MUSIC & GENETIC ALGORITHM

Yaswanth Jagilanka

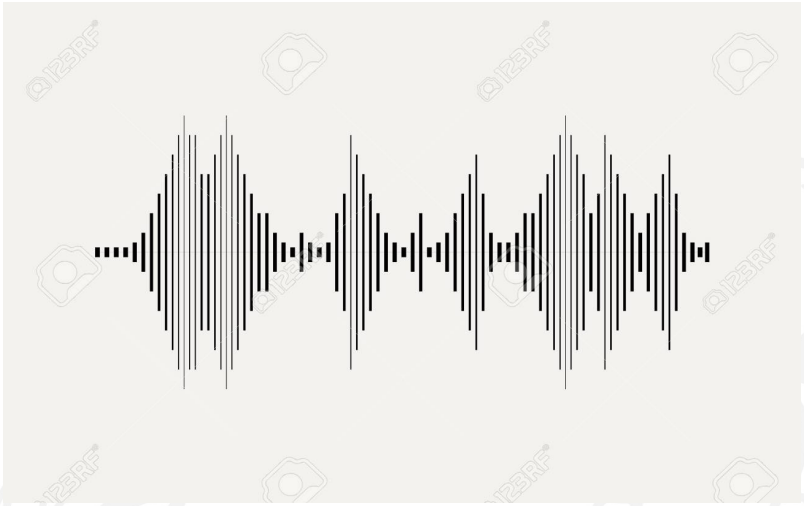
CSE 633 Parallel Algorithms

50365613

University at Buffalo
The Graduate School

Problem Introduction

- Using Genetic Algorithm for Optimal Search in given Music Space
- Finding nearest and best possible music related to a given class in a diverse multiclass music data
- Running this optimal search algorithm in parallel using Open MPI in C++ , by distributing computations over multiple processors



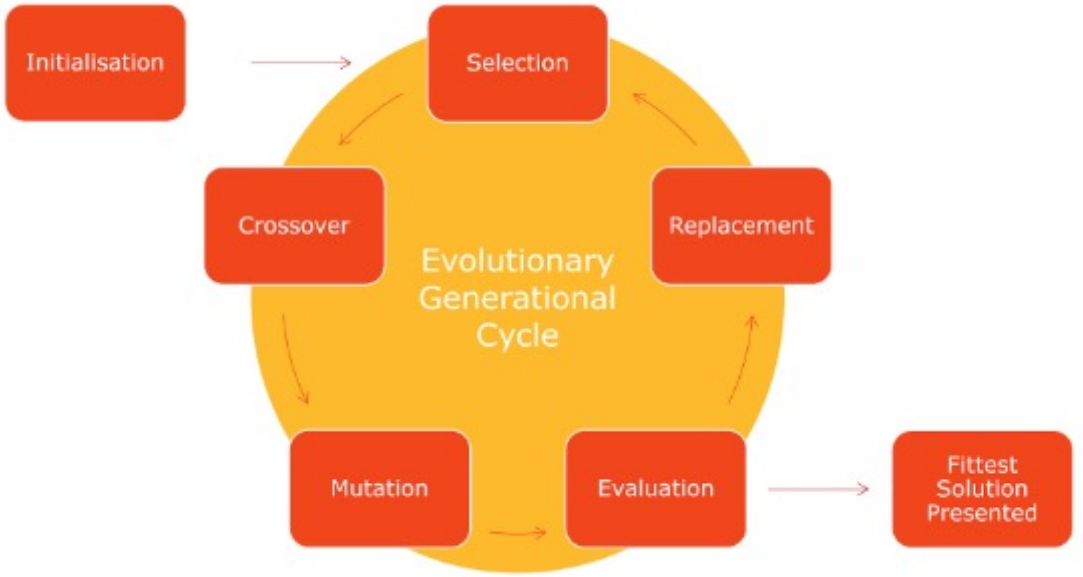
genetic algorithm

Genetic Algorithm(GA)

A genetic algorithm is **a search heuristic that is inspired by Charles Darwin's theory of natural evolution**. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Steps in Genetic Algorithm :

- Initialize Population
- Sample from Population
- Define Fitness Function
- Evaluate Fitness Function
- Select the best possible parents for next generation
- Optional --- Mutation and Crossover



Dataset

For this problem, we have chosen the dataset to be AudioSet : https://research.google.com/audioset/ by Google.

Insights on the Dataset :

- Region separated Data ---- EU , US and Asia
- Labelled and Class Annotated
- Wav files and link to Youtube
- Start and End Time of Audio

Data Preprocessing

Sample for 10s

IBM Max Pooling

Converts to bit data

Each wav file to int vector of 1280

10000 such samples

Class A : Highest Populated Class Type – 250 samples

Class B : Second Highest Populated Class – 250 samples

Population apart from classes extracted 5000+ samples

GA Customization

Initial Population

2D Vector with each row containing 1280 integers from max pooling. Size 5000*1280

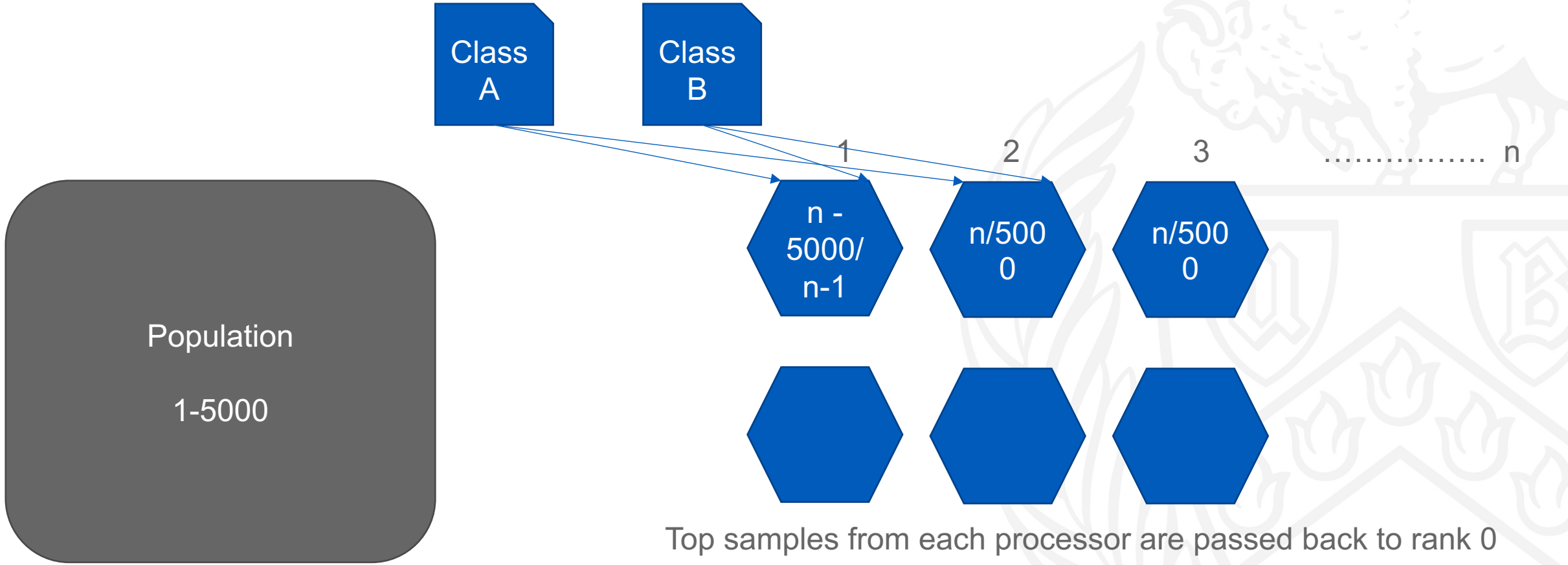
Fitness Function

Cosine Similarity with 250 class samples and mean with highest frequency.

Sampling Fittest Solution

Frequency of the highest cosine similarity values is used to find top 100 samples.

Parallel Approach 1 – Data Distribution



Top samples from each processor are passed back to rank 0

Parallel Approach 1 – Results Discussion

Execution Time

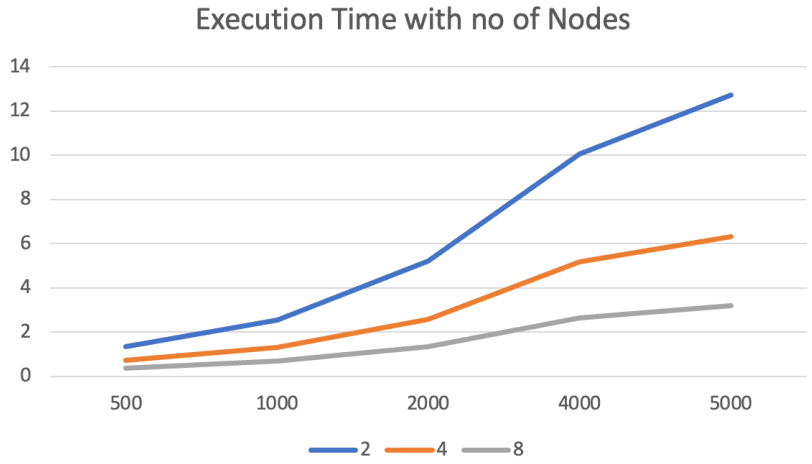
	Processor Size			
Population Size	**2**	**4**	**8**	**16**
500	1.32207	0.704074	0.371306	Failed
1000	2.55295	1.31019	0.680622	Failed
2000	5.20235	2.56478	1.3227	Failed
4000	10.0704	5.16431	2.64234	Failed
5000	12.723	6.32766	3.2056	Failed

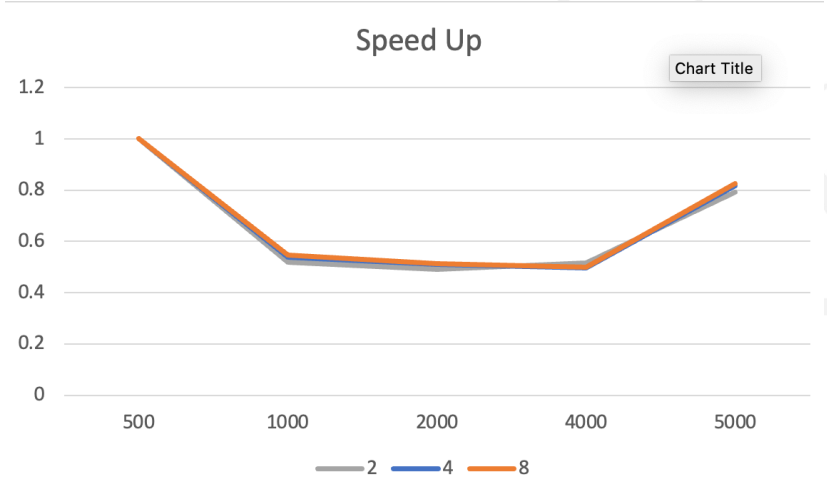
Parallel Approach 1 – Results Discussion

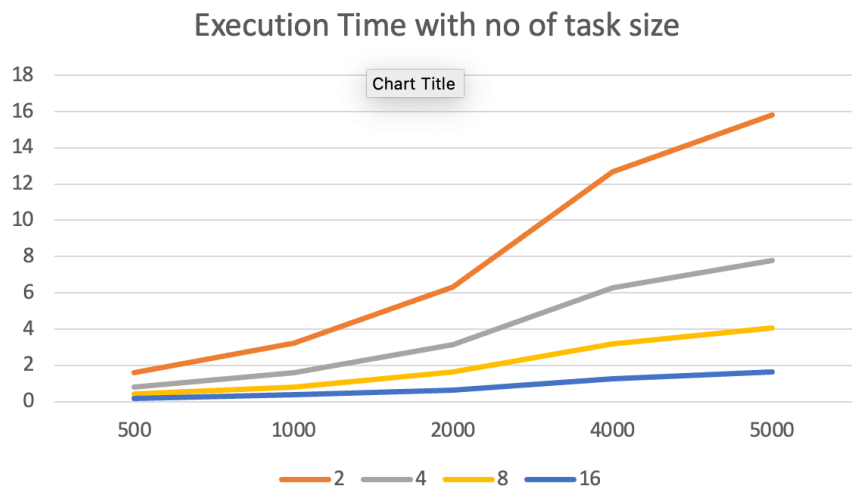
	Speed Up/Processor Size			
Population Size	**2**	**4**	**8**	**16**
500	1	1	1	#VALUE!
1000	0.5178597	0.5373831	0.5455392	#VALUE!
2000	0.4907302	0.5108391	0.5145702	#VALUE!
4000	0.5165981	0.4966356	0.500579	#VALUE!
5000	0.7915114	0.8161485	0.8242887	#VALUE!

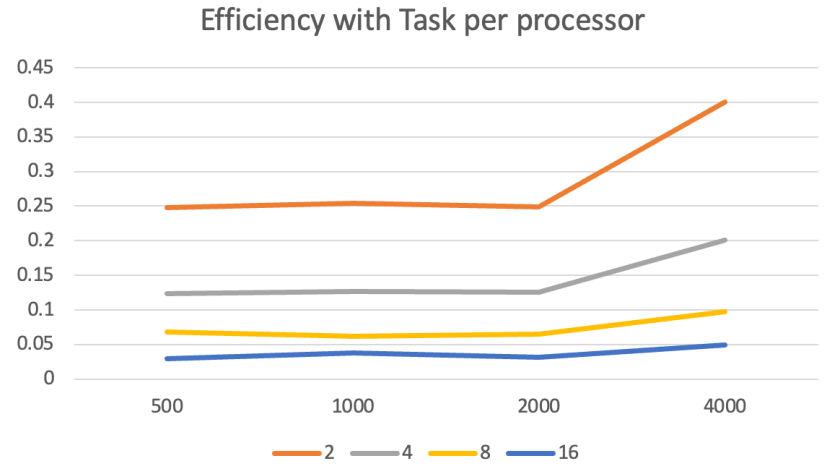
	Efficiency/Processor Size			
Population Size	**2**	**4**	**8**	**16**
500	1	1	1	Failed
1000	0.2589299	0.1343458	0.0681924	Failed
2000	0.2453651	0.1277098	0.0643213	Failed
4000	0.2582991	0.1241589	0.0625724	Failed
5000	0.3957557	0.2040371	0.1030361	Failed

Parallel Approach 1 – Results Discussion









Node vs Task Comparison

Segmentation fault because of smaller data size

	Processor Size			
Population Size	**2**	**4**	**8**	**16**
500	1.32207	0.704074	0.371306	Failed
1000	2.55295	1.31019	0.680622	Failed
2000	5.20235	2.56478	1.3227	Failed
4000	10.0704	5.16431	2.64234	Failed
5000	12.723	6.32766	3.2056	Failed

	Task size per Processor			
Population Size	**2**	**4**	**8**	**16**
500	1.58998	0.788665	0.445013	0.181297
1000	3.20975	1.59115	0.815808	0.378399
2000	6.30824	3.14851	1.64692	0.634747
4000	12.6856	6.25722	3.16547	1.27611
5000	15.8262	7.80561	4.06794	1.62899

	Difference in Task vs Nodes			
Population Size	**2**	**4**	**8**	**16**
500	-0.26791	-0.084591	-0.073707	#VALUE!
1000	-0.6568	-0.28096	-0.135186	#VALUE!
2000	-1.10589	-0.58373	-0.32422	#VALUE!
4000	-2.6152	-1.09291	-0.52313	#VALUE!
5000	-3.1032	-1.47795	-0.86234	#VALUE!

Parallel Approach 2 – Class Distribution

In this approach, we tend to distributed classes based on their number to the respective rank processor

And then respective samples at each processor are sent back to root for evaluation

In this way we can drill down to top 100 samples with continuous communication to processors

Advantages over Parallel 1 :

- Can work over multiple classes at a time
- Increases efficiency due to even workload across processors

Disadvantages over Parallel 1 :

- Communication Overhead may hit after a point of data size

Parallel Approach 2 – Results Discussion

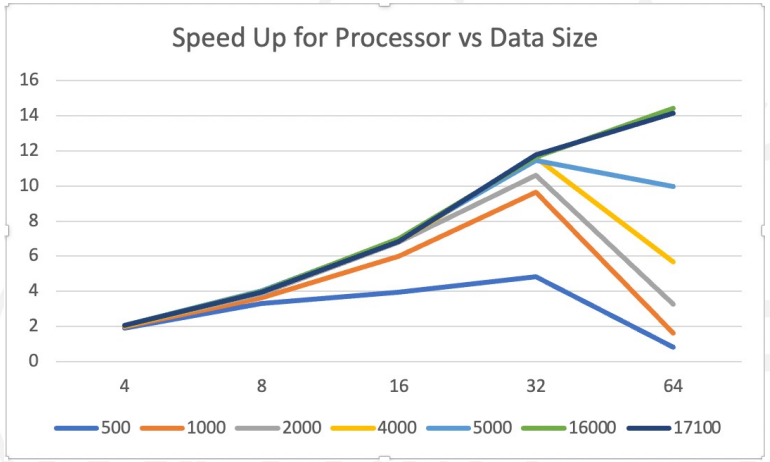
Execution Time

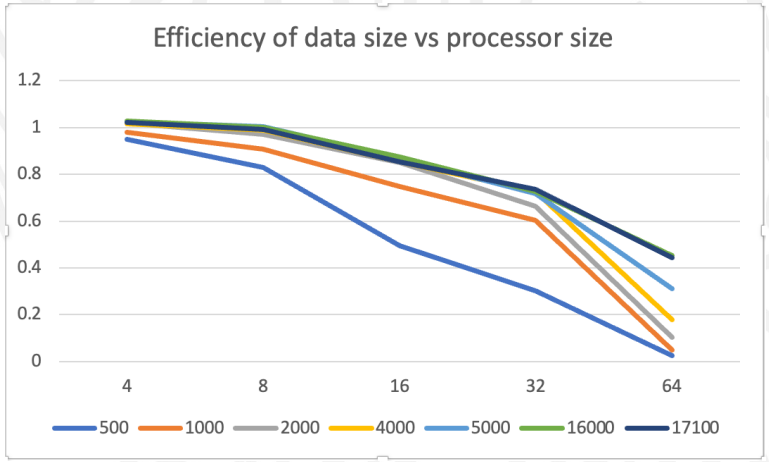
	Processor Size					
Population Size	**2**	**4**	**8**	**16**	**32**	**64**
500	0.7119523	0.3746498	0.2145342	0.1797262	0.1478768	0.8759772
1000	1.3944525	0.7108342	0.3842455	0.2328372	0.144598	0.8674586
2000	2.8046825	1.376445	0.7229608	0.4121856	0.264606	0.8604214
4000	5.57759	2.749682	1.40921	0.8177772	0.4802746	0.9831998
8000	10.99365	5.366728	2.7419633	1.581316	0.9591765	1.104752
16000	22.163925	10.7835	5.5373175	3.173294	1.903405	1.535612
17100	23.566875	11.53818	5.944305	3.45428	2.00071	1.66565

Parallel Approach 2 – Results Discussion

	Speed Up/Processor Size					
Population Size	**2**	**4**	**8**	**16**	**32**	**64**
500	1	1.900314	3.31859559	3.9613159	4.8144959	0.812752
1000	1	1.9617127	3.629066573	5.9889592	9.64365	1.6075148
2000	1	2.0376277	3.879439513	6.8044165	10.599467	3.2596615
4000	1	2.0284491	3.957955166	6.8204274	11.613335	5.6728958
5000	1	2.0484828	4.009408101	6.9522157	11.461551	9.9512379
16000	1	2.0553554	4.002646588	6.9845167	11.644356	14.433285
17100	1	2.0425123	3.96461403	6.8225144	11.779256	14.148756

	Efficiency/Processor Size					
Population Size	**2**	**4**	**8**	**16**	**32**	**64**
500	1	0.950157	0.829648897	0.4951645	0.300906	0.0253985
1000	1	0.9808564	0.907266643	0.7486199	0.6027281	0.0502348
2000	1	1.0188139	0.969859878	0.8505521	0.6624667	0.1018644
4000	1	1.0142246	0.989488792	0.8525534	0.7258335	0.177278
5000	1	1.0242414	1.002352025	0.869027	0.7163469	0.3109762
16000	1	1.0276777	1.000661647	0.8730646	0.7277722	0.4510401
17100	1	1.0212562	0.991153507	0.8528143	0.7362035	0.4421486





Further Exploration

- Mutation and Crossover ------ Interesting
- Actual Results Inference

References

- Audioset : https://research.google.com/audioset/
- Code Github : GA_MusicRecommendation
- Genetic Algorithm : https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3#:~:text=A%20genetic%20algorithm%20is%20a,offspring%20of%20the%20next%20generation.
- MPI : https://mpitutorial.com/tutorials/mpi-introduction/
- IBM Maxpooling : https://github.com/IBM/MAX-Audio-Classifier
- In progress.

University at Buffalo
The Graduate School

THANK YOU

Questions & Feedback