Hashing2Vec: Fast Embedding Generation for SARS-CoV-2 Spike Sequence Classification

Taslim Murad[*] **tmurad2@student.gsu.edu**
Prakash Chourasia[*] **pchourasia1@student.gsu.edu**
Sarwan Ali[*] **sali85@student.gsu.edu**
Murray Patterson[+] **mpatterson30@gsu.edu**
Department of Computer Science, Georgia State University, Atlanta, GA, USA

Abstract

Due to the ongoing coronavirus (COVID-19) pandemic, an unprecedented amount of SARS-CoV-2 sequence data is available. The scale of this data has out-paced traditional methods for its analysis, while machine-learning approaches aimed at clustering and classification of SARS-CoV-2 variants is becoming an attractive alternative. Since the SARS-CoV-2 genome is highly dimensional, considering the much smaller spike region can save a great deal of processing. As the spike protein mediates the attachment of the coronavirus to the host cell, most of the newer and more contagious variants can be characterized by alterations to the spike protein; hence it is often sufficient for characterizing the different SARS-CoV-2 variants. Another important consideration is to have a fast feature embedding generation, which is the subject of this work.

Applying any machine learning (ML) model to a biological sequence requires first transforming it into a fixed-length (numerical) form. While there exist several compact embeddings for SARS-CoV-2 spike protein sequences, the generation process is computationally expensive since the features, added to the resulting vectors, are indexed in a naïve fashion. To solve this problem, we propose a fast and alignment-free hashing-based approach to design a fixed-length feature embedding for spike protein sequences, called Hashing2Vec, which can be used as input to any standard ML model. Using real-world data, we show that the proposed embedding is not only efficient to compute but also outperforms current state-of-the-art embedding methods in terms of classification accuracy. In terms of runtime, we achieve up to a 99.8% improvement in the Hashing2Vec-based embedding generation as compared to the baselines on a set of 7K spike amino acid sequences. It also outperforms the baselines on this data in terms of predictive performance and achieves accuracy and ROC-AUC scores of 86% and 84.4%, respectively.

Keywords: COVID-19, SARS-CoV-2, Sequence Classification, Spike Sequence, Hashing, k-mers, Feature Embedding

1. Introduction

The COVID-19 disease caused by the severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) has had a lasting global impact. It has infected almost 1.36 million people from 219 countries as of April 2021 Uyangodage et al. (2021). According to a recent report (June 2022) by the centers for disease control and prevention (CDC) a total of 86, 379, 937

*. Equal Contribution
+. Corresponding Author

cases are reported in the United States alone. With the pandemic levels of this disease, an unprecedented amount of SARS-CoV-2 genomic sequencing data has been collected and is still ongoing. Such data is important for gaining a deeper understanding of the disease, which will help researchers to advise preventive measures and minimize its effects.

The SARS-CoV-2 genome, depicted in Figure 1, is composed of various regions that code for different proteins. Of these proteins, the spike (S) protein is notable because it is responsible for the attachment of the virus to the host cell membrane. Because of this, the mutation rate within the spike region is elevated compared to the remaining regions, as selection pressure for increased transmissibility is the driving force behind the emergence of new variants Harvey et al. (2021). As a result, many variants are characterized by particular mutations in the spike region, hence the spike protein often suffices to provide the necessary information for SARS-CoV-2 variant classification.

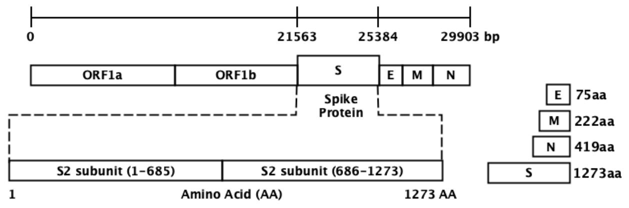

Figure 1: The SARS-CoV-2 genome, of roughly 30K base pairs in length, codes for both structural (S, E, M and N) and non-structural proteins (ORF1ab). Among the structural proteins, the S (spike), comprising roughly 1273 amino acids (see inset) is responsible for attaching the virus to the host cell membrane. Increased viral transmissibility is a result of advantageous mutations in the spike region.

Despite the savings in processing that considering only the spike protein affords, the number of sequences (millions) is several orders of magnitude beyond what can be handled by more traditional methods for analyzing sequencing data. As a result, machine learning (ML) approaches have begun to become an attractive alternative — many of the state-of-the-art variant classification and clustering methods being ML approaches Kuzmin et al. (2020); Ali et al. (2021b, 2022d, 2021a); Tayebi et al. (2021). Designing a fixed-length numerical representation (also called embedding) of such sequences is an important step in an ML-based classification pipeline Hu et al. (2022); Ali et al. (2022c); Ali (2022); Ali et al. (2021c); Ullah et al. (2020). Many existing methodologies are dealing with this conversion, such as Spike2Vec Ali and Patterson (2021), PWM2Vec Ali et al. (2022a) and string kernel Farhan et al. (2017); Ali et al. (2022b, 2021d). Alignment-based approaches are among the more computationally expensive due to the need for multiple sequence alignment operations as a preprocessing step Chowdhury and Garai (2017). Conversely, alignment-free approaches such as Spike2Vec generate a feature vector based on k-mers frequencies for a given sequence, while PWM2Vec uses k-mers weights to create the embedding. Although the existing embedding techniques yield promising classification results, they are still computationally costly to generate, especially for very long sequences (and large k). This is because they index the many features (*e.g.*, k-mers), in building the resulting vector, in a naïve and inefficient fashion. Therefore, to lower the computational overhead while keeping the classification performance high, we proposed a hashing-based embedding technique.

Hashing is a procedure for mapping data to fixed-size values. These values are treated as identifiers for the corresponding inputs. Hashing is a favorable technique for storing and retrieving data due to fast searches. Numerous methods use the underlying concept of hashing, such as bloom filters. A bloom filter is a hashing-based space-efficient probabilistic data structure used to verify the presence of an element (*e.g.*, a k-mer) in a set. Its space efficiency is due to using only bits to store these elements, but it can yield false-positive results if the hash function output overlaps for different input elements. The “counting” variant of a bloom filter stores elements along with their frequencies, supporting the deletion operation (by setting the frequency of an element to 0). Our approach uses a hash function — the Fowler-Noll-Vo (FNV) hash function to be precise — and keeps track of frequencies, but differently, to generate a feature vector that is amenable to downstream analysis with ML approaches, unlike a bloom filter, which is just a data structure to be used directly.

In summary, our contributions in this paper are the following:

1. We propose a fast, alignment-free, and efficient embedding method, called Hashing2Vec, which quickly computes a low dimensional numerical embedding for biological sequences.
2. We show that the proposed embedding method is easy to compute and can speed up the embedding generation time by up to 99.8% as compared to the baselines.
3. We also show that Hashing2Vec is efficient in terms of predictive performance on real-world SARS-CoV-2 spike sequence data using different machine learning classifiers. It outperforms the baselines and achieve up to 86% and 84.4% accuracy and ROC-AUC on a set of 7K spike amino acid sequences, respectively.
4. Using a t-distributed stochastic neighbor embedding (t-SNE) based visualization, we show that the overall structure of Hashing2Vec is not so different from the baseline embeddings.

The rest of the paper is organized as follows: The literature related to SARS-CoV-2 sequence classification is given in Section 2. Section 3 outlines the details of our proposed model along with existing embedding approaches (baselines). A description of the dataset used for experiments, and the evaluation metrics are discussed in Section 4. The results of these experiments are highlighted in Section 5, followed by Section 6 which concludes the paper and discusses future prospects.

2. Related Work

More traditional methods for analyzing sequencing data typically employ a phylogenetic approach Minh et al. (2020). However the number of sequences currently available for viruses such as SARS-CoV-2 (millions) is several orders of magnitude beyond the capabilities of such methods, making machine learning (ML) approaches, such as sequence classification, a more attractive alternative.

Sequence classification is a well-researched issue in bioinformatics Krishnan et al. (2021). Several alignment-based Kuzmin et al. (2020); Ali et al. (2022a) and alignment-free Ali and Patterson (2021) embedding approaches have been proposed recently for ML tasks such as classification and clustering. In Kuzmin et al. (2020), authors used a straightforward approach called One-Hot-Encoding (OHE) to generate the numerical representation for biological sequences, but the approach is not scalable due to the very high dimensionality of the feature vector. The authors in Ali et al. (2022a) generate feature embeddings for spike sequences using a position weight matrix (PWM) based method. However, one disadvantage is that it only functions for aligned sequences. The use of k-mers counts for phylogenetic applications was first explored in Blaisdell (1986), which proposed constructing accurate phylogenetic trees from several coding and non-coding nDNA sequences. The k-mers based approach is also used for sequence analysis in metagenomics Wood and Salzberg (2014). An alignment-free k-mers based approach for classifying SARS-CoV-2 sequences was proposed in Ali and Patterson (2021).

Although k-mers-based approaches achieve reasonable performance, their applications are inherently limited due to sparsity, and accurate k-mers calculation challenge for statistical methods. To counter this caveat, the authors in Singh et al. (2017) proposed the gapped k-mer.

Utilizing some idea of the similarity between biological sequences to create a kernel matrix Farhan et al. (2017); Ali et al. (2022b) is also used for classifying biological sequences. The quadratic ($O(n^2)$) memory cost of this method makes it nearly impossible to hold the kernel matrix in memory when n is large. In Shen et al. (2018), authors employ a neural network to extract the features using the Wasserstein distance (WD). The embedding generation has applications in many domains, such as in Hu et al. (2022), authors proposed backward compatible embedding for product recommendations. Feature embedding is an integral part of any machine learning-based solution.

Experiments on metagenomics datasets containing labels showed that Locality Sensitive Hashing (LSH) Shi and Chen (2019) can shorten training times for the model and achieve greater accuracy than competing approaches. The authors of Hash2Vec Argerich et al. (2016) have proposed a hash function to generate a (approximate) word embedding for language processing. The approximate nature of Hash2Vec allows for collision in the resultant vectors, which degrades the performance of the embedding. Authors in Laehnemann et al. (2016) demonstrated the use of bloom filters for error correction in raw read data to assist a de novo assembly. Authors in Ellis et al. (2019) use bloom filters to identify potentially overlapping reads to attain a less expensive all-to-all alignment. However, bloom filters have the disadvantage that collisions might happen.

3. Methodological Framework

This section discusses the existing systems available for sequence embedding, and we have used them as baseline systems for our experiments. Moreover, it also talks about our proposed approach in detail.

3.1. Efficient But Costly Baselines

There are several newly proposed approaches to create fixed-length numerical embeddings given spike sequences. They are expensive to produce even while their predictive performance is efficient. We start by discussing those methods in this section.

3.1.1. SPIKE2VEC ALI AND PATTERSON (2021)

This method aims to provide numerical embedding of the given input spike sequences to enable the application of ML models. Initially, it generates k-mers of the given spike sequence, as k-mers are known to preserve the ordering information of the sequence.

Definition 1 (k-mers) *It is referred to as a set of (consecutive) amino acids (also called mers) of length k for any given sequence (also called nGram in the NLP domain).*

The total number of k-mers generated for a sequence of length N is $N - k + 1$.

The Spike2Vec computes the frequency vector based on k-mers to map the k-mers alphabetical information into a numerical representation. This vector consists of the counts of occurrences of each k-mer of the sequence. The workflow of Spike2Vec is shown in Figure 2. The figure illustrates that for a given sequence with alphabet Σ of amino acids, its k-mers are computed and a feature vector of size $|\Sigma|^k$ is created to hold the k-mers frequencies (Figure 2 (b)). At the same time we use sliding window to generate k-mer for the spike sequence as shown in Figure 2 (c) and (d). Then for each k-mer, the k-mers counts are calculated Figure 2 e), and its respective bin in the feature vector is searched (bin searching see Defination 2) and updated with the frequency count. But this bin searching is an expensive operation, especially for long sequences, and high k. Our experiments are performed using $k = 3$.

Definition 2 (Bin searching) *Given a vector for all possible k-mers ($|\Sigma^k|$), each k-mer in the spike sequence is assigned to a bin in feature vector. Since we do not (initially) know the position of bin for a specific k-mer, we need to perform searching. This problem is called bin searching.*

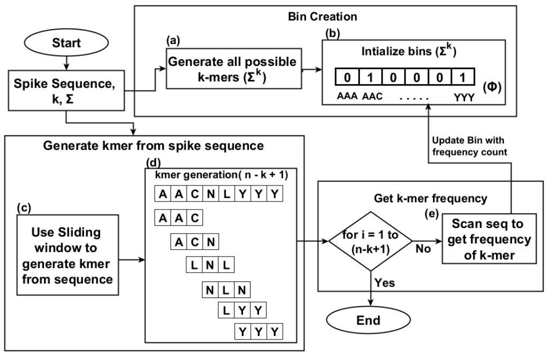
Start
Spike Sequence, k, Σ
Bin Creation
(a)
Generate all possible k-mers (Σ^k)
(b)
Intialize bins (Σ^k)
0 1 0 0 0 1
AAA AAC ..... YYY
(Φ)
Update Bin with frequency count
Generate kmer from spike sequence
(c)
Use Sliding window to generate kmer from sequence
(d)
kmer generation(n - k + 1)
A A C N L Y Y Y
A A C
A C N
L N L
N L N
L Y Y
Y Y Y
Get k-mer frequency
for i = 1 to (n-k+1)
No
(e)
Scan seq to get frequency of k-mer
Yes
End

Figure 2: Flow chart of Spike2Vec embedding.

3.1.2. PWM2Vec Ali et al. (2022a)

PWM2Vec is another approach for converting biological sequences to numerical form. It takes the sequence as input and yields the feature embedding as output. It follows the underlying concept of k-mers too, but rather than using constant frequency values of k-mers, it assigns weights to each amino acid of the k-mers. The weight determination of an amino acid is based on its location in a k-mer position weight matrix (PWM). PWM2Vec preserves the ordering information and considers amino acids' relative importance. We used $k = 9$ for this embedding in our experiments, which is decided using standard validation set approach Devijver and Kittler (1982).

3.1.3. String Kernel Farhan et al. (2017)

String kernel provides a way to measure the similarity between two sequences. The similarity is determined by measuring the number of matched and mismatched k-mers between two sequences, or if two k-mers are at a distance m, then they are considered to be matched. Furthermore, this approach proposed a closed-form solution for the size of the intersection of d-mismatch neighborhoods, which is a computationally efficient method. Moreover, it followed locality-sensitive hashing theory to estimate k-mers of two sequences at distance m, which further reduced the computational overhead. This methods design a kernel matrix of length $q \times q$, where q is the total number sequences in the data. We use the kernel matrix as input to kernel PCA to get the principal components based feature embeddings. Our experiments are carried out with $k = 3$ in this case. Since string kernel requires storing a kernel matrix, it have problem of storage overhead as q increases.

3.1.4. Wasserstein Distance Guided Representation Learning (WDGRL) Shen et al. (2018)

WDGRL is an unsupervised domain adoption technique. It uses the source and target encoded distributions to determine the Wasserstein distance (WD), which is utilized for extracting features from input data with the help of neural networks. It aims to determine the representation by minimizing the estimated WD and optimizing the feature extractor network. It uses the standard one-hot encoded (OHE) vector of a sequence as input. The OHE Kuzmin et al. (2020) is an algorithm for creating a fixed-length numerical representation of sequences. Due to WDGRL being based on a neural network, its requirement for training data can be expensive.

3.1.5. Spaced k-mers Singh et al. (2017)

Feature vectors for sequences based on k-mers frequencies are very large-sized and sparse, and their size and sparsity negatively impact the sequence classification performance. Spaced k-mers introduced the concept of using non-contiguous length k sub-sequences (g-mers) for generating compact feature vectors with reduced sparsity and size. Given a spike sequence as input, it first computed g-mers. From those g-mers, we compute k-mers, where $k < g$. We used $k = 4$ and $g = 9$ to perform the experiments. The size of the gap is determined by $g - k$. But this method still goes through computationally expensive operation of bin searching.

3.1.6. Auto-Encoder Xie et al. (2016)

This approach employs a deep neural network to learn the feature representation of data. It follows the technique of non-linear mapping from data space X to a lower-dimensional feature space Z, where it iteratively optimizes the objective. It takes the sequences as input. For our experiments, we have used a 2 layered network with an ADAM optimizer and MSE loss function.

3.1.7. SeqVec Heinzinger et al. (2019)

This work has proposed a way to represent the protein sequences in continuous vectors by using the language model named ELMO (Embeddings from Language Models) Sarzynska-Wawer et al. (2021). It captures the biophysical properties from the unlabeled data UniRef50 and creates the embeddings. This process is known as SeqVec (Sequence-to-Vector). It assigns the embeddings to a word by considering the context of a word.

3.2. Proposed Model: Hash the k-mers

Our goal in this paper is to generate numerical embeddings (ϕ) for spike sequences. A spike sequence is a long string of amino acids (characters). Let Σ represent the alphabet (the set of all unique amino acids comprised of $ACDEFGHIKLMNPQRSTVWXY$) and k is the length of a k-mer. The total number of k-mers of length k created from the given spike sequence will be $|\Sigma|^k$. Our model (called Hashing2Vec) uses a hashing-based technique for embedding, with a hash table size of m to reduce the bin searching overhead (see Definition 2). Moreover, the overall pipeline of Hashing2Vec for a given sequence is shown in Algorithm 1. For a given sequence s, k-mer length k and hash table size m, it returns the *exact* feature embedding of the sequence. The first step involves computing unique k-mers of size k in a spike sequence of length n. In the next step, we create a dictionary (local hash value to k-mers within a spike sequence) of size d, where $d \leq n - k + 1$ (since there will be repetitive k-mer in a sequence) and store the counts of each unique k-mer in the dictionary. The next step involves computing the (global) hash value for all possible k-mers in the data and assigning them a hash table position. For each (local) k-mer within a spike sequence, we use its global hash value and place its count in the hash table (we use this hash table as a feature embedding ϕ). The dimension of ϕ is the size m of the hash table. The final step is to use Principal Component Analyses (PCA) to get a low dimensional representation of ϕ. Each step is explained in more detail below:

Step 1: Generating k-mers:

To generate the fixed-length embedding for a given spike sequence, we generate all possible k-mers (see Definition 1) for a given spike sequence. An example of generating k-mers for a given spike sequence is shown in Figure 2(d).

Remark 3 *For Hashing2Vec embedding generation, we took $k = 3$, which is decided using standard validation set approach Devijver and Kittler (1982).*

Step 2: Counting the k-mers:

After creating the k-mers, we count the number of each k-mer by storing the unique k-mers in a dictionary (this can be thought of as "local" hashing of the k-mers within a specific spike sequence). After getting the k-mers count, the next step is to design an embedding ϕ, where each unique k-mer is assigned a bin that will contain its count as the numerical value. The k-mers that are not present in a given spike sequence will have zero value in ϕ.

To find the optimal bin for the k-mers in ϕ, a brute-force method is used to search the embedding to see which bin a specific k-mer belongs to. For each k-mer in a given spike sequence, this step must be repeated. In the worst case, this *bin search* for the relevant k-mers position can end up being an expensive process. Therefore, we intend to solve the problem of bin searching (see Definition 2) of ϕ in this paper, after the k-mers frequencies computation is done. Note that bin searching is not an optimization problem.

Remark 4 *Note that we are not concerned with the k-mers counting algorithm, rather our focus is to improve the ϕ generation as a result of k-mers counting (i.e., improving the traditional bin searching mechanism). There are many efficient and fast methods for k-mers counting, however, discussing (and using) those methods are out of the scope to this research article.*

Step 3: Assign Unique ID to k-mers Using Hashing:

We propose to use hashing to solve the *bin searching* problem. More specifically, we apply hashing (we refer to then as "Global" hash values) on the k-mers using a popular hash function, called Fowler–Noll–Vo Fowler et al. (2011), to assigns a unique ID (hash table entry) to each k-mer.

Definition 5 (Fowler–Noll–Vo (FNV)) *FNV is a non-cryptographic commonly utilized deterministic hash function.*

FNV (FNV-1a 32bit specifically) works by initializing a hash variable. Then it performs two major operations for each byte of data. First, an XOR of the byte and the hash is performed, and then the result is multiplied with a particular prime number. After mapping the k-mers to consistent hash values, we designed a frequency-based feature vector.

Remark 6 *Note that the hash values assigned in Step 3 are different from the dictionary of k-mers discussed in Step 2 (k-mers counting). In step 2, we are interested in designing a dictionary to locally count the unique k-mers within a specific spike sequence (which will be different for different spike sequences). Now, in order to design a general purpose feature embedding for a spike sequence, we need a "global" hash value for each possible k-mer in all spike sequences, so that any k-mer in any given spike sequence is mapped to same hash table entry (the resultant feature vector).*

For a given spike sequence, we now have a k-mers count (computed in step 2) and a global hash value (computed using FNV). At the global hash table entry, we place the k-mer count directly for all k-mers in a given spike sequence. All the other entries in the hash table will have the value 0. The pseudocode for Hashing2Vec is given in Algorithm 1.

Our experiments show that for $k = 3$, the optimal hash table size (m) is 404048. The optimality of m is determined (iteratively) by eliminating the collisions of hash values. Since the parameter m (hash table size) is a learned parameter, hence it guarantee zero collisions in the hash table. The value of m is learned by iteratively increasing the hash table size until each unique k-mer got a unique hash table id. Therefore our created feature embeddings are exact (not approximate embeddings). Moreover, the learning of m needed to be done only once in the start and we can then have $O(1)$ time complexity for placing each k-mers in relevant bins of feature vector (the hash table entry) afterwards for any number of sequences (hence it could be scaled to any size of data as no bin searching overhead is required). For Hashing2Vec, we only require one hash function. The resultant hash table (containing k-mers count) is considered as the final feature vector ϕ. At this point, for q spike sequences (total number of spike sequences in the data), we get a $q \times m$ dimensional matrix, where each row corresponds to a numerical representation of a spike sequence and each column corresponds to a specific k-mer count.

Algorithm 1: Hashing2Vec

```
Input: Spike Sequence $seq$, and integer $k$ and $m$
Output: Feature Vector $\phi$ based on Hash Values
$kmers = \emptyset$;
$\phi = \text{List}(0) \times \text{m}$;          // feature embedding vector of length $m$
for $i \leftarrow 1$ to $|seq|$ do
    $kmer$.append($seq[i : i + k]$);          // create k-mers using sliding window
end
$unique\_kmer, kmer\_count$ = createDictionary($kmers$);          // local hash value
/* create the FNV hash of size m for each k-mer using its count */
for $i \leftarrow 1$ to $|unique\_kmer|$ do
    $global\_hash\_value = FNV(unique\_kmer[i], m)$
    // map k-mer count to hash table for each unique kmer in dictionary
    $\phi[global\_hash\_value] = kmer\_count[i]$
end
$\phi$ = PCA($\phi$, 500);          // get the first 500 PCA components
return($\phi$)
```

Step 4: Low Dimensional Representation Using PCA:

Since the dimensionality (size) of the hash table (feature embedding ϕ) is very high, we applied Principle Component Analysis (PCA) Wold et al. (1987) to reduce it. PCA is a popular and widely followed technique for the reduction of data dimensionality. For our experiments, we have chosen the first 500 PCA components.

Hashing2Vec Workflow

The workflow of our proposed model is given in Figure 3. In the first step, we compute the unique k-mers frequency count for the spike sequence of length n as shown in the left box of Figure 3 (a), (b) and (c)). The k-mers are generated using a sliding window of size k. Along with it, a dictionary d is maintained to keep the counts of unique k-mers in a spike sequence, where the size $|d|$ of the dictionary is $|d| \leq n - k + 1$ (since we will have repeated k-mers in the sequence). Afterward, each unique k-mer in the dictionary d is passed through the FNV hash function to get a (global) corresponding hash value. The k-mer frequency count from the dictionary is mapped to the hash table using the respective computed (global) hash value for each unique k-mer in the dictionary as shown in Figure 3 (d)). Finally, the feature embedding (ϕ) of the sequence is generated using the hash table, and the length of the feature embedding (ϕ) is m since the size of the hash table is m.

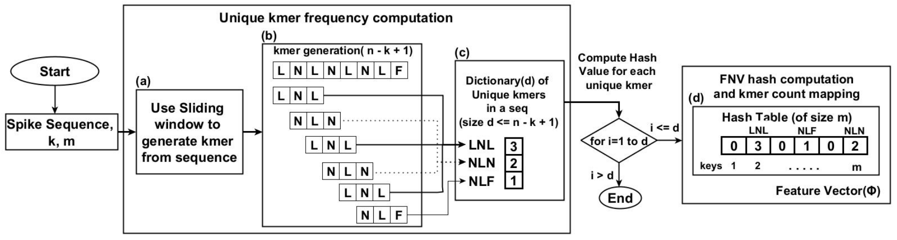
Unique kmer frequency computation
Start
Spike Sequence, k, m
(a)
Use Sliding window to generate kmer from sequence
(b)
kmer generation(n - k + 1)
L N L N L N L F
L N L
N L N
L N L
N L N
L N L
N L F
(c)
Dictionary(d) of Unique kmers in a seq (size d <= n - k + 1)
LNL 3
NLN 2
NLF 1
Compute Hash Value for each unique kmer
for i=1 to d
i <= d
i > d
End
(d)
FNV hash computation and kmer count mapping
Hash Table (of size m)
LNL
NLF
NLN
0 3 0 1 0 2
keys 1 2 ..... m
Feature Vector(Φ)

Figure 3: Flow chart of Hashing2Vec based embedding.

4. Evaluation Framework

In this section, we describe our dataset in detail. We also discuss the classifiers and evaluation metrics used for experimentation purposes. In the end, we show a visual representation of the data to analyze any (hidden) patterns from raw sequences. The datasets used are as follows,

Spike7k dataset: This dataset consists of SARS-CoV-2 spike sequences extracted from GISAID [1] along with information about the respective lineages of each sequence. The extracted data contains 22 coronavirus lineages within 7000 total sequences Ali et al. (2022b). The detail of each lineage i.e. name (count/distribution), in the dataset is as follow: B.1.1.7 (3369), B.1.617.2 (875), AY.4 (593), B.1.2 (333), B.1 (292), B.1.177 (243), P.1 (194), B.1.1 (163), B.1.429 (107), B.1.526 (104), AY.12 (101), B.1.160 (92), B.1.351 (81), B.1.427 (65), B.1.1.214 (64), B.1.1.519 (56), D.2 (55), B.1.221 (52), B.1.177.21 (47), B.1.258 (46), B.1.243 (36),and R.1 (32). For this dataset, our classification task is using the lineage as the class label. Table 1 illustrates demonstrates examples of spike sequences consisting of a long string of characters, where each character represents an amino acid, and variations/mutations in these amino acids form different lineages of the coronavirus. This table contains an example of Alpha, Beta, and Gamma variant sequences along with their respective mutations. The mutations column refers to amino acids, with their respective mutation position. For example, N501Y indicated that the amino acid N was to be replaced by Y at the 501 position of the sequence to create an alpha variant.

1. https://www.gisaid.org/

Coronavirus Host dataset: The spike sequences, along with their metadata (genus/ subgenus, infected host, etc), of the clades of the Coronaviridae family are extracted from ViPR Pickett et al. (2012); Ali et al. (2022a) and GISAID [2], and the spike sequence with its corresponding host information is utilized to form our Coronavirus Host dataset. The count of each host (class label) in our dataset is Bats (153), Bovines (88), Cats (123), Cattle (1), Equine (5), Fish (2), Humans (1813), Pangolins (21), Rats (26), Turtle (1), Weasel (994), Birds (374), Camels (297), Canis (40), Dolphins (7), Environment (1034), Hedgehog (15), Monkey (2), Python (2), Swines (558), and Unknown (2). We used a 5558 total number of spike sequences corresponding to 21 unique hosts. For this dataset, our classification tasks are using hostname as class label.

Sequence	Variant Name	Lineage	Mutations
MFVFLVLL...QPTYGVG...	Alpha	B.1.1.7	N501Y
MFVFL..N.K..QPTYGVG...	Beta	B.1.351	K417N,E484K,N501Y
MFVFL..T.K..QPTYGVG...	Gamma	P.1	K417T,E484K,N501Y

Table 1: An example of sequences for Alpha, Beta, and Gamma variants of coronavirus, along with their respective changes (marked red).

These two sequence datasets are used to perform experiments on code i5 processor-based windows 10 system having 2.40 GHz speed and 32 GB memory. Further, the Python programming language is used to carry out the experiments with 70-30% train and test split respectively, and average results are reported for 5 runs. Our code and pre-processed data are available online for reproducibility [3].

4.1. Evaluation Metrics and Classification Algorithms

Support Vector Machine (SVM), Naive Bayes (NB), Multi-Layer Perceptron (MLP), K-Nearest Neighbors (KNN), Random Forest (RF), Logistic Regression (LR), and Decision Tree (DT) classifiers are used as baseline models for the sequence classification. The performance of various models is evaluated using average accuracy, precision, recall, F1 (weighted), F1 (macro), Receiver Operator Characteristic Curve (ROC), Area Under the Curve (AUC), and training runtime metrics. Furthermore, the one-vs-rest approach is used to convert the binary evaluation metrics to multi-class ones.

4.2. Data Visualization

The t-distributed stochastic neighbor embedding (t-SNE) Van der Maaten and Hinton (2008) is utilized to identify any hidden patterns in the data. This method works by mapping the high dimensional input data into 2D space but preserves the pairwise distance between data points in high dimensions. This visualization aims to highlight if different embedding methods introduce any changes to the overall distribution of data. For various embedding methods, Figure 4 illustrated the t-SNE-based visualization (with variants as labels as shown in legends) of the Spike7k dataset. We can observe that overall the B.1.1.7 (Alpha) variant forms a single huge group (shown in yellow color) since its representation in the dataset is larger than other variants. Moreover, we can observe that Hashing2Vec is able to preserve the structure of the data similar to Spike2Vec, PWM2Vec, Spaced k-mers, and String kernel. The WDGRL shows a scattered t-SNE plot, which means that the overall

2. https://www.gisaid.org/
3. https://github.com/sarwanpasha/Hashing2Vec

structure of data, in that case, is disturbed, hence the performance of the embeddings will not be as good compared to other embedding methods (this behavior is also observed in classification results in the next section, in Table 2).

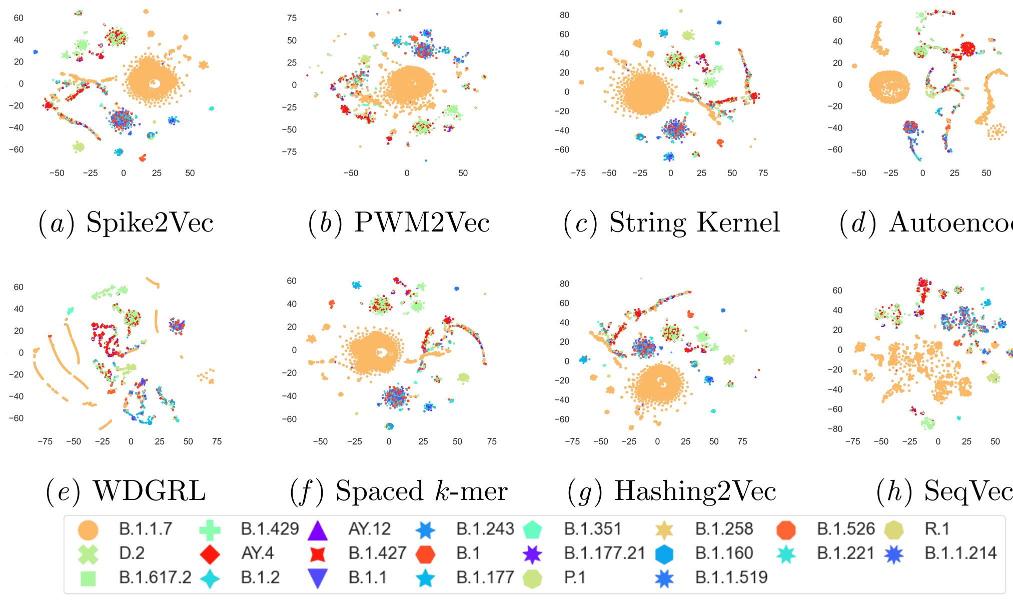

Figure 4: t-SNE plots using different embeddings for 7000 Spike7k dataset sequences. This figure is best seen in color.

5. Experimental Result

This section presents the classification results and embedding generation runtime for Hashing2Vec and compares it to several baseline approaches using various evaluation criteria.

5.1. Classification Results

We report the classification results for the proposed and the baseline methods in Table 2 for the Spike7k dataset. From this table, we can observe that the LR model using Hashing2Vec feature vectors outperformed all the other baselines embedding approaches in terms of average accuracy, precision, recall, F1 (weighted), and F1 (macro), while SVM using Hashing2Vec-based embedding has the highest ROC-AUC score as compared to other embedding methods. This indicates that the Hashing2Vec embedding approach is the best performing approach for sequence classification using ML models. Although WDGRL has a minimum train time, its classification performance, however, is the lowest. One reason for this behavior is that the WDGRL failed to preserve the overall structure of data in its embeddings as compared to the other embedding methods (see Figure 4 for visualization of embeddings). Although the prime goal of Hashing2Vec is to improve the embedding generation runtime, we can observe that it also outperforms all baselines for all but one evaluation metric.

Similarly, we also observe the behavior of Hashing2Vec for the Coronavirus Host dataset and report the results in Table 3. Since the original host classification is done using

Embedding	Algo.	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (Sec.)
Spike2Vec Ali and Patterson (2021)	SVM	0.855	0.853	0.855	0.843	0.689	0.843	61.112
	NB	0.476	0.716	0.476	0.535	0.459	0.726	13.292
	MLP	0.803	0.803	0.803	0.797	0.596	0.797	127.066
	KNN	0.812	0.815	0.812	0.805	0.608	0.794	15.970
	RF	0.856	0.854	0.856	0.844	0.683	0.839	21.141
	LR	0.859	0.852	0.859	0.844	0.690	0.842	64.027
	DT	0.849	0.849	0.849	0.839	0.677	0.837	4.286
PWM2Vec Ali et al. (2022a)	SVM	0.818	0.820	0.818	0.810	0.606	0.807	22.710
	NB	0.610	0.667	0.610	0.607	0.218	0.631	1.456
	MLP	0.812	0.792	0.812	0.794	0.530	0.770	35.197
	KNN	0.767	0.790	0.767	0.760	0.565	0.773	1.033
	RF	0.824	0.843	0.824	0.813	0.616	0.803	8.290
	LR	0.822	0.813	0.822	0.811	0.605	0.802	471.659
	DT	0.803	0.800	0.803	0.795	0.581	0.791	4.100
String Kernel Farhan et al. (2017)	SVM	0.845	0.833	0.846	0.821	0.631	0.812	7.350
	NB	0.753	0.821	0.755	0.774	0.602	0.825	0.178
	MLP	0.831	0.829	0.838	0.823	0.624	0.818	12.652
	KNN	0.829	0.822	0.827	0.827	0.623	0.791	0.326
	RF	0.847	0.844	0.841	0.835	0.666	0.824	1.464
	LR	0.845	0.843	0.843	0.826	0.628	0.812	1.869
	DT	0.822	0.829	0.824	0.829	0.631	0.826	0.243
WDGRL Shen et al. (2018)	SVM	0.792	0.769	0.792	0.772	0.455	0.736	0.335
	NB	0.724	0.755	0.724	0.726	0.434	0.727	0.018
	MLP	0.799	0.779	0.799	0.784	0.505	0.755	7.348
	KNN	0.800	0.799	0.800	0.792	0.546	0.766	0.094
	RF	0.796	0.793	0.796	0.789	0.560	0.776	0.393
	LR	0.752	0.693	0.752	0.716	0.262	0.648	0.091
	DT	0.790	0.799	0.790	0.788	0.557	0.768	**0.009**
Spaced k-mers Singh et al. (2017)	SVM	0.852	0.841	0.852	0.836	0.678	0.840	2218.347
	NB	0.655	0.742	0.655	0.658	0.481	0.749	267.243
	MLP	0.809	0.810	0.809	0.802	0.608	0.812	2072.029
	KNN	0.821	0.810	0.821	0.805	0.591	0.788	55.140
	RF	0.851	0.842	0.851	0.834	0.665	0.833	646.557
	LR	0.855	0.848	0.855	0.840	0.682	0.840	200.477
	DT	0.853	0.850	0.853	0.841	0.685	0.842	98.089
Auto-Encoder Xie et al. (2016)	SVM	0.699	0.720	0.699	0.678	0.243	0.627	4018.028
	NB	0.490	0.533	0.490	0.481	0.123	0.620	24.6372
	MLP	0.663	0.633	0.663	0.632	0.161	0.589	87.4913
	KNN	0.782	0.791	0.782	0.776	0.535	0.761	24.5597
	RF	0.814	0.803	0.814	0.802	0.593	0.793	46.583
	LR	0.761	0.755	0.761	0.735	0.408	0.705	11769.02
	DT	0.803	0.792	0.803	0.792	0.546	0.779	102.185
SeqVec Heinzinger et al. (2019)	SVM	0.796	0.768	0.796	0.770	0.479	0.747	1.0996
	NB	0.686	0.703	0.686	0.686	0.351	0.694	0.0146
	MLP	0.796	0.771	0.796	0.771	0.510	0.762	13.172
	KNN	0.790	0.787	0.790	0.786	0.561	0.768	0.6463
	RF	0.793	0.788	0.793	0.786	0.557	0.769	1.8241
	LR	0.785	0.763	0.785	0.761	0.459	0.740	1.7535
	DT	0.757	0.756	0.757	0.755	0.521	0.760	0.1308
Hashing2Vec	SVM	0.853	0.858	0.853	0.842	0.685	**0.844**	10.044
	NB	0.598	0.741	0.598	0.637	0.497	0.744	0.375
	MLP	0.759	0.763	0.759	0.752	0.554	0.776	10.972
	KNN	0.825	0.817	0.825	0.811	0.635	0.805	0.557
	RF	0.835	0.842	0.835	0.813	0.642	0.804	4.593
	LR	**0.860**	**0.862**	**0.860**	**0.847**	**0.699**	0.841	17.719
	DT	0.822	0.828	0.822	0.815	0.635	0.812	1.539

Table 2: Classification results for different evaluation metrics using the proposed and baseline methods for the Spike7k dataset. Best values are shown in bold.

PWM2Vec in Ali et al. (2022a), that is why we are comparing Hashing2Vec to PWM2Vec. Results illustrates that the RF method corresponding to Hashing2Vec outperforms PWM2Vec in terms of accuracy, precision, recall, and F1 weighted score, while the SVM of Hashing2Vec has the maximum AUC ROC score. Likewise, Hashing2Vec achieves minimum training time for NB. These results indicate that the Hashing2Vec method has better performance than PWM2Vec.

Embeddings	Algo.	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (Sec.)
PWM2Vec Ali et al. (2022a)	SVM	0.799	0.806	0.799	0.801	0.648	0.859	44.793
	NB	0.381	0.584	0.381	0.358	0.400	0.683	2.494
	MLP	0.782	0.792	0.782	0.778	0.693	0.848	21.191
	KNN	0.786	0.782	0.786	0.779	0.679	0.838	12.933
	RF	0.836	0.839	0.836	0.828	**0.739**	0.862	7.690
	LR	0.809	0.815	0.809	0.800	0.728	0.852	274.917
	DT	0.801	0.802	0.801	0.797	0.633	0.829	4.537
Hashing2Vec	SVM	0.815	0.825	0.815	0.818	0.725	**0.863**	5.591
	NB	0.588	0.649	0.588	0.583	0.585	0.791	**0.146**
	MLP	0.779	0.783	0.779	0.777	0.483	0.735	43.401
	KNN	0.812	0.809	0.812	0.809	0.642	0.817	0.499
	RF	**0.859**	**0.859**	**0.859**	**0.853**	0.735	0.846	5.767
	LR	0.573	0.479	0.573	0.493	0.213	0.591	4.638
	DT	0.800	0.804	0.800	0.800	0.660	0.840	1.855

Table 3: Classification results for different evaluation metrics using the proposed and baseline methods for Coronavirus Host dataset. Best values are shown in bold.

5.2. Embeddings Generation Time

To evaluate the computation time for different embedding generations, we report the runtime in Table 4 for the Spike7k dataset. We can observe that Hashing2Vec takes the lowest time to generate the feature vectors as compared to the baseline methods. The PWM2Vec is the second-best while the SeqVec takes the most time for feature vector generation. We observed the same behavior regarding the embedding generation runtime in the case of coronavirus host data as well.

We also provide % improvement for Hashing2Vec from PWM2Vec (second best in terms of runtime) and SeqVec(worst in terms of runtime) using the following expression for the Spike7k dataset:

$$\%\text{ improvement} \quad = \quad \frac{R_{Baseline} - R_{Hashing2Vec}}{R_{Baseline}} \times 100 \tag{1}$$

where $R_{Baseline}$ represents the runtime of baselines PWM2Vec and SeqVec embedding methods while $R_{Hashing2Vec}$ corresponds to the run-time for Hashing2Vec embedding computation. We can observe that Hashing2Vec improves the runtime performance by 68.7% and 99.8% as compared to PWM2Vec and SeqVec, respectively.

The runtime for computing PWM2Vec and Hashing2Vec with the increasing number of sequences is shown in Figure 5 for the Spike7k dataset. We can see that Hashing2Vec significantly outperforms the PWM2Vec (fastest among other embeddings) in terms of runtime with any number of sequences. Additionally, we can observe that the increasing runtime trend for Hashing2Vec is very slow as compared to the PWM2Vec, which makes it more suitable for Big Data.

Overall, we can observe that the proposed embedding, called Hashing2Vec not only performs slightly better in terms of predictive performance as compared to the baselines, but it also preserves the overall structure of the data similar to the recently proposed embedding methods. Moreover, Hashing2Vec can be generated very quickly as compared to the other methods, making it an ideal choice while dealing with larger-sized datasets because of its scalability property.

Embeddings	Runtime (Seconds)
Spike2Vec Ali and Patterson (2021)	354.061
PWM2Vec Ali et al. (2022a)	163.257
String Approx. Farhan et al. (2017)	2292.245
WDGRL Shen et al. (2018)	438.188
Spaced k-mers Singh et al. (2017)	12901.808
Auto-Encoder Xie et al. (2016)	181.70052
SeqVec Heinzinger et al. (2019)	32500.19
Hashing2Vec (ours)	**51.094**
% Improv. of Hashing2Vec from PWM2Vec	68.7%
% Improv. of Hashing2Vec from SeqVec	99.8%

Table 4: Embedding generation runtime for different methods using the Spike7k dataset. Best value is shown in bold. The percentage improvement of runtime (see Equation (1)) is also given for Hashing2Vec.

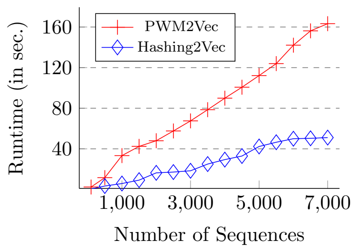

Figure 5: Runtime for embedding generation of PWM2Vec and Hashing2Vec with increasing number of sequences for the Spike7k dataset. The figure is best seen in color.

6. Conclusion

We propose an efficient and alignment-free method, called Hashing2Vec, to generate numerical embeddings for biological sequences. We show that Hashing2Vec-based embeddings are fast to compute and improve the classification results as compared to baselines. We performed extensive experiments on real-world biological data to validate the model using different evaluation metrics. Hashing2Vec shows 99.8% improvement in computational runtime as compared to the baselines for embedding generation using Spike7k data. It also achieves the maximum 86% accuracy and 84.4% ROC AUC score for the Spike7k data classification, and the highest 85.9% accuracy and 86.3% ROC AUC score for the Coronavirus Host data classification as compared with the given baseline models. Future work involves evaluating the Hashing2Vec for larger sets of sequence data (millions) and also applying Hashing2Vec to other virus data such as Zika. We also plan to explore deep learning models for spike sequence classification. Using other alignment-free baseline methods to compare their performance with Hashing2Vec is another potential future extension.

Acknowledgements

The authors would like to acknowledge funding from an MBD fellowship to Sarwan Ali and Georgia State University Computer Science Startup Grant to Prakash Chourasia and Murray Patterson.

References

Sarwan Ali. Information we can extract about a user from'one minute mobile application usage'. *arXiv preprint arXiv:2207.13222*, 2022.

Sarwan Ali and Murray Patterson. Spike2vec: An efficient and scalable embedding approach for covid-19 spike sequences. In *IEEE International Conference on Big Data (Big Data)*, pages 1533–1540, 2021.

Sarwan Ali, Tamkanat E Ali, Muhammad Asad Khan, Imdadullah Khan, and Murray Patterson. Effective and scalable clustering of sars-cov-2 sequences. In *International Conference on Big Data Research (ICBDR)*, pages 42–49, 2021a.

Sarwan Ali, Babatunde Bello, and Murray Patterson. Classifying covid-19 spike sequences from geographic location using deep learning. *arXiv preprint arXiv:2110.00809*, 2021b.

Sarwan Ali, Simone Ciccolella, Lorenzo Lucarella, Gianluca Della Vedova, and Murray Patterson. Simpler and faster development of tumor phylogeny pipelines. *Journal of Computational Biology*, 28(11):1142–1155, 2021c.

Sarwan Ali, Bikram Sahoo, Naimat Ullah, Alexander Zelikovskiy, Murray Patterson, and Imdadullah Khan. A k-mer based approach for sars-cov-2 variant identification. In *International Symposium on Bioinformatics Research and Applications*, pages 153–164, 2021d.

Sarwan Ali, Babatunde Bello, Prakash Chourasia, Ria Thazhe Punathil, Yijing Zhou, and Murray Patterson. PWM2Vec: An efficient embedding approach for viral host specification from coronavirus spike sequences. *Biology*, 11(3):418, 2022a.

Sarwan Ali, Bikram Sahoo, Muhammad Asad Khan, Alexander Zelikovsky, Imdad Ullah Khan, and Murray Patterson. Efficient approximate kernel based spike sequence classification. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2022b.

Sarwan Ali, Bikram Sahoo, Alexander Zelikovskiy, Pin-Yu Chen, and Murray Patterson. Benchmarking machine learning robustness in covid-19 genome sequence classification. *arXiv preprint arXiv:2207.08898*, 2022c.

Sarwan Ali, Yijing Zhou, and Murray Patterson. Efficient analysis of covid-19 clinical data using machine learning models. *Medical & Biological Engineering & Computing*, pages 1–16, 2022d.

Luis Argerich, Joaquín Torré Zaffaroni, and Matías J Cano. Hash2vec, feature hashing for word embeddings. *arXiv preprint arXiv:1608.08940*, 2016.

B Edwin Blaisdell. A measure of the similarity of sets of sequences not requiring sequence alignment. *Proceedings of the National Academy of Sciences*, 83:5155–5159, 1986.

Biswanath Chowdhury and Gautam Garai. A review on multiple sequence alignment from the perspective of genetic algorithm. *Genomics*, 109(5-6):419–431, 2017.

Pierre A Devijver and Josef Kittler. *Pattern recognition: A statistical approach*. Prentice hall, 1982.

Marquita Ellis, Giulia Guidi, et al. diBELLA: Distributed long read to long read alignment. In *Proceedings of the 48th International Conference on Parallel Processing*, pages 1–11, 2019.

Muhammad Farhan, Juvaria Tariq, Arif Zaman, Mudassir Shabbir, and Imdad Ullah Khan. Efficient approximation algorithms for strings kernel based sequence classification. *Advances in neural information processing systems*, 30, 2017.

Glenn Fowler, Landon Curt Noll, Kiem-Phong Vo, Donald Eastlake, and Tony Hansen. The fnv non-cryptographic hash algorithm. *IETF-draft: Fremont, CA, USA*, 2011.

W.T. Harvey, A.M. Carabelli, B. Jackson, et al. SARS-CoV-2 variants, spike mutations and immune escape. *Nature Reviews Microbiology*, 19:409–424, 2021.

Michael Heinzinger, Ahmed Elnaggar, Yu Wang, Christian Dallago, Dmitrii Nechaev, Florian Matthes, and Burkhard Rost. Modeling aspects of the language of life through transfer-learning protein sequences. *BMC bioinformatics*, 20(1):1–17, 2019.

Weihua Hu, Rajas Bansal, Kaidi Cao, Nikhil Rao, Karthik Subbian, and Jure Leskovec. Learning backward compatible embeddings. *arXiv preprint arXiv:2206.03040*, 2022.

Gokul S Krishnan, S Sowmya Kamath, et al. Predicting vaccine hesitancy and vaccine sentiment using topic modeling and evolutionary optimization. In *International Conference on Applications of Natural Language to Information Systems*, pages 255–263. Springer, 2021.

Kiril Kuzmin et al. Machine learning methods accurately predict host specificity of coronaviruses based on spike sequences alone. *BBRC*, 533(3):553–558, 2020.

David Laehnemann et al. Denoising DNA deep sequencing data—high-throughput sequencing errors and their correction. *Briefings in Bioinformatics*, 17(1):154–179, 2016.

B. Q. Minh et al. Iq-tree 2: New models and efficient methods for phylogenetic inference in the genomic era. *Molecular Biology and Evolution*, 37(5):1530–1534, 2020.

Brett E Pickett, Eva L Sadat, Yun Zhang, Jyothi M Noronha, R Burke Squires, et al. Vipr: an open bioinformatics database and analysis resource for virology research. *Nucleic acids research*, pages D593–D598, 2012.

Justyna Sarzynska-Wawer, Aleksander Wawer, Aleksandra Pawlak, Julia Szymanowska, Izabela Stefaniak, Michal Jarkiewicz, and Lukasz Okruszek. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304:114135, 2021.

Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation. In *AAAI conference on artificial intelligence*, 2018.

Lizhen Shi and Bo Chen. A vector representation of dna sequences using locality sensitive hashing. *BioRxiv*, page 726729, 2019.

Ritambhara Singh, Arshdeep Sekhon, et al. Gakco: a fast gapped k-mer string kernel using counting. In *Joint ECML and Knowledge Discovery in Databases*, pages 356–373, 2017.

Zahra Tayebi, Sarwan Ali, and Murray Patterson. Robust representation and efficient feature selection allows for effective clustering of sars-cov-2 variants. *Algorithms*, 14(12):348, 2021.

Asad Ullah, Sarwan Ali, Imdadullah Khan, Muhammad Asad Khan, and Safiullah Faizullah. Effect of analysis window and feature selection on classification of hand movements using emg signal. In *Proceedings of SAI Intelligent Systems Conference*, pages 400–415. Springer, 2020.

Lasitha Uyangodage, Tharindu Ranasinghe, and Hansi Hettiarachchi. Transformers to fight the COVID-19 infodemic. In *NLP for Internet Freedom: Censorship, Disinformation, and Propaganda*, pages 130–135, 2021.

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

S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, 15(3):1–12, 2014.

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