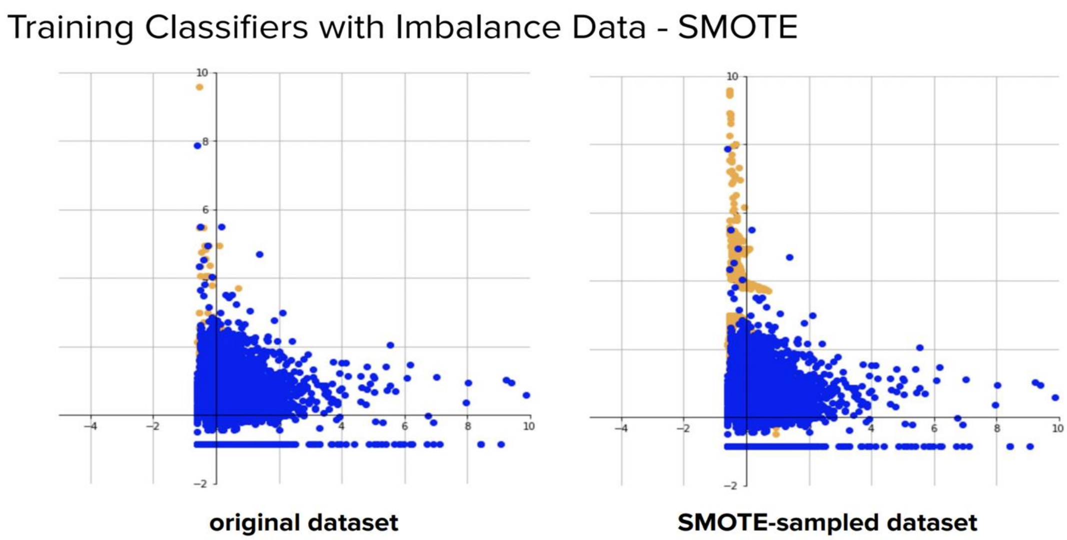
K-Nearest Neighbor on GPU

1. Summary

- In Classification Problems using Machine Learning, a dataset may have multiple classes but their sizes could be in different ratio.
    - For example, let's assume a dataset where we are trying to classify a student into 3 categories:
        1. "Good",
        2. "Better"
        3. "Best"
    - If the training set has many rows for the categories "Good" and "Better" but only few rows which belong to category "Best", then this problem becomes a **class imbalance problem** in machine learning
- A model trained on this imbalanced dataset will only be able to learn "Good" and "Better" classes perfectly but will lack learning the class "Best" significantly.
    - Hence, the predictions also will only belong to classes "Good" and "Better" and will not have "Best" Category. To solve this, I use SMOTE: ***S****ynthetic* ***M****inority* ***O****versampling* ***TE****chnique*
- **SMOTE** - It is a technique to create more data samples for the minority classes which is in this case "BEST" class. It will synthesize more data points in the dataset which belongs to the minority class making sure that the dataset have rows which have equal proportion of the classes



2. **Parallelization Aspects of the Problem**

SMOTE - Synthetic Minority Oversampling Technique

Uses k-nearest neighbor & Vector Distance algorithm to compute synthetic samples which belong to the minority class.

- For this project, I will be **parallelizing the following algorithms using GPU**
    - **k-Nearest Neighbor** is a technique in machine learning to associate a category to a datapoint based on it's k-nearest neighbors
    - **Vector Distance** is a technique to impute distance between 2 points in n-dimension space (Euclidean distance metric for calculating distance between 2 points)

The two different kernels implemented on GPU and which were used for time comparison are:

1) Naive GPU implementation of vector distance and k-NN

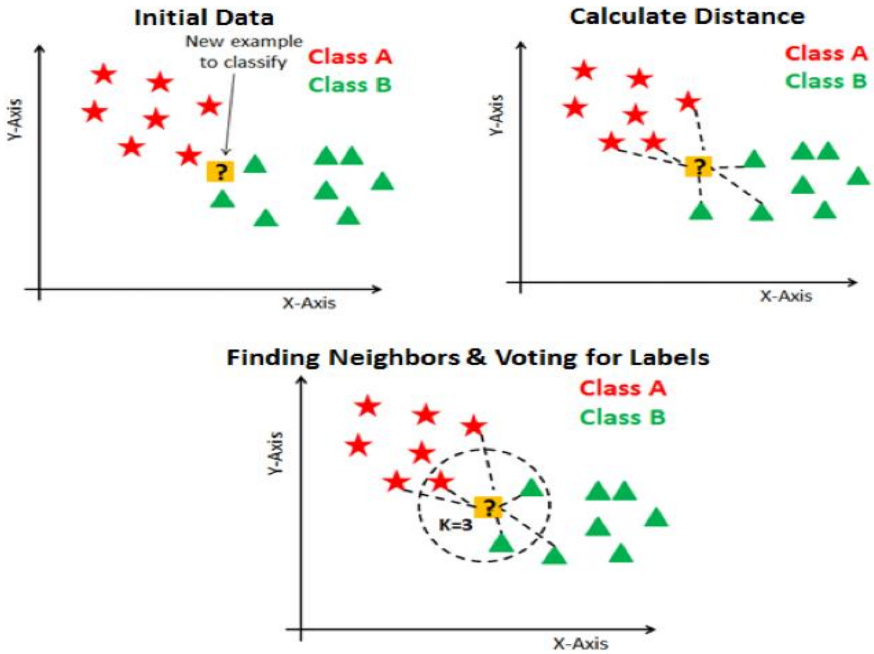
2) GPU implementation of vector distance and k-NN, with element sorting happening on CPU for k-NN.

3. **Smote Technique - Core Algorithms**

K-Nearest Neighbor & Vector Distance

- Here in this problem there are 2 classes, and I need to find the class for the unknown point.
- I started this problem by finding the vector distance between the unknown points with all the samples in the dataset.

● Now I have all the distances measured between this unknown point and all other points in the sample. So, for K=3 (randomly selected), I picked 3 nearest points to this unknown point. I noticed that 2 out of 3 points belong to class B and hence by majority vote, the unknown point was classified as **Class B.**



4. **Smote Technique**

The minority classes in the dataset are synthesized based on KNN and Vector Distance Algorithm.

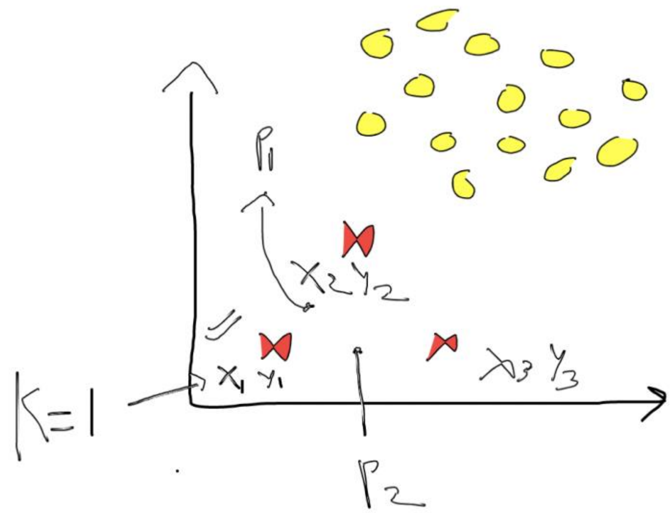
Example:

a. Assume there are 100 million data points which are segregated into 2 classes “Yes” and “No”

b. 90 million data points in class “NO”

c. 10 million data points in class “YES”

d. I can use SMOTE (which internally uses k-Nearest Neighbor and Vector Distance) to synthesize 40 million more samples for the data points which belong to class “YES”



5. **Detailed Description – Why**

- For solving this class imbalance problem, I am using vector distance as the metric, to impute distance of one point with all the other points in the sample. Calculating distance of one point with all the other points and repeating the same for all data points in the sample, makes it of the **order of N^2** which is very slow computation.
- Also, once I have the distance, based on the value of "K", I have to synthesize new data points from each data point. This process is also time taking if done serially as there can be **large number of samples which I want to synthesize.**
- Hence parallelizing both K-NN and Vector Distance will be very efficient in solving the class imbalance problem.

- Calculating distance between all the minority samples - 5 minority samples

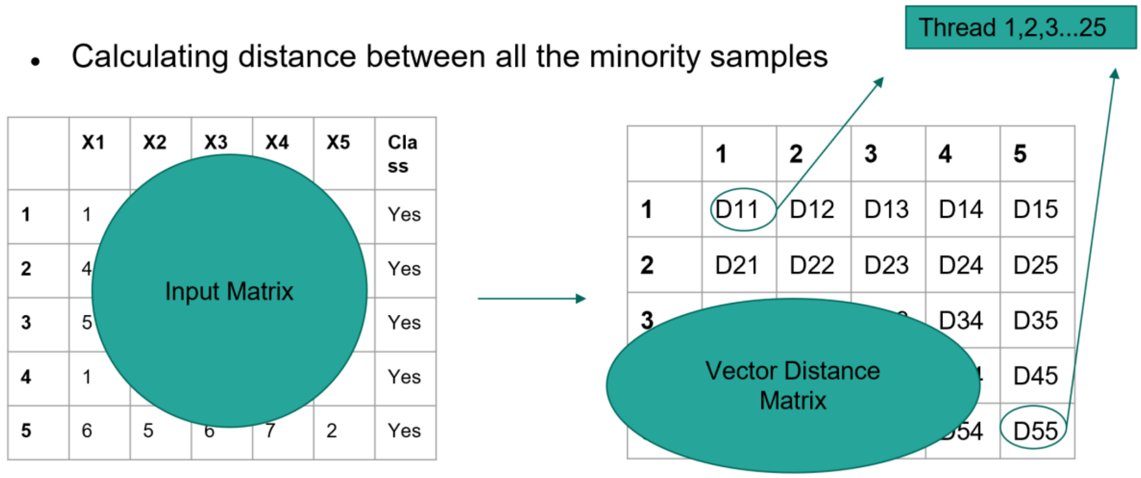
	X1	X2	X3	X4	X5	Class
1	1	3	4	5	4	Yes
2	4	3	2	5	7	Yes
3	5	9	2	8	6	Yes
4	1	2	4	6	7	Yes
5	6	5	6	7	2	Yes

	1	2	3	4	5
1	D11	D12	D13	D14	D15
2	D21	D22	D23	D24	D25
3	D31	D32	D33	D34	D35
4	D41	D42	D43	D44	D45
5	D51	D52	D53	D54	D55

Dij - Distance between ith and jth point

$$D_{12}=\sqrt{((M_{21}-M_{11})^2+(M_{22}-M_{12})^2+(M_{23}-M_{13})^2+(M_{24}-M_{14})^2+(M_{25}-M_{15})^2)}$$

- Calculating distance between all the minority samples



Dij - Distance between ith and jth point

$$D_{12}=\sqrt{((M_{21}-M_{11})^2+(M_{22}-M_{12})^2+(M_{23}-M_{13})^2+(M_{24}-M_{14})^2+(M_{25}-M_{15})^2)}$$

- Now we have the Distances between the Samples. Now we want to find various points based on varying “K”

	1	**2**	**3**	**4**	**5**
1	D11	D12	D13	D14	D15
2	D21	D22	D23	D24	D25
3	D31	D32	D33	D34	D35
4	D41	D42	D43	D44	D45
5	D51	D52	D53	D54	D55

- Now we have the Distances between the Samples. Now we want to find various points based on varying “K”

K=1

So, for k=1, let’s assume following nearest neighbor based on distance
1) For point 1, nearest neighbor is 3
2) For point 2, nearest neighbor is 5
3) For point 3, nearest neighbor is 2
4) For point 4, nearest neighbor is 1
5) For point 5, nearest neighbor is 2

	1	2	3	4	5
1	D11	D12	D13	D14	D15
2	D21	D22	D23	D24	D25
3	D31	D32	D33	D34	D35
4	D41	D42	D43	D44	D45
5	D51	D52	D53	D54	D55

- Now we have the Distances between the Samples. Now we want to find various points based on varying “K”

K=1

So, for k=1, let’s assume following nearest neighbor based on distance
1) For point 1, nearest neighbor is 3
2) For point 2, nearest neighbor is 5
3) For point 3, nearest neighbor is 2
4) For point 4, nearest neighbor is 1
5) For point 5, nearest neighbor is 2

	1	2	3	4	5	
1	D11	D12	D13	D14	D15	→ Point 6
2	D21	D22	D23	D24	D25	→ Point 7
3	D31	D32	D33	D34	D35	→ Point 8
4	D41	D42	D43	D44	D45	→ Point 9
5	D51	D52	D53	D54	D55	→ Point 10

→ Point 6 = Avg of Features for Point 1 & Point 6

- Now we have the Distances between the Samples. Now we want to find various points based on varying “K”

	1	2	3	4	5	
1	D11	D12	D13	D14	D15	→ Point 6
2	D21	D22	D23	D24	D25	
3	D31	D32	D33	D34	D35	
4	D41	D42	D43	D44	D45	
5	D51	D52	D53	D54	D55	

Thread 1 → Point 6

	X1	X2	X3	X4	X5	Class
1	1	3	4	5	4	Yes
3	5	9	2	8	6	Yes

Shared Memory

Result

	X1	X2	X3	X4	X5	Class
6	?	?	?	?	?	Yes

Point 6 = Avg of Features for Point 1 & Point 6

- Now we have the Distances between the Samples. Now we want to find various points based on varying “K”

K=2

So, for k=2, let’s assume following nearest neighbor based on distance

1) For point 1, nearest neighbor is 3 and 5
2) For point 2, nearest neighbor is 5 and 1
3) For point 3, nearest neighbor is 2 and 4
4) For point 4, nearest neighbor is 1 and 5
5) For point 5, nearest neighbor is 2 and 4

	1	2	3	4	5	
1	D11	D12	D13	D14	D15	→ Point 11
2	D21	D22	D23	D24	D25	→ Point 12
3	D31	D32	D33	D34	D35	→ Point 13
4	D41	D42	D43	D44	D45	→ Point 14
5	D51	D52	D53	D54	D55	→ Point 15

Example for Input Size: 5

Vector Distance Output:

Input Matrix →

[[1. 3. 4. 5. 4.]
 [4. 3. 2. 5. 7.]
 [5. 9. 2. 8. 6.]
 [1. 2. 4. 6. 7.]
 [6. 5. 6. 7. 2.]]

Shape of Input Matrix= (5, 5)

[[0. 4.61 .30
3.31 6.40]
[4.61 0. 6.85
3.87 7.28]
[.30 6.85 0.
8.60 7.08]
[3.31 3.87 8.60
0. 8.]
[6.40 7.28 7.08
8. 0.]]

K=2

New Points For the Matrix Through KNN Algorithm

[[1. 2.5 4.
5.5 5.5]
[2.5 2.5 3.
5.5 7.]
[4.5 6. 2.
6.5 6.5]
[1. 2.5 4.
5.5 5.5]
[3.5 4. 5.
6. 3.]
[2.5 3. 3.
5. 5.5]
[2.5 3. 3.
5. 5.5]
[5.5 7. 4.
7.5 4.]
[2.5 2.5 3.
5.5 7.]
[5.5 7. 4.
7.5 4.]]

Shape of Output Matrix = (10, 5)

6. Results

Number of Rows	Vector Distance CPU Naive (ms)	Vector Distance GPU Naive (ms)	k-NN CPU Naive (ms)	k-NN GPU Naive (ms)	k-NN GPU (CPU Sort) (ms)
64	28.07	2.086	8.192	3.922	3.567
256	441.469	2.915	68.781	24.115	48.985
1024	7306.5	31.242	964.638	465.625	859.034
2048	28304.937	124.387	3850.55	1926.440	3532.538
4096	113108.70	461.8450	15462.044	8273.1103	14616.88
8192	463131.68	1413.88	63622.26	39773.44	61138.83

Elapsed time in ms for varying row sizes in input matrix K = 3

The major improvements in elapsed time was observed in vector distance where the parallel algorithm on GPU resulted in approximately 300X improvement for the 8192 rows in the dataset. The K-nearest neighbors algorithm on the GPU resulted in 2X faster elapsed time.

Hardware (Google Colab):

➢ Machine type: n1-standard-4 (4 vCPUs, 15 GB memory)

➢ CPU platform: Intel Haswell

➢ GPUs: 1 x NVIDIA Tesla T4