Subzero Signals: Analyzing Icecube Neutrinos

Big Data Project Report
Rohaan Advani (rna3535), Meghna Savit (ms14622),
Chinmayan Pradeep (cpk286), Shrish Singhal (sks9405)

May 11th 2024

1 Introduction

This project endeavors to harness the potential of big data analytics in the realm of neutrino research through a comprehensive exploration of the "IceCube" dataset. The primary objectives include performing feature engineering to enhance data quality and utility, and evaluating the performance of various frameworks tailored to handling neutrino data. Additionally, the project aims to establish a robust key-value database and a fast querying system to facilitate efficient data retrieval and analysis.
Utilizing advanced clustering techniques, the project seeks to unravel patterns within positional data, enabling the grouping of neutrino events in deep space. Interactive data visualizations will be developed to facilitate in-depth analysis and outlier detection, empowering researchers with actionable insights.
Furthermore, the project will introduce a vector database specifically tailored for angular values, enabling streamlined vector queries. Demonstrating real-time processing capabilities, simulated data streaming will be implemented, showcasing the system's capacity for live data processing.
In a bid to enhance predictive capabilities, regression models and Convolutional Neural Networks (CNN) will be trained on the dataset to predict the point of origin for neutrino events. Ultimately, this project aims to culminate in a multi-faceted, integrated system designed to optimally query, process, visualize, and deploy predictive models on large-scale neutrino data.

2 Overview

2.1 Dataset Description

The dataset for this project is sourced from a Kaggle competition aimed at identifying the directional origins of neutrinos detected by the IceCube neutrino observatory. The primary objective is to enable rapid localization of neutrino sources, facilitating subsequent investigations by traditional telescopes for short-lived events like supernovae or gamma-ray bursts.
The dataset comprises several files, including **[train/test]_meta.parquet**, which contains metadata such as batch IDs, event IDs, and azimuthal and zenith angles representing the direction of neutrinos. Notably, the azimuth angle ranges from 0 to 2π, while the zenith angle ranges from 0 to π. These angles provide crucial directional information, aiding in associating neutrinos with their sources.
The **[train/test]/batch_[n].parquet** files contain detailed event data, each comprising thousands of pulses recorded by the 5160 IceCube photomultiplier sensors. For each pulse, information such as the time, sensor ID, charge (indicative of light intensity), and auxiliary flag (denoting pulse quality) is provided. Notably, the relative time of pulses within an event is significant for analysis, rather than their absolute timestamps.
Additionally, the dataset includes a **sample_submission.parquet** file, serving as a template for submitting predictions, and a **sensor_geometry.csv** file containing the x, y, and z coordinates of IceCube sensors. These coordinates facilitate conversion to azimuthal and zenith angles, crucial for interpreting directional information encoded in the dataset.
Overall, the dataset presents a rich resource for exploring large-scale neutrino data, enabling researchers to develop robust models for accurately predicting the directional origins of neutrino events, thus advancing our understanding of cosmic phenomena.

2.2 Architectural Design

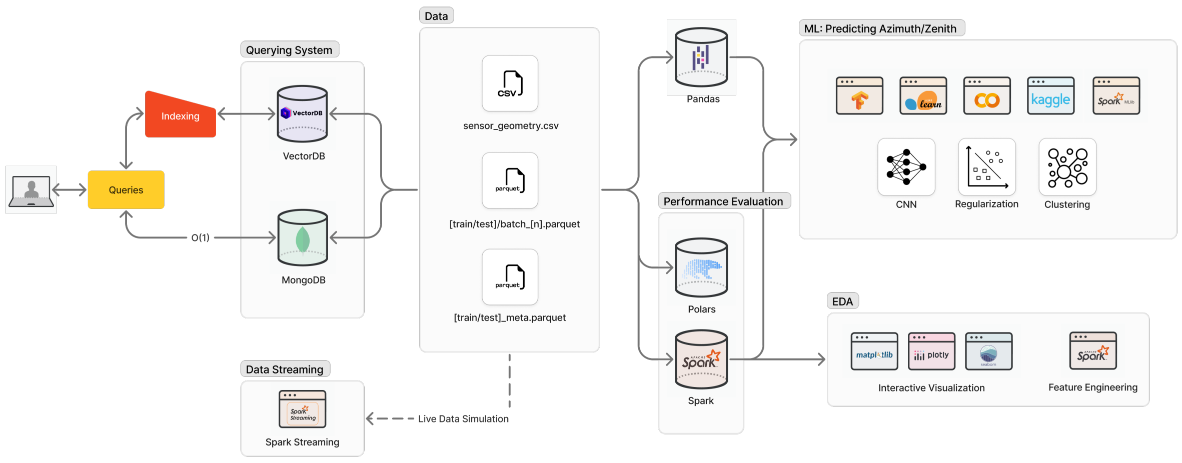


Figure 1: Architecture

We can categorize the system into six distinct segments that aim to achieve the objectives discussed in the introduction. They are as follows:

1. Exploratory Data Analysis
2. Performance Comparison With Polars
3. Data Streaming
4. Interactive Visualization
5. VectorDB Query System
6. Machine Learning

Figure 1 highlights how these systems are interconnected and what frameworks they utilize. The querying system, for example, uses both VectorDB and MongoDB. We also make use of Spark, Polars and Pandas to carry out our EDA, Visualization and Machine Learning tasks. For data streaming, we have made use of Spark Streaming.

3 Implementation

3.1 Exploratory Data Analysis

3.1.1 Methodology and Motivation

PySpark is ideal for EDA due to its scalability, speed, built-in functions, integration with big data tools, ease of use with Python, interactive analysis capabilities, and support for complex analytics tasks like machine learning. It enables efficient exploration of large datasets, offering fast processing and a user-friendly interface for data scientists and analysts.
In the feature engineering phase of the project, exploratory data analysis (EDA) was conducted using PySpark to gain insights into the characteristics of the dataset. The following steps were undertaken:

1. **Calculation of Number of Pulses:**
   The number of pulses per event was determined by subtracting the index of the first pulse from the index of the last pulse within each event.

$$n\_pulses = last\_pulse\_index - first\_pulse\_index \tag{1}$$

2. **Analysis of Unique Pulses and Sensors:**
   The count and percentage of unique pulses and unique sensors identifying each event were calculated. This provided an understanding of the diversity of sensor readings per event.

$$unique\% = 100 * n\_unique\_pulses/n\_pulses \tag{2}$$

3. **Utilization of Aggregate and Group By Functionality:**
   PySpark's aggregate and group by functionality was employed to efficiently analyze and summarize data across events.

4. **Statistical Analysis of Time and Charge:**
   Minimum, maximum, and average time in nanoseconds, as well as minimum, maximum, and average charge in photoelectrons, were computed from the batch data. This statistical analysis provided insights into the temporal and intensity aspects of the events.

5. **Exploration of Auxiliary Flag:**
   The count of true (not digitized) and false (digitized) auxiliary flags for events was determined. This allowed for understanding the proportion of fully digitized data within the dataset.

6. **Calculation of Digitized Data Percentage:**
   The percentage of data that is digitized was computed by dividing the count of false auxiliary flags by the total count of auxiliary flags.

$$dig\% = 100 * false/(true + false) \tag{3}$$

7. **Analysis of Sensors Detecting Events:**
   The percentage of sensors detecting a given event was calculated by dividing the count of sensors detecting an event by the total number of sensors (5160). This analysis provided insights into sensor coverage for each event.

$$sen\% = 100 * n\_unique\_pulses/n\_sensors \tag{4}$$

8. **Implementation of Window Functions:**
   Row_number and rank window functions were applied to sort batches by time and charge data. This facilitated the identification of patterns and trends within the dataset.

9. **Sensor Data Analysis and Conversion:**
   Sensor data was analyzed, and azimuth and zenith angles were calculated using inverse trigonometric functions. Additionally, x, y, and z coordinates of meta data were computed from azimuth and zenith using trigonometric formulas.

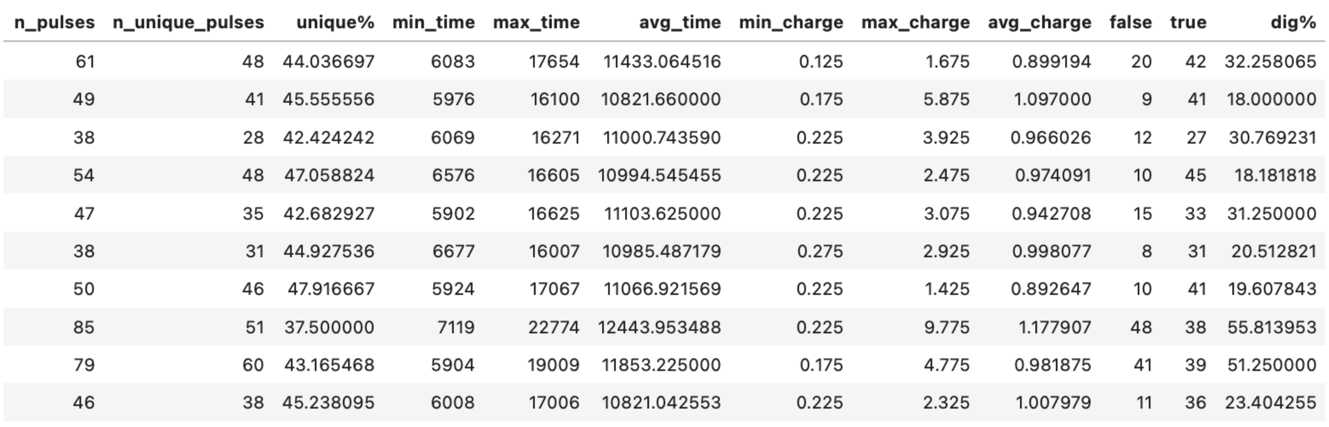
$$x = cos(azimuth) * sin(zenith) \tag{5}$$

$$y = sin(azimuth) * sin(zenith) \tag{6}$$

$$z = cos(zenith) \tag{7}$$

10. **Creation of Reusable Exploration Function:**
    A reusable function was developed to conduct the above exploration for any given batch, enhancing efficiency and scalability of the analysis process.

3.1.2 Results



n_pulses	n_unique_pulses	unique%	min_time	max_time	avg_time	min_charge	max_charge	avg_charge	false	true	dig%
61	48	44.036697	6083	17654	11433.064516	0.125	1.675	0.899194	20	42	32.258065
49	41	45.555556	5976	16100	10821.660000	0.175	5.875	1.097000	9	41	18.000000
38	28	42.424242	6069	16271	11000.743590	0.225	3.925	0.966026	12	27	30.769231
54	48	47.058824	6576	16605	10994.545455	0.225	2.475	0.974091	10	45	18.181818
47	35	42.682927	5902	16625	11103.625000	0.225	3.075	0.942708	15	33	31.250000
38	31	44.927536	6677	16007	10985.487179	0.275	2.925	0.998077	8	31	20.512821
50	46	47.916667	5924	17067	11066.921569	0.225	1.425	0.892647	10	41	19.607843
85	51	37.500000	7119	22774	12443.953488	0.225	9.775	1.177907	48	38	55.813953
79	60	43.165468	5904	19009	11853.225000	0.175	4.775	0.981875	41	39	51.250000
46	38	45.238095	6008	17006	10821.042553	0.225	2.325	1.007979	11	36	23.404255

Figure 2: Explored Dataset Post-Processing with PySpark

3.2 Performance Comparison With Polars

3.2.1 Methodology and Motivation

In our project, we conducted a comparative analysis of PySpark and Polars, two powerful frameworks for data processing and analysis. PySpark is a distributed computing framework built on top of Apache Spark, while Polars is a multi-threaded query engine written in Rust, designed for parallelism and optimized for modern processors.

1. **Parallelism and Performance:**

   Polars leverages vectorized and columnar processing, enabling cache-coherent algorithms and high performance on modern processors. Its parallel execution engine and efficient algorithms allow for significant performance gains compared to other solutions. In our analysis, Polars outperformed PySpark, achieving up to 30x performance gains, particularly in tasks involving reading, analyzing schema, distribution, and displaying large datasets.

2. **Comparative Analysis:**

   We conducted a comprehensive comparative analysis of Polars and PySpark across various operations, including reading, analyzing schema, distribution, and displaying a given dataset. The results of our analysis were very promising, showcasing the superior performance of Polars.

3. **Conclusion:**

   The comparative analysis highlights the strengths of Polars in terms of parallel execution, efficient algorithms, and vectorized processing. Its superior performance, especially in handling large datasets, makes it a compelling choice for data processing tasks. While PySpark remains a popular choice for distributed computing, Polars offers a viable alternative, particularly for applications where speed and efficiency are paramount.

3.2.2 Results

When compared to PySpark, the operations performed on Polars exhibited remarkable efficiency. For instance, while the same set of operations took 73 seconds to complete on PySpark, Polars accomplished them in just 27 seconds. This translates to a significant improvement, with Polars being **2.5 to 3 times faster** than PySpark in analyzing large datasets.

3.3 Data Streaming

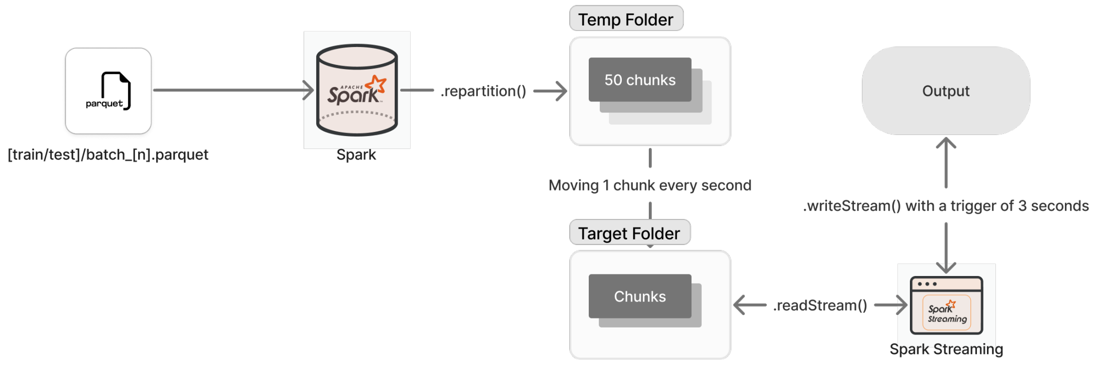

parquet
[train/test]/batch_[n].parquet
Spark
.repartition()
Temp Folder
50 chunks
Moving 1 chunk every second
Target Folder
Chunks
.readStream()
Spark Streaming
.writeStream() with a trigger of 3 seconds
Output

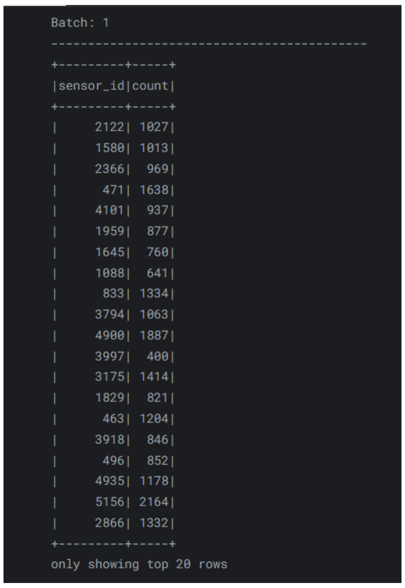
Figure 3: Data Streaming Architecture

3.3.1 Methodology and Motivation

Data Streaming would let us process incoming data in batches, allowing us to work with data dynamically. Since we do not have access to ”live” neutrino data, we have chosen to simulate streaming by chunking our data and periodically moving the chunks into a target folder that processes it. We do this by using Spark Streaming and pointing the *.readStream()* at this target folder. For the purpose of this demonstration we move the chunks in every second and have *.writeStream()* trigger every 3 seconds. On trigger, Spark Streaming processes the data, executes our arbitrary computation and reflects this in the output.

3.3.2 Results

In our simulation, we show a simple groupby and count, the correctness of which we verify by carrying out the same computation on the complete batch file independent of the streaming.



```
Batch: 1
-------------------------------------------
+---------+-----+
|sensor_id|count|
+---------+-----+
|     2122| 1027|
|     1580| 1013|
|     2366|  969|
|      471| 1638|
|     4101|  937|
|     1959|  877|
|     1645|  760|
|     1088|  641|
|      833| 1334|
|     3794| 1063|
|     4900| 1887|
|     3997|  400|
|     3175| 1414|
|     1829|  821|
|      463| 1204|
|     3918|  846|
|      496|  852|
|     4935| 1178|
|     5156| 2164|
|     2866| 1332|
+---------+-----+
only showing top 20 rows
```

Figure 4: .writeStream() Output Mid Computation

3.4 Interactive Visualizations

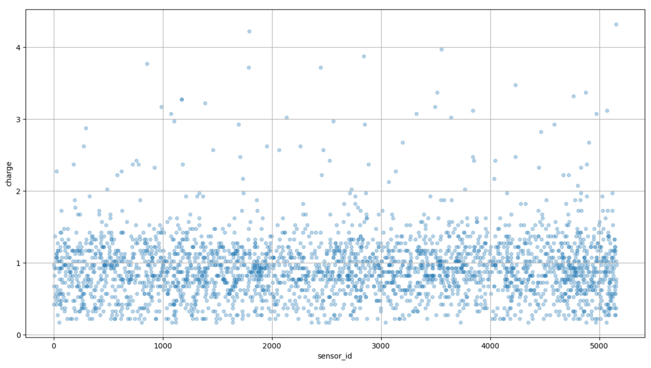


Figure 5: max_charge when aux=True

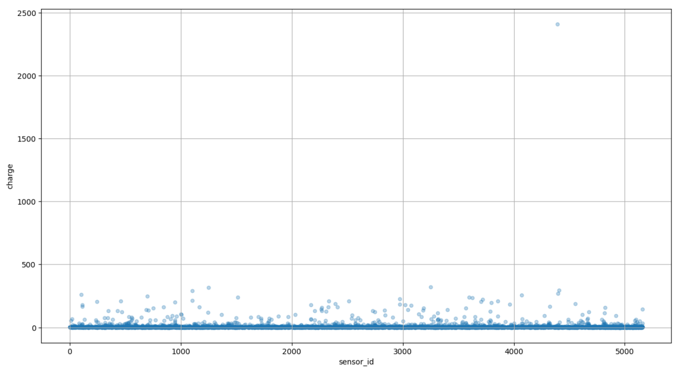


Figure 6: max_charge when aux=False

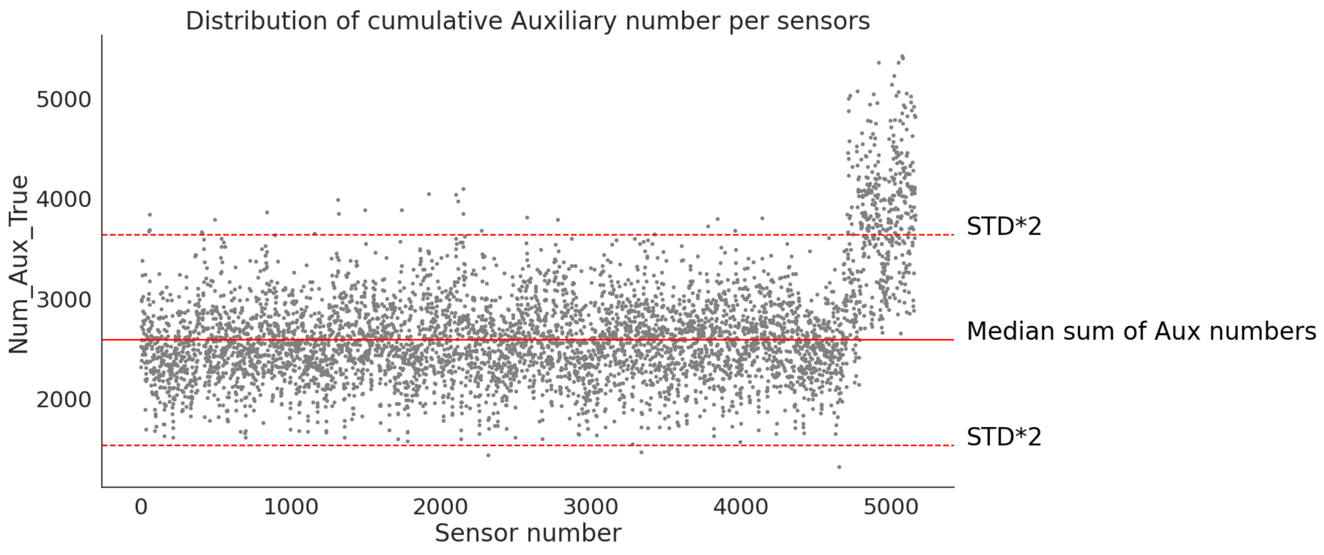


Figure 7: Number of partially digitized pulses detected by the sensors

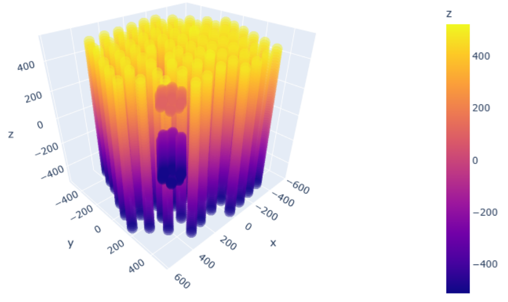


Figure 8: Sensor positions

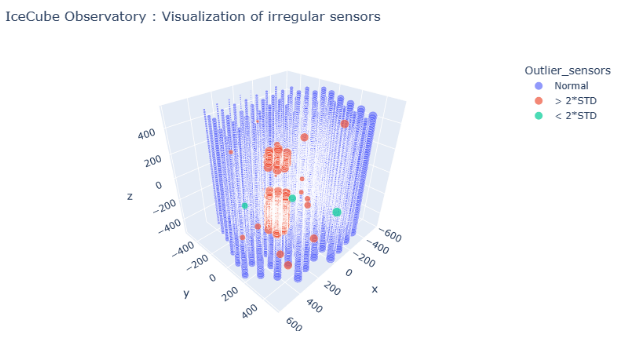


Figure 9: Visualization of noisy sensors

Figures 5 and 6 illustrate that the maximum charge detected in high-quality pulses (aux=False) exhibits more pronounced outliers compared to low-quality pulses (aux=True).

Figure 7 reveals a higher count of partially digitized pulses detected in sensors with IDs greater than 5000. We generated a 3D plot of the sensor positions in Figure 9, highlighting these noisy sensors in red.

3.5 MongoDB Search System

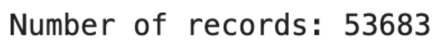
3.5.1 Methodology and Motivation

With data of this size, there is a big case to be made for fast retrieval and processing. We chose to build our fast-querying system using MongoDB. Here are some of the queries we tested:

1. Filtering events based on duration: Useful for analyzing events with longer durations, which may indicate significant or unusual neutrino interactions.
2. Filtering events based on digitization percentage: Helps in selecting subsets of data that meet certain quality standards or criteria for further analysis.
3. Retrieve events with nth largest values for a specified field: Events with the highest charge values may represent extreme or unusual neutrino interactions.
4. Find the events within a given distance from an event: Analyzing events within a specified distance from a reference event can be helpful to contextualize the given event within its spatial neighborhood.
5. Find the distribution of values: Obtain a concise overview of the charge distribution in the dataset, which can serve as a basis for further exploratory analysis or hypothesis testing.

3.5.2 Results

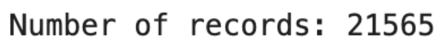
1. **Filtering events based on duration: Query to find events with duration greater than 12000 ns**



```
Number of records: 53683
```

Figure 10: Number of events with duration greater than 12000 ns

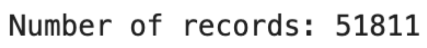
2. **Filtering events based on digitization percentage: Query to find events with digitization greater than 60%**



```
Number of records: 21565
```

Figure 11: Number of events with digitization greater than 60%

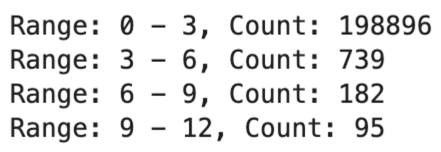
3. **Find the events within a given distance from an event: event_id = 4090686**



```
Number of records: 51811
```

Figure 12: Number of events within a distance of 1 from event_id = 4090686

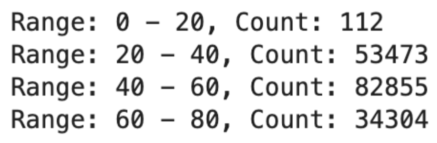
4. **Find distribution of average charge**



```
Range: 0 - 3, Count: 198896
Range: 3 - 6, Count: 739
Range: 6 - 9, Count: 182
Range: 9 - 12, Count: 95
```

Figure 13: Distribution of average charge

5. **Find distribution of number of unique pulses**



```
Range: 0 - 20, Count: 112
Range: 20 - 40, Count: 53473
Range: 40 - 60, Count: 82855
Range: 60 - 80, Count: 34304
```

Figure 14: Distribution of number of unique pulses

3.6 VectorDB Query System

3.6.1 Methodology and Motivation

Vector databases are specialized databases designed for storing and querying high-dimensional vectors or arrays efficiently. Unlike traditional relational databases that primarily deal with tabular data, vector databases are optimized for handling data points represented as vectors, often used in applications such as machine learning, data mining, recommendation systems, and similarity search.

In our project, we utilized vector databases to handle the positional and angular aspects of our dataset. To achieve this, we employed the Faiss library, a Python library specifically designed for efficient similarity search and clustering of large-scale datasets.

We utilized the Faiss library to create a vector database and perform vector queries. Specifically, we created an L2 index over azimuth and zenith angles using Faiss. In Faiss, the IndexFlatL2 index is employed, which computes the L2 (Euclidean) distance between vectors. This index type is well-suited for our dataset, which involves positional and angular data.

Queries Supported by Vector Databases:

1. **K Nearest Neighbors (KNN):**
   This query type enables the identification of the k points closest to a query point based on the Euclidean distance. It is useful for tasks requiring finding the nearest neighbors to a given point in the vector database.

2. **Range Search:**
   Range search allows for the identification of points within a specified radius from a query point in the vector database. It helps in retrieving points that fall within a certain proximity to a given reference point.

3. **Approximate Nearest Neighbors (ANN):**
   ANN search aims to find approximate nearest neighbors to a query point, prioritizing performance over exactness in high-dimensional spaces. Faiss achieves this by using a quantizer to map high-dimensional vectors to lower-dimensional centroids, enhancing search efficiency.

4. **Cosine Similarity Search:**
   This query type computes the cosine of the angle between two vectors in the vector database, providing a measure of similarity irrespective of vector magnitude. It is beneficial for tasks where the directionality of vectors is more relevant than their magnitude.

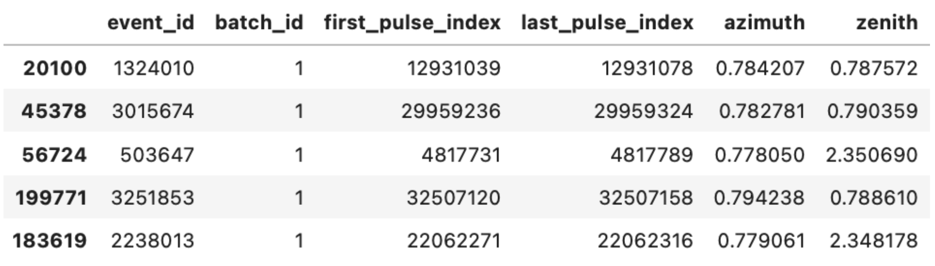
5. **Batch Search:**
   Batch search enables the simultaneous execution of queries for multiple query points in the vector database. This optimization enhances computational efficiency by leveraging parallel processing and optimizing resource utilization.

Vector databases, facilitated by the Faiss library, offer versatile capabilities for handling high-dimensional data, such as positional and angular data in our dataset. Through various query types supported by Faiss, including KNN, range search, ANN, cosine similarity search, and batch search, we were able to efficiently retrieve relevant information from the vector database, facilitating effective analysis and exploration of our dataset.

3.6.2 Results

The test query point for Azimuth and Zenith were [0.5, 0.5] for the below solutions:

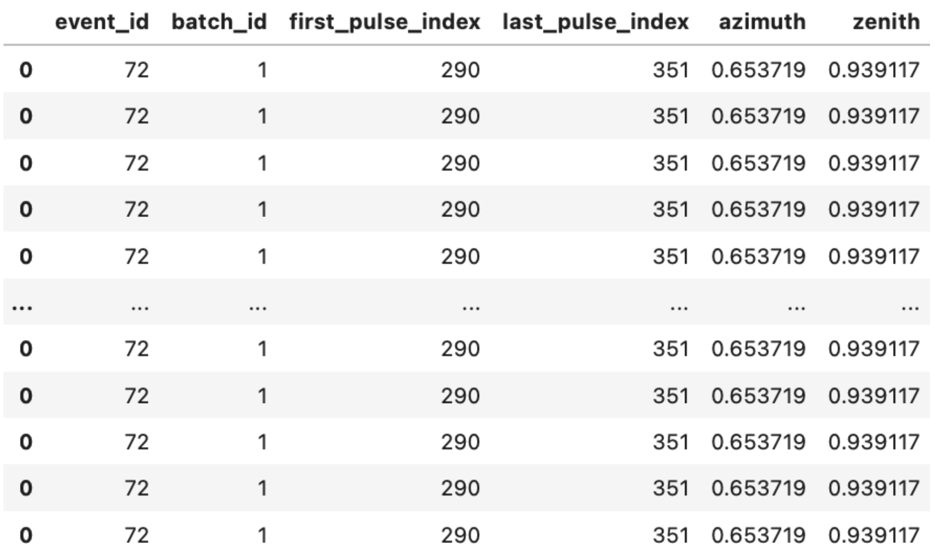
1. **K Nearest Neighbors (KNN): K = 5**



	event_id	batch_id	first_pulse_index	last_pulse_index	azimuth	zenith
20100	1324010	1	12931039	12931078	0.784207	0.787572
45378	3015674	1	29959236	29959324	0.782781	0.790359
56724	503647	1	4817731	4817789	0.778050	2.350690
199771	3251853	1	32507120	32507158	0.794238	0.788610
183619	2238013	1	22062271	22062316	0.779061	2.348178

Figure 15: K Nearest Neighbors (KNN): K = 5

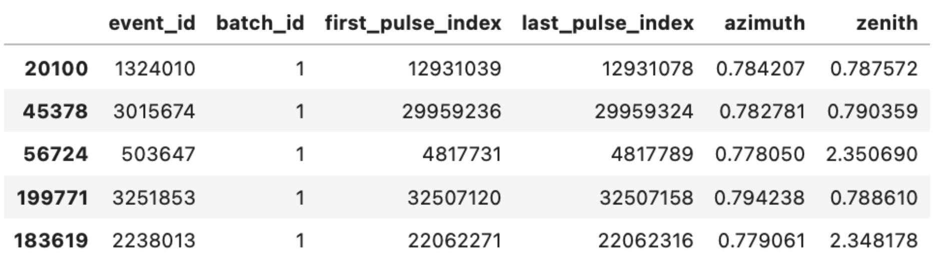
2. **Range Search: Radius = 0.1**



	event_id	batch_id	first_pulse_index	last_pulse_index	azimuth	zenith
0	72	1	290	351	0.653719	0.939117
0	72	1	290	351	0.653719	0.939117
0	72	1	290	351	0.653719	0.939117
0	72	1	290	351	0.653719	0.939117
0	72	1	290	351	0.653719	0.939117
...	...	...	...	...	...	...
0	72	1	290	351	0.653719	0.939117
0	72	1	290	351	0.653719	0.939117
0	72	1	290	351	0.653719	0.939117
0	72	1	290	351	0.653719	0.939117
0	72	1	290	351	0.653719	0.939117

Figure 16: Range Search: Radius = 0.1

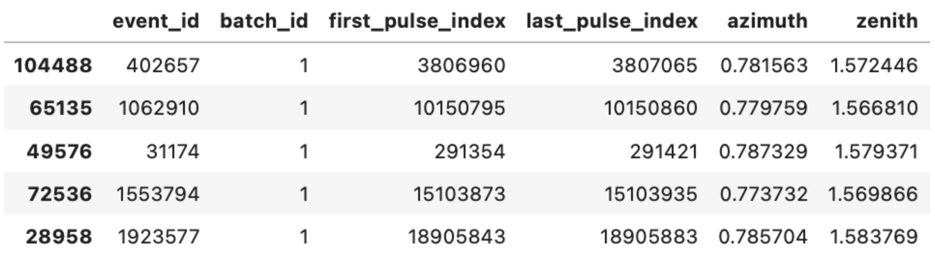
3. **Approximate Nearest Neighbors (ANN): K = 5**



	event_id	batch_id	first_pulse_index	last_pulse_index	azimuth	zenith
20100	1324010	1	12931039	12931078	0.784207	0.787572
45378	3015674	1	29959236	29959324	0.782781	0.790359
56724	503647	1	4817731	4817789	0.778050	2.350690
199771	3251853	1	32507120	32507158	0.794238	0.788610
183619	2238013	1	22062271	22062316	0.779061	2.348178

Figure 17: Approximate Nearest Neighbors (ANN): K = 5

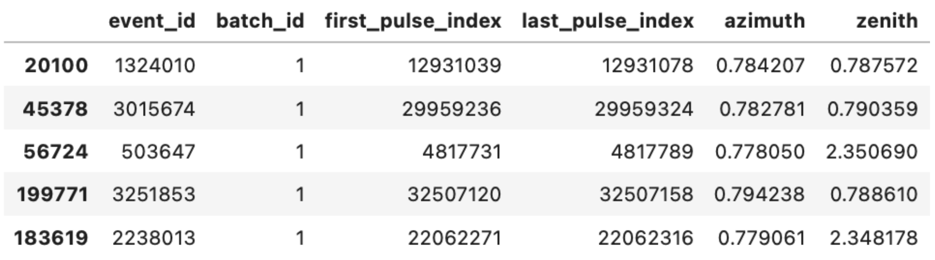
4. **Cosine Similarity Search: K = 5**



	event_id	batch_id	first_pulse_index	last_pulse_index	azimuth	zenith
104488	402657	1	3806960	3807065	0.781563	1.572446
65135	1062910	1	10150795	10150860	0.779759	1.566810
49576	31174	1	291354	291421	0.787329	1.579371
72536	1553794	1	15103873	15103935	0.773732	1.569866
28958	1923577	1	18905843	18905883	0.785704	1.583769

Figure 18: Cosine Similarity Search: K = 5

5. **Batch Search: Query points: [0.5, 0.5], [0.2, 0.8], [0.7, 0.3]**



	event_id	batch_id	first_pulse_index	last_pulse_index	azimuth	zenith
20100	1324010	1	12931039	12931078	0.784207	0.787572
45378	3015674	1	29959236	29959324	0.782781	0.790359
56724	503647	1	4817731	4817789	0.778050	2.350690
199771	3251853	1	32507120	32507158	0.794238	0.788610
183619	2238013	1	22062271	22062316	0.779061	2.348178

Figure 19: Batch Search: Query points: [0.5, 0.5], [0.2, 0.8], [0.7, 0.3]

3.7 Machine Learning

3.7.1 Clustering using Pyspark ML:

In our big data project, we employed PySpark ML to perform clustering analysis on positional data derived from the ”IceCube” dataset. Clustering is a fundamental technique used to group similar data points together, enabling us to identify patterns and structures within the data. The following steps outline our approach to clustering using PySpark ML:

1. **VectorAssembler for Feature Vector Creation:**

   The first step in our clustering analysis involved using the VectorAssembler transformer provided by PySpark ML. This transformer allowed us to create a feature vector containing the positional data for each event calculated during the data exploration phase. By assembling the relevant features into a single vector, we prepared the data for input into the clustering algorithm.

2. **Fit a KMeans Model with Experimental k:**

   Next, we experimented with fitting a KMeans model to the data. The KMeans algorithm is a popular clustering technique that partitions the dataset into k distinct clusters. In our experimental setup, we initially set k to a predefined value (e.g., $k = 3$) and fit the KMeans model to the feature vector created earlier. This step involved iterating over different values of k to find an optimal number of clusters that best capture the underlying structure of the data.

3. **Calculate Silhouette Score:**

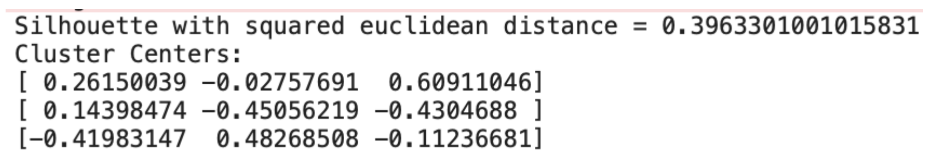
   To assess the quality of the clustering results, we calculated the Silhouette score using the squared Euclidean distance metric. The Silhouette score is a measure of how well each data point fits into its assigned cluster relative to other clusters. It ranges from -1 to 1, where a high score indicates that the data point is well-matched to its own cluster and poorly matched to neighboring clusters. By evaluating the Silhouette score, we gained insights into the cohesion and separation of the clusters, helping us assess the effectiveness of the clustering algorithm.

4. **Extract Cluster Centers:**

   Finally, we extracted the cluster centers from the fitted KMeans model. Cluster centers represent the central tendency of data within each cluster and play a crucial role in evaluating clustering quality and facilitating predictive modeling. By examining the cluster centers, we gained a deeper understanding of the characteristics and properties of each cluster, which in turn enabled us to interpret the results of the clustering analysis and make informed decisions.

Conclusion:

Clustering positional data using PySpark ML has proven to be a valuable technique for grouping neutrino events in deep space. By leveraging clustering algorithms such as KMeans, we were able to identify meaningful patterns and structures within the data, facilitating further analysis and interpretation. Moreover, the clustering results can be utilized for predictive modeling, enabling the assignment of new data points to existing clusters and aiding in the prediction of future events. Overall, clustering analysis has enhanced our understanding of the underlying structure of the "IceCube" dataset and provided valuable insights into the distribution of neutrino events in space.



```
Silhouette with squared euclidean distance = 0.3963301001015831
Cluster Centers:
[ 0.26150039 -0.02757691  0.60911046]
[ 0.14398474 -0.45056219 -0.4304688 ]
[-0.41983147  0.48268508 -0.11236681]
```

Figure 20: Silhouette Score and Cluster Centers

3.7.2 Regression using Pyspark ML:

In our big data project, we utilized PySpark ML to perform regression analysis on the "IceCube" dataset. Regression analysis is a statistical method used to predict the value of a dependent variable based on one or more independent variables. Here, we elaborate on the steps performed for regression analysis:

1. **Feature Engineering:**

   The first step in our regression analysis involved feature engineering to obtain relevant predictors for the regression model. We extracted position and time information relative to the first sensor that receives a pulse for each event. Additionally, we identified the top and bottom 10 sensors by the time that received a pulse. These features were crucial in capturing the spatial and temporal characteristics of the neutrino events.

2. **Selection of Final Columns:**

   From the extracted features, we selected the final columns for regression modeling. These columns included the event_id, sensor's position and time information, azimuth, and zenith. The azimuth and zenith angles are essential parameters in astrophysics, representing the direction from which the neutrino originated. By incorporating these features into the regression model, we aimed to predict the azimuth and zenith angles with high accuracy.

3. **Training a Linear Regression Model:**

   Next, we trained a linear regression model on a subset of the dataset containing close to 20,000 events. The linear regression model is a widely used technique for predicting continuous outcomes by fitting a linear relationship between the independent variables and the dependent variable. In our case, we aimed to predict the azimuth and zenith angles based on the selected features derived from the positional and temporal data.

4. **Evaluation of Results:**

   After training the linear regression model, we evaluated its performance using appropriate metrics. The root mean square error (RMSE) was calculated to assess the accuracy of the predictions. For the azimuth angle, the RMSE was found to be 1.8, while for the zenith angle, the RMSE was 0.69. These RMSE values provide insights into the average error of the model predictions relative to the actual values. Additionally, we noted that the azimuth angle ranges from 0 to 6.28 radians, while the zenith angle ranges from 0 to 3.14 radians.

Conclusion:

Our regression analysis using PySpark ML yielded promising results in predicting the azimuth and zenith angles of neutrino events based on spatial and temporal features. The trained linear regression model demonstrated relatively low RMSE values, indicating good predictive performance. By accurately predicting the direction of neutrino events, our regression model contributes to advancing our understanding of astrophysical phenomena and provides valuable insights for future research in observational astronomy and particle physics.

3.7.3 Regression using Pycaret ML:

In our big data project, we leveraged PyCaret, a Python library for machine learning, to conduct regression analysis on the ”IceCube” dataset. PyCaret provides a convenient interface for quickly testing different regression models and evaluating their performance. Below, we elaborate on the steps performed for regression analysis using PyCaret:

1. **Using PyCaret for Regression Modeling:**

   The initial step in our analysis involved utilizing PyCaret to test various regression models on a subset of the dataset containing 10,000 events. PyCaret simplifies the process of building, comparing, and evaluating machine learning models by automating many of the tedious tasks involved in the workflow.

2. **Testing Different Regression Models:**

   With PyCaret, we were able to test a wide range of regression models with minimal effort. These models include popular algorithms such as Linear Regression, Decision Tree Regression, Random Forest Regression, and Gradient Boosting Regression, among others. By testing multiple models, we aimed to identify the one that best fits the data and yields the most accurate predictions.

3. **Evaluation of Model Performance:**

   After training the regression models, we evaluated their performance using appropriate metrics. In this case, we focused on the root mean square error (RMSE) to assess the accuracy of the predictions. The RMSE measures the average deviation of the predicted values from the actual values, with lower values indicating better predictive performance.

4. **Identifying the Best Performing Model:**

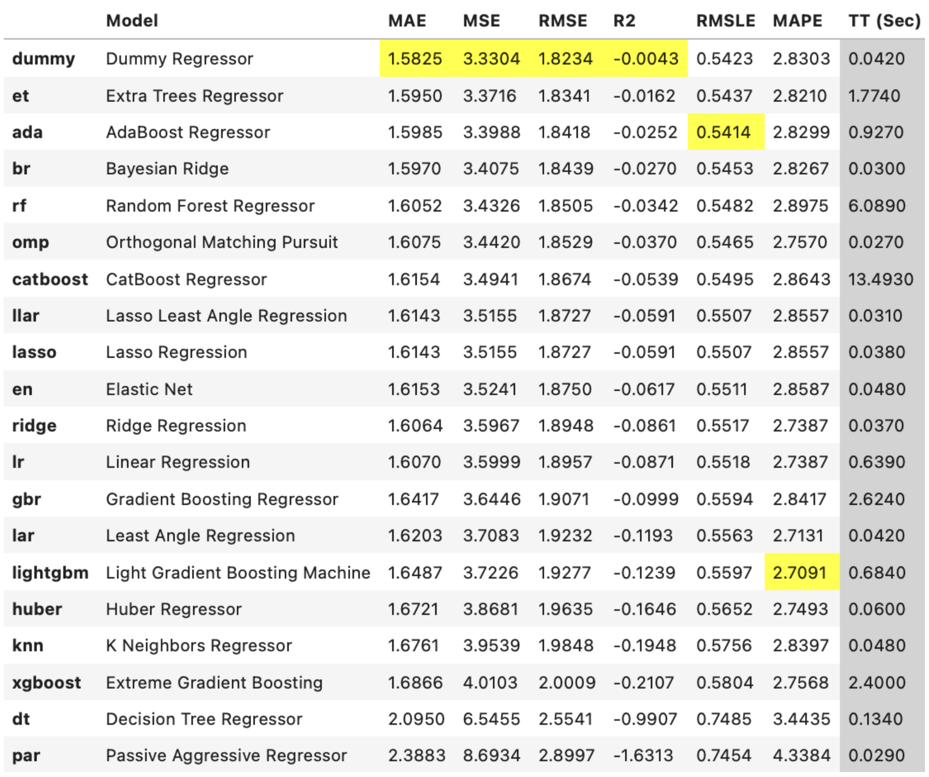
   Among the tested regression models, the Dummy Regressor emerged as the best performer based on the RMSE metrics. The Dummy Regressor is a simple baseline model that predicts the mean or median of the target variable. Despite its simplicity, the Dummy Regressor achieved the lowest RMSE values for both azimuth and zenith angles compared to other regression models tested.

5. **Results:**

   The evaluation results showed that the Dummy Regressor achieved an RMSE of 1.83 for the azimuth angle and 0.75 for the zenith angle. These RMSE values indicate the average deviation of the predicted angles from the true angles in radians. While the Dummy Regressor may not be the most sophisticated regression model, its performance highlights the importance of establishing a baseline for comparison and evaluating more complex models against it.

Conclusion:

In conclusion, our regression analysis using PyCaret provided valuable insights into the performance of different regression models on the ”IceCube” dataset. By leveraging PyCaret’s automation capabilities, we were able to quickly test multiple models and identify the best performing one. While the Dummy Regressor may not be suitable for all scenarios, its performance serves as a benchmark for evaluating more advanced regression models and guiding further analysis. Overall, PyCaret facilitated a streamlined and efficient approach to regression analysis, enabling us to make informed decisions and derive actionable insights from the data.



	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
dummy	Dummy Regressor	1.5825	3.3304	1.8234	-0.0043	0.5423	2.8303	0.0420
et	Extra Trees Regressor	1.5950	3.3716	1.8341	-0.0162	0.5437	2.8210	1.7740
ada	AdaBoost Regressor	1.5985	3.3988	1.8418	-0.0252	0.5414	2.8299	0.9270
br	Bayesian Ridge	1.5970	3.4075	1.8439	-0.0270	0.5453	2.8267	0.0300
rf	Random Forest Regressor	1.6052	3.4326	1.8505	-0.0342	0.5482	2.8975	6.0890
omp	Orthogonal Matching Pursuit	1.6075	3.4420	1.8529	-0.0370	0.5465	2.7570	0.0270
catboost	CatBoost Regressor	1.6154	3.4941	1.8674	-0.0539	0.5495	2.8643	13.4930
llar	Lasso Least Angle Regression	1.6143	3.5155	1.8727	-0.0591	0.5507	2.8557	0.0310
lasso	Lasso Regression	1.6143	3.5155	1.8727	-0.0591	0.5507	2.8557	0.0380
en	Elastic Net	1.6153	3.5241	1.8750	-0.0617	0.5511	2.8587	0.0480
ridge	Ridge Regression	1.6064	3.5967	1.8948	-0.0861	0.5517	2.7387	0.0370
lr	Linear Regression	1.6070	3.5999	1.8957	-0.0871	0.5518	2.7387	0.6390
gbr	Gradient Boosting Regressor	1.6417	3.6446	1.9071	-0.0999	0.5594	2.8417	2.6240
lar	Least Angle Regression	1.6203	3.7083	1.9232	-0.1193	0.5563	2.7131	0.0420
lightgbm	Light Gradient Boosting Machine	1.6487	3.7226	1.9277	-0.1239	0.5597	2.7091	0.6840
huber	Huber Regressor	1.6721	3.8681	1.9635	-0.1646	0.5652	2.7493	0.0600
knn	K Neighbors Regressor	1.6761	3.9539	1.9848	-0.1948	0.5756	2.8397	0.0480
xgboost	Extreme Gradient Boosting	1.6866	4.0103	2.0009	-0.2107	0.5804	2.7568	2.4000
dt	Decision Tree Regressor	2.0950	6.5455	2.5541	-0.9907	0.7485	3.4435	0.1340
par	Passive Aggressive Regressor	2.3883	8.6934	2.8997	-1.6313	0.7454	4.3384	0.0290

Figure 21: Results of Regression using Pycaret

3.7.4 Convolutional Neural Network:

In our big data project, we employed a Convolutional Neural Network (CNN) to address the spatial configurations of the sensors in the ”IceCube” dataset. CNNs are particularly effective for tasks involving spatial data, making them suitable for analyzing the intricate sensor layout in our dataset. Below, we detail the steps involved in training and evaluating the CNN:

1. **CNN Training for Spatial Analysis:**

   Recognizing the importance of spatial information in our dataset, we opted to train a CNN. Unlike traditional regression models, CNNs are capable of capturing spatial patterns in data, making them well-suited for analyzing the arrangement of sensors in the IceCube neutrino observatory.

2. **3D Volume Representation:**

   To input the sensor data into the CNN, we represented it as a 3D volume. Each point within this volume corresponded to the time at which a pulse was received by a sensor. This representation allowed the CNN to process the temporal information from the sensors while preserving their spatial relationships.

3. **CNN Architecture:**

   The CNN architecture comprised two convolutional layers followed by max pooling operations to extract relevant features from the input volume. Subsequently, the output was flattened and passed through a fully connected neural network for further processing. This architecture enabled the CNN to learn hierarchical representations of the sensor data, capturing both local and global patterns.

4. **Evaluation and Performance:**

   After training the CNN, we evaluated its performance using root mean square error (RMSE) metrics. The RMSE values for azimuth and zenith angles were found to be 1.5 and 0.54, respectively. These metrics quantified the average deviation of the predicted angles from the true angles, indicating the accuracy of the CNN's predictions.

5. **Comparative Analysis:**

   Notably, the CNN outperformed all regression models previously tested in our analysis. Its superior performance, particularly in terms of RMSE values, underscores the effectiveness of leveraging CNNs for spatial analysis tasks. By capturing complex spatial patterns inherent in the sensor layout, the CNN demonstrated its utility in improving predictive accuracy for azimuth and zenith angles.

Conclusion:
In conclusion, our utilization of a Convolutional Neural Network (CNN) proved instrumental in addressing the spatial complexities present in the IceCube dataset. Through its ability to extract and learn spatial features from the sensor data, the CNN achieved superior performance compared to traditional regression models. The CNN's capacity to accurately predict azimuth and zenith angles highlights its effectiveness in capturing intricate spatial configurations, reaffirming its value as a powerful tool for spatial analysis tasks in big data projects.

4 Challenges

Throughout our big data project, we encountered various challenges that required creative solutions to overcome. These challenges, along with their corresponding solutions, are elaborated below:

4.1 Uploading Big Data to Jupyterhub:

Challenge: One of the initial hurdles we faced was uploading large datasets to Jupyterhub for analysis. The size of the dataset exceeded the storage capacity and processing capabilities of the local environment.

Solution: To address this challenge, we leveraged the interfaces provided by Kaggle and Google Colab. These platforms offer cloud-based computing resources and support for large datasets, allowing us to upload and analyze the data seamlessly.

4.2 Configuration of MongoDB Atlas to Kaggle:

Challenge: Integrating MongoDB Atlas, a cloud-based database service, with the Kaggle environment posed configuration challenges due to platform compatibility issues.

Solution: To overcome this challenge, we utilized Jupyterhub to create a MongoDB search system. Jupyterhub provided a flexible and customizable environment where we could set up and configure MongoDB Atlas to work seamlessly with our project requirements.

4.3 Unable to Train ML Models on Large Samples due to Platform Restrictions:

Challenge: Platform restrictions imposed limitations on the size of datasets that could be used for training machine learning (ML) models, hindering our ability to leverage large samples for model training.

Solution: To address this challenge, we optimized our data preprocessing pipeline to select representative subsets of the dataset for model training. Additionally, we explored techniques such as data augmentation and feature engineering to enhance the diversity and richness of the training data without significantly increasing its size.

4.4 No Live Data for Streaming:

Challenge: The absence of live data for streaming posed a challenge in demonstrating the real-time processing capabilities of our system.

Solution: To address this challenge, we developed a simulation using random batch data to mimic the behavior of live streaming data. By generating synthetic data streams that emulated the characteristics of real-world data, we were able to showcase the live processing capabilities of our system effectively.

4.5 Scalability Issues with Data Processing:

Challenge: As our dataset grew in size, we encountered scalability issues with data processing. Traditional data processing tools and techniques struggled to handle the increased volume, resulting in slow performance and resource limitations.

Solution: To address scalability issues, we adopted distributed computing frameworks such as Apache Spark and MongoDB. These frameworks enable parallel processing of data across multiple nodes, allowing for efficient handling of large-scale datasets.

4.6 Complexity of Data Integration:

Challenge: Integrating data from disparate sources with different formats and structures posed a significant challenge. The complexity of data integration hindered our ability to consolidate and analyze data effectively.

Solution: To tackle the complexity of data integration, we employed data integration tools and techniques using Pyarrow. These tools facilitate seamless data ingestion, transformation, and integration across heterogeneous data sources.

4.7 Data Quality and Consistency Issues:

Challenge: Ensuring data quality and consistency across disparate datasets proved to be a daunting task. Inaccuracies, inconsistencies, and missing values in the data posed challenges for meaningful analysis and interpretation.

Solution: To address data quality and consistency issues, we implemented data cleansing and preprocessing techniques. This involved identifying and resolving data errors, inconsistencies, and outliers.

5 Future Work

As our big data project progresses, there are several future work possibilities that we can explore to enhance its effectiveness and impact. Below are twelve potential areas for further development:

1. **Further Data Exploration using Polars:**

   Expanding our data exploration efforts using Polars, a multi-threaded query engine, presents an opportunity to gain deeper insights into the dataset. Polars' vectorized and columnar processing capabilities can enhance our ability to analyze and visualize large-scale data efficiently.

2. **Exploring Additional Frameworks for Improved Performance:**

   Continuing our quest for performance optimization, we plan to explore additional frameworks and technologies that offer superior scalability and efficiency. By benchmarking and comparing the performance of different frameworks, such as Dask, Vaex, or Apache Arrow, we can identify the most suitable solution for our specific use case.

3. **Scaling ML Models for the Entire Dataset to Improve RMSE:**

   To improve the accuracy of our machine learning (ML) models, we intend to scale them to accommodate the entire dataset. By training ML models on the full dataset, rather than subsets, we can capture more comprehensive patterns and relationships within the data, leading to improved predictive performance.

4. **Dynamically Predict on Live Data Captured by Spark Streaming:**

   Expanding our predictive capabilities to real-time data streams captured by Spark Streaming represents a significant advancement in our project. By dynamically predicting outcomes and trends on live data, we can provide timely insights and actionable intelligence to stakeholders.

5. **Create MongoDB Search System for Entire Dataset and Implement Web API Interface:**

   Developing a comprehensive MongoDB search system for the entire dataset and implementing a user-friendly Web API interface will enhance accessibility and usability for end-users. By indexing and organizing the dataset in MongoDB, users can efficiently search, retrieve, and analyze observatory data based on various criteria.

6. **Integration of Advanced Analytics and AI Techniques:**

   Exploring advanced analytics techniques, such as deep learning, natural language processing (NLP), and reinforcement learning, can uncover complex patterns and insights within the dataset. By integrating AI-driven approaches, we can enhance predictive modeling, anomaly detection, and pattern recognition capabilities.

7. **Real-time Anomaly Detection and Alerting:**

   Implementing real-time anomaly detection algorithms on streaming data can enable early identification of irregularities, deviations, or unusual patterns in the data. By setting up alerting mechanisms, stakeholders can receive notifications in real-time, allowing for proactive intervention and mitigation of potential issues or threats.

8. **Automated Data Governance and Compliance Frameworks:**

   Developing automated data governance and compliance frameworks ensures adherence to regulatory requirements, data privacy standards, and organizational policies. By implementing robust data governance mechanisms, such as data lineage tracking, access controls, and audit trails, we can ensure data integrity, security, and compliance throughout the data lifecycle.

9. **Optimization of Data Storage and Retrieval Mechanisms:**

   Exploring alternative data storage and retrieval mechanisms, such as data lakes, data warehouses, or NoSQL databases, can improve scalability, performance, and cost-effectiveness. By optimizing data storage and retrieval processes, we can streamline data access, accelerate query performance, and reduce infrastructure costs associated with data management.

10. **Implementation of Data Quality Monitoring and Feedback Loops:**

    Establishing data quality monitoring frameworks enables continuous assessment and improvement of data quality metrics, such as accuracy, completeness, and consistency. By implementing feedback loops and automated data quality checks, we can identify and rectify data quality issues in real-time, ensuring the reliability and trustworthiness of the data for analysis and decision-making.

11. **Integration of Predictive Maintenance and Optimization Techniques:**

    Integrating predictive maintenance and optimization techniques enables proactive maintenance of equipment, infrastructure, or systems based on predictive analytics insights. By leveraging predictive models to forecast equipment failures or performance degradation, organizations can schedule maintenance activities preemptively, minimize downtime, and optimize resource utilization.

These future work possibilities offer opportunities to enhance the sophistication, effectiveness, and impact of the big data project. By embracing these initiatives, organizations can unlock new insights, drive innovation, and achieve greater value from their data-driven endeavors.

6 Conclusions

Based on the comprehensive exploration, challenges faced, solutions found, and future work possibilities outlined throughout the project, we emphasize the following key points:

1. **Achievements and Insights:** The project has made significant strides in creating an integrated system for exploring and analyzing the "IceCube" dataset, leveraging advanced technologies such as Polars, PySpark, MongoDB, VectorDB, Pandas, Streaming, Neural Networks and machine learning algorithms. Through data exploration, feature engineering, and model training, valuable insights have been gleaned regarding neutrino detection, event localization, and predictive modeling.

2. **Overcoming Challenges:** Despite encountering various challenges, such as scalability issues, data integration complexities, and resource constraints, the project team successfully devised innovative solutions and workarounds. By leveraging distributed computing frameworks, data integration tools, and cloud-based resources, the project overcame these obstacles and continued to progress towards its objectives.

3. **Future Directions:** The project's journey does not end here; there are numerous avenues for further exploration and enhancement. Future work possibilities include delving deeper into advanced analytics techniques, implementing real-time anomaly detection, optimizing data storage and retrieval mechanisms, and much more.

4. **Value and Impact:** The project's findings and insights hold significant value for scientific research, astrophysics, and data science communities. By laying the groundwork for an integrated big data processing system, the project has the potential to drive advancements in observational astronomy by making the research process more streamlined.