Classifying Corner-Clipping Events in IceCube

Submitted in partial fulfillment of the requirements for the degree

of Bachelor of Science in Physics, Drexel University, Philadelphia,

PA

Michael Bogert

Advisor: Dr. Naoko Kurahashi-Neilson

5/16/2021

Contents

Chapter 1: Abstract

The IceCube Neutrino Observatory that is located in the South Pole aims to observe highly energetic, astrophysical events through the use of their cubic-kilometer detector submerged in the south pole's ice. The detector records Cherenkov radiation given off by neutrinos interacting with other particles inside the ice, and then reconstructs the path the neutrinos took to reach the detector. The reconstructed path is then used to identify sources of neutrinos, and that is how the collaboration maps the universe. IceCube detects 10^{11} events per year, most of which are background. However, not all of these events are useful. There are events called 'corner-clippers' which, as the name suggests, occur on the edge or outside the detector so that the whole event is not recorded. Generally, the reconstruction for these events cannot be trusted since the whole event might not have been recorded, so the reconstructed path also cannot be trusted and therefore these events must be thrown out. While it is unknown what percentage of all recorded events are corner-clippers, their reconstructions are known to be a significant waste of computational power and time. In addition, alerts are automatically given out for high-energy/interesting events, but if the event is a corner-clipper the alert has to be retracted, which hurts IceCube's goal of trying to do things in real-time. In order to save computational time and power, prevent alerts from being retracted, and stop future corner-clippers from being recorded, I came up with an algorithm to predict whether an event

is a corner-clipper.

Chapter 2: Introduction

Nicknamed ‘the ghost particle’ because of their extremely small size, mass, and electrical neutrality, neutrinos were first discovered in 1956 (1). They are fundamental particles, and one of the most abundant particles in the universe that come from almost everywhere. Some of the most important places that neutrinos come from are high energy sources out in the universe are Gamma Ray Bursts and Active Galactic Nuclei (2) are all sources of high energy neutrinos, which are what IceCube wants to examine.

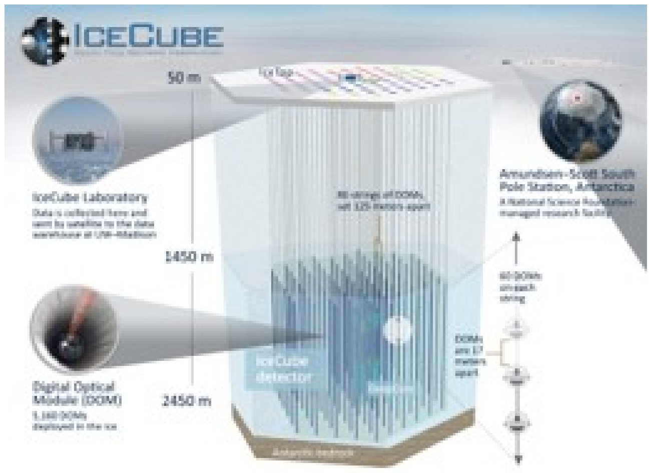

ICECUBE
50 m
IceCube Laboratory
Data is collected here and sent by satellite to the data warehouse at UW–Madison
1450 m
Digital Optical Module (DOM)
5,160 DOMs deployed in the ice
2450 m
IceCube detector
Amundsen–Scott South Pole Station, Antarctica
A National Science Foundation-managed research facility
60 DOMs on each string
DOMs are 17 meters apart

Figure 2.1: Graphic of the IceCube detector, from https://icecube.wisc.edu/science/icecube/

IceCube is an experiment located in the South Pole, where a cubic-kilometer detector is submerged in ice. The detector consists of 86 ‘strings’ with 60 digital optical modules (DOMs) (2) attached to each string. Neutrinos rarely interact, but when they do, Cherenkov radiation is given off by the resultant particle after the interaction. The DOMs in IceCube's detector observe and record this radiation as it traverses through the ice, and then a reconstruction of the event is made. The reconstruction contains information about the event such as the particle's path/trajectory, the number of photo-electrons (PEs) recorded at each DOM, the particle's energy, and the time when individual PEs were detected. Of these events, high energy neutrinos (events with energy in the range of 10^{12} eV) are important to IceCube because their high energy allows for an accurate energy reconstruction to be made. These neutrinos can be used to map the universe since they are not deflected by electric/magnetic fields and are small enough to pass through most matter, so the paths from their origin to IceCube's detector can be considered a straight line.

However, the detector is active all day and every day, and 3,000 detections are made each second. This can lead to 10^{11} events being reconstructed every year, which is a lot of data. Among the data are events called 'corner-clippers'. Corner-clippers are events where an interaction occurs outside the detector, but some of the residual Cherenkov radiation manages to be recorded by some DOMs on the outside or corner of the detector. These events cannot be used since most of the light will not reach the detector, which leads to an inaccurate reconstructions which cannot be trusted and ends up becoming a waste of computational resources. The event shown in Fig. 2.2 is a corner-clipper because the reconstruction of the particle's path is completely vertical, and the detected Cherenkov radiation is concentrated onto only one string. Overall, this does not give a lot of information about the energy of the particle, what type of particle was detected, and the path the particle traveled, meaning this event could be thrown out.

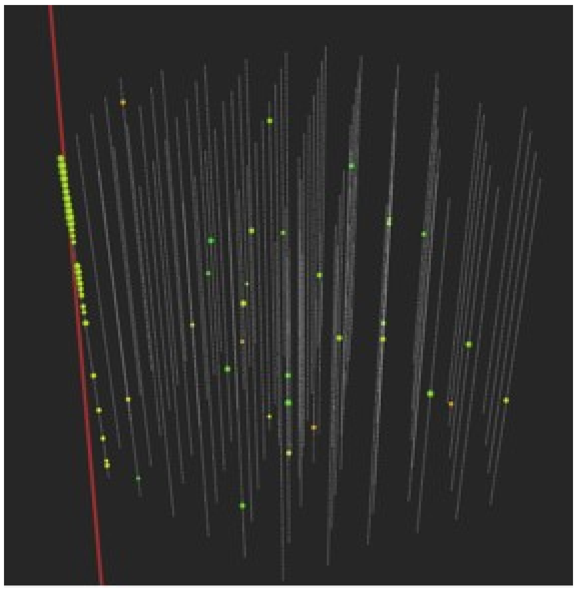


Figure 2.2: Example of a corner-clipping event. The gray lines are the strings inside the ice that make up the detector. The red line is the reconstructed path, and the colored dots indicate where a PE was detected.

The goal of my research is to develop a process that can describe events to a computer and then have it classify events as either corner-clippers or non-corner-clippers so that reconstruction and analysis on corner-clippers can be skipped. However, there is no concrete "definition" for what a corner-clipper is. They are just another type of event, except we know we can't trust these events based on how their reconstruction looks. Creating this process to identify corner-clippers is difficult because it is something done by eye. A computer can't "see" the event and automatically know whether an event is a corner-clipper or not. Instead, the event must be described to it so that it can make it's own decision on whether the event is a corner-clipper or not. In order to describe events to a computer, we can give the computer values for a set of common attributes (or features) that every event has, and then tell it whether or not the event is normal or a corner-clipper. By giving the computer a set of things that can describe an event and then telling it whether the event is a normal event or a corner-clipper, it can learn what a corner-clipper "looks like" based on what the values for the event features are for normal events and for corner-clippers. This is what machine learning is.

Machine learning algorithms are methods that make predictions about a set of data by making connections between attributes of the data. For example, a machine learning algorithm can learn to distinguish handwritten numbers 0-9 by converting the image of the number into an 8x8 matrix that contains grayscale values. The attributes of this dataset would be the grayscale values in each matrix. By comparing thousands, or even hundreds of thousands of images, a pattern can be recognized for each number that allows the algorithm to correctly identify each handwritten number. For the problem I am working on, a classification algorithm is best because I am trying to sort neutrino events into one of two groups: a group of corner-clippers and a group of non-corner-clippers.

Chapter 3: Methods

In order to classify events as corner-clippers or non-clippers, the first thing that had to be done was to establish an outer layer of the detector, since corner-clippers generally occur on the outside/edge of the detector. The outer layer served to define an area of the detector where any PEs detected were considered part of a set of "outer layer of PEs". The outer layer was defined as the outermost strings of the detector. Additionally, an upper and lower set of DOMS were also included from the middle strings, since the event could also clip the top/bottom of the detector.

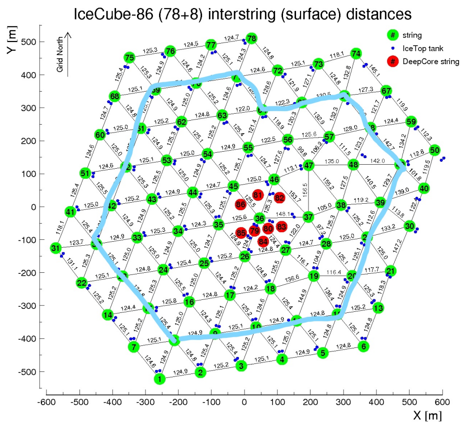


Figure 3.1: Top-down view of the outer layer I defined. Any strings outside the teal line were considered part of the outer layer, while anything on/inside the teal line were not considered in the outer layer.

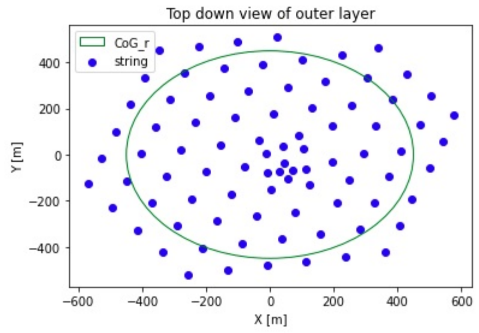
This definition of the outer layer allowed for the first attribute of an event to be defined: the ratio of outer PEs divided by total PEs. Since a corner-clipper is generally recorded on the outer portion of the detector, more light should be detected on the outside than on the inside. So for corner-clippers, the ratio of outer PEs divided by total PEs should be close to 1, as opposed to normal events who should not have a ratio close to 1. If a majority of PEs are detected in the outer layer of the detector, there is a good chance the event could be a corner-clipper.

The second and third attributes came from a thesis done by Tessa Carver (3), who also worked on IceCube. The attributes are Center of Gravity for the event in the Radial direction ($\mathrm{CoG_r}$) and Center of Gravity for the event in the Vertical direction ($\mathrm{CoG_z}$). The Center of Gravity for events is analogous to normal center of gravity, except Center of Gravity can be a misleading label, since the number of PEs detected at a location is used instead of mass. By determining the location of $\mathrm{CoG_r}$ and $\mathrm{CoG_z}$, it allowed us to see if either centers fell close to the outer layer. If so, it is another piece of evidence that the event may be a corner-clipper. The equation for finding the center of gravity is

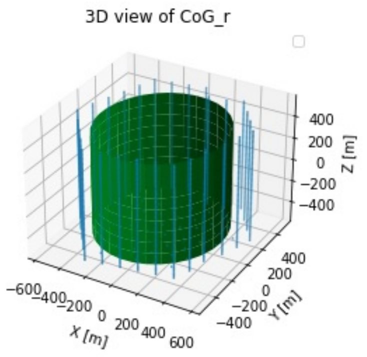
$$CoG(r) = \frac{\sum_{i=0}^{N_{\text{DOMs}}} q_{\text{i}} r_{\text{i}}}{\sum_{i=0}^{N_{\text{DOMs}}} q_{\text{i}}} \tag{3.1}$$

$$r = x^2 + y^2 \tag{3.2}$$

In Equation 1, q_{i} is the number of photo-electrons detected at the i^th DOM. r is the radial distance from the center of the detector, with x and y being the coordinates of the DOM where photo-electrons were detected. To find CoG_{z}, simply replace r with z.



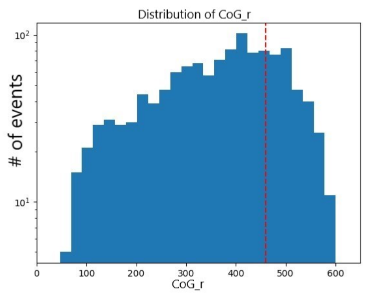
(a) 3.2a



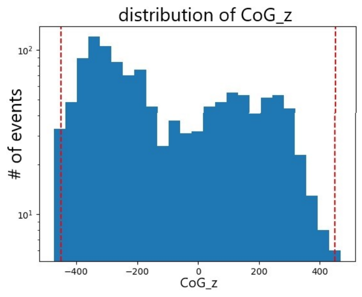
(b) 3.2b

Figure 3.2: A top-down view (left) of CoG_{r} and 3D view (right) of CoG_{r} and CoG_{z} combined. Events with a CoG_{r} value outside the ring in the top-down view or a CoG_{z} value above/below the cylinder in the 3D view had a high chance of being corner-clippers.

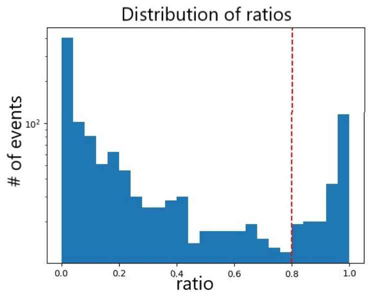
Afterwards, the distributions of all three attributes were plotted from a file containing 1300 events in order to get a general feel of how many events had CoG's outside the outer layer, and how many had a ratio >= 0.8, which was chosen so that a majority of PEs were detected in the outer layer and could potentially be corner-clippers. Below are histograms showing the distributions of $\mathrm{CoG_r}$, $\mathrm{CoG_z}$, and ratios for the 1300 real events detected by IceCube.



(a) 3.3a



(b) 3.3b



(c) 3.3c

Figure 3.3: Distributions of $\mathrm{CoG_r}$, $\mathrm{CoG_z}$, and ratios. The dashed red lines indicate where the cutoff value for potential corner-clippers were (0.8 for ratio, more than 450 m from the center for $\mathrm{CoG_r}$, and 450 m above or below the center of the detector for $\mathrm{CoG_z}$.

As can be seen from the distributions, there was a good number of events who had ratio values above or equal to 0.8, and who had $\mathrm{CoG_r}$ values more than 450 m. However, not many events had $\mathrm{CoG_z}$ values above or below 450 m from the center of the detector. As I hypothesized before, corner-clippers are events that pass by/on the edges of the detector, so in general corner-clippers should have a large $\mathrm{CoG_r}$ value, along with a high ratio of outer PE'S/total PE's. The histograms show that there are a good number of events that fall in the corner-clipper range for their ratio and $\mathrm{CoG_r}$ values, so these events were thought to be corner-clippers. To test this theory, a 2D histogram of $\mathrm{CoG_r}$ values and ratios was made. If corner-clippers do have large $\mathrm{CoG_r}$ and ratio values, a cluster of events should be seen in the top-right corner of the histogram. This turned out to be true, as can be seen from the cluster of events contained in the top-right corner.

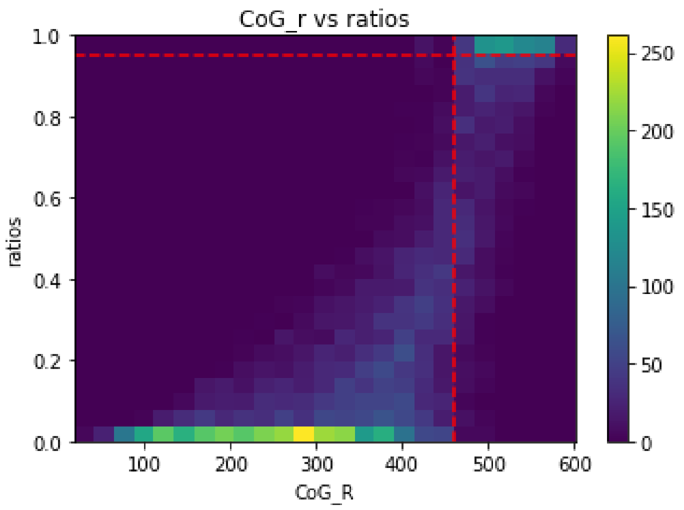
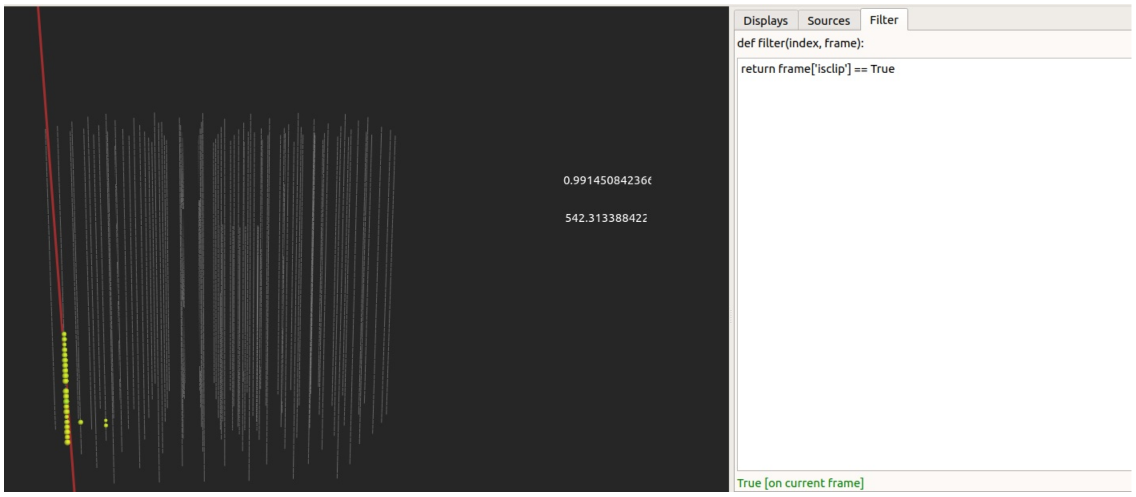


Figure 3.4: 2D histogram of event's ratio values and $\mathrm{CoG_r}$ values. It was hypothesized that a 2D histogram of ratio vs $\mathrm{CoG_r}$ should show a bunch of events in the top-right corner, since corner-clippers should have high ratio and $\mathrm{CoG_r}$ values. This turned out to be true, as can be seen from the bins in the top right enclosed by the two dashed red lines. The colorbar represents the number of events with the given ratio and $\mathrm{CoG_r}$ values, so the fact that there are green bins indicating 150-200 events in the top right corner, those events could be corner-clippers.

It was determined that these events highlighted in the top-right corner of the 2D histogram had high probability to be corner-clippers. In order to see whether or not this idea was correct, the next step was to look at these events by eye and determine whether they were corner-clippers or not. To do this, I used the collaboration's visualization software, Steamshovel.

3.1 Steamshovel

Steamshovel is a visualization software that the IceCube collaboration uses to view the reconstruction of events (5). Steamshovel allows us to open .i3 files, which is the file type that stores information about the event and it's reconstruction. The position of the DOMs, photo-electrons and the time they were detected by the DOMs, total energy of the event, along with the path the particle took are just a few examples of what is contained inside an .i3 file. We are also able to toggle whether certain information appears or not, so that only the relevant information is present. In addition, these files can be edited using a python module, so additional custom tags/values can be added onto each frame. In this case, custom tags for $\mathrm{CoG_r}$, $\mathrm{CoG_z}$, ratios, and a new tag 'isclip' was added to every frame. Isclip is a binary value, with a value of 1 being assigned to events that fell within the top-right corner of the 2D histogram since they would most likely be corner-clippers, and 0 for normal events. Steamshovel also allows for filtering of events based on the values for certain tags, so finding these events in the top-right corner since I could filter for events that had isclip equals 1.



```
def filter(index, frame):
return frame['isclip'] == True
```

```
True [on current frame]
```

Figure 3.5: Window view of steamshovel. Also included in the window are the values for the event's ratio and $\mathrm{CoG_r}$

After filtering and inspecting by eye, the events were obviously corner-clippers, confirming that the upper-right region highlighted in Fig. 3.3 contained corner-clippers. Now that I knew there way to find corner-clippers without machine learning, I could try using machine learning to see whether or not it improved the number of events successfully classified as corner-clippers or not. Since I wanted to use machine learning, I wanted to get more features to describe the events, to see whether I missed a feature that was also very useful in identifying corner-clippers. I added 6 more features: qtot, zenith, energy, ratio2, ratio3, and DOM ratio. Qtot was the total number of PEs seen for the event. Energy is the total energy of the event. Ratio2 and ratio3 are similar to my first ratio, except ratio2 took the two outer-most layer of strings as the outer layer, and ratio3 took the three outer-most. DOM ratio is the ratio of total DOMs in the outer layer that saw detected something during the event divided by the total number of DOMs in the outer layer. Lastly, zenith is the angle that the path of the particle took with respect to the vertical axis.

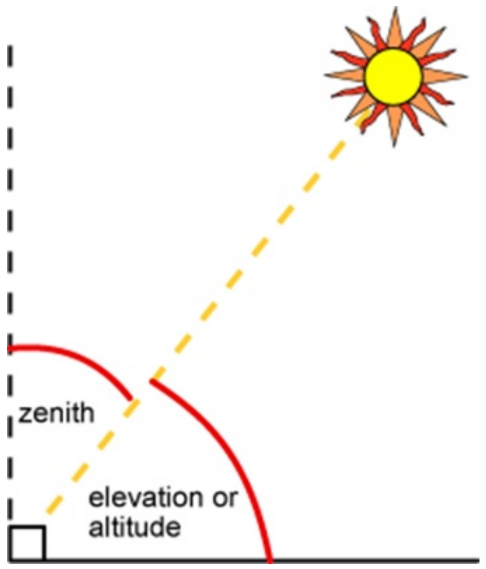


Figure 3.6: Graphic showing the zenith angle, from
https://www.pveducation.org/pvcdrom/properties-of-sunlight/elevation-angle

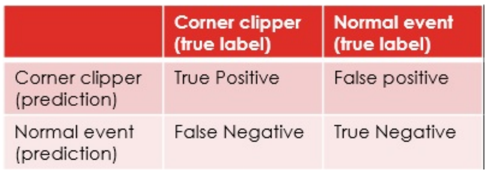
Now that I had all the attributes I wanted to use to describe events, the next step was to try different machine learning algorithms and seeing which worked best. As described in the introduction, by giving the computer a set of attributes and their values and then telling it whether or not an event is a corner-clipper, the algorithm can learn how to classify events. This is known as supervised learning. In a supervised method, the algorithm works on an already well-labeled set of data (4). However, the events I was looking at were unlabeled, which meant that I had to label the data myself. In total, I went through 3,517 events (not fun!), 737 of which I labeled as corner-clippers, which means 1/5 of the events I looked at were corner-clippers. I gave each event a label; 0 for normal events and 1 for corner-clippers. Now that I had labels for every event, I assigned each label to its corresponding event, along with the values for each feature, and began trying different machine learning algorithms. Half of these 3,517 events would be used to train the model (a training set), and then the algorithm would make its predictions based on the other half (a test set), and it's predictions would be compared to the labels I made for the other half to measure how well the models were working. To name a few, I tried K-Nearest Neighbors (KNN), Gaussian Naive Bayes (GaussNB), a decision tree and bagged decision tree, Random Forest (RanFor), Support Vector Classifier (SVC), and MPL Regression (MPLReg).

I tried many different algorithms, but I had to come up with a way to determine which was best. I decided the metrics to determine the best method would be to measure the precision and completeness of each method. Precision can be thought of as the percentage of how many events that were labeled as positive were actually positive (6). In other words, how many events labeled as corner-clippers were actually corner-clippers. Completeness is the percentage of how many positives were successfully labeled as positive out of all positives. In other words, how many corner-clippers were labeled as corner-clippers out of the total number of corner-clippers. The equations for completeness and precision are:

$$precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{3.3}$$

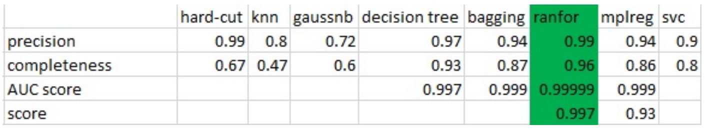
$$precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{3.4}$$

True positives are corner-clippers that are labeled corner-clippers, true negatives are normal events labeled as normal, false positives are normal events labeled as corner-clippers, and false negatives are corner-clippers labeled as normal.



	Corner clipper (true label)	Normal event (true label)
Corner clipper (prediction)	True Positive	False positive
Normal event (prediction)	False Negative	True Negative

(a) 3.7a



	hard-cut	knn	gaussnb	decision tree	bagging	ranfor	mplreg	svc
precision	0.99	0.8	0.72	0.97	0.94	0.99	0.94	0.9
completeness	0.67	0.47	0.6	0.93	0.87	0.96	0.86	0.8
AUC score				0.997	0.999	0.99999	0.999	
score						0.997	0.93	

(b) 3.7b

Figure 3.7: The left table describes what true positives, true negatives, false positives, and false negatives are for corner-clippers. The right table shows the comparisons between multiple algorithm's precision and completeness scores, along with some other metrics.

In the end, Random Forest turned out to perform the best, and is the final algorithm I decided to use for my project.

3.2 Random Forest

Random Forest is a classification algorithm that is based off of decision trees, which is another algorithm I tried. To understand how a Random Forest works, we first have to go over how a decision tree works. A decision tree works by asking a yes or no question, or setting a condition, about the data. Then, depending on the answer, it goes down one branch of the tree, where it is met with either another question or it is placed into a category. This process continues from the very first question all the way down the tree until all events are sorted into their corresponding categories. Below is an example of what a decision tree may look like for my corner clipping project.

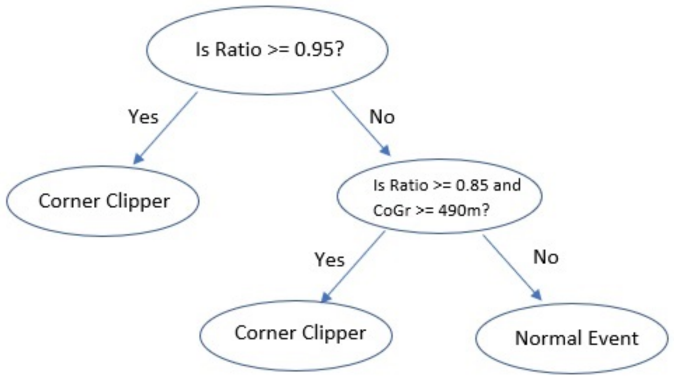
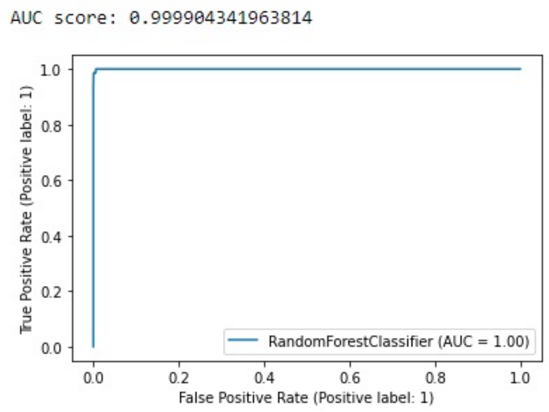


Figure 3.8: Example of how a decision tree would work for my corner clipping project. First, the question of whether an event's ratio is >= 0.95. If yes, the event is called a corner-clipper. If not, it is asked another question: is ratio >= 0.85 and $\mathrm{CoG_r}$ >= 490 m? If yes, the event is called a corner-clipper. If not, it is a normal event.

As mentioned before, Random Forest is a type of algorithm based off of a decision tree. In a Random Forest, multiple decision trees are made, and each tree randomly selects selects how many and which attributes to use from the data. These multiple decision trees make up the forest, and after all the trees are made, they are combined into one tree which becomes the final decision tree (7). This combination of multiple trees, some of which work well and others which don't work well, allows for the algorithm to find out what works and what doesn't and come up with a final result that should encapsulate the best part of each model.

Chapter 4: Results

As can be seen in the figure below, Random Forest performed the best in both precision and completeness out of all algorithms. It also had a very high AUROC score, which stands for Area Under the Receiving Operating Characteristic Curve. The ROC curve and AUC score are both performance metrics that can tell you how well a classifier works (8). The closer the value is to 1, the better the performance.



```
AUC score: 0.999904341963814
```

Figure 4.1: The AUC score and ROC curve for my Random Forest classifier. The AUC score is .9999, which is extremely close to 1, indicating that the algorithm performs extremely well at correctly identifying corner-clippers.

The performance of the Random Forest can also be seen through a comparison of scatter plots of the events with labels done by my eye and the labels done by the Random Forest. Nearly every one of the events that I labeled as corner-clippers are also labeled as corner-clippers by the Random Forest, indicating high accuracy.

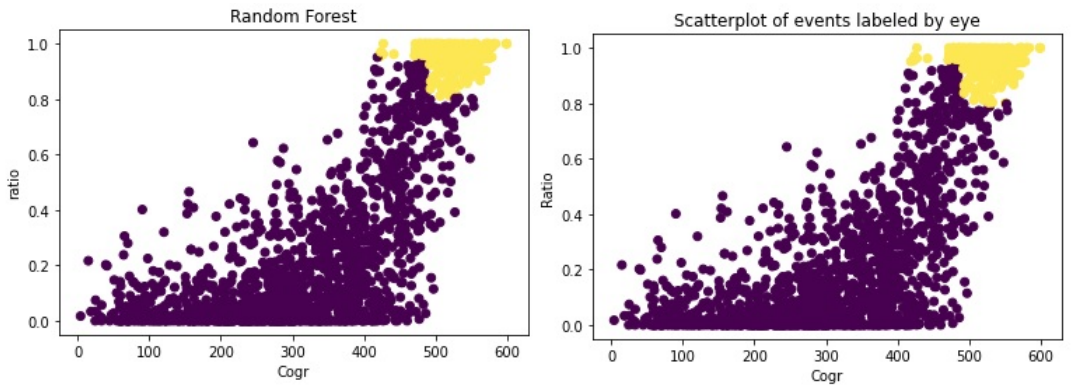


Figure 4.2: Comparison of event labels done by the Random Forest and my eye. The purple dots represent normal events, and the yellow dots represent corner-clippers. Notice the similarity to the 2D histogram from before, indicating corner-clippers are located in that top-right corner.

Out of the 1,759 events in the test set, almost all of the events are correctly labeled, with only 7 being labeled incorrectly; 5 of them being false negatives and 2 false positives. This means less than 1% of events are mislabeled, so the algorithm works very well at classifying events correctly.

Chapter 5: Discussion

Overall, I believe the method I have come up with to classify corner-clippers works very well. I am very satisfied with how well it performs in terms of the precision and completeness, as well as how the project turned out. That is not to say that it has no room for improvement though.

One thing that could be improved is the completeness and contamination, even though it almost classifies everything correctly. Below is an example of an event that was classified as a false negative. Clearly, it should be labeled as a corner-clipper seeing as how the reconstructed path does not even go through the detector, yet it is labeled as a normal event.

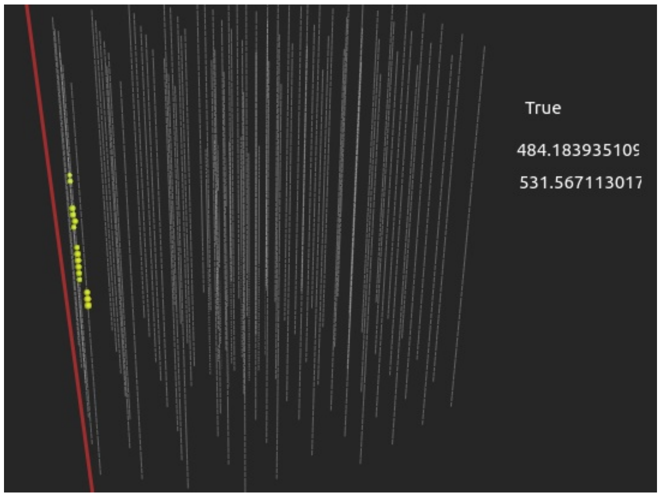


Figure 5.1: Example of a false negative. The 'True' text in the image indicates that it is a false negative (I added a special tag to find an example). The two numbers below, 484 and 531, are two different $\mathrm{CoG_r}$ values. 531 is my value, and 484 is a $\mathrm{CoG_r}$ value that was already inside steamshovel. This large difference in $\mathrm{CoG_r}$ values may explain why this event was labeled as a corner-clipper, although I don't know how the 484 value was calculated.

One possible explanation for this could be because of the asymmetry of the detector. Looking at Fig. 3.1 the detector is not a normal shape. It looks like a hexagon, but has a corner cut out so that it is actually an octagon. All sides of the detector are also not the same size, so this weird geometry could contribute to the $\mathrm{CoG_r}$ value being calculated incorrectly. The solution I came up with does not seem to handle events that could be classified as false negatives very well, which can be seen by looking at the confidence scores of these events.

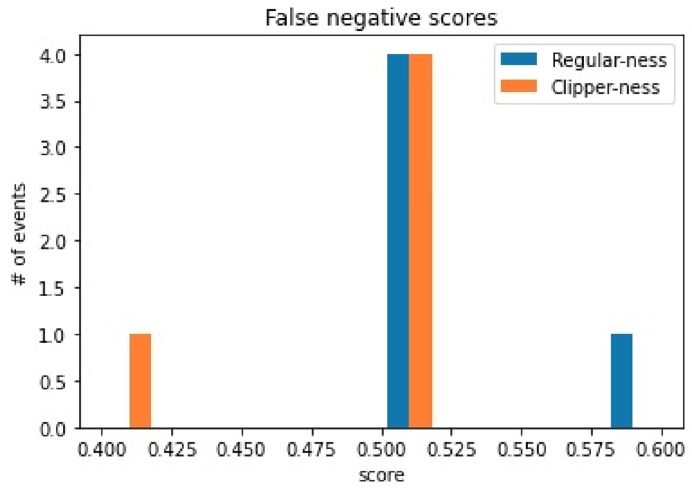
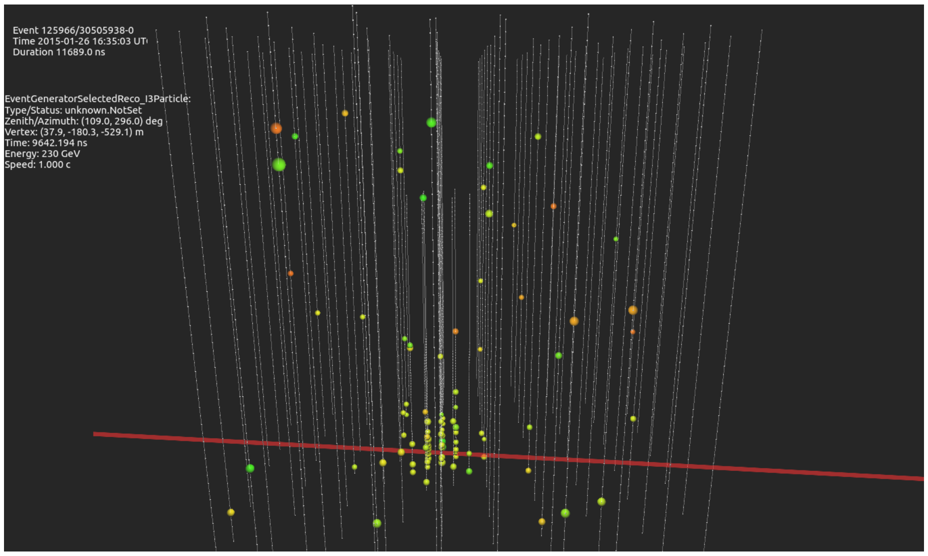


Figure 5.2: Confidence scores of false negative events. The blue bar indicates how confident the Random Forest is in it's prediction that the events are normal events, and the orange bar represents its confidence in the events being corner-clippers.

With the exception of one of the false negatives, the Random Forest is only slightly more confident that the events are corner-clippers, yet it still labels them as normal events. Future work could be done on optimizing the parameters, such as the number of trees or how many maximum branches each tree can have, given to the Random Forest so that it can better identify these hard to classify events.

Another area for future work/testing is adapting my program to work with cascade events. So far, all events that I have labeled, trained, and tested on were track events. Track events are defined by their relatively low energy and straight path, which can be seen in 2.2, 3.5, and 5.1. However, there is another type of event that IceCube observes, called a cascade. Cascades are characterized by the large amount of PEs detected and their "cascading" shape. The attributes I came up with may not work well when trying to identify corner clipping cascades. As can be seen from Fig. 5.3 below, there is a distinct area where a majority of the light is detected. However, for normal energy cascades like the one shown there are still some similarities between track-like events, like the path taken through the detector. These similarities can potentially cause an issue in my algorithm, since the two types of events are visually different (like corner-clippers and normal events), but it is hard to describe how they are different to a computer. Adapting my algorithm to make it work for tracks, cascades, or both at the same time would greatly improve it.



```
Event 125966/30505938-0
Time 2015-01-26 16:35:03 UT
Duration 11689.0 ns

EventGeneratorSelectedReco_I3Particle:
Type/Status: unknown.NotSet
Zenith/Azimuth: (109.0, 296.0) deg
Vertex: (37.9, -180.3, -529.1) m
Time: 9642.194 ns
Energy: 230 GeV
Speed: 1.000 c
```

Figure 5.3: Example of a cascade event.

Lastly, one of the cons of using a Random Forest algorithm is that it is slow. Since the algorithm's accuracy will increase with more trees, this means the most accurate Random Forests are the ones with the most trees. However, these trees all have to be run through, and adding more trees will slow down the run-time (9), making Random Forest not ideal for doing real-time analysis. A future improvement could be to change the machine learning algorithm to a faster one, which would improve on my algorithm and make it better for real time use.

Chapter 6: References

[1] C. L. Cowan Jr.; F. Reines; F. B. Harrison; H. W. Kruse; A. D. McGuire (July 20, 1956). "Detection of the Free Neutrino: a Confirmation" [2] Thomas Gaisser, Francis Halzen - Annu. Rev. Nucl. Part. Sci. 2014. 64:101–23

[3] CARVER, Tessa. Time Integrated searches for Astrophysical Neutrino Sources using the IceCube Detector and Gender in Physics studies for the Genera Project. Université de Genève. Thèse, 2019. doi: 10.13097/archive-ouverte/unige:120924 https://archive-ouverte.unige.ch/unige:120924

[4] Supervised vs Unsupervised Learning: Key Differences. (n.d.). Retrieved January 25, 2021, from https://www.guru99.com/supervised-vs-unsupervised-learning.html

[5] Basic usage. (n.d.). Retrieved February 24, 2021, from https://docs.icecube.aq/combo/trunk/projects/steamshovel/basics.html

[6] "Sklearn.metrics.precision_score." Scikit, scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html.

[7] Donges, Niklas. "A Complete Guide to the Random Forest Algorithm." Built In, builtin.com/data-science/random-forest-algorithm.

[8] Draelos, Rachel. “Measuring Performance: AUC (AUROC).” Glass Box, 2 Feb. 2020, glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/.

[9] Trehan, Daksh. “Why Choose Random Forest and Not Decision Trees.” Towards AI - The Best of Tech, Science, and Engineering, 17 Nov. 2020, towardsai.net/p/machine-learning/why-choose-random-forest-and-not-decision-trees.