EN.625.609.83.SP22 MATRIX THEORY PROJECT: DIMENSIONALITY REDUCTION EVALUATION

NICHOLAS A. LINES

1. Introduction

Dimensionality reduction (DR) is defined somewhat broadly as a transformation of numeric data from a high-dimensional representation to a lower-dimensional representation that preserves some important qualities or structures. Consider the situation in which a dataset is summarized by describing each of the d data points $\boldsymbol{x}_i$ as a row vector in $\mathbb{R}^n$ where n is large. The data is therefore represented as the $d \times n$ matrix $\boldsymbol{X}$ whose rows are observations and whose columns represent features in the feature space. This situation is the typical overture to machine learning tasks like supervised classification (i.e. applying labels to new data based on observed data), unsupervised clustering (i.e. grouping data into structures without labels), and graph-based learning. DR in these situations amounts to transforming each data point to a new, smaller feature space via an embedding process that (hopefully) preserves and perhaps even emphasizes important structures in the dataset.

In this paper we will discuss the problem of evaluating DR techniques. We will first familiarize ourselves with three popular tools representative of several classes of DR techniques, and then introduce several evaluation methods that focus on specific success metrics. Finally, we will apply these evaluation methods to review various solutions to two real-world DR problems.

1.1. **The value of dimensionality reduction.**

For the past half a century many computational standards have grown exponentially, proportionate to Moore's Law. One notable result of this trend is the rise of "Big Data." While many authorities agree that this pattern of growth has decayed into a less predictable pattern [3], the reality remains that dataset sizes continue to increase and demand novel analysis techniques that adapt to this trend. DR has proved itself one of the most potent solutions to this problem.

There are many different applications and motivations for DR, beyond the elementary purpose of shrinking the data size. In statistical learning with errors a common problem is *overfitting*, where a model is trained to be overly sensitive to its training data. This typically results in a model that is more complex than we can justify given the uncertainty in the data, as shown in the example in Figure 1. DR can be used to collapse the feature space into a lower dimensional space that emphasizes only the features that best explain the data, thereby reducing the likelihood of model overfitting. Similarly, DR may be applied in signal processing problems to filter noise and emphasize the signal(s) of interest.

Another common application of DR is data visualization. Humans cannot easily conceptualize higher dimensional representations of data, so it is common to project

Date: April 17, 2022.

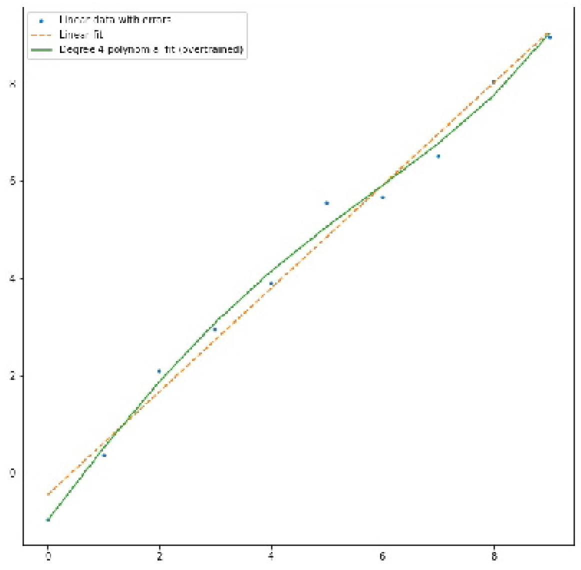


FIGURE 1. In this example of overfitting, we measure data from $\boldsymbol{y} = \boldsymbol{x}$ with Gaussian additive errors $\epsilon \sim Normal(0, 0.5)$. The linear fit is justifiable given the underlying phenomenon measured. The degree 5 polynomial fit comes closer to matching the training data, but as a result is learning random errors.

high dimensional data to a two or three-dimensional embedding for the purpose of plotting. This embedding can be optimized to attempt to preserve local or global distance structure between data points for any given metric. When DR is designed to preserve pairwise distance information, it often goes by the name *multidimensional scaling.*

Graphical models are particularly benefited by this process: generating visualizations for very large graphs (in terms of number of nodes or number of edges) can be extremely computationally expensive. Using DR, we can cluster the nodes into communities and visualize the relationships between these communities, an abstraction that is much simpler to produce.

When handling large text data sets, it is common in Natural Language Processing to represent documents as long sparse vectors (e.g. using counts of important vocabulary words in each document) and cluster these. DR of the original document embedding can perform this clustering task, and in this context is given the name *topic modeling.*

There are many other useful aspects to DR. These techniques can be considered as data compression to reduce storage and computation requirements –although the techniques vary in their degree of reversibility. DR can also solve the problem of data redundancy, eliminating colinear or heavily interdependent features. We

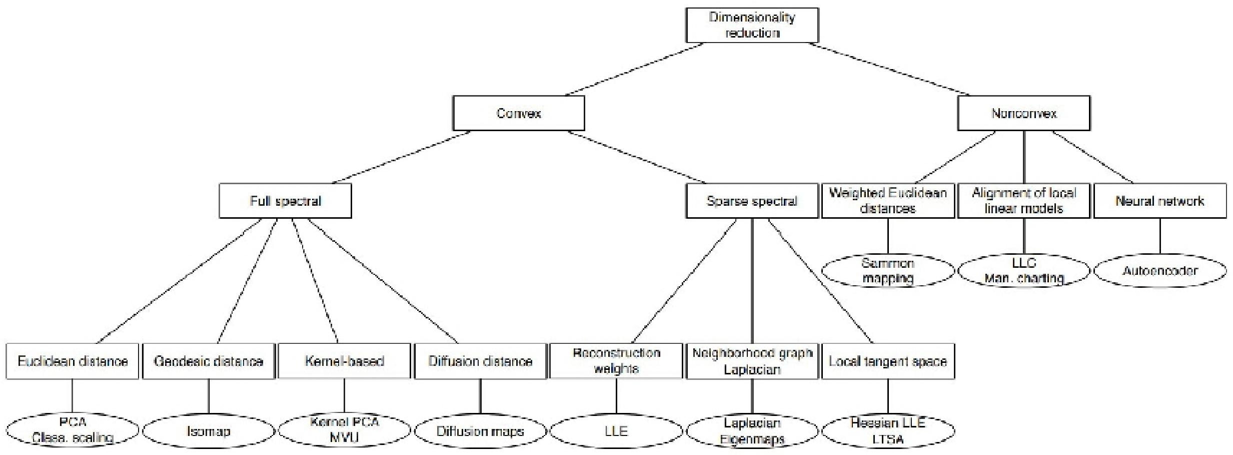


Figure 2. A taxonomy of DR techniques, reproduced from [11].

can also use these tools to identify the most explanatory features, i.e. those that account for the most variance in the data. In short, DR is used broadly in machine learning and data science for many diverse applications.

1.2. **Evaluating DR Performance.** Many DR techniques were introduced to meet a particular need, which is expressed as the objective function optimized by the technique. This objective function serves as a convenient proxy for determining the success of the technique. However, when comparing DR tools, it is necessary to consider other efficacy measures.

Some popular DR tools, such as Principle Component Analysis and Non-negative Matrix Factorization, operate through approximate matrix decompositions which can be reversed by multiplying the factors they produce. For these techniques, it is possible to directly compare the original matrix and the reconstructed matrix, allowing us to evaluate the efficacy of these compression tools. However, this method does not indicate the value of the actual low-dimensional representation of the data. It is important, therefore, to identify evaluation methods that directly compare important properties of the high-dimensional and low-dimensional representations of the data. These are the main subject of our review.

2. Some popular DR techniques

Before we discuss common methods for evaluating DR tools, we must introduce a few such tools. DR techniques come in a wide variety. Figure 2 shows a popular taxonomy of DR techniques, proposed by [11]. We will introduce only three techniques here, emphasizing the most elementary. We will consider Principal Component Analysis in detail, primarily to introduce one DR algorithm completely for any reader who has not previously encountered one. We will then briefly introduce Non-Negative Matrix Factorization and Uniform Manifold Projection.

2.0.1. *Principal Component Analysis (PCA).* PCA is closely related to Singular Value Decomposition (SVD). It identifies the components of the data associated with the highest singular values as principal, or most significant. Recall our general example where $\boldsymbol{X}$ is a real $d \times n$ matrix whose rows $\boldsymbol{x}_i$ are data points expressed in n-dimensions. PCA seeks a compressing $m \times n$ matrix $\boldsymbol{W}$ and a decompression $n \times m$ matrix $\boldsymbol{U}$ such that $m < n$ and the lower-dimensional representation $\boldsymbol{WX}$ encodes as much information about $\boldsymbol{X}$ as possible. To this end we phrase the PCA problem as

$$\arg\min_{\boldsymbol{W},\boldsymbol{U}} \sum_{i=1}^{d} \left\|\boldsymbol{x}_i - \boldsymbol{UW}\boldsymbol{x}_i\right\|_2^2 . \tag{2.1}$$

We emphasize that the norm used here is typically the Euclidean 2-norm.

This optimization is solved by applying two convenient theorems (as described in [9, p. 324-336]).

Theorem 1. *For any $(\boldsymbol{U}, \boldsymbol{W})$ that solves 2.1, we have $\boldsymbol{W} = \boldsymbol{U}^T$ and $\boldsymbol{WU} = \boldsymbol{I}_m$, where $\boldsymbol{I}_m$ is the $m \times m$ identity matrix.*

This theorem indicates that $\boldsymbol{U}$ is orthonormal.

Proof. Let $(\boldsymbol{U}, \boldsymbol{W})$ be a solution to 2.1. The range R of the function $f(\boldsymbol{X}) = \boldsymbol{UWX}$ is a linear subspace of $\mathbb{R}^n$. Suppose $\boldsymbol{V}$ is a $m \times d$ matrix composed so its columns form an orthonormal basis of R. This means that $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}_m$ and the range of $\boldsymbol{V}$ is $\{\boldsymbol{Vx} : \boldsymbol{x} \in \mathbb{R}^n\} = R$. We can represent all vectors in R with this basis, so for all $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$ it follows that

$$\begin{aligned}
\left\|\boldsymbol{x} - \boldsymbol{Vy}\right\|_2^2 &= \left\|\boldsymbol{x}\right\|^2 + \boldsymbol{y}^T\boldsymbol{V}^T\boldsymbol{Vy} - 2\boldsymbol{y}^T\boldsymbol{V}^T\boldsymbol{x} \\
&= \left\|\boldsymbol{x}\right\|^2 + \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T\boldsymbol{V}^T\boldsymbol{x} \\
&= \left\|\boldsymbol{x}\right\|^2 + \left\|\boldsymbol{y}\right\|^2 - 2\boldsymbol{y}^T\left(\boldsymbol{V}^T\boldsymbol{x}\right).
\end{aligned}$$

We minimize this by differentiating with respect to $\boldsymbol{y}$ and setting the expression equal to zero, yielding

$$\begin{aligned}
2(\nabla\boldsymbol{y})\,\boldsymbol{y} &= 2(\nabla\boldsymbol{y})\left(\boldsymbol{V}^T\boldsymbol{x}\right) \\
\boldsymbol{y} &= \boldsymbol{V}^T\boldsymbol{x}.
\end{aligned}$$

Therefore the minimization

$$\arg\min_{\boldsymbol{x}^* \in R} \left\|\boldsymbol{x} - \boldsymbol{x}^*\right\|_2^2 = \boldsymbol{V}\left(\boldsymbol{V}^T\boldsymbol{x}\right)$$

holds for all $\boldsymbol{x} \in \mathbb{R}^n$, including the row vectors $\boldsymbol{x}_i$ that compose $\boldsymbol{X}$. It follows that $\left(\boldsymbol{V}, \boldsymbol{V}^T\right)$ satisfies

$$\sum_{i=1}^{d} \left\|\boldsymbol{x}_i - \boldsymbol{UW}\boldsymbol{x}_i\right\|_2^2 \geq \sum_{i=1}^{d} \left\|\boldsymbol{x}_i - \boldsymbol{VV}^T\boldsymbol{x}_i\right\|_2^2,$$

and since $(\boldsymbol{U}, \boldsymbol{W})$ is a minimal solution to 2.1 we see that $\boldsymbol{U} = \boldsymbol{V}$ and $\boldsymbol{W} = \boldsymbol{V}^T = \boldsymbol{U}^T$. □

This theorem simplifies our PCA optimization problem to

$$\arg\min_{\boldsymbol{U}} \sum_{i=1}^{d} \left\|\boldsymbol{x}_i - \boldsymbol{UU}^T\boldsymbol{x}_i\right\|_2^2 .$$

We may also use the manipulation

$$\begin{aligned}\left\|\boldsymbol{x}-\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x}\right\|_2^2 &= \|\boldsymbol{x}\|_2^2 - 2\boldsymbol{x}^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x} \\ &= \|\boldsymbol{x}\|_2^2 - 2\boldsymbol{x}^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x} \\ &= \|\boldsymbol{x}\|_2^2 - \boldsymbol{x}^T\boldsymbol{U}\boldsymbol{U}^T\boldsymbol{x} \\ &= \|\boldsymbol{x}\|_2^2 - \text{trace}\left[\boldsymbol{U}^T\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{U}\right],\end{aligned}$$

and the fact that the trace is a linear operator to rewrite the PCA optimization as

$$\arg\max_{U\in\mathbb{R}^{n,m}:\boldsymbol{U}^T\boldsymbol{U}=\boldsymbol{I}_m} \text{trace}\left[\boldsymbol{U}^T\left(\sum_{i=1}^d \boldsymbol{x}_i\boldsymbol{x}_i^T\right)\boldsymbol{U}\right]. \tag{2.2}$$

Consider the symmetric matrix $\boldsymbol{A}=\sum_{i=1}^d \boldsymbol{x}_i\boldsymbol{x}_i^T$ in the above expression: since $\boldsymbol{A}$ is real-valued it is a normal matrix, and the Singular Value Decomposition [10, Theorem 5.17] gives us

$$\boldsymbol{A}=\boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T$$

where $\boldsymbol{D}$ is a diagonal matrix whose nonzero entries are the eigenvalues of $\boldsymbol{A}$ and the columns of $\boldsymbol{V}$ are the eigenvectors, with $\boldsymbol{V}^T\boldsymbol{V}=\boldsymbol{V}\boldsymbol{V}^T=\boldsymbol{I}_m$. (Under these advantageous circumstances the SVD is called the *Spectral Decomposition.*) The fact that $\boldsymbol{A}$ is positive semidefinite tells us that these eigenvalues are all nonnegative. If this ordering is not already in place, we rewrite $\boldsymbol{D}$ and $\boldsymbol{V}$ so the eigenvalues are in descending order, i.e. $\boldsymbol{D}_{1,1}\geq\boldsymbol{D}_{2,2}\geq\cdots\geq\boldsymbol{D}_{n,n}$. Let $\boldsymbol{U}^*$ be the $n\times m$ matrix whose columns are the first m eigenvectors of $\boldsymbol{A}$ whose eigenvalues are maximal. We wish to show that $\boldsymbol{U}^*$ solves 2.2.

Theorem 2. *(23.2 in* [9]*) Given that $\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_d\in\mathbb{R}^n$, $\boldsymbol{A}=\sum_{i=1}^d \boldsymbol{x}_i\boldsymbol{x}_i^T$, and $\boldsymbol{U}^*$ the matrix whose columns $\boldsymbol{u}_1,\boldsymbol{u}_2,\ldots,\boldsymbol{u}_m$ are the first m eigenvectors of $\boldsymbol{A}$ whose corresponding eigenvalues are maximal, $\boldsymbol{U}^*$ solves 2.2 (and therefore satisfies the PCA problem 2.1 with $\boldsymbol{W}=(\boldsymbol{U}^*)^T$.*

Proof. We will only sketch the proof for brevity. We can obtain the spectral decomposition $\boldsymbol{A}=\boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T$, propose some arbitrary matrix with orthonormal columns called $\boldsymbol{U}\in\mathbb{R}^{n,m}$, and finally, create the matrix $\boldsymbol{B}=\boldsymbol{V}^T\boldsymbol{U}$. Since

$$\boldsymbol{V}\boldsymbol{B}=\boldsymbol{V}\boldsymbol{V}^T\boldsymbol{U}=\boldsymbol{U}$$

we may write

$$\begin{aligned}\boldsymbol{U}^T\boldsymbol{A}\boldsymbol{U} &= \boldsymbol{B}\quad{}^T\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{B} \\ &= \quad\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}.\end{aligned}$$

This means that

$$\text{trace}\left[\boldsymbol{U}^T\boldsymbol{A}\boldsymbol{U}\right]=\sum_{j=1}^n \boldsymbol{D}_{j,j}\sum_{i=1}^m \boldsymbol{B}_{j,i}^2$$

We can show $\boldsymbol{B}$ has orthonormal columns and from this obtain$\sum_{j=1}^n\sum_{i=1}^m \boldsymbol{B}_{j,i}^2=m$. Suppose $\tilde{\boldsymbol{B}}$ is a matrix in $\mathbb{R}^{n,n}$ whose columns are the first n columns of $\boldsymbol{B}$. Further suppose that $\tilde{\boldsymbol{B}}^T\tilde{\boldsymbol{B}}=\boldsymbol{I}_n$. This would mean that $\sum_{i=1}^m \tilde{\boldsymbol{B}}_{j,i}^2=1$ and thus

$\sum_{i=1}^{m} \boldsymbol{B}_{j,i}^2 \leq 1$. We wrap up by using these facts to state

$$\begin{aligned} \operatorname{trace}\left[\boldsymbol{U}^T \boldsymbol{A} \boldsymbol{U}\right] &\leq \sum_{j=1}^{n} \boldsymbol{D}_{j,j} \\ &= \max_{\beta \in [0,1]^n : \|\beta\|_1 \leq n} \sum_{j=1}^{n} \boldsymbol{D}_{j,j} \beta_j . \end{aligned}$$

Since $\operatorname{trace}\left[\left(\boldsymbol{U}^*\right)^T \boldsymbol{A} \boldsymbol{U}^*\right] = \sum_{j=1}^{n} \boldsymbol{D}_{j,j}$, this concludes the sketched proof. □

Because the SVD is extremely fast to compute, PCA remains one of the most popular DR tools in use today.

2.0.2. *Non-negative Matrix Factorization (NMF).* NMF is yet another extremely efficient DR tool with a long history founded on simple linear algebra. When $\boldsymbol{X}$ happens to be non-negative, we may state the DR problem

$$\arg\min_{\boldsymbol{T},\boldsymbol{D}} \|\boldsymbol{X} - \boldsymbol{T}\boldsymbol{D}\|_F$$

such that $\boldsymbol{T}$ and $\boldsymbol{D}$ are nonnegative, where $\|\cdot\|_F$ is the Frobenius norm. This optimization is typically performed using Hierarchical Alternating Least Squares methods, which take turns optimizing each matrix.

2.0.3. *Uniform Manifold Projection (UMAP).* UMAP is a comparatively recent addition to the DR toolbox. It fills a gap as a DR technique that optimizes the preservation of local distance structures while also providing provable performance bounds [7]. While the details are outside the scope of this review, UMAP works by first establishing a fuzzy topological model for the high-dimensional data, and then using a spectral embedding to project this into a lower-dimensonal space, finally applying a force-directed algorithm to optimize the embedding. UMAP has become the de facto favorite tool for producing two and three-dimensional visualizations of high-dimensional datasets.

3. Some popular DR evaluation techniques

Strategies for evaluating DR performance may be classed in several ways, but perhaps the most intuitive is in terms of the structural level they focus on. *Local* strategies aim to gauge a DR tool's preservation of information about pairwise distances in local neighborhoods within the dataset. *Global* strategies concern themselves instead with a DR tool's preservation of information about the global distance structure. We will consider four DR evaluation tools here: two that focus on local neighborhood structures, one that emphasizes global structure, and one that mixes the two. For the following discussions, let n be the dimensionality of the original data and let m be the lower dimension the data will be represented in. Thus $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_d$ are real vectors in $\mathbb{R}^n$ and $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_d$ are the corresponding real vectors in $\mathbb{R}^m$.

3.0.4. *Kruskal Stress (KS).* The term *stress* in the context of DR evaluation means some measure of the inconsistency between global corresponding pairwise distances in the higher and lower-dimensional data representations, and several such measures of stress exist. Kruskal stress at its simplest is essentially a residual sum of squares computation,

$$ks_D(\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_d) = \left(\sum_{i \neq j = 1 \ldots d} \left(\delta_{ij} - \|\zeta_{ij}\|^2 \right) \right)^{\frac{1}{2}},$$

where δ_{ij} and ζ_{ij} represent the corresponding pairwise (Euclidean) distances in the higher and lower-dimensional embeddings (respectively)[4]. First proposed in 1964, KS is one of the oldest DR evaluation tools in use.

3.0.5. *The Coranking Matrix.* It is common to present the pairwise distances in both embeddings using distance matrices. The distance matrix corresponding to an embedding $\boldsymbol{M} = [m_{ij}]$ uses $m_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ to represent the distance from the ith data point to the jth data point. Since symmetric distance functions are typically used, this matrix is usually symmetric.

When we are interested in describing the integrity of localized distance preservation, these distance matrices are excessive. We may instead choose to work with ranking matrices $\boldsymbol{R} = [r_{ij}]$ where

$$r_{ij} = \# \{k : m_{ik} < m_{ij} \text{ or } (m_{ik} = m_{ij} \text{ and } k < j)\}$$

is a positive integer indicating the ranked distance of point j from point i, i.e. the number of other points closer to j. (Note that we use # to represent set cardinality.)

From this we may abstract one step further to the coranking matrix [12]. Let $\boldsymbol{R}$ and $\boldsymbol{R}'$ be the ranking matrices for the higher and lower-dimensional embeddings (respectively). We define the coranking matrix $\boldsymbol{C} = [c_{kl}]$ by

$$c_{kl} = \# \left\{ (i,j) : r_{ij} = k \text{ and } r'_{ij} = l \right\}.$$

In other words, the entry c_{kl} represents the number of times the DR transformation reordered the kth-ranked point as the lth-ranked point in the ranking matrices. This structure is extremely convenient, as it groups the local and distant errors into separate regions of the matrix, as shown in figure 3. When the DR transformation brings a distant point closer, we call this an *intrusion.* When the DR transformation pushes a nearby point further away, we call this an *extrusion.* Nonzero elements in the upper-right and lower-left corners of the matrix represent extreme ("hard") changes to local structure, i.e. signs that local relative distances are not preserved by the transformation. Nonzero elements in the top left region represent "mild" local changes, which are typically less concerning. Nonzero elements in the lower right region represent distant changes, i.e. signs that non-local relative distances were disturbed by the transformation. Typically these errors the least concerning of any, and this area is called the trivial region. The boundaries of these regions are determined by some choice of k between 1 and d, the number of data points. Smaller k choices favor tighter local neighborhoods. The diagonal represents unaltered rankings, i.e. ranks that are unchanged by the transformation. The ideal coranking matrix would be a diagonal matrix.

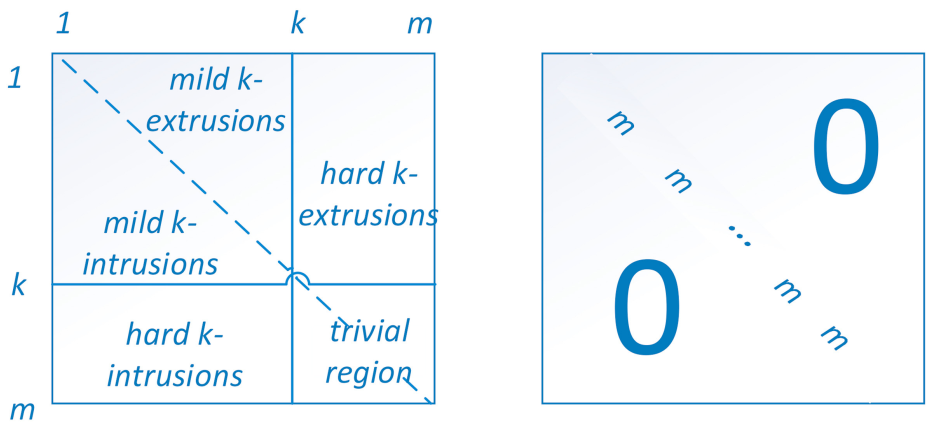


FIGURE 3. The anatomy of the coranking matrix.The matrix on the left shows the regions within the corankng matrix (note that m represents the number of data points, elsewhere refered to as d). The matrix on the right represents an "ideal" coranking matrix. Reproduced from [12].

We will designate these regions with the following notation.

Notation	Meaning
$\mathbb{F}_k = \{1, \ldots, k\}$	First k
$\mathbb{L}_k = \{k+1, \ldots, d-1\}$	Last $d-k$
$\mathbb{UL}_k = \mathbb{F}_k \times \mathbb{F}_k$	Upper-left block
$\mathbb{UR}_k = \mathbb{F}_k \times \mathbb{L}_k$	Upper-right block, hard k-extrusions
$\mathbb{LL}_k = \mathbb{L}_k \times \mathbb{F}_k$	Lower-left block, hard k-intrusions
$\mathbb{UL}_k = \mathbb{L}_k \times \mathbb{L}_k$	Lower-right block, trivial region
$\mathbb{D}_k = \{(i,i) : 1 \leq i \leq k\}$	Diagonal up to k
$\mathbb{LT}_k = \{(i,j) : 1 < i \leq k \text{ and } j < i\}$	Mild k-intrusions
$\mathbb{UT}_k = \{(i,j) : 1 \leq i < k \text{ and } j > i\}$	Mild k-extrusions

Several DR evaluation measures are derived by taking weighted sums over elements in the coranking matrix that favor regions of interest [5]. We will consider two.

3.0.6. *Coranking Behavior Criterion.* We may express the fraction of mild k-intrusions as

$$U_N(k) = \frac{1}{kd} \sum_{(k,l) \in \mathbb{LT}_k} c_{kl}$$

and the fraction of mild k-extrusions as

$$U_X(k) = \frac{1}{kd} \sum_{(k,l) \in \mathbb{UT}_k} c_{kl}.$$

We define the coranking behavior indicator as

$$B_{NX}(k) = U_X(k) - U_N(k).$$

This indicates how extrusive or intrusive the DR transformation is using more negative or more positive outputs. The coranking behavior criterion is computed by sampling the value of $B_{NX}(k)$ for various k inputs and computing the area under the curve (AUC) of this function. By computing the AUC we favor more local neighborhoods.

3.0.7. *Coranking Quality Criterion.* We compute the fraction of unchanged rank assignments within a k-neighborhood as

$$U_P\left(k\right) = \frac{1}{kd} \sum_{(k,l)\in\mathbb{D}_k} c_{kl}.$$

This gives us the coranking quality indicator

$$Q_{NX}\left(k\right) = U_P\left(k\right) + U_N\left(k\right) + U_X\left(k\right),$$

which is simply the sum over elements in $\mathbb{UL}_k$. This represents how often ranks were kept within the k-neighborhood by the transformation. Once again we sample $Q_{NX}\left(k\right)$ using various k values and compute the AUC to obtain the coranking quality criterion. This criterion also emphasizes local consistency over global.

3.0.8. *Mean Relative Rank Errors (MRRE).* Let v_i^k represent the k-neighborhood around data point i in the original embedding, and let n_i^k represent the k-neighborhood about that point in the new low-dimensional embedding. The two MRRE are defined as

$$\begin{aligned} W_n\left(k\right) &= \frac{1}{H_k}\sum_{i=1}^{d}\sum_{j\in n_i^k}\frac{\left|r_{ij}-r'_{ij}\right|}{r_{ij}} = \frac{1}{H_k}\sum_{(k,l)\in\mathbb{UL}_k\cup\mathbb{UR}_k}\frac{|k-l|}{l}q_{kl} \\ W_v\left(k\right) &= \frac{1}{H_k}\sum_{i=1}^{d}\sum_{j\in v_i^k}\frac{\left|r_{ij}-r'_{ij}\right|}{r'_{ij}} = \frac{1}{H_k}\sum_{(k,l)\in\mathbb{UL}_k\cup\mathbb{UR}_k}\frac{|k-l|}{l}q_{kl}\end{aligned}$$

where

$$H_k = d\sum_{\kappa=1}^{k}\frac{|d-2\kappa+1|}{\kappa}$$

is a normalizing factor that varies only in k [5]. The first MRRE measures all k-intrusions and the mild k-extrusions, while the second measures all k-extrusions and only the mild k-intrusions. The MRRE give a mixed local and global perspective about the perturbations to the pointwise distance structure caused by the DR transformation.

4. A demonstration of DR evaluation

We will devote the remainder of this work to a practical demonstration of the DR techniques and evaluation methods discussed above.

4.1. **Example datasets.** Our first real-world problem comes from the realm of social network analysis. For this demonstration we will use a dataset that represents a hero social network in the Marvel comic universe [1]. This network contains 224k+ weighted edges, which is more than enough to pose a challenge for visualization. For our demonstration, however, we will use only the first 2k weighted edges. From this we form a sparse adjacency matrix.

Our second dataset is taken from the 20-Newsgroup text data [2][6].While this classic dataset from early internet history is subdivided into twenty discussion forums, we use only the 910 posts from the "talk.politics.guns" message board. We preprocess this data using an elementary English language stemmer and vectorization strategy to reduce the vocabulary to 3631 meaningful words, and use a bag-of-words model to represent each post in the dataset. This means that our data is represented as a matrix whose row vectors are posts, and each row vector's entries represent the number of occurrences of a given vocabulary word in that post. This preprocessing step is quite elementary compared to the typical overhead of setting up a topic modeling problem, but it is sufficient for our demonstration.

4.2. **Methodology.** The entire demonstration is provided as an appendix to this paper, in the form of a Jupyter Notebook. All of our work is done in Python. We take advantage of two dimensionality reduction evaluation libraries. The first is pyDRMetrics, a recently developed toolkit for this purpose [12]. It unfortunately suffers from extreme latency, which rendered it difficult to apply to our problems at hand. The second library is the backend of the tool TALE [8]. This is a promising tool for evaluating low-dimensional embeddings that is still under construction. Unfortunately it appears that their implementation of MRRE is faulty, as this returned only zeros during our experiment. A further review of the code will be necessary to identify the cause of this issue.

For each dataset and each DR technique, we use default parameters and reduce to two different dimension sizes, 2 and 20. The former represents the task of visualizing the data, and the latter represents a clustering approach.

4.3. **Results.** The experimental results are summarized in Figure 4. In both tasks we found that PCA minimized Kruskal Stress better than NMF or UMAP, especially in the higher dimensional embedding, meaning that its global distance preservation was best out of the three. UMAP performed the best in terms of coranking quality in the lower dimension embeddings, meaning that it preserved local distances best, but this advantage did not hold in the higher-dimensional graph embeddings. The ideal coranking score is zero, and this metric varied between datasets and embedding dimensions, but was always positive, demonstrating that all three DR tools are more intrusive than extrusive.

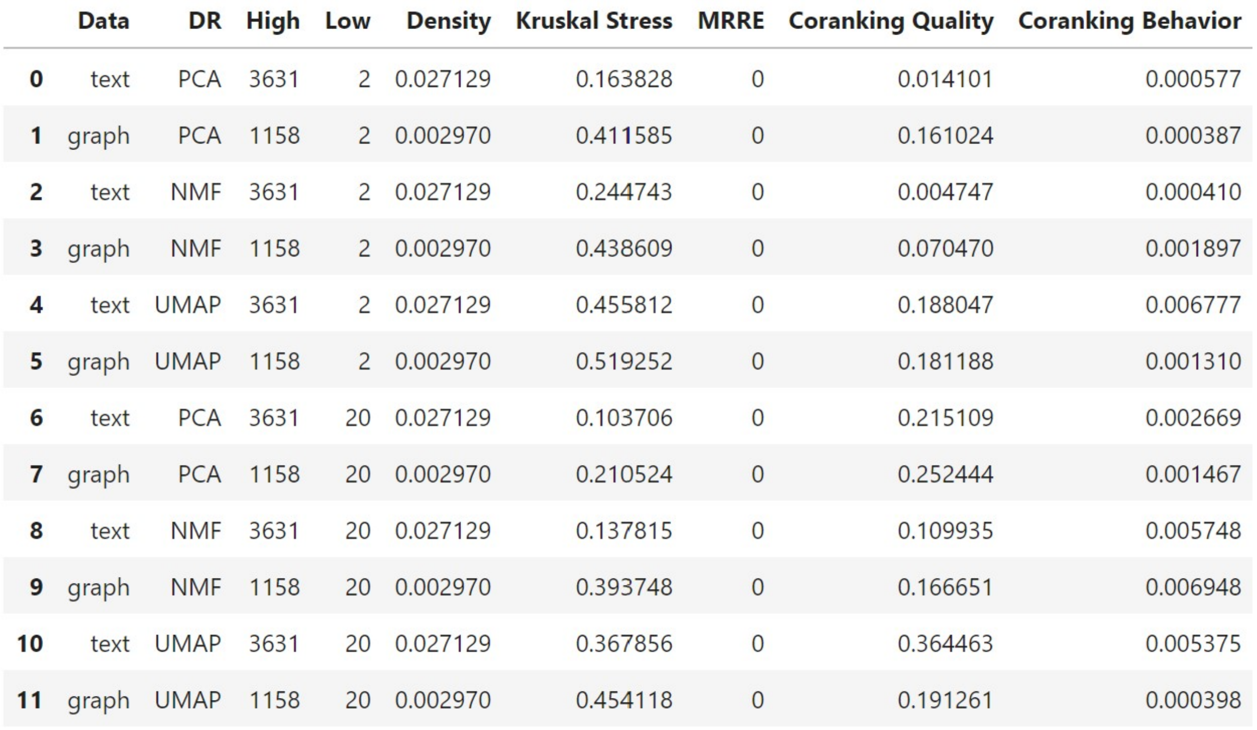
These results suggest (correctly, according to our bias) that the best DR tool of the three for visualization tasks is UMAP, and the best tool for global distance preservation is PCA.

5. Conclusion

We have seen that many tools for evaluating DR performance exist, and that these vary greatly in purpose. These methods provide us with a variety of insights into the comparative functions of various DR techniques, and enable thorough post-mortem review of DR performance. We selected only a few evaluation methods for review here, and we could certainly add more to this toolbox. In particular, there is room for evaluation tools that combine sensitivity to both local and global distance integrity, favoring local neighborhood preservation while also warning against unacceptable levels of global perturbation.

Another important follow-on to this work would be to scale the experiments to more realistically-challenging data sizes. This would likely require writing new implementations of the DR evaluation tools that are more computationally efficient.

A final but important caution is that we did not exhaustively explore the parameter space of the DR techniques used. All of our conclusions voiced in the



	Data	DR	High	Low	Density	Kruskal Stress	MRRE	Coranking Quality	Coranking Behavior
0	text	PCA	3631	2	0.027129	0.163828	0	0.014101	0.000577
1	graph	PCA	1158	2	0.002970	0.411585	0	0.161024	0.000387
2	text	NMF	3631	2	0.027129	0.244743	0	0.004747	0.000410
3	graph	NMF	1158	2	0.002970	0.438609	0	0.070470	0.001897
4	text	UMAP	3631	2	0.027129	0.455812	0	0.188047	0.006777
5	graph	UMAP	1158	2	0.002970	0.519252	0	0.181188	0.001310
6	text	PCA	3631	20	0.027129	0.103706	0	0.215109	0.002669
7	graph	PCA	1158	20	0.002970	0.210524	0	0.252444	0.001467
8	text	NMF	3631	20	0.027129	0.137815	0	0.109935	0.005748
9	graph	NMF	1158	20	0.002970	0.393748	0	0.166651	0.006948
10	text	UMAP	3631	20	0.027129	0.367856	0	0.364463	0.005375
11	graph	UMAP	1158	20	0.002970	0.454118	0	0.191261	0.000398

FIGURE 4. A summary of the experimental results. More detail is available in the appendix.

Results section above apply exclusively to the default settings for each algorithm. It is quite possible that these conclusions would not apply if we properly tuned the parameters. However, if they did hold under more thorough investigation, these observations would provide significant objective evidence to support popular rules of thumb for DR tool selection.

References

1. Ricardo Alberich, Joe Miro-Julia, and Francesc Rosselló, *Marvel universe looks almost like a real social network*, arXiv preprint cond-mat/0202174 (2002).
2. Gracinda Carvalho, David Martins de Matos, and Vitor Rocio, *Document retrieval for question answering: a quantitative evaluation of text preprocessing*, Proceedings of the ACM first Ph. D. workshop in CIKM, 2007, pp. 125–130.
3. Mike Fritze, Patrick Cheetham, Jennifer Lato, and Paul Syers, *The death of mooreâĂŹs law*, STEPS: Science. Technology. Engineering and Policy Studies (2016), no. 3, 35.
4. Joseph B Kruskal, *Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis*, Psychometrika **29** (1964), no. 1, 1–27.
5. John A Lee and Michel Verleysen, *Quality assessment of dimensionality reduction: Rank-based criteria*, Neurocomputing **72** (2009), no. 7-9, 1431–1443.
6. Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf,

Datasets: A community library for natural language processing, Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (Online and Punta Cana, Dominican Republic), Association for Computational Linguistics, November 2021, pp. 175–184.

7. Leland McInnes, John Healy, and James Melville, *Umap: Uniform manifold approximation and projection for dimension reduction*, arXiv preprint arXiv:1802.03426 (2018).
8. Raphael Mitsch, *Tale*, `https://github.com/rmitsch/TALE`, 2013.
9. Shai Shalev-Shwartz and Shai Ben-David, *Understanding machine learning: From theory to algorithms*, Cambridge university press, 2014.
10. Thomas S Shores, *Applied linear algebra and matrix analysis*, vol. 2541, Springer, 2007.
11. Laurens Van Der Maaten, Eric Postma, Jaap Van den Herik, et al., *Dimensionality reduction: a comparative*, J Mach Learn Res **10** (2009), no. 66-71, 13.
12. Yinsheng Zhang, Qian Shang, and Guoming Zhang, *pydrmetrics-a python toolkit for dimensionality reduction quality assessment*, Heliyon **7** (2021), no. 2, e06199.

E-mail address: `nicholasalines@gmail.com`

Dimensionality_Reduction_Evaluation_Experiments

April 17, 2022

1 Dimensionality Reduction Evaluation Code Appendix

This code is intended to accompany the final project paper for EN.625.609.83.SP22 Matrix Theory.

```
[1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

1.1 Enviornment preparation

```
[33]: import pickle
      from tqdm.notebook import tqdm
```

The following packages are introduced so we can manage the Marvel graph data.

```
[2]: import networkx as nx
     import pandas as pd
```

These next packages are imported so we can handle the 20 Newsgroups text data.

```
[3]: import Stemmer
     from datasets import load_dataset
     from sklearn.feature_extraction.text import CountVectorizer
```

The following packages are used to perform dimensionality reduction.

```
[4]: from sklearn.decomposition import PCA, NMF
     from umap import UMAP
     import plotly.express as px
```

The next cells are for tools for evaluating dimensionality reduction.
For this next cell you'll first need to run

```
pip install git+https://github.com/samueljackson92/coranking.git
```

```
[66]: from coranking.metrics import trustworthiness, continuity
```

pyDRMetrics is a great library, but very, very slow.

```
[5]: from pyDRMetrics.pyDRMetrics import *
```

TALE is a standalone tool in Python and Java for reviewing low-dimensional projections. We will pull in some classes and functions from the backend. It is still under construction, and the backend was not intended for this kind of use, so our implementation is a little hacky. All credit for that backend code goes to the TALE team.

```
[6]: from coranking.metrics import trustworthiness, continuity
     import six
     import sys
     sys.modules['sklearn.externals.six'] = six
     import talebackend
```

```
[67]: from importlib import reload
```

```
[132]: talebackend = reload(talebackend)
```

1.2 Data preparation

1.2.1 Marvel Character Network

Before running this next cell, you must download and extract the data from the Kaggle dataset, placing it in the superdirectory `Data`.

```
[7]: hdf = pd.read_csv("../Data/hero-network.csv")
```

```
[8]: hdf.shape
```

```
[8]: (574467, 2)
```

```
[9]: udf = hdf.drop_duplicates()
```

```
[10]: udf.shape
```

```
[10]: (224181, 2)
```

```
[11]: columns = hdf.columns.tolist()
      hdf["weights"] = 1
      #hdf = hdf.groupby(columns)["weights"].count().reset_index()
      tdf = hdf.groupby(columns)["weights"].count()
      tdf = tdf.reset_index()
```

```
[12]: tdf.weights.value_counts()
```

```
[12]: 1      138829
      2       39326
```

```
3       16532
4        8548
5        5061
        …
168         1
351         1
295         1
167         1
184         1
Name: weights, Length: 200, dtype: int64
```

[13]:
```
tdf[tdf.weights>=150]
```

[13]:
```
                       hero1                 hero2  weights
3090    ANGEL/WARREN KENNETH  BEAST/HENRY &HANK& P      156
3346    ANGEL/WARREN KENNETH  ICEMAN/ROBERT BOBBY       162
4408    ANT-MAN/DR. HENRY J.       CAPTAIN AMERICA      154
4623    ANT-MAN/DR. HENRY J.  IRON MAN/TONY STARK       156
5049    ANT-MAN/DR. HENRY J.  WASP/JANET VAN DYNE       195
…                        …                     …       …
212460  WASP/JANET VAN DYNE        CAPTAIN AMERICA      193
212669  WASP/JANET VAN DYNE   IRON MAN/TONY STARK       152
214080  WATSON-PARKER, MARY   SPIDER-MAN/PETER PAR      321
218871      WOLVERINE/LOGAN   COLOSSUS II/PETER RA      159
219555      WOLVERINE/LOGAN   STORM/ORORO MUNROE S      198

[75 rows x 3 columns]
```

[87]:
```
mini_tdf = tdf[tdf.index<=2000]
```

[88]:
```
G = nx.from_pandas_edgelist(df=mini_tdf, source="hero1", target="hero2",␣
 ↪edge_attr="weights")
```

[89]:
```
nx.draw_spring(G)
```

[90]:
```
marvel_matrix = nx.adjacency_matrix(G,weight="weights")
```

```
<ipython-input-90-278252ef8d39>:1: FutureWarning: adjacency_matrix will return a
scipy.sparse array instead of a matrix in Networkx 3.0.
  marvel_matrix = nx.adjacency_matrix(G,weight="weights")
```

[137]:
```
n = marvel_matrix.count_nonzero()
x,y = marvel_matrix.shape
dm = n / (x * y)
print("The Marvel matrix has density", dm)
```

```
The Marvel matrix has density 0.0029695055199095576
```

1.2.2 A bag of words model matrix

[18]:
```
# Override TfidfVectorizer
class StemmedCountVectorizer(CountVectorizer):
    def build_analyzer(self):
        analyzer = super(CountVectorizer, self).build_analyzer()
        return lambda doc: stemmer.stemWords(analyzer(doc))
```

[19]:
```
dataset = load_dataset("newsgroup",'18828_talk.politics.guns')
text_data = dataset['train']['text']
stemmer = Stemmer.Stemmer('en')
analyzer = CountVectorizer().build_analyzer()
vectorizer = StemmedCountVectorizer(stop_words='english', min_df=5, max_df=0.5)
text_matrix = vectorizer.fit_transform(text_data)
vocab = vectorizer.get_feature_names_out()
```

```
Reusing dataset newsgroups (/home/nick/.cache/huggingface/datasets/newsgroups/18
828_talk.politics.guns/3.0.0/8ea0c9dc025ecfbfd96a2c1e22caa1e1281f361946dac082054
48d66f78398f5)

  0%|          | 0/1 [00:00<?, ?it/s]
```

[145]:
```
text_matrix.shape
```

[145]:
```
(910, 3631)
```

[136]:
```
n = text_matrix.count_nonzero()
x,y = text_matrix.shape
dt = n / (x * y)
print("The text matrix has density", dt)
```

```
The text matrix has density 0.027128723658605233
```

1.3 Performing dimensionality reduction

[92]:
```
low = 2
med = 20
X_t = text_matrix.toarray()
X_m = marvel_matrix.toarray()
```

[93]:
```
pcatlow = PCA(n_components = low) # keep the first low components
pcatlow.fit(X_t)
Z_t_pca_low = pcatlow.transform(X_t)

pcatmed = PCA(n_components = med) # keep the first med components
pcatmed.fit(X_t)
Z_t_pca_med = pcatmed.transform(X_t)
```

[94]:
```
pcamlow = PCA(n_components = low) # keep the first low components
pcamlow.fit(X_m)
Z_m_pca_low = pcamlow.transform(X_m)

pcammed = PCA(n_components = med) # keep the first med components
pcammed.fit(X_m)
Z_m_pca_med = pcammed.transform(X_m)
```

[95]:
```
nmftlow = NMF(n_components = low, init='nndsvda', beta_loss="kullback-leibler",␣
 ↪solver="mu", max_iter=(1000)) # keep the first low components
nmftlow.fit(X_t)
Z_t_nmf_low = nmftlow.transform(X_t)

nmftmed = NMF(n_components = med, init='nndsvda', beta_loss="kullback-leibler",␣
 ↪solver="mu", max_iter=(1000)) # keep the first med components
nmftmed.fit(X_t)
Z_t_nmf_med = nmftmed.transform(X_t)
```

[96]:
```
nmfmlow = NMF(n_components = low, init='nndsvda', beta_loss="kullback-leibler",␣
 ↪solver="mu", max_iter=(1000)) # keep the first low components
nmfmlow.fit(X_m)
Z_m_nmf_low = nmfmlow.transform(X_m)

nmfmmed = NMF(n_components = med, init='nndsvda', beta_loss="kullback-leibler",␣
 ↪solver="mu", max_iter=(1000)) # keep the first med components
nmfmmed.fit(X_m)
Z_m_nmf_med = nmfmmed.transform(X_m)
```

[97]:
```
umaptlow = UMAP(n_components = low, init='random')
umaptlow.fit(X_t)
Z_t_umap_low = umaptlow.transform(X_t)

umaptmed = UMAP(n_components = med, init='random')
umaptmed.fit(X_t)
Z_t_umap_med = umaptmed.transform(X_t)
```

[98]:
```
umapmlow = UMAP(n_components = low, init='random')
umapmlow.fit(X_m)
Z_m_umap_low = umapmlow.transform(X_m)

umapmmed = UMAP(n_components = med, init='random')
umapmmed.fit(X_m)
Z_m_umap_med = umapmmed.transform(X_m)
```

This next cell just backs up our work so far.

[99]:
```
mats = [X_t, X_m]
lows =␣
 ↪[Z_t_pca_low,Z_m_pca_low,Z_t_nmf_low,Z_m_nmf_low,Z_t_umap_low,Z_m_umap_low]
meds =␣
 ↪[Z_t_pca_med,Z_m_pca_med,Z_t_nmf_med,Z_m_nmf_med,Z_t_umap_med,Z_m_umap_med]
with open("../Data/matrices.pkl", 'wb') as pfile:
    pickle.dump([mats, lows, meds], pfile)
```

1.4 Evaluate reductions

Kruskal stress

```
[122]: lows_ks = []
       for i in tqdm(range(len(lows))):
           s = talebackend.Stress(mats[i%2],lows[i])
           kruskal_stress = s.compute()
           lows_ks.append(kruskal_stress)
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

```
[124]: meds_ks = []
       for i in tqdm(range(len(meds))):
           s = talebackend.Stress(mats[i%2],meds[i])
           kruskal_stress = s.compute()
           meds_ks.append(kruskal_stress)
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

Coranking-based measures

```
[134]: lows_ms = []
       lows_bc = []
       lows_qc = []
       for i in tqdm(range(len(lows))):
           c = talebackend.CorankingMatrix(high_dimensional_data=mats[i%2],
                                           low_dimensional_data=lows[i],
                                           distance_metric='euclidean',)
           b = talebackend.
        ↪CorankingMatrixBehaviourCriterion(high_dimensional_data=mats[i%2],
                                                                 ␣
        ↪low_dimensional_data=lows[i],
                                                                 ␣
        ↪distance_metric='euclidean',
                                                                  coranking_matrix=c,
                                                                  )
           q = talebackend.
        ↪CorankingMatrixQualityCriterion(high_dimensional_data=mats[i%2],
                                                                 ␣
        ↪low_dimensional_data=lows[i],
                                                                 ␣
        ↪distance_metric='euclidean',
                                                                  coranking_matrix=c,
                                                                  )
           m = talebackend.MRRE(high_dimensional_data=mats[i%2],
                                low_dimensional_data=lows[i],
                                distance_metric='euclidean',
                                coranking_matrix=c,
                                )
           bc = b.compute()
           ms = m.compute()
           qc = q.compute()
           lows_ms.append(ms)
           lows_bc.append(bc)
           lows_qc.append(qc)
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

[135]:

```
meds_ms = []
meds_bc = []
meds_qc = []
for i in tqdm(range(len(meds))):
    c = talebackend.CorankingMatrix(high_dimensional_data=mats[i%2],
                                    low_dimensional_data=meds[i],
                                    distance_metric='euclidean',)
    b = talebackend.
 ↪CorankingMatrixBehaviourCriterion(high_dimensional_data=mats[i%2],
                                                                ␣
 ↪low_dimensional_data=meds[i],
                                                                ␣
 ↪distance_metric='euclidean',
                                                                 coranking_matrix=c,
                                                                 )
    q = talebackend.
 ↪CorankingMatrixQualityCriterion(high_dimensional_data=mats[i%2],
                                                                ␣
 ↪low_dimensional_data=lows[i],
                                                                ␣
 ↪distance_metric='euclidean',
                                                                 coranking_matrix=c,
                                                                 )
    m = talebackend.MRRE(high_dimensional_data=mats[i%2],
                         low_dimensional_data=meds[i],
                         distance_metric='euclidean',
                         coranking_matrix=c,
                         )
    bc = b.compute()
    ms = m.compute()
    qc = q.compute()
    meds_ms.append(ms)
    meds_bc.append(bc)
    meds_qc.append(qc)
```

```
  0%|          | 0/6 [00:00<?, ?it/s]
```

```
[143]: types = ["text", "graph"]
       dims = [X_t.shape[1], X_m.shape[1]]
       densities = [dt, dm]
       metrics = pd.DataFrame()
       metrics["Data"] = types * len(lows)
       metrics["DR"] = ["PCA","PCA","NMF","NMF","UMAP","UMAP"]*2
       metrics["High"] = dims * len(lows)
       metrics["Low"] = [low] * len(lows) + [med]*len(meds)
       metrics["Density"] = densities * len(lows)
       metrics["Kruskal Stress"] =  lows_ks + meds_ks
       metrics["MRRE"] =  lows_ms + meds_ms
       metrics["Coranking Quality"] = lows_qc + meds_qc
       metrics["Coranking Behavior"] = lows_bc + meds_bc
```

```
[144]: metrics
```

```
[144]:      Data    DR  High  Low   Density  Kruskal Stress  MRRE  Coranking Quality  \
       0    text   PCA  3631    2  0.027129        0.163828     0           0.014101
       1   graph   PCA  1158    2  0.002970        0.411585     0           0.161024
       2    text   NMF  3631    2  0.027129        0.244743     0           0.004747
       3   graph   NMF  1158    2  0.002970        0.438609     0           0.070470
       4    text  UMAP  3631    2  0.027129        0.455812     0           0.188047
       5   graph  UMAP  1158    2  0.002970        0.519252     0           0.181188
       6    text   PCA  3631   20  0.027129        0.103706     0           0.215109
       7   graph   PCA  1158   20  0.002970        0.210524     0           0.252444
       8    text   NMF  3631   20  0.027129        0.137815     0           0.109935
       9   graph   NMF  1158   20  0.002970        0.393748     0           0.166651
       10   text  UMAP  3631   20  0.027129        0.367856     0           0.364463
       11  graph  UMAP  1158   20  0.002970        0.454118     0           0.191261

           Coranking Behavior
       0             0.000577
       1             0.000387
       2             0.000410
       3             0.001897
       4             0.006777
       5             0.001310
       6             0.002669
       7             0.001467
       8             0.005748
       9             0.006948
       10            0.005375
       11            0.000398
```

```
[146]: metrics[metrics["Data"]=="text"]
```

```
[146]:     Data    DR  High  Low   Density  Kruskal Stress  MRRE  Coranking Quality  \
       0   text   PCA  3631    2  0.027129        0.163828     0           0.014101
       2   text   NMF  3631    2  0.027129        0.244743     0           0.004747
       4   text  UMAP  3631    2  0.027129        0.455812     0           0.188047
       6   text   PCA  3631   20  0.027129        0.103706     0           0.215109
       8   text   NMF  3631   20  0.027129        0.137815     0           0.109935
       10  text  UMAP  3631   20  0.027129        0.367856     0           0.364463

           Coranking Behavior
       0             0.000577
       2             0.000410
       4             0.006777
       6             0.002669
       8             0.005748
       10            0.005375
```

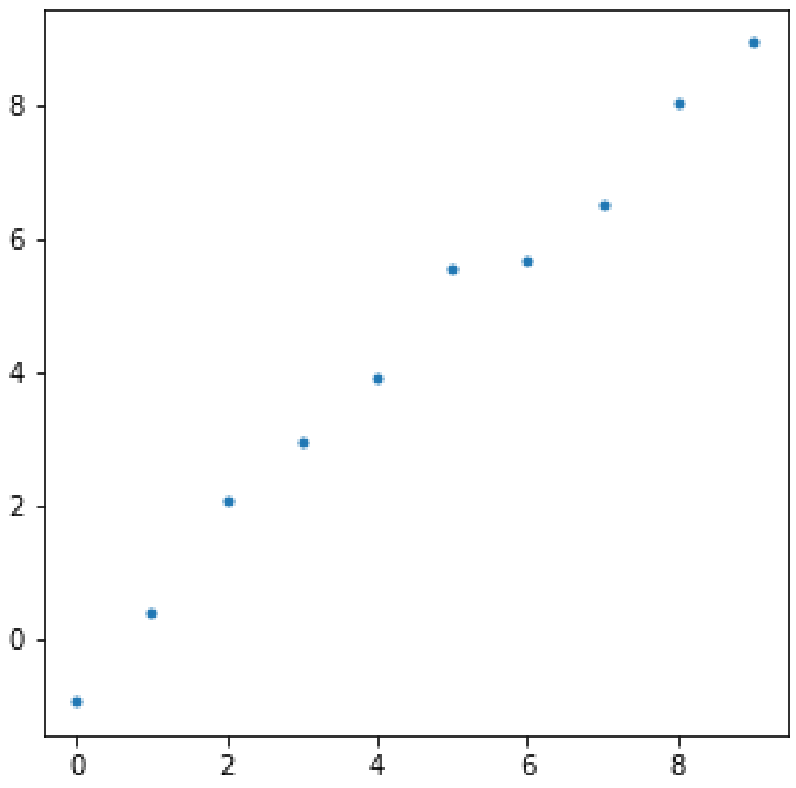
```
[147]: metrics[metrics["Data"]=="graph"]
```

```
[147]:      Data    DR  High  Low  Density  Kruskal Stress  MRRE  Coranking Quality  \
       1   graph   PCA  1158    2  0.00297        0.411585     0           0.161024
       3   graph   NMF  1158    2  0.00297        0.438609     0           0.070470
       5   graph  UMAP  1158    2  0.00297        0.519252     0           0.181188
       7   graph   PCA  1158   20  0.00297        0.210524     0           0.252444
       9   graph   NMF  1158   20  0.00297        0.393748     0           0.166651
       11  graph  UMAP  1158   20  0.00297        0.454118     0           0.191261

           Coranking Behavior
       1             0.000387
       3             0.001897
       5             0.001310
       7             0.001467
       9             0.006948
       11            0.000398
```

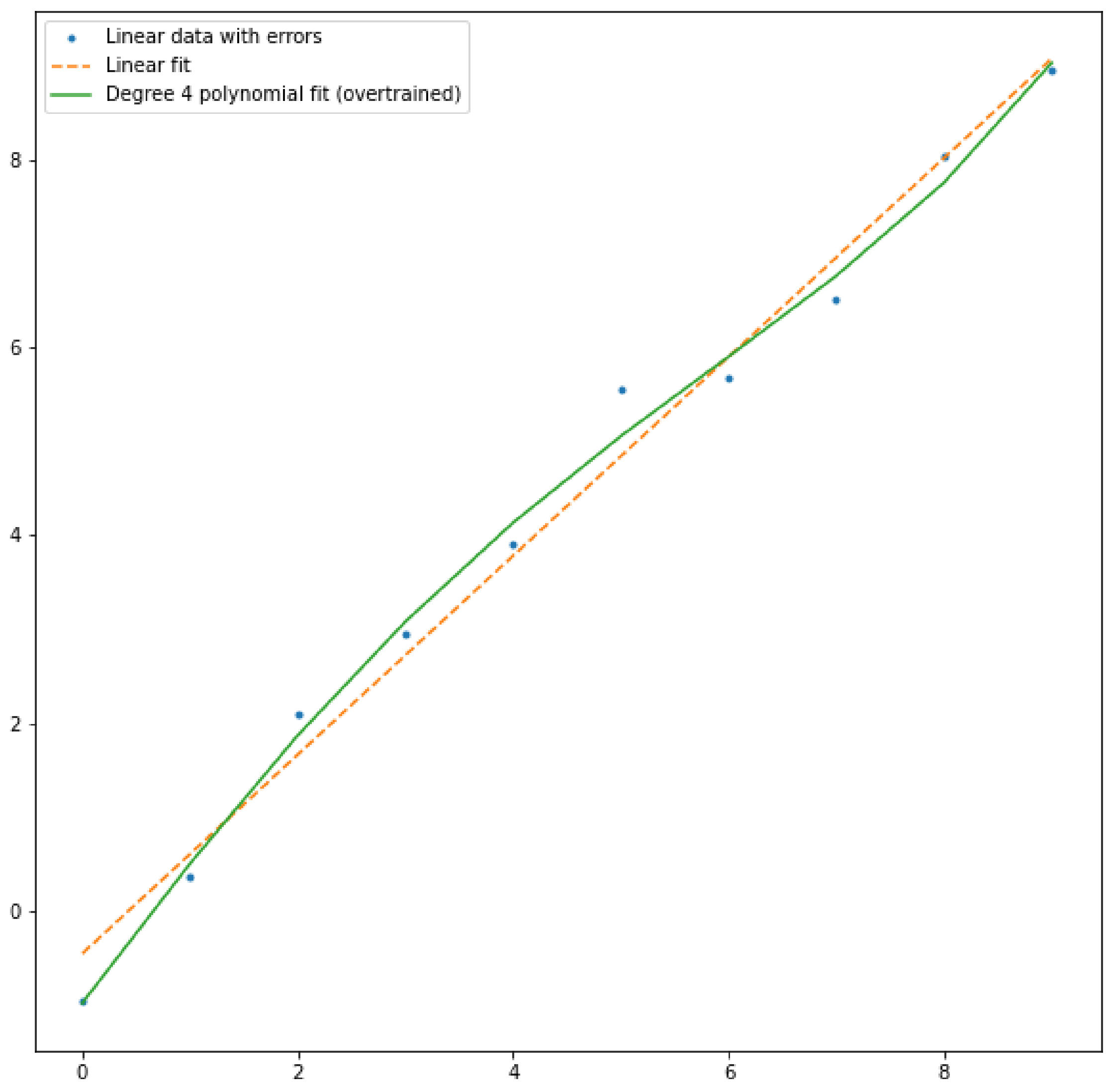
1.5 Overfitting figure

```
[47]: figure(figsize=(5,5))
      d = 10
      x = list(range(10))
      y = x
      eps = np.random.normal(0,0.5,d)
      yeps = y + eps
      plot(x,yeps,'.');
```



[51]:
```
m1 = polyfit(x, yeps, 1)
p1 = poly1d(m1)
m5 = polyfit(x, yeps, 4)
p5 = poly1d(m5)
```

[58]:
```
f = figure(figsize=(10,10))
plot(x,yeps,'.', label="Linear data with errors");
plot(x, p1(x), '--', label="Linear fit")
plot(x, p5(x), label="Degree 4 polynomial fit (overtrained)")
legend();
f.savefig("../Images/overfitting.jpg",)
```


Linear data with errors
Linear fit
Degree 4 polynomial fit (overtrained)
0
2
4
6
8
0
2
4
6
8