Efficient Sequence Embedding For SARS-CoV-2 Variants Classification

SARWAN ALI (sali85@student.gsu.edu)

➔ Visiting Researcher at Bento's Lab

➔ 2nd Year Ph.D. Student at Georgia State University, Atlanta, GA

Table of Content

Motivation

- In-depth studies of alterations in the spike protein to classify and predict amino acid changes in SARS-CoV-2 are crucial in understanding the immune invasion and host-to-host transmission properties of SARS-CoV-2 and its variants
- Knowledge of mutations and variants will help identify transmission patterns of each variant that will help devise appropriate public health interventions to prevent rapid spread
- This will also help in vaccine design and efficacy

Research Problem

- How can we design a fixed length representation of protein sequences that can enable us to apply sophisticated Classification models on the protein sequences

Real World Applications

- Genomic surveillance: Tracking the spread of pathogens in terms of genomic content
- Real time identification of new and rapidly emerging variants
- Track the spread of known variants in new municipalities, regions, countries and continents

Challenges

- Mutations happen disproportionately in the spike region of the genome
- Since new variants are emerging, not much information is available about these variants
- Generating fixed-length feature vectors from variable length sequences

Previous Work

- Some efforts have been done to perform classification of SARS-CoV-2 spike sequences
- However, those methods are not generalizable to disproportionality of mutations
- Although they were successful in getting high predictive accuracy, it is not clear if the proposed methods are robust and will give the same predictive performance on different types of data

Kernel Method

- A method that allows us to apply linear classifiers to non-linear problems by mapping non-linear data into a higher-dimensional space
- Kernel-based methods (e.g., SVM) are proven useful for several machine learning (ML) tasks such as sequence classification
- There are two challenges involved with kernel methods in general
  - Kernel computation (requires exponential complexity to compute dot product)
  - scalability (storing n x n matrix in memory is not possible when n, the number of data points, is too large)
- The computational complexity problem can be solved using approximate methods
- The scalability issue remains for the typical kernel methods in general

Our Contribution

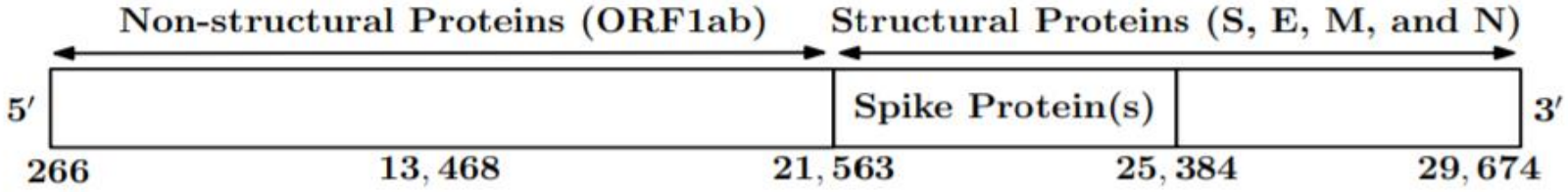
- We propose an efficient embedding method that encompasses the qualities of kernel methods while avoiding both computation and scalability challenges
- To address the computational challenge, given a biological sequence, we take the k-mers, compute a sketch of the sequence and take the dot product, which avoids computing the whole spectrum (frequency count)
- Since our method computes with low dimensional vectors (sketches) for sequences rather than an n x n matrix or full-length spectrum, it can easily be scaled to a large number of sequences, hence addressing the scalability problem
- Our proposed fast and alignment-free spectrum method can be used as input to any distance (e.g., k nearest neighbors) and non-distance (decision tree) based ML methods for classification and clustering tasks

Proposed Approach

- Use Of Spike Sequence
- Using Hashing To Generate Sketches
- Applying Classification Models

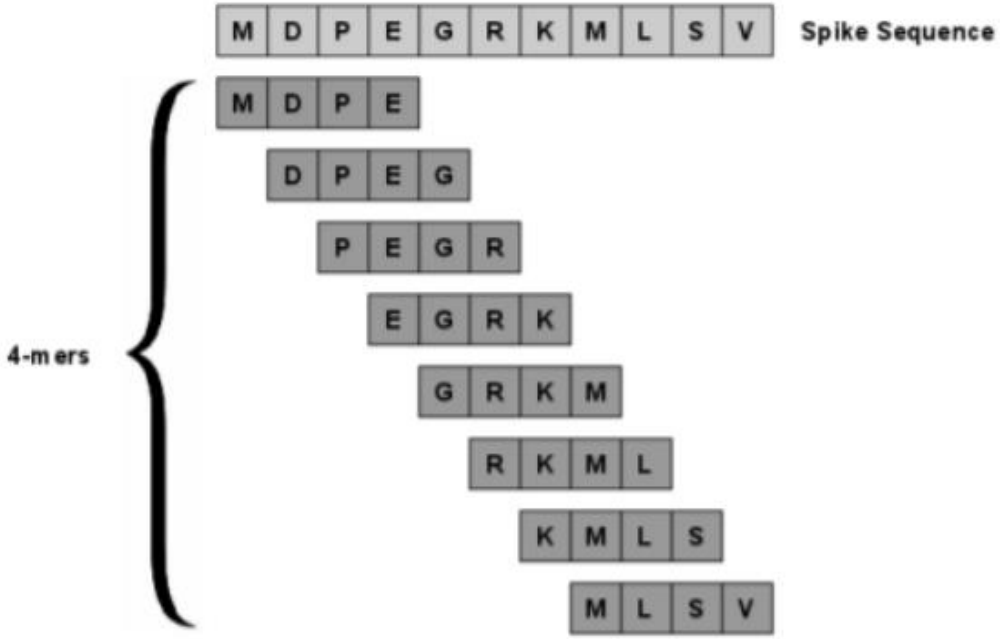
Spike Sequence

- Since the spike protein is the entry point of the virus to the host cell, it is an important characterizing feature of a coronavirus
- the mRNA vaccines (e.g., Pfizer and Moderna) for COVID-19 are designed to target only the SARS-CoV-2 spike protein (unlike traditional vaccines which comprise an entire virome)
- Since the spike region is sufficient to characterize most of the important features of a viral sample, yet is much smaller in length, we focus on an embedding approach tailored to the spike region of the sequences



Using Hashing To Generate Sketches

Build k-mers



Using Hashing To Generate Sketches

Numerical Representation Of k-mers

- Given a kmer and Alphabet Σ => ACDEFGHIKLMNPQRSTVWXY
- For each character in k-mer
  - Find index ***i*** of the character in alphabet
  - Sort the k-mer
  - Find position ***n*** of character in sorted k-mer
  - The final numerical value **v** of the character is **i x $|\Sigma|^n$**
  - Repeat the above process for all characters in k-mers and concat **v** to get **nk-mer**
- Repeat the process for all k-mers

Using Hashing To Generate Sketches

Apply Hash Function

- Initialize Local Sketch **sk** having **m** dimensions
  - m is tunable parameter => 2^{10}
- Compute Hash Value
  - hVal = (a1 x nk-mer + b1) % p) % m
  - a1 => random value between 2 and m-1
  - b1 => random value between 0 and m-1
  - p => any 4 digit prime number
- Increment **sk[hVal]** by 1
- Repeat the process for all k-mers within a sequence to get final sketch
- Normalize **sk** by dividing it by **sum(sk) x h**
  - h => number of hash functions

Using Hashing To Generate Sketches

Using Multiple Hash Function

- We can use multiple hash function **h**
- In that case, we compute normalized **sk** for each hash function separately
- Concat all **sk** together to get final sketch
- Repeat the process for all sequences

Algorithm 1: Peplomer2Vec Computation

Input: Set of Sequences S, integers k, m, p, Σ,h
Output: Φ
function COMPUTESKETCH(S, k, m, p, Σ,h)
    Φ = []
    m = 2^{10} ▷ take integer power of 2
    p=4999 ▷ any 4 digit prime number, $p > m$
    for $s \in S$ **do** ▷ for each sequence
        kmersSet = BUILDKMERS(s,k)
        LSketchArr = [] ▷ Local Sketch Array
    ▷ starting loop for multiple hash functions for each s
        for $hashLoop \leftarrow 1$ to h **do** ▷ # of Hash Func.
            LocalSketch = [0]*m
            a1 = RANDOMINT(2, m-1) ▷ range 2 to m-1
            b1 = RANDOMINT(0, m-1) ▷ range 0 to m-1
            for $kmer \in kmersSet$ **do** ▷ kmers in s
                NumKmer = 0
                for $kmersIndex \in kmer$ **do**
                    charPosition = Σ.index(kmersIndex)
                    sKmer = SORT(kmer)
                    position = sKmer.index(kmersIndex)
                    pos = charPosition $\times$ ($|\Sigma|^{position}$)
                    NumKmer = NumKmer + pos
                hVal = ((a1 * NumKmer + b1) % p) % m
                LocalSketch[hVal] ++
            denum = sum(LocalSketch) $\times$ h
            nLocalSketch = $\frac{LocalSketch}{denum}$ ▷ point-wise divide
            LSketchArr.Concat(nLocalSketch)
        Φ.append(LSketchArr)
    return Φ

Dataset

- Extracted 7000 sequences from GISAID https://gisaid.org/
- 22 variants used as class label

Lineage	Region	Labels	No. Mutation S/Gen.	No. of sequences
B.1.1.7	UK (Galloway et al. 2021)	Alpha	8/17	3369
B.1.617.2	India	Delta	8/17	875
AY.4	India	Delta	-	593
B.1.2	USA	-	-	333
B.1	USA	-	-	292
B.1.177	Spain (Hodcroft et al. 2020)	-	-	243
P.1	Brazil (Naveca et al. 2021)	Gamma	10/21	194
B.1.1	UK		-	163
B.1.429	California	Epsilon	3/5	107
B.1.526	New York (West Jr et al. 2021)	Iota	6/16	104
AY.12	India	Delta	-	101
B.1.160	France	-	-	92
B.1.351	South Africa (Galloway et al. 2021)	Beta	9/21	81
B.1.427	California (Zhang et al. 2021)	Epsilon	3/5	65
B.1.1.214	Japan	-	-	64
B.1.1.519	USA	-	-	56
D.2	Australia	-	-	55
B.1.221	Netherlands	-	-	52
B.1.177.21	Denmark	-	-	47
B.1.258	Germany	-	-	46
B.1.243	USA	-	-	36
R.1	Japan	-	-	32
Total	-	-	-	7000

Table 1: SARS-CoV-2 dataset statistics for 22 variants. The character '-' means that information not available. The fourth column shows the total number of mutations in S (spike/peplomer region) and full length genome (Gen.).

Existing Baseline Models

- Spike2Vec
- PWM2Vec
- String Kernel
- Wasserstein Distance Guided Representation Learning (WDGRL)
- Spaced k-mers

Results

Results

- Changing number of Hash functions

Parameter h	Algo.	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (sec.)
Number of Hash Functions: 1	SVM	0.845	0.848	0.845	0.836	0.675	0.837	4.034
	NB	0.612	0.739	0.612	0.635	0.447	0.731	**0.708**
	MLP	0.819	0.820	0.819	0.813	0.604	0.800	12.754
	KNN	0.806	0.821	0.806	0.801	0.616	0.797	0.965
	RF	0.854	0.855	0.854	0.844	0.680	0.836	1.705
	LR	0.482	0.243	0.482	0.318	0.030	0.500	3.492
	DT	0.841	0.844	0.841	0.833	0.663	0.829	0.327
Number of Hash Functions: 2	SVM	0.848	0.858	0.848	0.841	0.681	0.848	9.801
	NB	0.732	0.776	0.732	0.741	0.555	0.771	1.440
	MLP	0.835	0.825	0.835	0.825	0.622	0.819	13.893
	KNN	0.821	0.818	0.821	0.811	0.616	0.803	1.472
	RF	**0.863**	**0.867**	**0.863**	**0.854**	**0.703**	**0.851**	2.627
	LR	0.500	0.264	0.500	0.333	0.031	0.500	11.907
	DT	0.845	0.856	0.845	0.841	0.683	0.839	0.956
Number of Hash Functions: 3	SVM	0.842	0.845	0.842	0.832	0.678	0.840	14.189
	NB	0.639	0.741	0.639	0.655	0.474	0.736	2.100
	MLP	0.817	0.816	0.817	0.809	0.608	0.802	18.490
	KNN	0.811	0.812	0.811	0.804	0.616	0.797	1.981
	RF	0.852	0.852	0.852	0.841	0.689	0.842	2.966
	LR	0.482	0.233	0.482	0.314	0.030	0.500	8.324
	DT	0.841	0.846	0.841	0.833	0.679	0.837	1.279

Table 2: Classification results showing the effect of changing number of hash functions with k=3 for k-mers in Peplomer2Vec method. Best values are shown in bold.

Results

- Comparison with SOTA

Embeddings	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. 2022)	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, Sekhon 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
Peplomer2Vec (k=3)	SVM	0.848	0.858	0.848	0.841	0.681	0.848	9.801
	NB	0.732	0.776	0.732	0.741	0.555	0.771	1.440
	MLP	0.835	0.825	0.835	0.825	0.622	0.819	13.893
	KNN	0.821	0.818	0.821	0.811	0.616	0.803	1.472
	RF	**0.863**	**0.867**	**0.863**	**0.854**	**0.703**	**0.851**	2.627
	LR	0.500	0.264	0.500	0.333	0.031	0.500	11.907
	DT	0.845	0.856	0.845	0.841	0.683	0.839	0.956

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

Results

- Effect of k for k-mers

Parameter k	Algo.	Acc.	Prec.	Recall	F1 (Weig.)	F1 (Macro)	ROC AUC	Train Time (sec.)
k=3	SVM	0.848	0.858	0.848	0.841	0.681	0.848	9.801
	NB	0.732	0.776	0.732	0.741	0.555	0.771	1.440
	MLP	0.835	0.825	0.835	0.825	0.622	0.819	13.893
	KNN	0.821	0.818	0.821	0.811	0.616	0.803	1.472
	RF	**0.863**	**0.867**	**0.863**	**0.854**	**0.703**	**0.851**	2.627
	LR	0.500	0.264	0.500	0.333	0.031	0.500	11.907
	DT	0.845	0.856	0.845	0.841	0.683	0.839	**0.956**
k=5	SVM	0.850	0.847	0.850	0.836	0.680	0.839	8.827
	NB	0.640	0.715	0.640	0.640	0.463	0.721	1.432
	MLP	0.826	0.823	0.826	0.816	0.629	0.813	13.375
	KNN	0.818	0.824	0.818	0.812	0.621	0.801	1.319
	RF	0.857	0.853	0.857	0.843	0.690	0.842	2.322
	LR	0.483	0.237	0.483	0.315	0.030	0.500	7.219
	DT	0.844	0.840	0.844	0.833	0.667	0.834	0.987
k=7	SVM	0.853	0.854	0.853	0.841	0.691	0.846	9.782
	NB	0.642	0.721	0.642	0.644	0.452	0.721	1.398
	MLP	0.831	0.826	0.831	0.821	0.634	0.818	13.363
	KNN	0.823	0.827	0.823	0.817	0.637	0.816	1.378
	RF	0.856	0.854	0.856	0.844	0.692	0.845	2.644
	LR	0.485	0.236	0.485	0.317	0.030	0.500	7.942
	DT	0.842	0.841	0.842	0.833	0.656	0.830	1.090
k=9	SVM	0.849	0.847	0.849	0.838	0.676	0.836	10.099
	NB	0.644	0.714	0.644	0.651	0.437	0.707	1.540
	MLP	0.833	0.830	0.833	0.825	0.625	0.810	12.938
	KNN	0.820	0.826	0.820	0.815	0.622	0.802	1.385
	RF	0.853	0.852	0.853	0.842	0.679	0.835	2.634
	LR	0.485	0.236	0.485	0.317	0.030	0.500	8.140
	DT	0.836	0.836	0.836	0.828	0.647	0.821	1.127

Table 4: Classification results showing the effect of k for k-mers with $h = 2$ for Peplomer2Vec. Best values are shown in bold.

Results

- Embedding Generation time

Embeddings	Runtime (Sec.)
Spike2Vec (Ali and Patterson 2021)	354.061
PWM2Vec (Ali et al. 2022)	163.257
String Approx. (Farhan et al. 2017)	2292.245
WDGRL (Shen et al. 2018)	438.188
Spaced k-mers (Singh, Sekhon et al. 2017)	12901.808
Peplomer2Vec	**47.401**
% Improv. of Peplomer2Vec from PWM2Vec	70.9%
% Improv. of Peplomer2Vec from Spaced k-mers	99.6%

Table 5: Embedding generation runtime for different methods. Best value is shown in bold. The percentage improvement of runtime is also given for Peplomer2Vec.

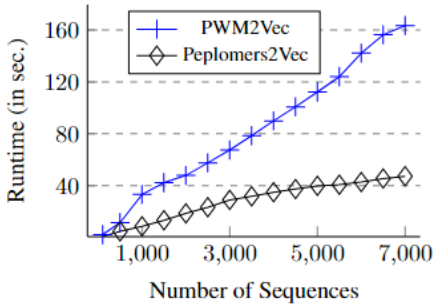


Figure 2: Embedding generation runtime of PWM2Vec and Peplomer2Vec with increasing number of sequences.

Data Visualization

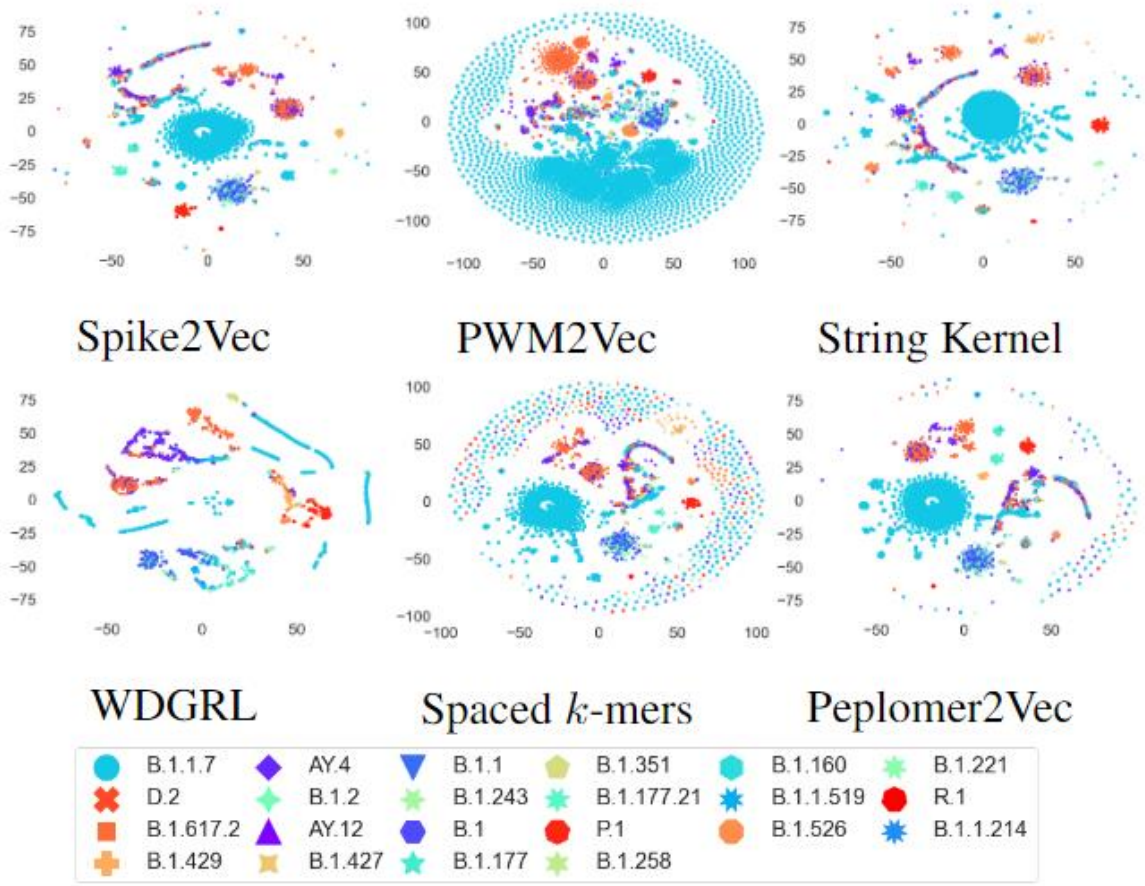
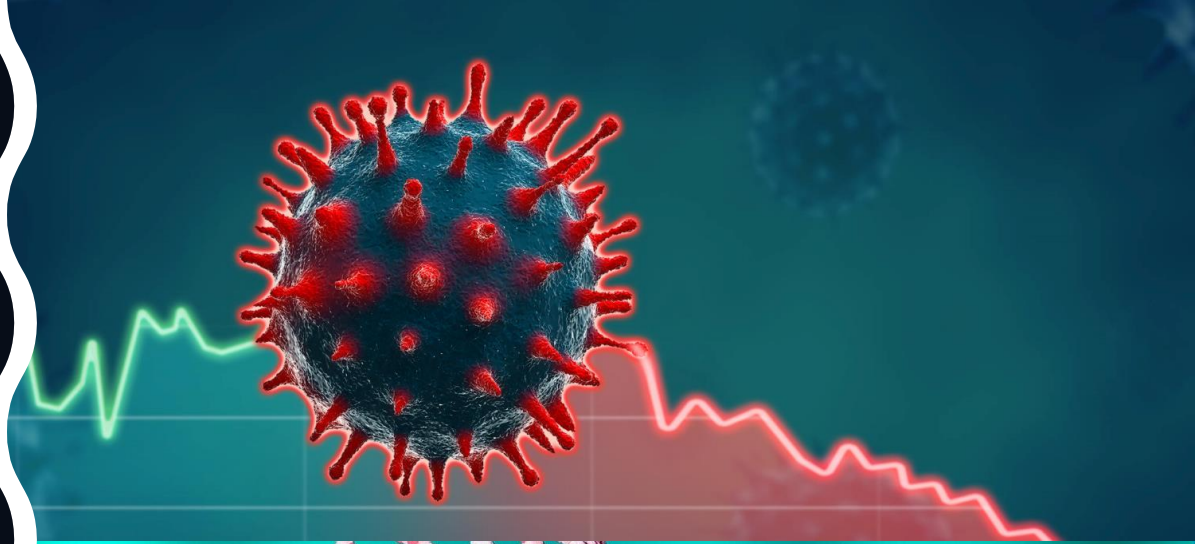
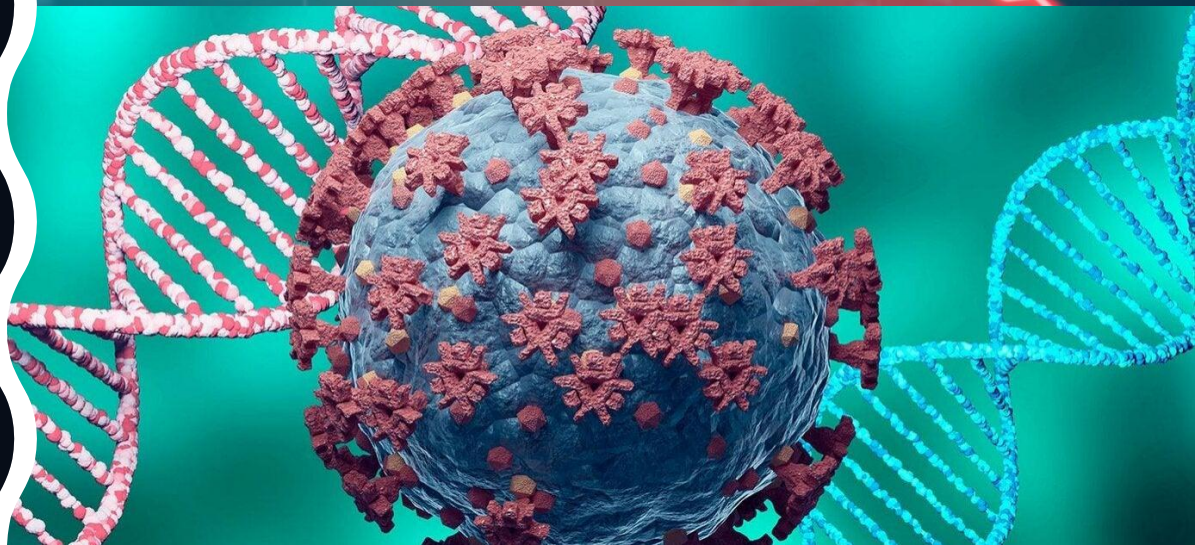


Figure 3: t-SNE plots using different embeddings for 7000 spike sequences. This figure is best seen in color.

Conclusion

- We propose an efficient and alignment-free method to generate sketches for the spike protein sequences using the idea of hashing
- We show that our method is not only generated quickly but also improved the classification results compared to SOTA
- We performed extensive experiments on real-world biological protein sequence data to validate the proposed model using different evaluation metrics





Future Work

- Evaluating the method for larger sets of sequence data (multi-million sequences)
- Applying the proposed method to other virus data such as Zika
- Use Deep Learning models
- Evaluate the robustness

Questions!!

Do Reach Out For Any Questions

- **Email:** sali85@student.gsu.edu
- **Website:** https://sarwanpasha.github.io/
- **Google Scholar:** https://scholar.google.com/citations?user=9dtXSoAAAAAJ&hl=en