THESIS

EMBEDDING BASED CLUSTERING OF TIME SERIES DATA USING DYNAMIC TIME WARPING

Submitted by

R.A.C.Laksheen Mendis

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2022

Master's Committee:

Advisor: Sangmi Lee Pallickara

Shrideep Pallickara
Stephen Hayne

ABSTRACT

EMBEDDING BASED CLUSTERING OF TIME SERIES DATA USING DYNAMIC TIME WARPING

Voluminous time-series observational data impose challenges pertaining to storage and analytics. Identifying patterns in such climate time-series data is critical for many geospatial applications. Over the recent years, clustering has become a key computational technique for identifying patterns/clusters. However, data with complex structures and high dimensions could lead to uninformative clusters and hinder the quality of clustering.

In this research, we use the state-of-the-art autoencoders with LSTMs, Bidirectional LSTMs and GRUs to learn highly non-linear mapping functions by training the networks with subsequences of timeseries to perform data reconstruction. Next, we extract the trained encoders to generate embeddings which are lightweight. These embeddings are more space efficient than the original time series data and require less computational power and resources for further processing.

In the final step of clustering, instead of using common distance-based metrics like Euclidean distance, we use DTW, an algorithm for computing similarity between time series by ignoring variations in speed, to calculate similarity between the embeddings during the application of k-Means algorithm. Based on Silhouette score, this method generates clusters which are better than other reduction techniques.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my parents, Boniface Mendis and Nayana Perera, for being my pillars of support and strength throughout my life. I specially thank them for believing in me and pushing me to do better. Next, I like to thank my siblings Dr. Shehani Mendis, Dr. Hiranya Mendis and brother-in-law Dr. Gayan Wijeratne for the tremendous support and guidance they have rendered throughout this journey.

I am very grateful to my advisor Dr. Sangmi Lee Pallickara for her guidance and continued support throughout my career here at CSU. As I came from the industry after working for four years, research was not easy for me and I thank her for being patient with me and providing valuable advice always. I thank my committee members, Dr. Shrideep Pallickara and Dr. Stephen Hayne, for promptly agreeing to be on my Thesis committee. Thank you for your valuable inputs.

Finally, I would like to thank all the faculty and staff here at the Computer Science Department for being so helpful and making life at the university pleasant.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Observations that occur over time is regarded as sequence or time-series data [1]. A time series T is a sequence of d-dimensional records described using the vector T = <R_0, ..., R_{n-1}>, where R_i = (a_0, ..., a_{d-1}) is a record at time i, for $0 \leq i \leq (n-1)$ and a_{ij} is the j^{th} attribute of i^{th} record [2]. Existing data analysis approaches assume that the differences between the timestamps of any two consecutive records are nearly the same [2]. A time series can be univariate ($d = 1$) or multivariate ($d > 1$) [3]. A univariate time series has one time-dependent attribute. For example, a univariate time series can consist of daily temperatures recorded sequentially over 24-hour increments. A multivariate time series is used to simultaneously capture the dynamic nature of multiple attributes. For example, a multivariate time series for a climate dataset can consist of precipitation, humidity, wind speed, snow depth, and temperature. In this paper, we experiment with MACA historic dataset [4].

Time series data generated at high frequencies (daily, hourly, etc) comes with an obvious problem: you end up with a lot of data. This creates problems in storing, training machine learning models, computations and visualization. Therefore, it is important to find innovative ways to reduce the size of time series data, in a way which preserves the underlying features. Thus, generating embeddings in a lower dimension have been popular among the research community. However, in this research we are trying out a subsequence based approach for creating embeddings.

Sequence data such as time series has various patterns which are unique to them. Hence, we will explore the effectiveness of embeddings generated by Deep Learning approach to explore climate patterns. In this regard, a centroid based clustering method was used along with a distance metric appropriate for sequence/time series data.

1.1 Research Questions

The primary objective of this study was to facilitate clustering over compressed, voluminous time series data. The specific research questions that we explored included the following:

- **RQ1:** How can we represent geo-spatial time series data in reduced size, while preserving trends and patterns?
- **RQ2:** How can we generate a time series clustering model using embeddings?

1.2 Overview of Our Approach

In this approach, we first normalize the time series data and generate subsequences of time series. Our approach can be distinguished from other methods in literature by using multidimensional subsequences generated using a window size and a step size. Afterwards, three different autoencoders; namely LSTM, Bidirectional LSTM and GRU (variants of RNNs), are trained using multidimensional subsequences. Then we extract the encoder from the trained autoencoder and predict the embeddings/latent vectors for the test data.

These embeddings are used for clustering with k-Means. An important point to note is, instead of using Euclidean distance, we use Dynamic Time Warping (DTW) to identify similarity between sequences by ignoring speed differences. However, our goal is to see whether we can use these embeddings in place of original time series.

1.3 Paper Organization

The remainder of this paper is organized as follows. In section 2, we discuss some related work in this field along with time series, auto encoders, and embeddings. Section 3, deals with the methodology used in our time series data reduction operation, and it is followed by a discussion on clustering the reduced time series data. In section 4, we evaluate the performance of our approach. This includes few comparisons with other data reduction techniques. It is followed by a conclusion in Section 5.

Chapter 2

Background and Related Work

2.1 Related Work

2.1.1 Time-series Analysis Over Reduced Data

Enormous amounts of data generated periodically by sensors, IoT devices, etc. imposes challenges for data storage, transmission and computing. Although technology has advanced, computing on such data volumes is quite expensive and time consuming. A survey of scientific data management systems is available in [5]. Spatial time-series data storage systems include MongoDB [6], Postgress [7], and Galileo [8]. Galileo is based on distributed hashtables and includes support for queries such as ad hoc [9], analytic [10], and geometry [11], [12] constrained queries. Stream processing systems include those specifically designed for time-series data based on sketches for spatial [13] and IoT data [14], [15]. Frameworks designed to visualize time-series data include systems such as Glance [16], [17] that leverage deep networks to render visualizations and Stash [18] that leverages distributed caching to alleviate disk accesses; these systems may leverage imputations [19], [20] that are performed using deep networks.

Thus, if we represent the data more efficiently, we can save more on storage, bandwidth and computing. In recent times, people have used different mechanisms to reduce voluminous time series data.

Fourier Transformation is one of the most prominent mechanisms used in many different domains to represent periodic time series/sequence data in the frequency domain [21]. Discrete Fourier Transform (DFT) of a sequence is obtained by decomposing the sequence of values into components of different frequencies [22]. This type of compressed data allows to perform fault analysis, remove noise, and condition monitoring of machines or systems.

Principal Component Analysis, widely known as PCA [23] is a very popular statistical technique used for data transformation, dimensionality reduction, exploration, and visualization. Ac-

cording to [24], a variation of PCA which is particularly designed for time series is known as Functional PCA (FPCA). However, PCA assumes that underlying subspace is linear. Therefore, it is not suitable for this scenario.

In recent years, time series databases have remained the fastest growing category of databases due to large volumes of time series data collected and the power it gives to entities to analyze and derive meaningful insights. TimescaleDB is an open source database for time series data and it is an extension of PostgreSQL. It combines the best of both relational databases (RDBMS) and NoSQL databases, while employing some of the best compression algorithms to enable greater storage efficiency [25].

2.1.2 Autoencoders (AE)

This is a type of artificial neural network considered as a generative model. An autoencoder comprises of two components known as an encoder and a decoder. It is heavily used to learn efficient encodings also known as latent vectors of unlabeled data. First, the encoder takes the input and generates an encoding/latent vector. Next, the decoder tries to reconstruct the original input (given to encoder) using the generated encoding. In this procedure, autoencoders are forced to reconstruct the original input approximately by preserving only the most important features, also known as latent features of the data [26]. Most basic autoencoder is known as undercomplete autoendcoder [27] (Refer Figure 2.1).

In recent years, research in this area has heavily contributed to innovative variations of autoencoders; Denoising autoencoder [28], Sparse autoencoder [29], Contractive autoencoder [30], Convolutional autoencoder [31], Adversarial autoencoder [32] and Variational autoencoder (VAE) [33] are some of those variants. According to [34], they can be roughly divided into two groups which are (1) undercomplete autoencoders and (2) overcomplete autoencoders. In general, undercomplete autoencoders are used to learn the underlying structure of data and used for visualisation/clustering [35] like PCA. In contrast, overcomplete autoencoders are used for classification

based on the assumption that higher dimensional features are better for classification [36]. All the autoencoders focus on learning features in a unsupervised manner.

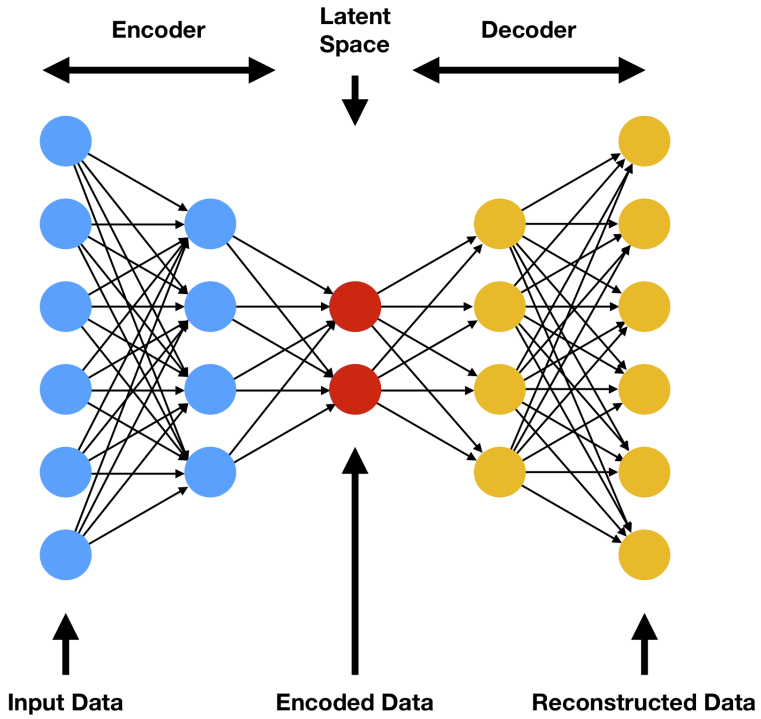

Figure 2.1: Architecture of a basic autoencoder

In undercomplete autoencoders, the latent space representation can be considered as a compressed representation of the input, when the feature space of the embedding has a lower dimensionality than the input space. This concept is heavily used in various domains including video streaming services [37] [38].

2.1.3 Time Series Clustering

In [39], Ali et al. have conducted an extensive benchmark for time series clustering using eight clustering algorithms representing three categories of clustering (Hierarchical, Partitional, and Density-based) and three types of distance measures (Euclidean, Dynamic Time Warping (DTW), and Shape-based) on 112 time series datasets. A phased evaluation framework has been designed such that in each phase only one of the two building blocks of a clustering method – algo-

rithm and distance measure – is varied at a time. Benchmark results show the overall performance of the eight algorithms to be similar with high sensitivity to the datasets, indicating that no method is superior to the others for all datasets.

A fast clustering method for large-scale time series data named YADING has been developed by Ding et al. [40]. In this, time series objects were allocated to clusters that were initially induced based on sampled subsets of the input data. However, this does not take into account the possible subspaces of a time series data set. Further, it is not applicable for time series with variable lengths.

[41] propose a two-stage approach for clustering time series data. In the first stage, they introduce a methodology to create cluster labels and thus enable transforming unsupervised learning to supervised learning for time series data. In the second stage, an autoencoder-based deep learning algorithm is built to model clustering time series data is presented.

2.1.4 Clustering Embeddings

Deep Embedded Clustering (DEC) [35] is a methodology which simultaneously learns feature representations and clustering assignments using deep neural networks. Unlike the traditional clustering algorithms (which focus on distance functions and grouping algorithms), DEC learns a mapping from the data space to a lower-dimensional feature space. It then iteratively optimizes a clustering objective in this lower dimensional space.

Ienco et al. proposed Deep Time Series Embedding Clustering (DeTSEC) [42] which includes two stages. At first, a recurrent autoencoder exploits attention and gating mechanisms to produce a preliminary embedding representation. Next, a clustering refinement stage is introduced to stretch the embedding manifold towards the corresponding clusters.

An autoencoder based data clustering method is proposed in [43]. They use a new objective function embedded into the autoencoder model. It contains two parts: the reconstruction error and the distance between data and their corresponding cluster centers in the new space. During optimization, data representation and cluster centers are updated iteratively.

2.1.5 Distributed Learning

Deriving valuable insights from big data has become an important, yet common practice in many sectors. Often these insights are based on leveraging algorithms that fit models to the data. However, processing big data is always associated with numerous challenges. Including but not limited to scalability and storage [44]. To mitigate such challenges, it requires an increased amount of storage and computational resources that are beyond the capacity of a single machine. Thus, a solution for this problem involves, using a distributed system which comprises of a large number of commodity machines.

However, processing big data on a distributed system requires a lot of effort toward providing efficient computations. While addition of more machines increases the capability of the system, it also increases the probability of failure. Further, it should also be scalable.

To fit models on voluminous datasets, several software libraries for deep learning have been suggested. They hide most of the complexity that is associated with the distributed environment. Abadi et al. [45] proposed the open-source software library for deep learning known as Tensorflow. It relies on a computational graph to enable expressing computations and executing them on different computational devices (CPUs, GPUs, TPUs). In this computational graph, the nodes represent mathematical operations and the edges represent the data flow between the graph nodes. Multidimensional data arrays known as tensors, transmit the data between graph nodes. The TensorFlow engine uses the available computational devices to execute the graph nodes. On the other hand, there is PyTorch [46]. It also uses a computational graph. However, in Tensorflow, the computational graph is generated in a static way before the code is run. But in PyTorch computational graph is generated dynamically.

Chapter 3

Methodology

3.1 Dataset

3.1.1 MACAv2 Historical Dataset

This archive [47] contains two different downscaled products/datasets covering the Continental US plus Alaska and Hawaii (CONUS-plus). Both use a common set of 20 Global Climate Models (GCMs) of Coupled Model Inter-Comparison Project 5 (CMIP5) that provides daily output of requisite variables for historical (1950-2005) and future experiments under RCP4.5 and RCP8.5 (RCP-Representative Concentration Pathways).

A requirement for any statistically downscaled product is historical data, or training data, that GCM output is bias corrected to. The two datasets/products use different training data. Briefly, one dataset uses the 6 km (1/16th degree) daily product of Livneh et al. [48] from 1950-2011 that also incorporates the Canadian portion of the Columbia River Basin. The other uses the gridMet daily dataset at a 4 km grid (1/24th degree) from 1979-2012 [47].

The experiments described in this research uses only the historical GCM forcings of three years (1995-1997). It contains daily climate variables for each county (identified by a Geography Identifier known as GISJOIN).

The MACAv2 dataset currently has data for the following variables;

- min specific humidity
- max specific humidity
- min precipitation
- max precipitation
- min surface downwelling shortwave flux
- max surface downwelling shortwave flux
- min max air temperature
- max max air temperature
- min min air temperature
- max min air temperature

- min eastward wind
- max eastward wind
- min northward wind
- max northward wind
- min vpd
- max vpd

3.1.2 Köppen-Geiger climate classification

The Köppen climate classification is one of the most widely used climate classification systems. It was first published by German-Russian climatologist Wladimir Köppen in 1884 [49], with several modifications by himself, notably in 1918 and 1936. Later, the climatologist Rudolf Geiger introduced some changes to the classification system, and it is sometimes called the Köppen–Geiger climate classification system [50].

This classification divides climates into five main climate groups based on seasonal precipitation and temperature patterns. The five main groups are A (tropical), B (dry), C (temperate), D (continental), and E (polar). Each group and subgroup is represented by a letter. All climates are assigned a main group (the first letter) and all climates except for those in the E group are assigned a seasonal precipitation subgroup (the second letter) [50]. The system assigns a temperature subgroup for all groups other than those in the A group, indicated by the third letter for climates in B, C, and D, and the second letter for climates in E. Figure 3.1 shows the climate classification for North America.

According to Figure 3.1, it is clear that most of the continental US belong to three main climate classes; Cfa (lime green), Dfa (cyan) and BSk (orange). Further, 1694, 597 and 413 counties are covered by Cfa, Dfa and BSk respectively. Therefore, these climate classes are used for the experiments in this research. [51] provides a text file which includes all the counties and their respective climate classes. Therefore, the text file was processed to extract relevant GISJOINs for required counties. Next, these GISJOINS were used to query MACAv2 historical dataset hosted in MongoDB by Sustain project [52].

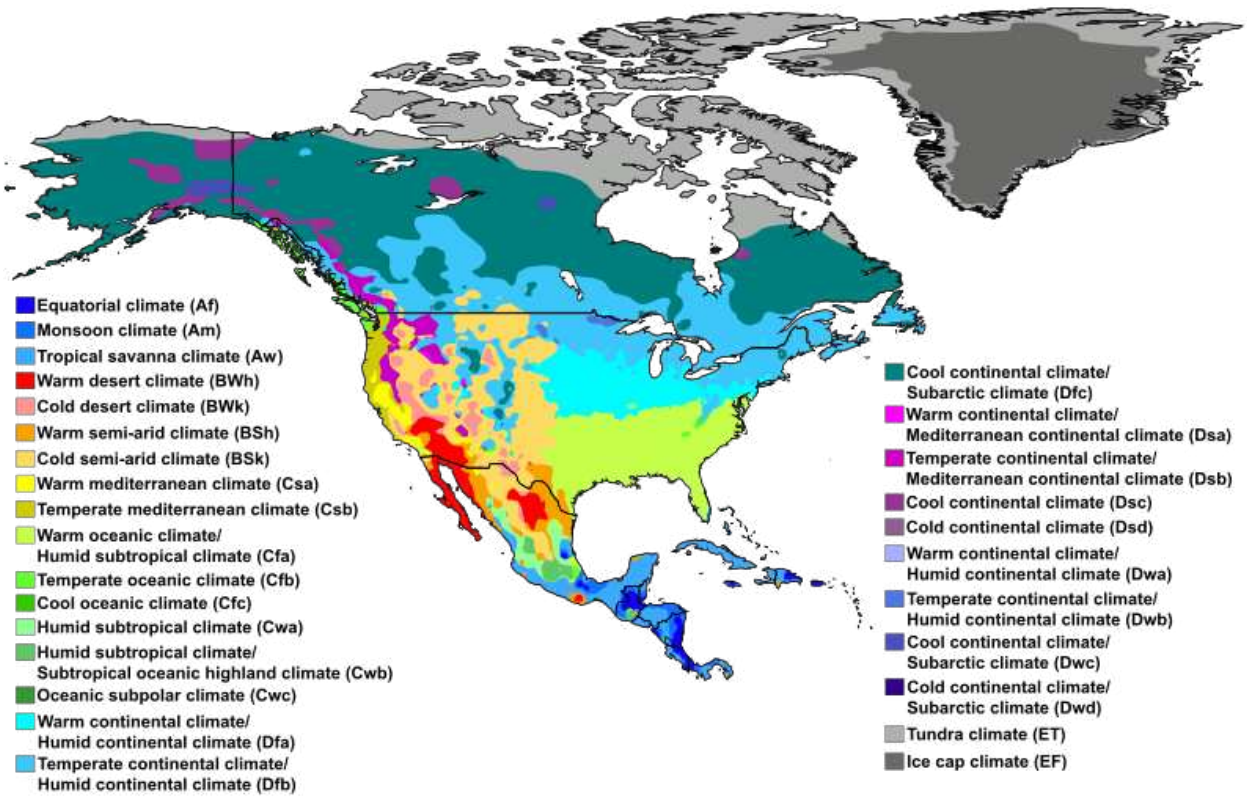

Figure 3.1: North America map of Köppen–Geiger climate classification (from Wikimedia Commons)

3.2 Data Selection

3.2.1 Data Pre-Processing and Feature Selection

In Machine Learning, a preliminary step is to explore available datasets and prepare data in a suitable manner. As shown in Table 3.1, values in each numerical feature lies between different ranges. If there is a vast difference in the range, ranging from thousands to tens, models make underlying assumptions that higher ranging numbers are more important than others. Therefore, such significant numbers start playing a more decisive role during the training process. Thus, feature scaling is considered essential for machine learning algorithms which calculate distances between data points.

Normalization is a technique for scaling features, and the data is scaled between 0 and 1. This avoids the problems by creating new values that maintain the general distribution and ratios in the

source data, while keeping values within a scale applied across all numeric columns used in the model.

Table 3.1: Ranges of available features

	Minimum	Maximum
min specific humidity	0	2.600000e-02
max specific humidity	0	2.900000e-02
min precipitation	0	2.138140e+02
max precipitation	0	2.848650e+02
min surface downwelling shortwave flux in air	1.235700e+01	4.080010e+02
max surface downwelling shortwave flux in air	1.620500e+01	4.537880e+02
min max air temperature	2.409670e+02	3.186830e+02
max max air temperature	2.454170e+02	3.271460e+02
min min air temperature	2.270000e+02	3.022600e+02
max min air temperature	2.355210e+02	3.111960e+02
min eastward wind	-2.126100e+01	1.575500e+01
max eastward wind	-1.362500e+01	2.027800e+01
min northward wind	-1.961500e+01	1.278300e+01
max northward wind	-1.332100e+01	1.934900e+01
min vpd	0	5.260000e+02
max vpd	0	9.960000e+02

3.3 Generating Embeddings

One of the main goals of this research is to explore ways in which we can reduce the size of lengthy, high dimensional geo-spatial time-series data. With that in mind, we used an approach which generates embeddings, using deep learning techniques (i.e. non-linear mappings).

The process of generating embeddings consists of two important steps. First, we generate subsequences and more details can be found in Section 3.3.1. Next, we train autoencoders with generated subsequences (Section 3.3.2). Once training is completed the encoder is separated from the autoencoder and it is capable of generating embeddings (Section 3.3.3).

A summary of this process for data of one county is depicted in Figure 3.2.

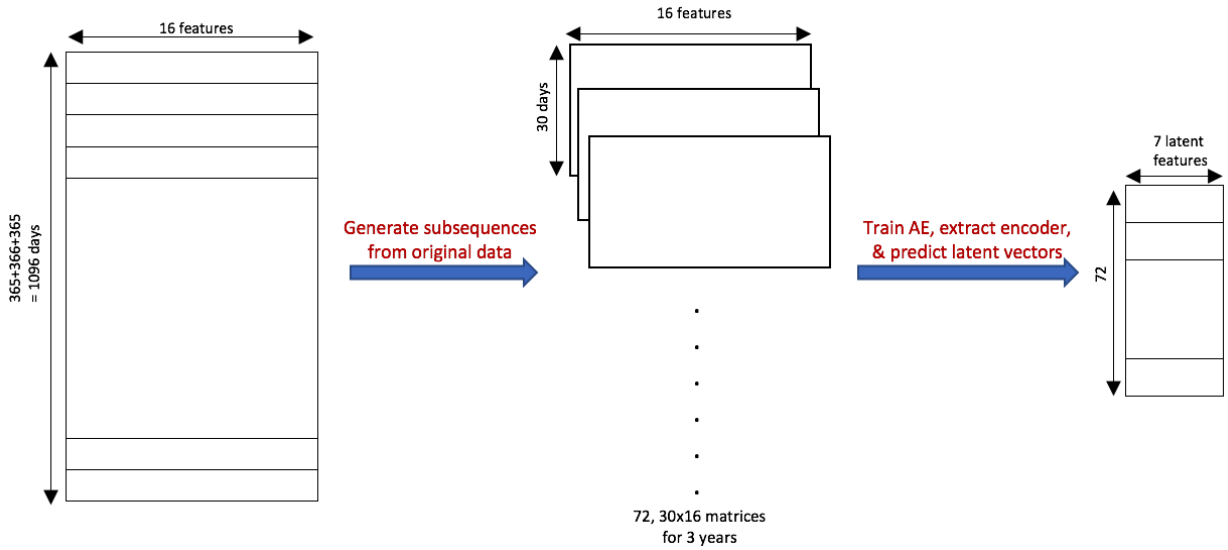

Figure 3.2: Summary of the process for data of one county

3.3.1 Generating Training Dataset

Once the preprocessing is done, we perform an important step. This is known as generating subsequences of time series. In order to do that we need two parameters; window size and step size.

- Window size - length of the subsequence (it determines how many adjacent observations from the time series should be in one subsequence)
- Step size - the number of observation shifts over the original time series

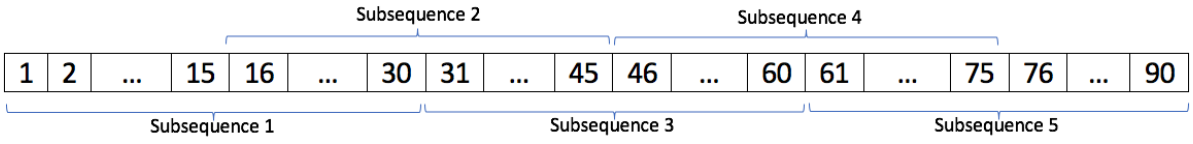

Figure 3.3: Example of generating subsequences

For instance, assume Figure 3.3 represents a time series for 90 days and each cell represents a day. Further, 16 attributes are reported each day (this is not explicitly shown). When the window

size is 30 and step size is 15, the first subsequence starts at day 1 and ends on day 30. Second subsequence starts at day 16 because the step size/stride is 15. It ends on day 45. Similarly, with the whole 90 day time series, we can generate 5 subsequences each having a shape of 30x16.

As mentioned in Section 3.1.1, our experiments use a range of 3 years, therefore each county has 1096 days (i.e. 365 x 3 = 1095 + 1 (1996 is a leap year)). Therefore, we are able to generate 72 subsequences each having a shape of 30x16. This is shown in the middle step of Figure 3.2.

3.3.2 Training Models

In this section we will be discussing about the neural networks used for this research. First, it is important to mention that we selected *Recurrent Neural Networks* (RNNs) over traditional neural networks [53]. RNNs are widely accepted neural network architecture for sequential or time series data. They differentiate from other neural networks due to their ability to memorize information from prior inputs to influence the current input and output.

Figure 3.4 shows a basic RNN. The *Rolled RNN* represents the entire neural network and the *Unrolled RNN* represents the individual layers or time steps. In this figure it is very clear that the output of RNNs depend on the prior elements within the sequence. Another distinguishing characteristic of recurrent networks is that they share the same weight parameter across each layer of the network. The weights are adjusted through the process of back propagation and gradient descent. RNNs leverage back propagation through time (BPTT) algorithm to determine the gradients, which is slightly different from traditional backpropagation as it is specific to sequence data [54].

In this research, three variants of Recurrent Neural Network (RNN) are used to compose the network of the autoencoder. They are; *Long Short Term Memory* (LSTM) [55], *Bidirectional LSTMs* [56] and *Gated Recurrent Units* (GRU) [57]. In order to compare these models, all three have a similar architecture (refer Figure 3.5, Figure 3.6 and Figure 3.7). All three models take subsequences as inputs (each with size 30x16) and tries to predict the same, by generating an intermediate embeddding of dimensionality 7. Therefore, our model is an undercomplete autoencoder

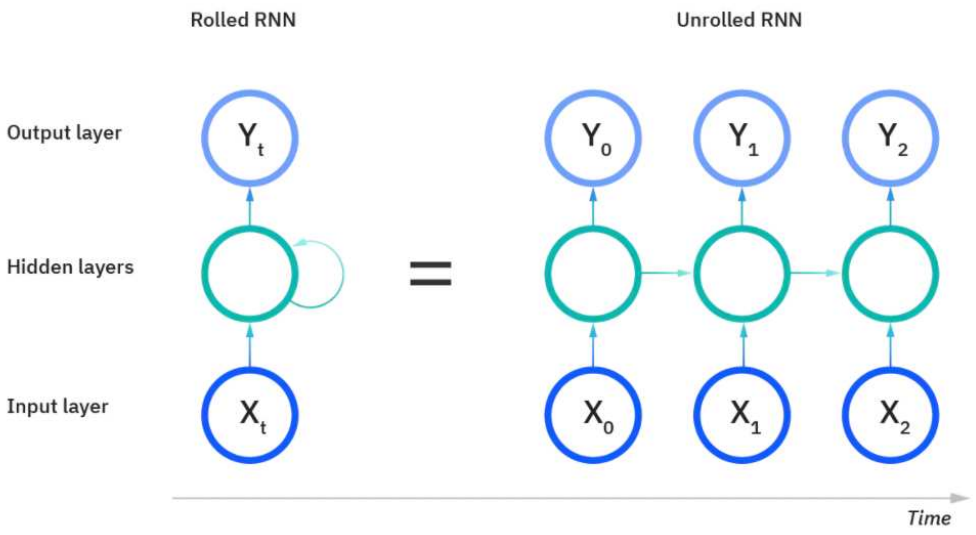

Figure 3.4: Rolled versus unrolled RNN (from IBM Cloud Learn Hub)

(latent space has lower dimensionality than the input space). First layer in the encoder section and first layer in the decoder section has an output shape of 30x32.

LSTM Auto-Encoder

Long Short Term Memory (LSTM) are a type of RNNs that is useful in learning order dependence in sequence prediction problems. This was introduced in [55] as a solution to the vanishing gradient problem which was present in basic RNNs. Also, it addresses the problem of long-term dependencies. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. These gates control the flow of information which is needed to predict the output in the network [54].

Bidirectional LSTM Auto-Encoder

This is an extension of the regular LSTM and consists of two LSTMs; one taking the input in a forward direction, and the other in a backwards direction. This operation makes it different from the regular LSTM. Therefore, this allows to preserve past and future information [56].

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 32)            6272

lstm_1 (LSTM)                (None, 7)                 1120

repeat_vector (RepeatVector  (None, 30, 7)             0
)

lstm_2 (LSTM)                (None, 30, 32)            5120

lstm_3 (LSTM)                (None, 30, 16)            3136

=================================================================
Total params: 15,648
Trainable params: 15,648
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.5: Summary of LSTM model generated by tf.keras.Model.summary()

```
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectiona  (None, 30, 32)            12544
l)

bidirectional_1 (Bidirectio  (None, 7)                 2240
nal)

repeat_vector_1 (RepeatVect  (None, 30, 7)             0
or)

bidirectional_2 (Bidirectio  (None, 30, 32)            10240
nal)

bidirectional_3 (Bidirectio  (None, 30, 16)            6272
nal)

=================================================================
Total params: 31,296
Trainable params: 31,296
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.6: Summary of Bidirectional LSTM model generated by tf.keras.Model.summary()

GRU Auto-Encoder

Gated Recurrent Unit [57] is similar to LSTM and it also addresses the short-term memory issue of vanilla RNNs. However, instead of three gates, it has two; a reset gate and an update gate. These gates control how much and which information to retain.

3.3.3 Generating Embeddings using an Encoder

Final step in this process is to separate the encoder from the trained autoencoder. For example, if we consider the Bidirectional model shown in Figure 3.6, first two layers; bidirectional and bidirectional_1 belongs to the encoder. Once we separate it, we are capable of generating embeddings

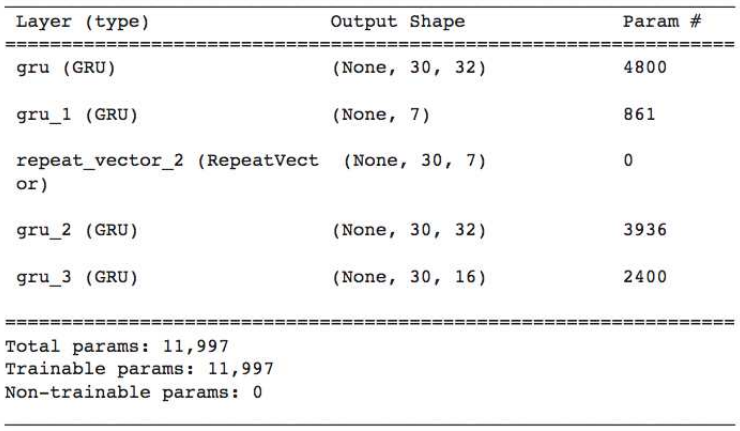

Figure 3.7: Summary of GRU model generated by tf.keras.Model.summary()

with a much lower dimension (7 using above mentioned models) for subsequences generated with test data (more details in Section 4.1).

3.4 Clustering Embeddings

Second goal of this research is to generate a time series clustering model, which resembles the climate classes (i.e. ground truth) using the embeddings generated in the previous step.

Clustering is the task of grouping objects into clusters, so that objects in the same cluster are more similar to each other than to objects in a different cluster. During our research we have used a well known clustering algorithm, *k-Means* [58]. More details on k-Means algorithm is available in Section 3.4.1. Calculating distances among data points is an important step in k-Means algorithm and Euclidean distance is frequently used for this calculation. However, this distance metric does not ignore variations in speed when calculating distances between sequences/time series. Therefore, it is not quite suitable for distance calculations between sequences. Hence, we used a different distance metric, know as Dynamic Time Warping (DTW), and details are included in 3.4.2.

3.4.1 k-Means algorithm

k-Means is a centroid based clustering technique which was introduced by Lloyd in 1982 [58]. This is an iterative algorithm that tries to partition n observations into k distinct non-overlapping subgroups, known as clusters. According to the pseudo-code in Figure 3.8, first it randomly selects k objects in D. These initially represents a cluster center/centroid. Next, each of the remaining objects are assigned to the cluster to which it is the most similar, based on a distance metric calculated between the object and the cluster center/centroid. k-Means algorithm iteratively improves the within-cluster variation, also known as sum of squared error. Then for each cluster, it computes the new mean (i.e. new centroid) using the objects assigned to the cluster in the previous iteration. Next, all the objects are reassigned using the updated means as the new cluster centers. The iterations continue until the clusters formed in the current round are similar to those formed in the previous round [59].

The k-Means algorithm does not guarantee to converge to the global optimum and often terminates at a local optimum. Further, the results may depend on the initial random selection of cluster

```
Input:
    D= {t1, t2, …. Tn }   // Set of elements
    K                     // Number of desired clusters
Output:
    K                     // Set of clusters
K-Means algorithm:
   Assign initial values for m1, m2,…. mk
   repeat
      assign each item ti    to the clusters which has the closest mean;
      calculate new mean for each cluster;
   until convergence criteria is met;
```

Figure 3.8: Pseudocode for k-Means algorithm (from [58])

centers. To obtain good results in practice, it is common to run the k-Means algorithm multiple times with different initial cluster centers [59].

3.4.2 Dynamic Time Warping (DTW)

Dynamic Time Warping (DTW) [60] is a mapping of points between a pair of unequal length time series/sequences, T_1 and T_2, by ignoring variations in speed, known as warping (refer Figure 3.9). It is designed to minimize the pairwise Euclidean distance between points and is considered the state-of-the-art, most accurate similarity measures for time series data [61]. The optimal mapping should adhere to three rules.

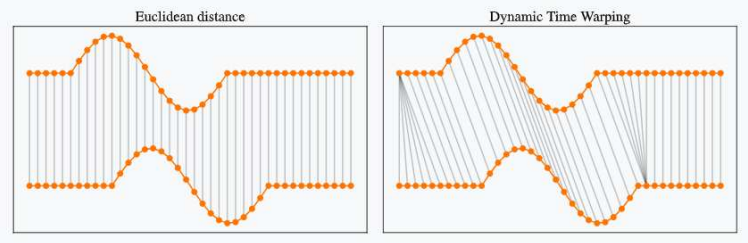

Figure 3.9: Comparison of Euclidean and DTW (from [62])

- Every point from T_1 must be aligned with one or more points from T_2, and vice versa.
- The first and last points of T_1 and T_2 must align.

- No cross-alignment is allowed, that is, the warping path must increase monotonically.

Figure 3.10 shows the pseudo-code for DTW distance matrix calculation for two time series [63]. S[n,m] contains the DTW distance between two time series. For the experiments, we have used a package named *tslearn* [64], and it is capable of measuring DTW similarity between multidimensional time series as well.

```
DTW(v1, v2)  {
//where the vectors v1=(a1,...,an), v2=(b1,...,bm) are the time series with n and m
time points
    Let a two dimensional data matrix S be the store of similarity measures
such that S[0,...,n, 0,...,m], and i, j, are loop index, cost is an integer.
    // initialize the data matrix
    S[0, 0]:= 0
    FOR i := 1 to m DO LOOP
        S[0, i]:= ∞
    END
    FOR i := 1 to n DO LOOP
            S[i, 0]:= ∞
    END
    // Using pairwise method, incrementally fill in the similarity matrix
    with the differences of the two time series
    FOR i := 1 to n DO LOOP
        FOR j := 1 to m DO LOOP
        // function to measure the distance between the two points
            cost := d(v1[i], v2[j])
            S[i, j] := cost + MIN(S[i-1, j],           // increment
                                  S[i, j-1],     // decrement
                                  S[i-1, j-1])  // match
        END
    END
    Return S[n, m]
}
```

Figure 3.10: Pseudocode for DTW algorithm (from [63])

Chapter 4

Evaluation

In this chapter we contrast different methods which were used to evaluate and compare our methodology. Section 4.1 includes details on how data was utilized for experiments. Section 4.2 includes experiments on training models, while section 4.3, includes details and experiments on clustering original time series data versus embeddings. Distributed training experiments are presented in section 4.4.

Unless noted explicitly, hyper parameters in table 4.1 are used for model training during the experiments described in following sections.

Table 4.1: Default hyper parameters for model training

Hyper parameter	Default value
Learning Rate	0.001
Epochs	200
Batch Size	40

In addition, parameters in table 4.2 are used when generating time series sub-sequences (as a preprocessing step) and embeddings (using autoencoders).

Table 4.2: Default parameters for sub-sequences and embeddings

Parameter	Default value
Window Size	30
Step Size	15
Embedding Dimension	7

4.1 Data for training and testing

As mentioned in Section 3.1, MACA dataset is used for all the experiments and we have specifically focused on 3 different climate classes covering CONUS; BSk, Cfa and Dfa. According to [51]

Cfa, Dfa and BSk cover 1694, 597 and 413 counties respectively. However, some counties belong to multiple climate classes. Therefore, to make the experiments more clear, we have removed such overlapping counties. After the removal we are left with 1542, 476 and 369 counties for Cfa, Dfa and BSk climate classes respectively.

If we use varying number of counties for the 3 climate classes, the trained models will be more biased towards the climate classes which have more data. Therefore, it was necessary to reduce the data size to the limiting climate type. In this scenario, it was BSk. Thus, 369 counties per each climate class was used for training and testing.

Out of each 369 counties per climate class, 80% (i.e. 295) was used as training data and 20% (i.e. 74) as testing data. Hence, a total of 885 counties were used to train models, while 222 counties were used for testing.

4.2 Embeddings

In this section, we discuss the experiments performed during training of models. Mainly we have used 3 types of autoencoders; LSTM, Bidirectional LSTM and GRU, each having a similar architecture except for the units used within.

4.2.1 Model training and testing

Figure 4.1 corresponds to the training error of 3 models, when training with default hyperparameters mentioned in Table 4.1. *Mean Squared Error* (MSE) was used as the cost function for models. MSE measures the average squared difference between an observation's actual and predicted values. From an autoencoder's point of view, this is the reconstruction loss. The goal was to minimize MSE to improve the accuracy of the models.

Additionally it is important to keep in mind that the models were trained with subsequences of time series generated at an intermediate step. Each county has 72 matrices and each matrix has a size of 30x16.

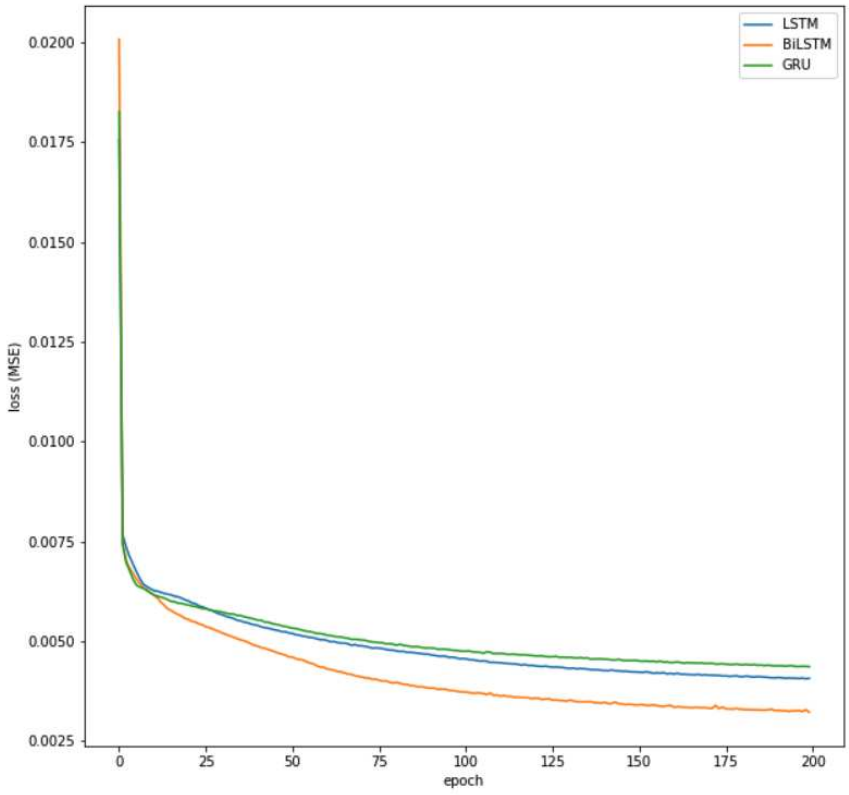

Figure 4.1: Training error for 3 models

Over 200 epochs, all three models improved significantly by reducing the MSE. Both LSTM and GRU models have reduced the MSE from 0.018 to o.0.004. However, Bidirectional LSTM outperforms LSTM and GRU models by reaching a MSE of 0.003. In addition, we evaluated the trained models with test data (using tf.keras.Model.evaluate()). This function calculates the average of MSE values for each sample in the test dataset. Table 4.3 includes the average MSE values achieved by each model. Therefore, we selected Bidirectional LSTM based autoencoder as the best model and used it for all other experiments.

Table 4.3: Average MSE values for testing data

Model	Loss
LSTM	0.0047
BiLSTM	0.0040
GRU	0.0051

4.2.2 Hyperparameters

Batch Size

As shown in Figure 4.2, batch size of 40 resulted in the lowest MSEs at the end of training for all 3 models. Further, the same batch size achieves lower MSEs for almost all the epochs, compared to other two batch sizes 80 and 120. In addition, although it is not reported, the training times decreased when the batch size increased. This is due to less number of weight updates taking place within an epoch.

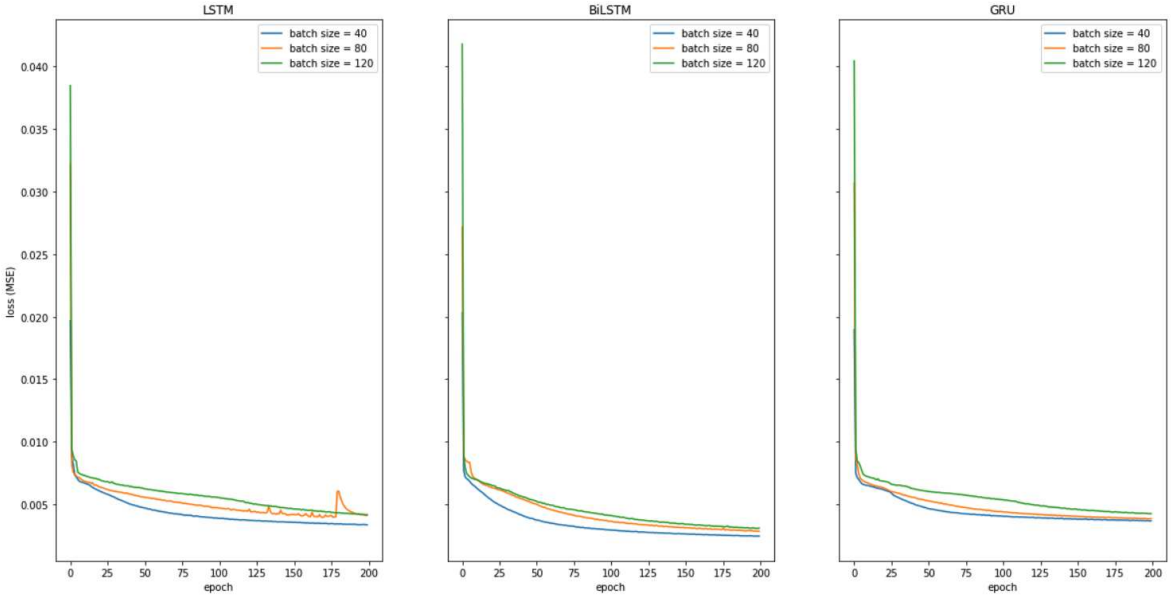

Figure 4.2: Training error for 3 models with different batch sizes

Learning Rate

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Therefore selecting a suitable learning rate is challenging. In Figure 4.3 we have experimented with three different learning rates and plotted the MSE/loss. For LSTM and BiLSTM models, a learning rate of 0.1 resulted in Nan values. That is the reason for not seeing a green line in first two graphs. The reason for this is, when learning rate is too large, Stochastic Gradient Descent can diverge into infinity. Even for GRU, loss begins to increase when learning rate is 0.1.

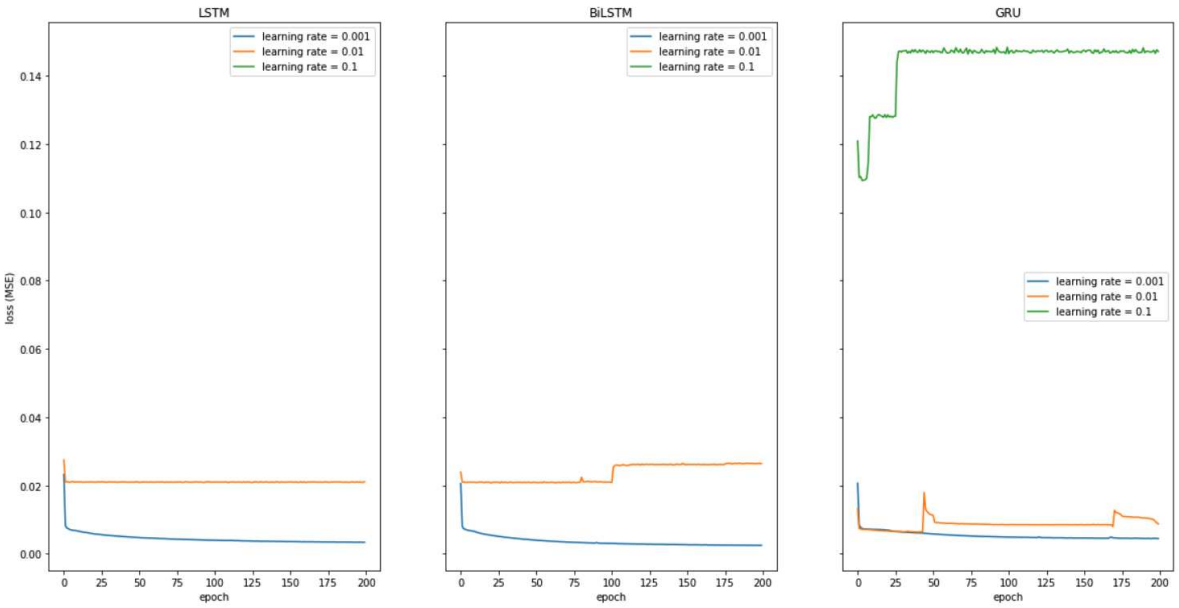

Figure 4.3: Training error for 3 models with different learning rates

Even for a learning rate of 0.01 the loss function does not seem to be consistent for all three models, as it is for 0.001. Therefore, we have selected 0.001 as the learning rate for all our experiments.

4.3 Clustering

This section contains details about clustering experiments performed using embeddings versus other alternative methods. Section 4.3.2 presents the metrics used to measure the quality of generated clusters. Remaining sections include details about clustering experiments.

4.3.1 t distributed Stochastic Neighbor Embedding

It is important to note that *t distributed Stochastic Neighbor Embedding* [65] widely known as t-SNE, introduced by Van der Maaten et al. has been used in next few sections to visualize high dimensional data. It is an extension of *Stochastic Neighbor Embedding* (SNE) [66], created and published in 2003 by Hinton et al..

t-SNE is a non-linear dimensionality reduction technique used to visualize high-dimensional data, by giving each data point a location in a two or three-dimensional map. Further, it produces significantly better visualizations by reducing the tendency to crowd points together in the center of the map [65].

First, it starts by converting high-dimensional Euclidean distances between datapoints into conditional probabilities that represent similarities. The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|i}$, that x_i would pick x_j as its neighbor, if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i. For nearby datapoints, $p_{j|i}$ is relatively high, whereas for widely separated datapoints, $p_{j|i}$ will be extremely small (refer Figure 4.4) [65].

Mathematically, conditional probability $p_(j|i)$ is given by first equation in Figure 4.5. σ_i is the variance of the Gaussian that is centered on datapoint x_i.

Moreover, [65] states that SNE can also be applied to datasets that consist of pairwise similarities between objects rather than high-dimensional vector representations of each object, provided these similarities can be interpreted as conditional probabilities. The same concept was used when utilizing the t-SNE implementation available in *sklearn.manifold.TSNE* [67]. We provided a dis-

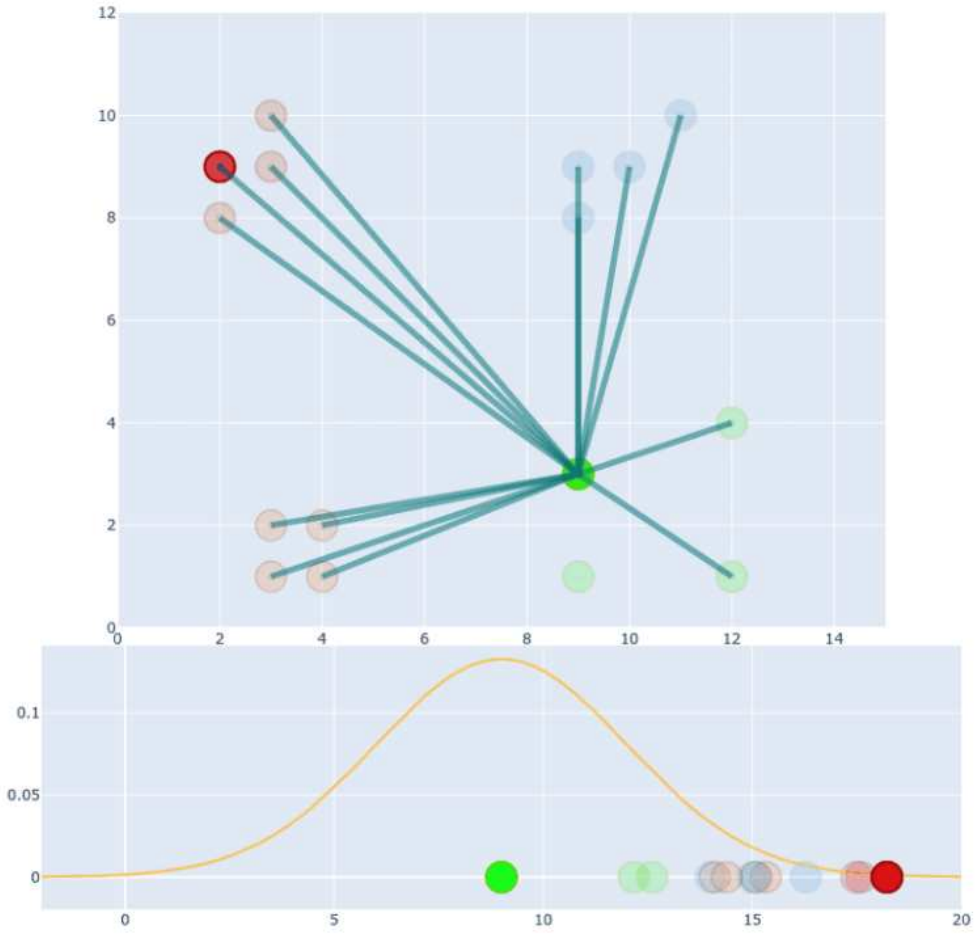

Figure 4.4: Probability distribution of original high dimensional data

$$p_{ij} = \frac{\exp(-\left\|x_i - x_j\right\|^2 / 2\sigma_i^2)}{\sum_{k \neq l} \exp(-\left\|x_k - x_l\right\|^2 / 2\sigma_i^2)}$$

$$q_{ij} = \frac{(1 + \left\|y_i - y_j\right\|^2)^{-1}}{\sum_{k \neq l} (1 + \left\|y_k - y_l\right\|^2)^{-1}}$$

Figure 4.5: Conditional probabilities for high dimensional and low dimensional data

tance matrix consisting of DTW pairwise similarities between time-series (original/embeddings). This was required because scikit-learn implementation does not support 3-dimensional datasets.

Dimensions of our datasets;

- original time series –> 222, 1096, 16
- embeddings –> 222, 72, 7

In the high-dimensional space, [65] convert distances into probabilities using a Gaussian distribution. However, in the low-dimensional map, they use a probability distribution that has much heavier tails than a Gaussian to convert distances into probabilities, known as *Student t-distribution* (refer Figure 4.6).

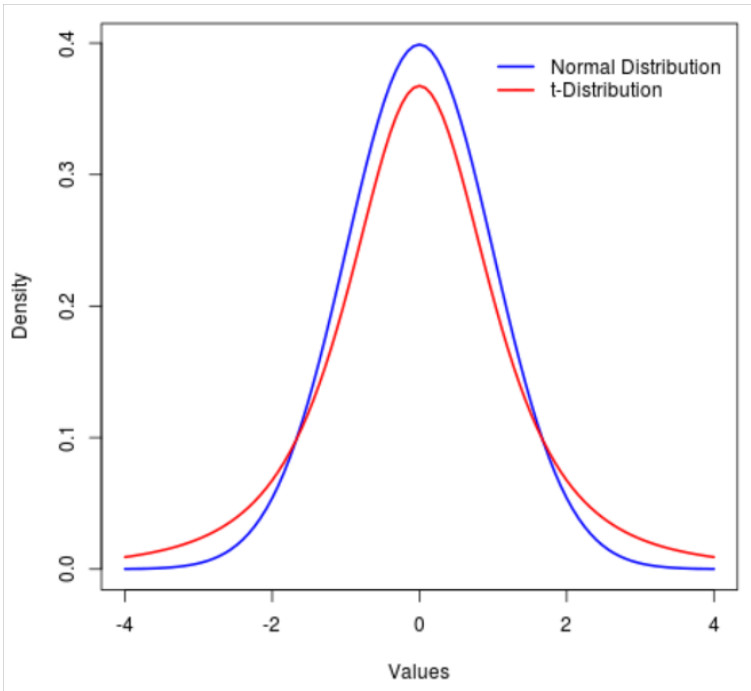

Figure 4.6: Comparison of Gaussian versus Student-t distribution

Thus, for the low-dimensional counterparts y_i and y_j, of the high-dimensional datapoints x_i and x_j, it is possible to compute a similar conditional probability, which is denoted by $q_(j|i)$. If the map points y_i and y_j correctly model the similarity between the high-dimensional datapoints x_i and x_j, the conditional probabilities $p_(j|i)$ and $q_(j|i)$ will be equal. Motivated by this observation, SNE aims to find a low-dimensional data representation that minimizes the mismatch between $p_(j|i)$ and $q_(j|i)$ [65].

A natural measure of the faithfulness with regards to $p_(j|i)$ and $q_(j|i)$ is the Kullback-Leibler divergence [65]. t-SNE minimizes the sum of *Kullback-Leibler divergences* [68] over all datapoints using a gradient descent method. Then gradient is treated as repulsion and attraction between points. Thus, the low-dimensional space is adjusted in small steps over the iterations, based on the calculated gradients.

Unless noted explicitly, Dynamic Time Warping (DTW) is used to calculate distances between geo-spatial time series (both original and embeddings) for all visualization purposes.

4.3.2 Metrics

The silhouette coefficient [69] measures how similar an object is to its own cluster (tightness), compared to other clusters (separation). This is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is $(b - a)/max(a, b)$. The silhouette coefficient is only defined if; 2 <= number of clusters <= (number of samples - 1). This score ranges from -1 to +1. Values near 0 indicate overlapping clusters. In general, negative values indicate that an object has been assigned to the wrong cluster [70]. On the other hand, silhoutte coefficient can be used to determine the appropriate number of clusters.

4.3.3 Climate classes of original versus embeddings (without clustering)

Each point in Figure 4.7 (a) represents the time series for 3 years for a particular county (i.e. 1096x16 matrix) and each point in Figure 4.7 (b) represents the compressed time series (i.e. 72x7) for a specific county based on the embeddings. Further, it represents how each time series (original and embeddings based) belongs to the three climate classes (no clustering is performed).

In order to know how well the three climate classes are represented as clusters with original versus embeddings, we have calculated the Silhouette score. It is clear in Figure 4.7 that the climate classes in embeddings representation (right) are more tight and separated than the original representation (left). Further, it is reflected in the calculated Silhouette scores; Original: 0.235, Embeddings: 0.563.

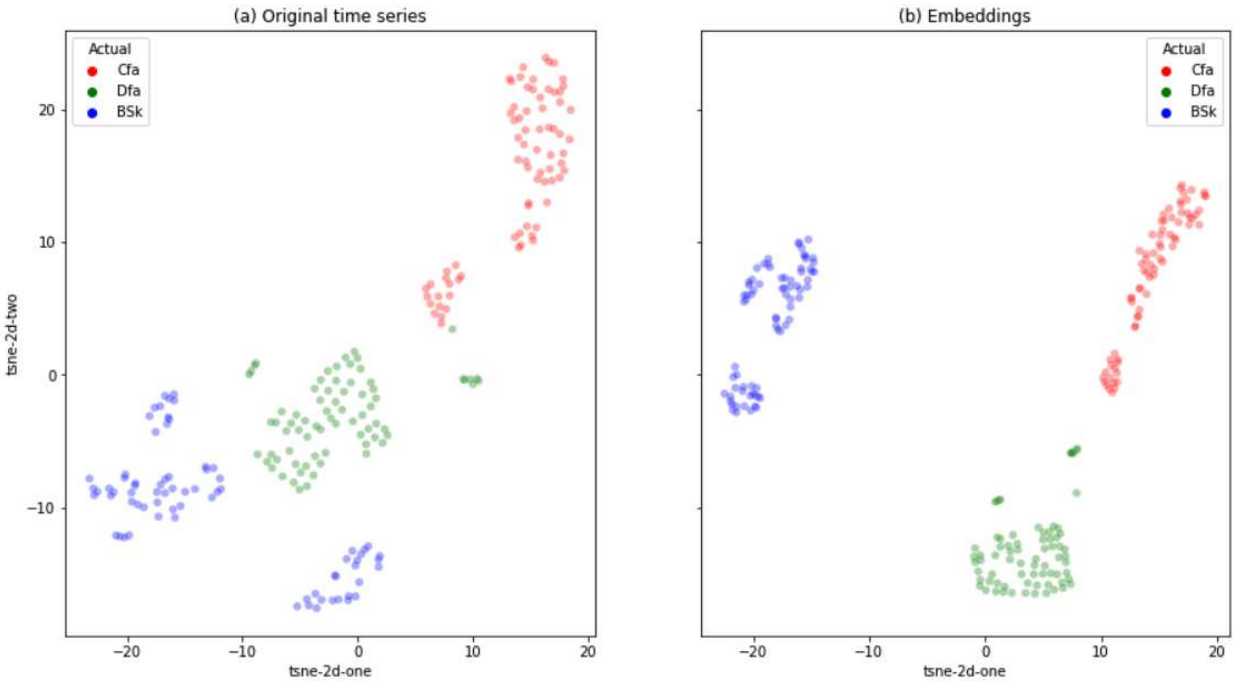

Figure 4.7: Climate classes of original time series versus embeddings, visualized using 2D t-SNE plots

4.3.4 Comparison between climate regions versus clustering of embeddings

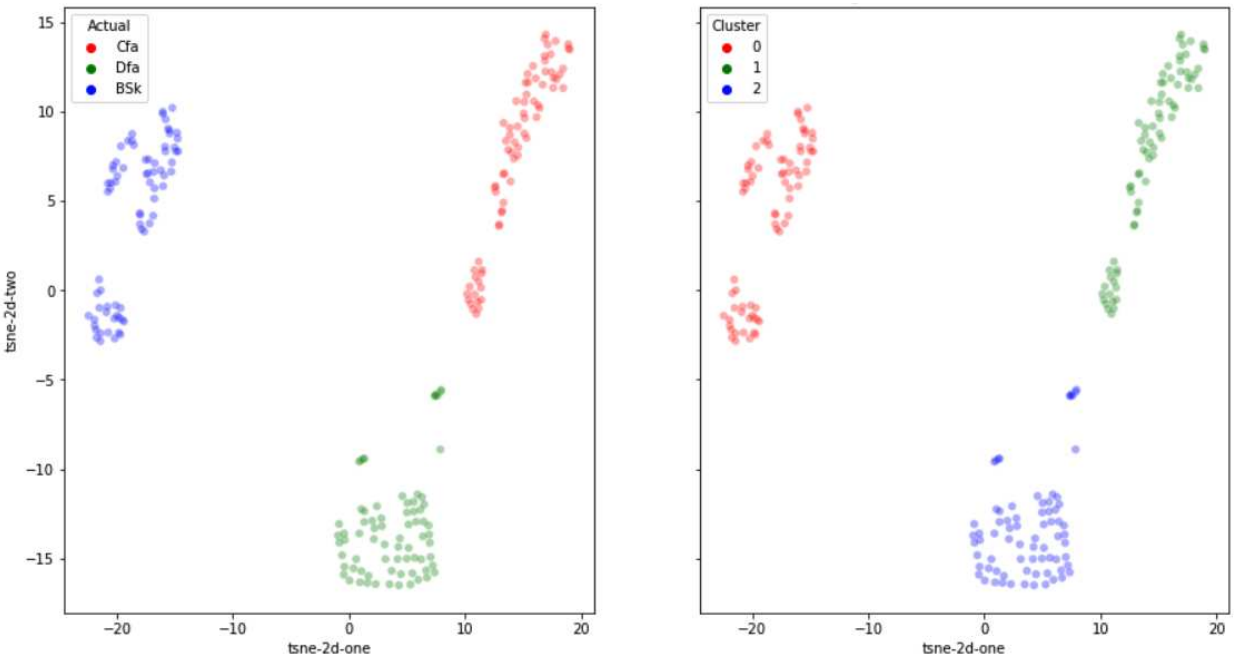

Figure 4.8: Climate regions and clustering of embeddings

Both plots in Figure 4.8 are related to embeddings. Left plot shows how the points belong to the three climate classes, while right plot shows how the same embeddings are clustered by k-Means, when using DTW as the distance metric. The clusters are numbered as 0,1 and 2. It is very clear that k-Means algortihm has identified clusters, identical to actual climate class clustering. Therefore, both plots yield a Silhouette score of 0.563.

4.3.5 Reduced features from Random Forest versus embeddings

Random Forest (RF) is a widely known classification model and it is an ensemble of decision tree algorithms. However, Random Forests are frequently used for feature selection during data science workflows. Such suitability is there because the tree-based strategies used by RFs, ranks the features by how well they improve the purity of the node. Therefore, a RF with 100 decision trees were trained using normalized training data. Once the training is over, we could find out the feature importance of each feature. Table 4.4 contains importance score for each feature.

Table 4.4: Feature Importance using Random Forest Classifier

Feature	Feature Importance
min specific humidity	0.058
max specific humidity	0.044
min precipitation	0.019
max precipitation	0.034
min surface downwelling shortwave flux in air	0.066
max surface downwelling shortwave flux in air	0.072
min max air temperature	0.083
max max air temperature	0.077
min min air temperature	0.085
max min air temperature	0.061
min eastward wind	0.053
max eastward wind	0.055
min northward wind	0.058
max northward wind	0.063
min vpd	0.05
max vpd	0.122

One important thing to note is, the sum of feature importances add up to 1 and features with higher feature importance are more important than others. Therefore, we used a threshold value of 0.06 to select features from the original 16 features. By using this threshold value, we were able to reduce the dimensionality by half. Selected features are;

- max vpd
- min min air temperature
- min max air temperature
- max max air temperature
- max surface downwelling shortwave flux
- min surface downwelling shortwave flux
- max northward wind
- max min air temperature

Next, we extract the selected features from testing dataset and normalized them. Afterwards, timeseries matrices were formed for each county, which are of size 1096x8. These time series were plotted using t-SNE according to climate classes (refer Figure 4.9 left) and performed a k-Means clustering with DTW distance metric (refer Figure 4.9 right). Silhouette score for climate classes is 0.315 and clustering is 0.332.

Figure 4.10 shows how the clustering created using our new approach (i.e. embeddings) compare to the clustering using reduced features by RF. Embedding based clustering is having a higher Silhouette score of 0.563 while the other is having only 0.332.

4.3.6 Reduced features from PCA versus embeddings

Principal Component Analysis known as PCA is a popular dimensionality reduction technique. In this section, we have tried to use this method and compare it against our new embeddings approach. The first step in this process is to fit the normalized training data using PCA. When doing so, PCA will create a similar number of new principal components (16 with this dataset), and each principal component is a linear combination of the original features. Then we need to identify how many components we need to keep for the rest of the process. To make this decision accurately we can use the cumulative variance plot.

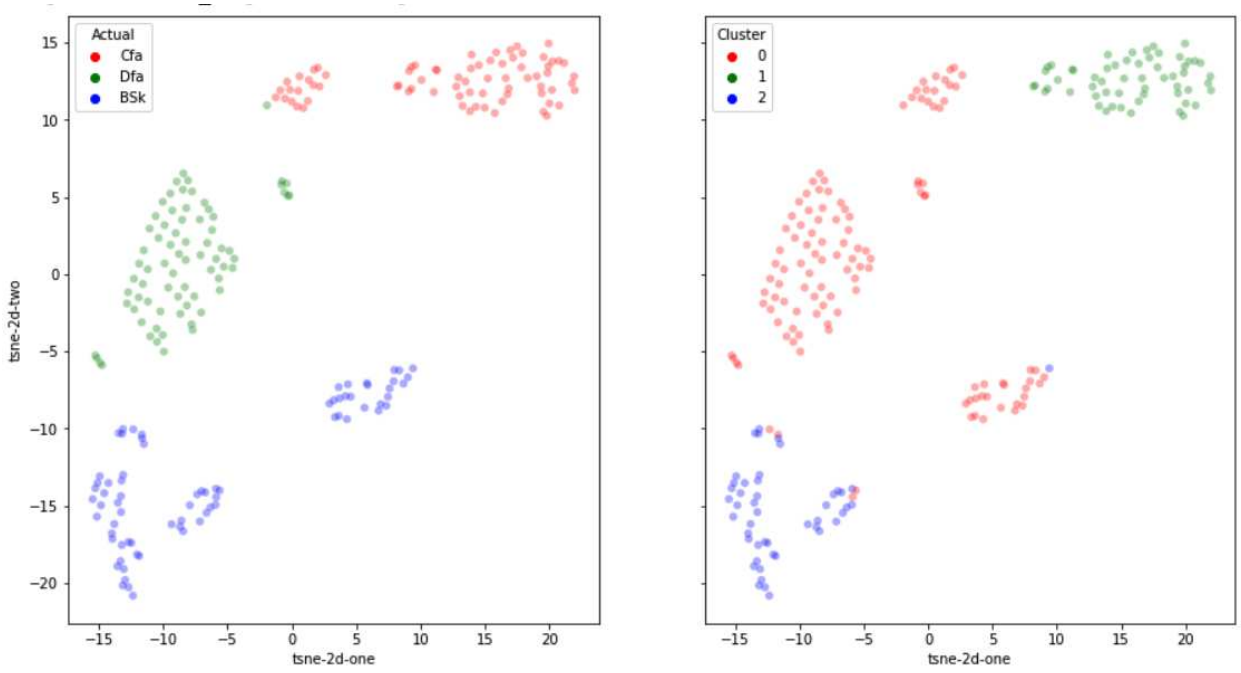

Figure 4.9: RF selected features based climate classes versus clustering

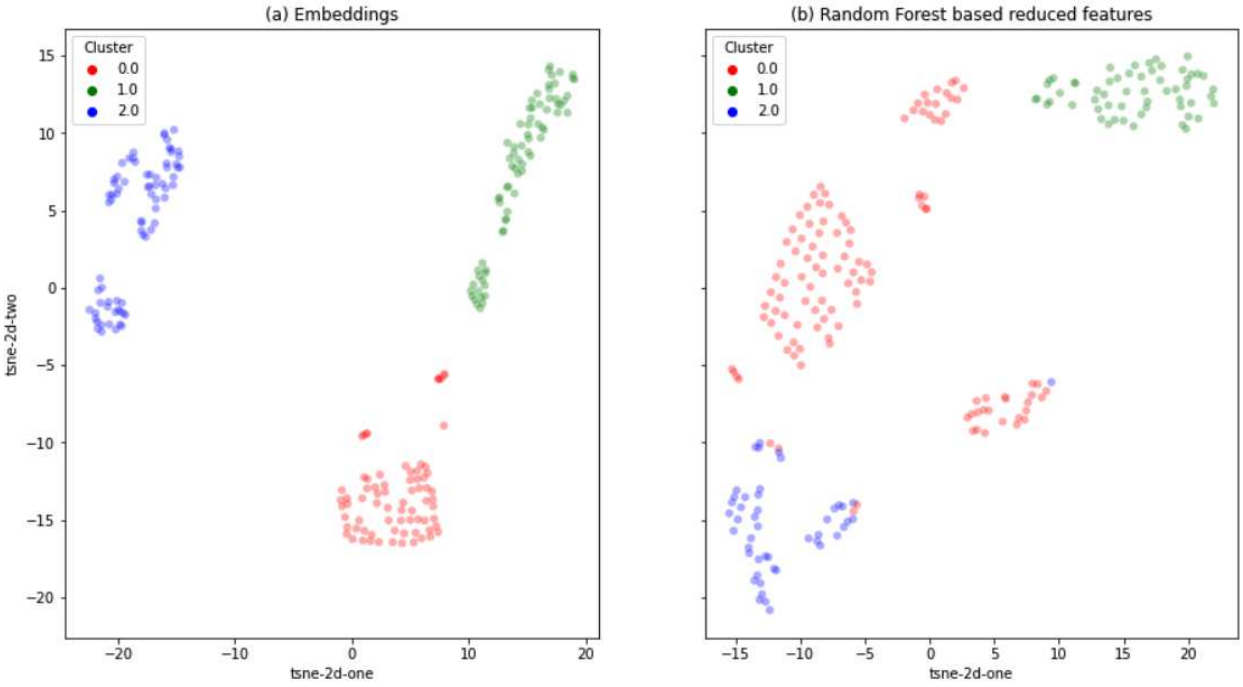

Figure 4.10: Embeddings clustering versus RF selected features based clustering

On the y-axis of Figure 4.11 shows the amount of variance captured, depending on the number of components we include. A rule of thumb is to preserve around 80% of the variance. In this

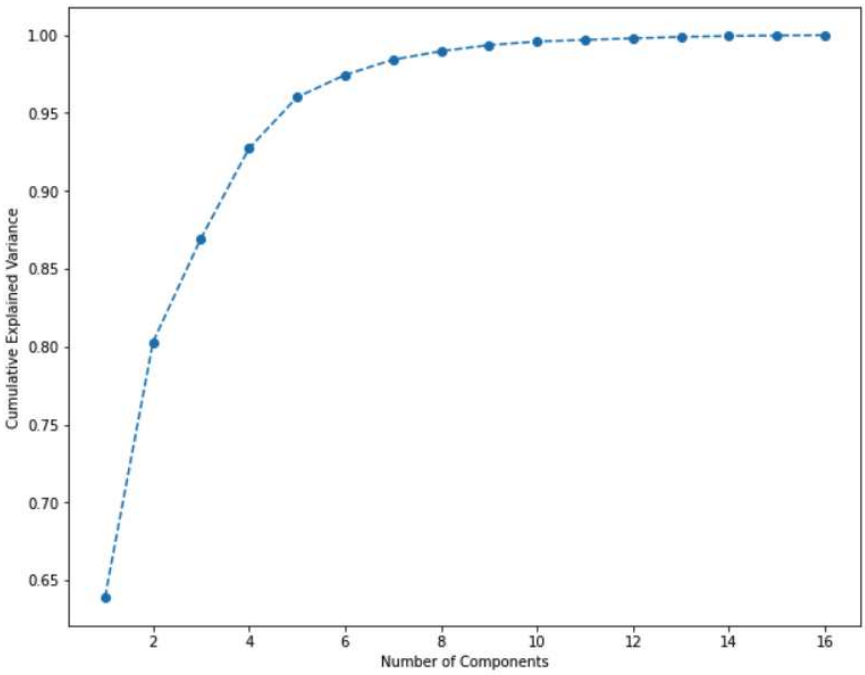

Figure 4.11: Cumulative explained variance by PCA components

instance, cumulative variance of first two components cover just above 80%. Thus, we decided to use the first two components. Then we reduce the dimensionality of normalized testing data from 16 to 2. Hence, for each county we will have a matrix of 1096x2. These time series were plotted using t-SNE according to climate classes (refer Figure 4.12 left) and performed a k-Means clustering with DTW distance metric (refer Figure 4.12 right). Silhouette score for climate classes is 0.309 and clustering is 0.343.

Figure 4.13 shows how the clustering created using our new approach (i.e. embeddings) compare to the clustering based on PCA components. Embedding based clustering is having a higher Silhouette score of 0.563 while the PCA based method is having only 0.343.

4.3.7 DTW versus other distance metrics

All the experiments that were discussed in this chapter used Dynamic Time Warping (DTW) as the distance metric (refer Section 3.4.2) during distance calculation between multidimensional time series for clustering and visualization (t-SNE plots). Hence, we wanted to see how it compares with other distance metrics. In this section, we have experimented and reported the results by

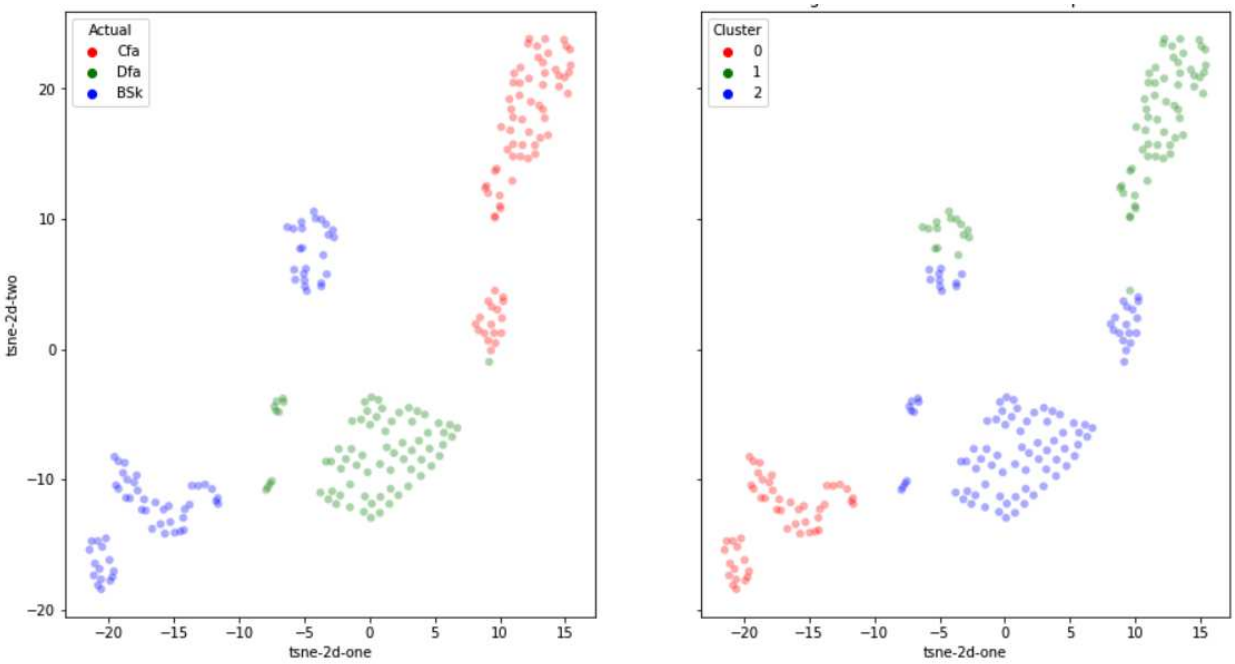

Figure 4.12: PCA component based climate classes versus clustering

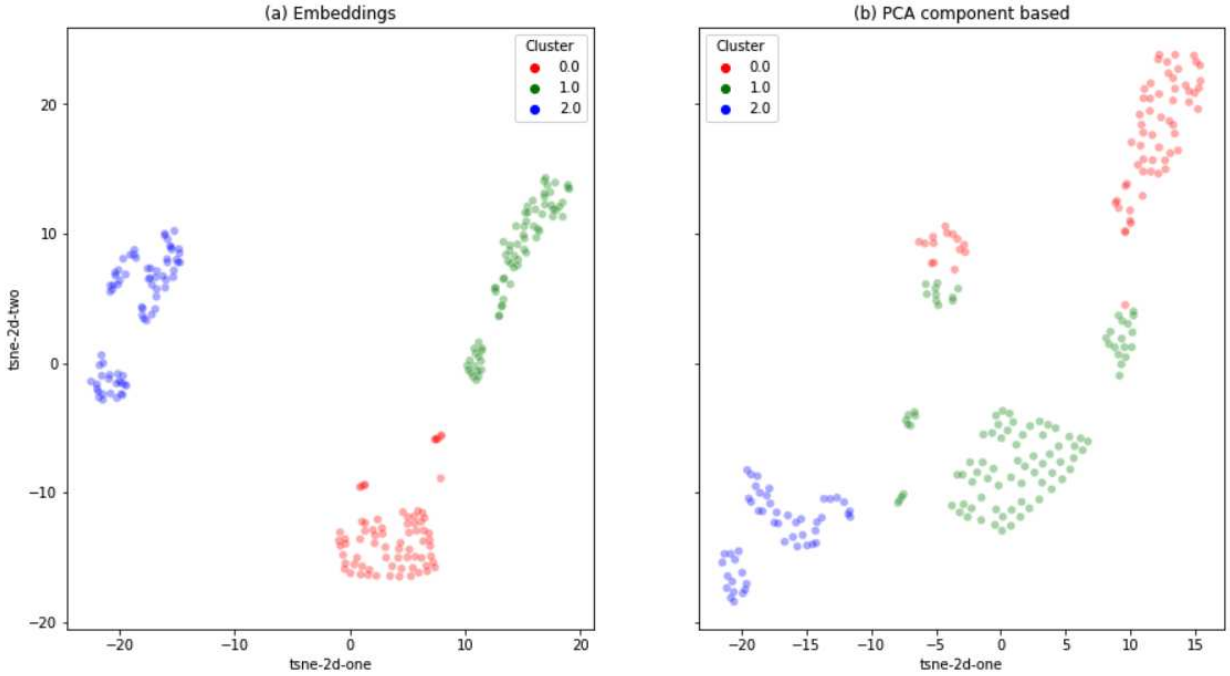

Figure 4.13: Embeddings clustering versus PCA component based clustering

using another popular distance metric known as Euclidean distance. In general, for points given by Cartesian coordinates in n-dimensional Euclidean space, the distance is calculated using the below formula [71],

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

In order to calculate the Euclidean distance, it is necessary for the two time series to be of same dimension and same size. Data in this dataset fulfills this requirement. Therefore, we were able to calculate Euclidean distances of original time series as well as embeddings.

Hence, we tried to cluster original time series and embeddings using k-Means clustering algorithm. Once using DTW as the distance metric and next time using Euclidean distance. It is important to note that t-SNE plots in both Figure 4.14 and Figure 4.15 are drawn using DTW distance (Otherwise, the visualization of the points becomes confusing and uncomparable).

Figure 4.14(a) shows a comparison of original time series climate classification (a point represents a 1096x16 matrix for a county). Figure 4.14(b) and Figure 4.14(c) represents the three clustering groups (0.0, 1.0 and 2.0) when DTW and Euclidean distances are used for clustering original time series respectively. As depicted in table 4.5, the Silhouette score for the Euclidean distance based representation is 0.321 and it is higher than that for DTW distance based representation, which is only 0.210. This might be due to more separation between climate classes in Euclidean based representation.

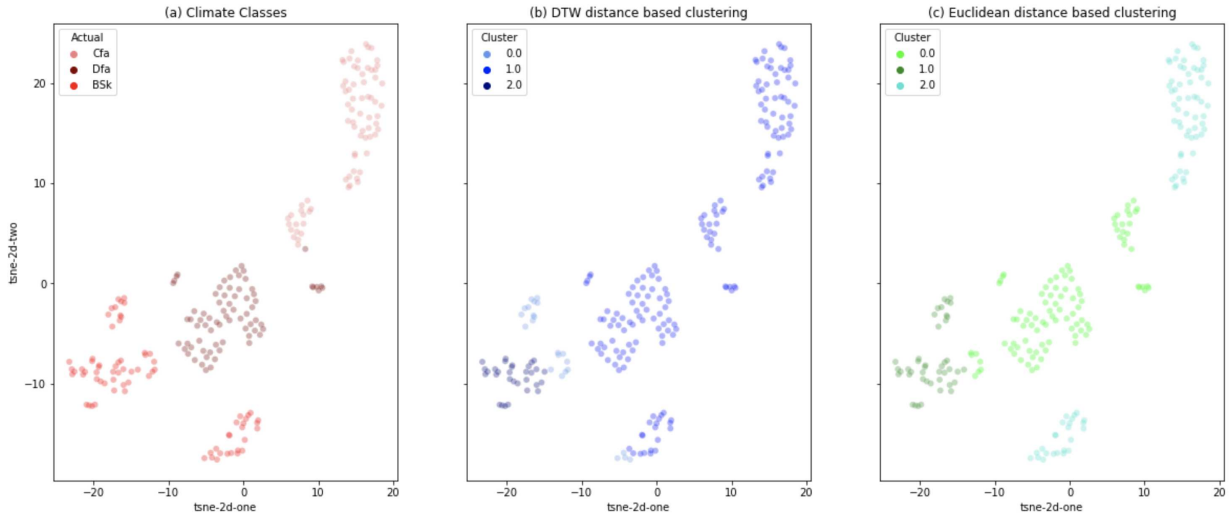

Figure 4.14: DTW versus Euclidean distance for original time series

Figure 4.15: DTW versus Euclidean distance for Embeddings

Next, we tried out DTW and Euclidean distance metrics to cluster embeddings. Refer Figure 4.15. Climate classification of embeddings is plotted in Figure 4.15(a). Figure 4.15(b) and Figure 4.15(c) represents the three clustering groups (0.0, 1.0 and 2.0) when DTW and Euclidean distances are used to cluster embeddings respectively. As shown in table 4.5, a Silhouette score of 0.563 is achieved by DTW based clustering of embeddings. And this is higher than that of Euclidean based one (0.546).

As shown in table 4.5, DTW based embeddings clustering is much better than any other.

Table 4.5: Silhouette scores for DTW versus Euclidean distance

Dist. Calc. Method	Original clustering	Embeddings clustering
DTW	0.210	0.563
Euclidean	0.321	0.546

4.3.8 Discussion

Table 4.6 represents the Silhouette scores for clusterings based on three dimensionality reduction techniques which were experimented. Namely, Random Forest based reduced feature clustering, Principal Components based clustering and embeddings based clustering. By consid-

ering Silhouette scores, it is evident that embeddings based clustering is outperforming other two methods. On the other hand, embeddings based clustering generates 3 clusters which is exactly similar to the 3 climate classes (i.e. ground truth) (refer Figure 4.8). Thus, we can conclude that an efficient time series clustering model can be built using embeddings.

Table 4.6: Comparison of Silhouette scores

Reduction Method	Silhouette Score
RF	0.332
PCA	0.343
Embeddings	0.563

4.4 Training time and cluster size

4.4.1 Cluster set up

In this study, we used clusters with 4-10 nodes. Each node has 64 GB RAM and 16 CPUs (Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz). Further, each node is running CentOS (version 8.4.2105). However, in these experiments, each node is utilizing only 1 CPU during the training process. Additionally, we use version 3.6.8 of Python, version 2.6.2 of tensorflow and version 0.23.0 of Horovod (more details on Horovod in Section 4.4.2).

4.4.2 Distributed training

As depicted in table 4.7, training models using a single machine was very time consuming. To be precise, LSTM, Bidirectional LSTM and GRU models were trained for 2 hours 52 minutes, 3 hours 23 minutes, and 3 hours 3 minutes respectively. Hence, it was quite important to try and reduce the training times. To achieve this, *Horovod* [72] was used. It is a distributed deep learning training framework for *TensorFlow* [45], *Keras* [73], *PyTorch* [46], and *Apache MXNet* [74], developed by Uber. This enables distributed deep learning fast and easy to use by bringing model training times down from days and weeks to hours and minutes. Further, with Horovod, an

existing training script can be scaled up to run on hundreds of CPUs/GPUs in just a few lines of Python code.

To achieve distributed learning, two approaches are available; Model parallelization and Data parallelization. In our experiments, we used data parallelization. In this, all the workers own a replica of the model. The global batch of data is split into multiple minibatches, and processed by different workers. Each worker computes the corresponding loss and gradients with respect to the data it possesses. Before the updating of the parameters at each epoch, the loss and gradients are averaged among all the workers through a collective operation [75].

For instance, this is how the data was split into multiple minibatches, when there was 4 nodes in the distributed cluster;

From each climate class there were 295x72 = 21240 subsequences as training data (295 counties per climate class). Therefore, a total of 21240x3 = 63720 subsequences were available (3 climate classes were used). Each of the nodes were responsible for training the model with quarter of the full training data (i.e. 15930 subsequences), since there are 4 nodes in the cluster. Similarly, when there were more nodes in the cluster, training data was split across nodes accordingly.

Table 4.7: Training times of models

	1 Node	4 Nodes	6 Nodes	8 Nodes	10 Nodes
LSTM	172m 22s	47m 37s	33m 5s	25m 13s	21m 21s
BiLSTM	203m 51s	56m 57s	39m 29s	30m 17s	25m 19s
GRU	183m 30s	50m 38s	35m 20s	27m 21s	22m 44s

As it is shown, when there are n number of nodes in the cluster, the training times have decreased by almost n fold. For instance time taken by BiLSTM model when running with only one machine is 203 minutes. With a 4 node cluster, time taken in reduced to 56 minutes. Therefore, the speedup can be calculated by 203 minutes/56 minutes = 3.6. The reason for this could be directly attributed to the data parallel architecture used during distributed training. However, the reason for not achieving a speed up of exactly 4, might be due to the overheads involved in distributed

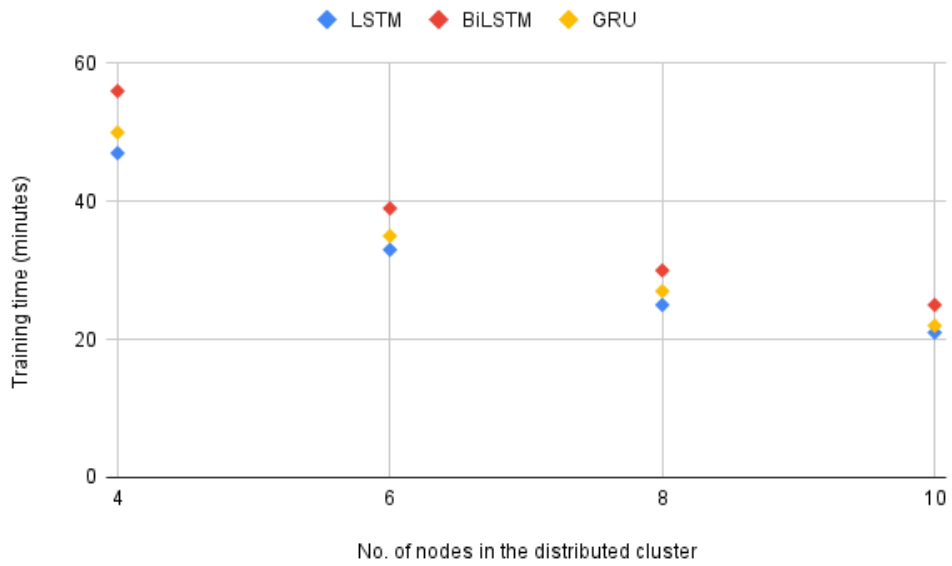

Figure 4.16: Distributed training times of models

training. Thus, it is evident that $scalability$ can be achieved using a distributed training framework like Horovod.

Chapter 5

Conclusions

We presented our methodology to effectively represent a geo-spatial, climate time series dataset using embeddings. This reduced dataset could be used for analytical applications such as visualization. Further, it enables to reduce the storage overheads, computational cost and time for analytical tasks.

5.1 Research Question 1(RQ1)

Representing geo-spatial time series data in reduced size (35x less) is achieved by generating subsequences of the original time series, training a Bidirectional LSTM autoencoder and generating embeddings from the extracted encoder of the autoencoder. The embeddings contain the most relevant features of the original time series.

5.2 Research Question 2(RQ2)

Quality of the embeddings based clustering using k-Means and DTW (as distance metric), outperformed the Random Forest based reduced feature clustering and Principal Components based clustering. Further, with embeddings based clustering, distance metric DTW performed better compared to Euclidean distance.

Bibliography

[1] Guozhu Dong and Jian Pei. *Sequence data mining*, volume 33. Springer Science & Business Media, 2007.

[2] Tung Kieu, Bin Yang, and Christian S Jensen. Outlier detection for multidimensional time series using deep neural networks. In *2018 19th IEEE International Conference on Mobile Data Management (MDM)*, pages 125–134. IEEE, 2018.

[3] Tian Guo, Zhao Xu, Xin Yao, Haifeng Chen, Karl Aberer, and Koichi Funaya. Robust online time series prediction with recurrent neural networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 816–825. Ieee, 2016.

[4] John T Abatzoglou and Timothy J Brown. A comparison of statistical downscaling methods suited for wildfire applications. *International journal of climatology*, 32(5):772–780, 2012.

[5] Sangmi Lee Pallickara, Shrideep Pallickara, and Marlon Pierce. Scientific data management in the cloud: A survey of technologies, approaches and challenges. In *Handbook of Cloud Computing*, pages 517–533. Springer, 2010.

[6] Yunhua Gu, Shu Shen, Jin Wang, and Jeong-Uk Kim. Application of nosql database mongodb. In *2015 IEEE International Conference on Consumer Electronics-Taiwan*, pages 158–159. IEEE, 2015.

[7] Michael Stonebraker and Lawrence A Rowe. The design of postgres. *ACM Sigmod Record*, 15(2):340–355, 1986.

[8] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Galileo: A framework for distributed storage of high-throughput data streams. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 17–24. IEEE, 2011.

[9] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing*, 5(1):28–42, 2015.

[10] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Analytic queries over geospatial time-series data using distributed hash tables. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1408–1422, 2016.

[11] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Evaluating geospatial geometry and proximity queries using distributed hash tables. *Computing in Science & Engineering*, 16(4):53–61, 2014.

[12] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Polygon-based query evaluation over geospatial data using distributed hash tables. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 219–226. IEEE, 2013.

[13] Thilina Buddhika, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2552–2566, 2017.

[14] Thilina Buddhika, Matthew Malensek, Shrideep Pallickara, and Sangmi Lee Pallickara. Living on the edge: Data transmission, storage, and analytics in continuous sensing environments. *ACM Transactions on Internet of Things*, 2(3):1–31, 2021.

[15] Thilina Buddhika, Sangmi Lee Pallickara, and Shrideep Pallickara. Pebbles: Leveraging sketches for processing voluminous, high velocity data streams. *IEEE Transactions on Parallel and Distributed Systems*, 32(8):2005–2020, 2021.

[16] Saptashwa Mitra, Daniel Rammer, Shrideep Pallickara, and Sangmi Lee Pallickara. A generative approach to visualizing satellite data. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 815–816. IEEE, 2021.

[17] Saptashwa Mitra, Daniel Rammer, Shrideep Pallickara, and Sangmi Lee Pallickara. Glance: A generative approach to interactive visualization of voluminous satellite imagery. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 359–367. IEEE, 2021.

[18] Saptashwa Mitra, Paahuni Khandelwal, Shrideep Pallickara, and Sangmi Lee Pallickara. Stash: Fast hierarchical aggregation queries for effective visual spatiotemporal explorations. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11. IEEE, 2019.

[19] Paahuni Khandelwal, Daniel Rammer, Shrideep Pallickara, and Sangmi Lee Pallickara. Mind the gap: Generating imputations for satellite data collections at myriad spatiotemporal scopes. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 92–102. IEEE, 2021.

[20] Kevin Bruhwiler, Paahuni Khandelwal, Daniel Rammer, Samuel Armstrong, Sangmi Lee Pallickara, and Shrideep Pallickara. Lightweight, embeddings based storage and model construction over satellite data collections. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 246–255. IEEE, 2020.

[21] Peter Bloomfield. *Fourier analysis of time series: an introduction*. John Wiley & Sons, 2004.

[22] Michael Heideman, Don Johnson, and Charles Burrus. Gauss and the history of the fast fourier transform. *IEEE ASSP Magazine*, 1(4):14–21, 1984.

[23] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[24] Han Lin Shang. A survey of functional principal component analysis. *AStA Advances in Statistical Analysis*, 98(2):121–142, 2014.

[25] Timescaledb: Sql made scalable for time-series data. https://pdfs.semanticscholar.org/049a/af11fa98525b663da18f39d5dcc5d345eb9a.pdf, 2017. [Accessed 28 October 2021].

[26] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[27] Domenico Buongiorno, Cristian Camardella, Giacomo Donato Cascarano, Luis Pelaez Murciego, Michele Barsotti, Irio De Feudis, Antonio Frisoli, and Vitoantonio Bevilacqua. An undercomplete autoencoder to extract muscle synergies for motor intention detection. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[28] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.

[29] Jun Deng, Zixing Zhang, Erik Marchi, and Björn Schuller. Sparse autoencoder-based feature transfer learning for speech emotion recognition. In *2013 humaine association conference on affective computing and intelligent interaction*, pages 511–516. IEEE, 2013.

[30] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Icml*, 2011.

[31] Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. *Advances in neural information processing systems*, 29:2802–2810, 2016.

[32] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[33] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[34] Elyor Kodirov, Tao Xiang, and Shaogang Gong. Semantic autoencoder for zero-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3174–3183, 2017.

[35] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487. PMLR, 2016.

[36] Minmin Chen, Zhixiang Xu, Kilian Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. *arXiv preprint arXiv:1206.4683*, 2012.

[37] Adam Golinski, Reza Pourreza, Yang Yang, Guillaume Sautiere, and Taco S Cohen. Feedback recurrent autoencoder for video compression. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[38] Amirhossein Habibian, Ties van Rozendaal, Jakub M Tomczak, and Taco S Cohen. Video compression with rate-distortion autoencoders. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7033–7042, 2019.

[39] Ali Javed, Byung Suk Lee, and Donna M Rizzo. A benchmark study on time series clustering. *Machine Learning with Applications*, 1:100001, 2020.

[40] Rui Ding, Qiang Wang, Yingnong Dang, Qiang Fu, Haidong Zhang, and Dongmei Zhang. Yading: fast clustering of large-scale time series data. *Proceedings of the VLDB Endowment*, 8(5):473–484, 2015.

[41] Neda Tavakoli, Sima Siami-Namini, Mahdi Adl Khanghah, Fahimeh Mirza Soltani, and Akbar Siami Namin. Clustering time series data through autoencoder-based deep learning models. *arXiv preprint arXiv:2004.07296*, 2020.

[42] Dino Ienco and Roberto Interdonato. Deep multivariate time series embedding clustering via attentive-gated autoencoder. *Advances in Knowledge Discovery and Data Mining*, 12084:318, 2020.

[43] Chunfeng Song, Feng Liu, Yongzhen Huang, Liang Wang, and Tieniu Tan. Auto-encoder based data clustering. In *Iberoamerican congress on pattern recognition*, pages 117–124. Springer, 2013.

[44] Jianqing Fan, Fang Han, and Han Liu. Challenges of big data analysis. *National science review*, 1(2):293–314, 2014.

[45] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[47] What is maca? http://www.climatologylab.org/maca.html. [Accessed 4 November 2021].

[48] Ben Livneh, Eric A Rosenberg, Chiyu Lin, Bart Nijssen, Vimal Mishra, Kostas M Andreadis, Edwin P Maurer, and Dennis P Lettenmaier. A long-term hydrologically based dataset of land surface fluxes and states for the conterminous united states: Update and extensions. *Journal of Climate*, 26(23):9384–9392, 2013.

[49] Hylke E Beck, Niklaus E Zimmermann, Tim R McVicar, Noemi Vergopolan, Alexis Berg, and Eric F Wood. Present and future köppen-geiger climate classification maps at 1-km resolution. *Scientific data*, 5(1):1–12, 2018.

[50] Köppen climate classification. https://en.wikipedia.org/wiki/K%C3%B6ppen_climate_classification. [Accessed 4 November 2021].

[51] World map of the kÖppen-geiger climate classification updated map for the united states of america.

[52] Sustain. https://urban-sustain.org/index.php. [Accessed 1 July 2021].

[53] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[54] Recurrent neural networks. https://www.ibm.com/cloud/learn/recurrent-neural-networks. [Accessed 24 November 2021].

[55] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[56] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

[57] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[58] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[59] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[60] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[61] CI Johnpaul, Munaga VNK Prasad, S Nickolas, and GR Gangadharan. Trendlets: A novel probabilistic representational structures for clustering the time series data. *Expert Systems with Applications*, 145:113119, 2020.

[62] An introduction to dynamic time warping. https://rtavenar.github.io/blog/dtw.html. [Accessed 27 November 2021].

[63] Simon Fong. Using hierarchical time series clustering algorithm and wavelet classifier for biometric voice classification. *Journal of Biomedicine and Biotechnology*, 2012, 2012.

[64] tslearn.metrics.dtw. https://tslearn.readthedocs.io/en/latest/gen_modules/metrics/tslearn.metrics.dtw.html. [Accessed 27 November 2021].

[65] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[66] Geoffrey Hinton and Sam T Roweis. Stochastic neighbor embedding. In *NIPS*, volume 15, pages 833–840. Citeseer, 2002.

[67] sklearn.manifold.tsne. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html. [Accessed 10 June 2021].

[68] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[69] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[70] sklearn.metrics.silhouette_score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html. [Accessed 8 November 2021].

[71] John Tabak. *Geometry: the language of space and form*. Infobase Publishing, 2014.

[72] Horovod. https://horovod.ai/. [Accessed 14 November 2021].

[73] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[74] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: a flexible and efficient machine learning library for heterogeneous distributed systems (2015). *arXiv preprint arXiv:1512.01274*, 2015.

[75] A quick guide to distributed training with tensorflow and horovod on amazon sagemaker. https://towardsdatascience.com/a-quick-guide-to-distributed-training-with-tensorflow-and-horovod-on-amazon-sagemaker\-dae18371ef6e. [Accessed 15 October 2021].