise naturally but rather is the result of the Alibaba model embedding words in a vector space, a natural first question is whether or not the data actually does lie in a lower-dimensional manifold of $\mathbb{R}^{768}$. First, PCA and Isomap were used to visualize the data in 2 dimensions.

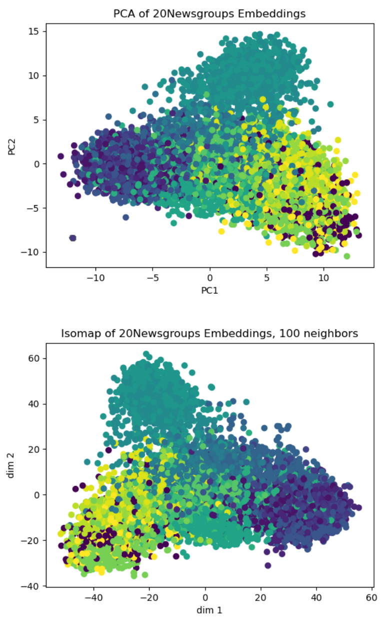

Figure 1

We argue that while do not show that the data is 2-dimensional, the visible geometric structure likely means the data is not 768-dimensional, as a projection from 768 dimensions down to 2 would be too lossy to be as well-behaved as the data appears in these plots.

To verify, we use the Levina-Bickel maximum likelihood dimension estimator[6]. The estimator tells us that the data is 26 dimensional, so even with the Levia-Bickel estimator's tendency to underestimate dimensionality, the data is not 768-dimensional, which lends more credence to the idea that manifold learning may meaningfully improve embeddings.

To check this, we examine the explained variance ratio plot for PCA. While this is a very crude method of estimating dimension, as PCA is a linear dimensionality reduction technique, it serves as a sanity check that the actual dimension of our data is lower than 768.

4.2 Refined Zero-shot Classification

We explore the performance of zero-shot classification on the 20 Newsgroups dataset for various refinement methods. Different manifold learning methods were run on a range of different dimensions and compared to the performance of the baseline Alibaba model. The results are included below.

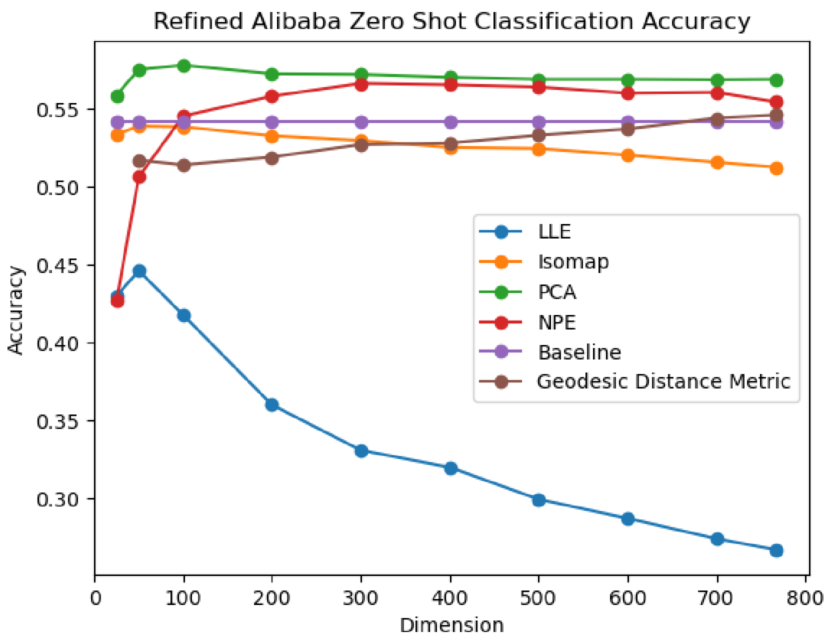

Figure 2: Zero-shot classification accuracy for various embedding refinement methods

Perhaps most interesting is that the linear dimensionality reduction methods (PCA, NPE) performed the best. This is further explored in the discussion section, but we believe this is because the data lies in a lower-dimensional linear subspace of $\mathbb{R}^{768}$, as opposed to simply a general manifold. This hypothesis is further supported by noting that Isomap and LLE actually worsen performance.

4.3 k-Nearest Neighbors

First we explored the performance of various manfiold based dimensionality reduction techniques and PCA on the performance of kNN on the multiclass embeddings classification task.

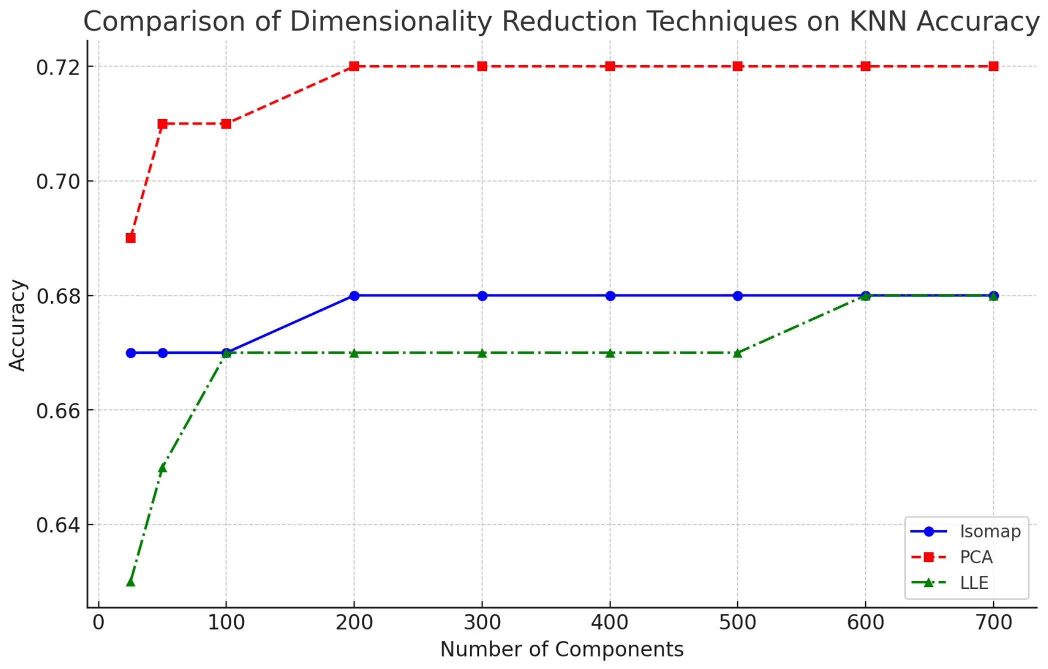

Figure 3

4.4 Pairwise Cosine Similarities

In Table 1, we also examine how different embedding refinement techniques impact the average pairwise cosine similarities amongst embeddings of the same class. The base similarities are notably low, and reflect unrefined embeddings data. We report how the average pairwise similarity changes when refining using the attractor/repeller, attractor/repeller using barycenters, centroid labels, and label embeddings based refinement methods described above.

Evidently, except for our barycenter based attractor/repeller refinement method, the refinements tended to increase pairwise cosine similarity, with the push towards label embeddings yielding the strongest increase. Surprisingly, the embeddings refined with the label embeddings did not yield a substantial improvement in kNN accuracy, as discussed later in the results section, suggesting the presence of potential outliers in the data. We hypothesize that this may also be caused due to the hierarchical nature of the labels, which we don't account for in our methods.

4.5 Supervised Methods

4.5.1 NPE Variants

In Table 2, we compare the baseline with five refinement strategies: Neighborhood Preserving Embedding (NPE), Supervised NPE as described in 2.4.3 (proposed by Bao et al. [1]), our modified Supervised NPE algorithm, Kernel NPE, and Supervised Kernel NPE. While the differences are marginal, we observe better performance with the supervised variants.

Performance at Lower k Values (10 to 50): The supervised approaches, specifically Supervised NPE and Supervised Kernel NPE, consistently outperform the baseline and unsupervised methods. This enhancement is attributed to the use of label information during the embedding refinement, which encourages embeddings to form clusters aligning well with class labels.

Performance at Higher k Values (100 to 500): As k increases, the advantage of using supervised methods diminishes slightly. This phenomenon is typical in kNN, where higher k values tend to include more neighbors, potentially introducing noise and diluting the effect of refined embeddings. Notably, the performance of all methods tends to converge at larger k values, with Kernel NPE and Supervised Kernel NPE showing slightly better robustness, likely due to their ability to capture complex, non-linear structures in the data. A few observations:

Table 1: Average Pairwise Similarities for Various Classes and Refinement Techniques

Class	Base Sim.	Attractor/Repeller	Centroid Label	Label Embedding	Barycenter
talk.politics.mideast	0.332	0.380	0.669	0.762	0.188
comp.graphics	0.334	0.372	0.668	0.761	0.185
talk.politics.guns	0.334	0.373	0.667	0.754	0.185
rec.motorcycles	0.349	0.391	0.684	0.756	0.201
comp.windows.x	0.334	0.374	0.667	0.745	0.187
sci.med	0.363	0.420	0.696	0.770	0.226
rec.sport.baseball	0.325	0.381	0.660	0.758	0.191
comp.os.ms-windows.misc	0.303	0.342	0.635	0.749	0.160
sci.space	0.310	0.350	0.644	0.748	0.165
rec.sport.hockey	0.366	0.441	0.699	0.775	0.241
talk.politics.misc	0.379	0.468	0.713	0.783	0.263
sci.electronics	0.400	0.458	0.731	0.780	0.259
alt.atheism	0.296	0.324	0.627	0.733	0.143
comp.sys.mac.hardware	0.272	0.304	0.601	0.719	0.129
rec.autos	0.308	0.345	0.642	0.747	0.160
soc.religion.christian	0.368	0.445	0.701	0.776	0.246
sci.crypt	0.352	0.408	0.687	0.758	0.213
misc.forsale	0.368	0.443	0.703	0.752	0.241
talk.religion.misc	0.300	0.334	0.633	0.728	0.151
comp.sys.ibm.pc.hardware	0.332	0.374	0.668	0.759	0.184

- **Supervision:** By incorporating label information, supervised methods tailor the embeddings to respect class boundaries, enhancing classification accuracy, particularly at lower k values.
- **Kernel Methods:** Kernel transformations in NPE methods seem to help in capturing additional relationships that are not apparent in the original feature space.

Table 2: KNN Accuracies Across Different Numbers of Neighbors for NPE Methods

k	Baseline	NPE	SNPE (Bao et al.)	SNPE (Ours)	Kernel NPE	Kernel SNPE
10	0.7138	0.7166	0.7109	0.7239	0.7230	0.7244
20	0.7196	0.7222	0.7092	0.7219	0.7228	0.7254
30	0.7152	0.7154	0.7064	0.7214	0.7174	0.7195
50	0.7146	0.7115	0.7101	0.7187	0.7127	0.7150
100	0.7081	0.7087	0.7117	0.7131	0.7081	0.7088
200	0.7035	0.6969	0.7083	0.7011	0.7027	0.7039
300	0.6955	0.6913	0.7071	0.6957	0.6944	0.6961
500	0.6856	0.6774	0.7113	0.6840	0.6806	0.6836

4.5.2 Centroid and Attractor/Repeller Embedding Refinement

In Table 3, we compare the performance of k-Nearest Neighbors (kNN) on embeddings refined using Label Embedding, Label Centroid, Attractor/Repeller, and Barycenter methods, relative to the unrefined baseline.

Firstly, the label centroid based method achieves the highest accuracy across all k values. By moving embeddings towards their class medians, it minimizes intra-class variance and enhances separability in the embedding space. We think that the tighter clustering of same-class embeddings allows kNN to more easily identify true neighbors, improving classification.

The attractor/repeller refinement method we implemented shows the second-best performance. Inspired by manifold learning, it pulls embeddings towards class attractors and pushes them away from repellers, amplifying class separation.

The barycenter based attractor/repeller refinement, based on the Wasserstein barycenter, yields mild improvements over the baseline by finding a representative embedding that minimizes intra-class

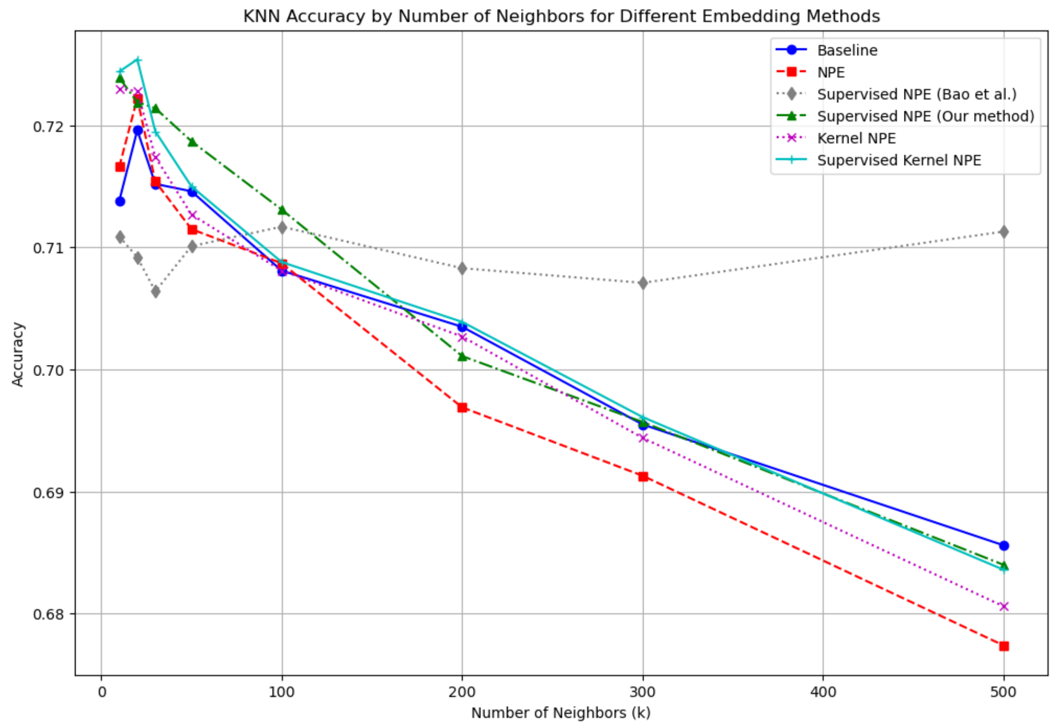

Figure 4

distances. However, its performance lags behind the former two, suggesting that the geometric averaging may not adequately capture class-specific characteristics. Likely, there is more tuning and implementation fixes that need to be made to extract better Wasserstein barycenter estimates.

Surprisingly, the method with Label Embedding refinement performs worse than the baseline. The predefined label embeddings fail to align with the test data's class structure. This occurs possibly due to overfitting or inadequate separation of label spaces in the embedding. We find that over-maximizing the intra-class cosine similarities of embeddings does not directly translate to improved embeddings quality or performance.

Across all methods except for Supervised NPE proposed by [1], accuracy declines with increasing k. We think that this occurs since larger neighborhoods may introduce more potential for cross-class contamination, as distant and less relevant neighbors are included in the majority vote, leading to misclassifications.

Table 3: KNN Accuracies for Different Numbers of Neighbors and Embedding Refinements

k	**Baseline**	**Label Embedding**	**Label Centroid**	**Attractor/Repellor**	**Barycenter**
10	0.7138	0.6872	0.7267	0.7192	0.7039
20	0.7196	0.6753	0.7238	0.7220	0.7151
30	0.7152	0.6693	0.7208	0.7196	0.7131
50	0.7146	0.6664	0.7139	0.7170	0.7073
100	0.7081	0.6593	0.7097	0.7112	0.7028
200	0.7035	0.6501	0.7053	0.7033	0.7004
300	0.6955	0.6458	0.7021	0.7016	0.6936
500	0.6856	0.6407	0.6952	0.6949	0.6842

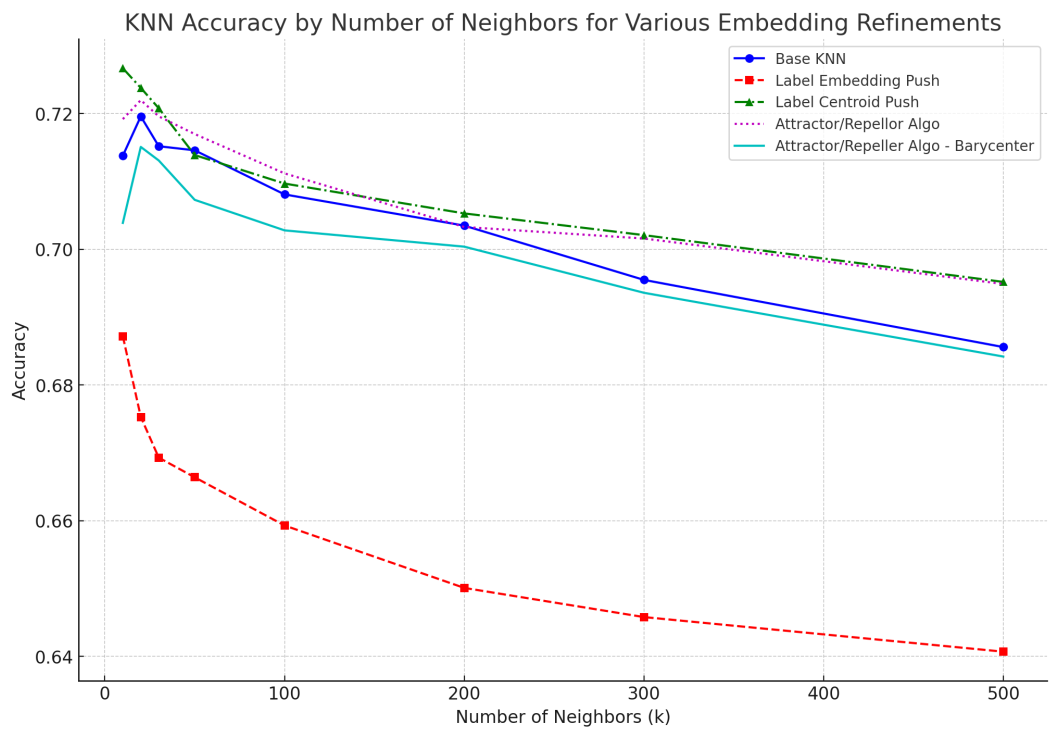

Figure 5

5 Discussion

Our experiments on refining text embeddings for zero-shot classification reveal that linear dimensionality reduction methods outperform traditional manifold learning algorithms. Previous studies argue that semantic concepts are indeed represented as low-dimensional linear subspaces of the embedding space; a concept represents some individual fact about the embedded text (its language, sentiment, etc.). But our findings, perhaps as expected, take this argument a step further: that linearity is not only sufficient to capture individual concepts, but also the geometry of the entire embedding space. The broader appeal of this linearity is that we can interpret and control language models using simple linear algeraic operations, such as orthogonal projections, directly in the embedding space.

We also demonstrate that in a supervised setting, incorporating label information can be helpful when doing nonlinear dimensionality reduction. We investigate a number of embedding refinement methods, both novel and/or inspired by previous work. However, we emphasize that our embedding refinement methods for zero-shot classification–which are fully unsupervised–answer a fundamentally different question than the experiments we run in a supervised setting. The former assesses whether we can improve text embeddings as a whole by refining them; the latter only aims to improve embeddings for a specific downstream task, which we do not expect to generalize.

6 Code Availability

This Github repository houses the code for all our experiments.

7 Group Contributions

William focused on NPE methods, NPE variants (supervised NPE, kernel NPE, supervised kernel NPE), centroid and label embedding based refinement, and experimentation with "attractors" and "repellers".

Albert focused on obtaining the text dataset and retrieving embeddings, implementing supervised NPE, and investigating geodesic distance as a similarity measure.

Alan focused on implementing base NPE, dimension estimation and other exploratory data analysis, and implementing zero-shot classification and other unsupervised methods.

Everyone contributed equally to paper writing.

References

[1] Xing Bao, Li Zhang, Bangjun Wang, and Jiwen Yang. A supervised neighborhood preserving embedding for face recognition. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 278–284, 2014.

[2] Yonghe Chu, Heling Cao, Yufeng Diao, and Hongfei Lin. Refined sbert: Representing sentence bert in manifold space. *Neurocomputing*, 2023.

[3] Yonghe Chu, Hongfei Lin, Liang Yang, Yufeng Diao, Shaowu Zhang, and Fan Xiaochao. Refining word representations by manifold learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pages 5394–5400, 2019.

[4] Jacob Devlin, Ming-Wei Chang, and Kristina Lee, Kenton andToutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. pages 4171–4186. Association for Computational Linguistics, 2019.

[5] Xiaofei He, Deng Cai, Shuicheng Yan, and Hong-Jiang Zhang. Neighborhood preserving embedding. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1208–1213 Vol. 2, 2005.

[6] Elizaveta Levina and Peter Bickel. Maximum likelihood estimation of intrinsic dimension. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004.

[7] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023.

[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 2013.

[9] Tom Mitchell. Twenty Newsgroups. UCI Machine Learning Repository, 1999. DOI: https://doi.org/10.24432/C5C323.

[10] Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. 2023.

[11] Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. *arXiv preprint arXiv:2311.03658*, 2023. Accepted for an oral presentation at NeurIPS 2023 Workshop on Causal Representation Learning.

[12] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992. Association for Computational Linguistics, 2019.

[13] Lawrence Saul and Sam Roweis. An introduction to locally linear embedding. *Journal of Machine Learning Research*, 7, 01 2001.

[14] Xiaoyan Tao, Shufu Dong, Qiaoxia Zhao, and Zhongxiang Han. Kernel neighborhood preserving embedding and its essence analysis. In *2009 WRI Global Congress on Intelligent Systems*, volume 4, pages 446–450, 2009.

[15] Joshua B. Tenenbaum, Vin de Silva, and John C. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290, 2000.

[16] Alexander M. Turner, Lukas Thiergart, Madeleine Udell, George Leech, Uday Mini, and Mike MacDiarmid. Activation addition: Steering language models without optimization. 2023.