School of Physics
and Astronomy

Senior Honours Project
Physics 4

Muon Filtering Algorithms

Alan Rowan
4th December 2020

Abstract

This project aims to identify a method for removing background noise from ANNIE detector data. Muons from the surrounding rock are vetoed by the detector's front muon veto but it was not in operation for the initial data taking. A deep learning neural network has been created as part of this report to try and remove these from the data. Variables were identified that appeared to show differences between the dirt muons and the muons from the neutrino beam, and these were then selected as input variables for the classification algorithm to use. The algorithm was able to classify the muons with a maximum 86.5% accuracy.

Declaration

I declare that this project and report is my own work.

Signature: ALAN ROWAN Date: 4th December 2020
Supervisor: Dr E Drakopoulou 10 Weeks

Contents

1 Introduction 1

2 Background 4

3 Methods 6
3.1 Real ANNIE Data Collection Procedure 6
3.2 Data Simulation 7
3.3 Variable Identification 7
3.4 Algorithms 10

4 Results 11
4.1 K-Fold Cross-Validation 11
4.2 Model Testing 13
4.2.1 Case 1 - Only PMT Timings 13
4.2.2 Case 2 – PMT Timings and PMT Charge 15
4.2.3 Case 3 - PMT Timings, PMT Charge, and Number of Photoelectrons 17
4.2.4 Case 4 - Only PMT Charge and Number of Photoelectrons 19

5 Conclusions and Future Outlook 21

6 Acknowledgements 21

7 References 21

8 Appendix 21
8.1 Data Plotting Script 22
8.2 Classifier Algorithm 27

1 Introduction

Neutrinos are effectively massless, neutral particles that interact so weakly with other matter that studying them is one of the great challenges in physics today. First conjectured as a 'new' particle by Wolfgang Pauli in 1930, the neutrino made it possible to resolve then-present issues such as the existence of a continuous β-decay spectrum [reference 1].

The Accelerator Neutrino Neutron Interaction Experiment (ANNIE) is a neutrino detector located at Fermilab, and it aims to develop physicists' understanding of neutrino interactions with matter. It is a water-based Cherenkov detector, meaning it measures Cherenkov radiation emitted by neutrino interactions in its water tank. The detector is situated on Fermilab's booster neutrino beam (BNB). The BNB accelerates protons to 8.89GeV using an accelerator. These protons are then fired at a beryllium target, where interactions generate pions and kayons of mixed charges. Negative pions are selected for by an electromagnet, and these then decay into muons and muon neutrinos:

$$\pi^- \to \mu^- + \nu_\mu$$

The muons are stopped by an absorber, leaving a beam of neutrinos. ANNIE uses these neutrinos to advance multiple research avenues.

The neutrinos from the BNB enter into ANNIE's main tank. Neutrino interactions within this tank can produce muons that are more easily studied than the neutrinos themselves ([H. Gallagher et al., 2011]). One of ANNIE's key components are the photo-multiplier tubes (or PMTs) that act as detectors of light within the tank. They are located around the edge of the tank, as shown in figure 1.

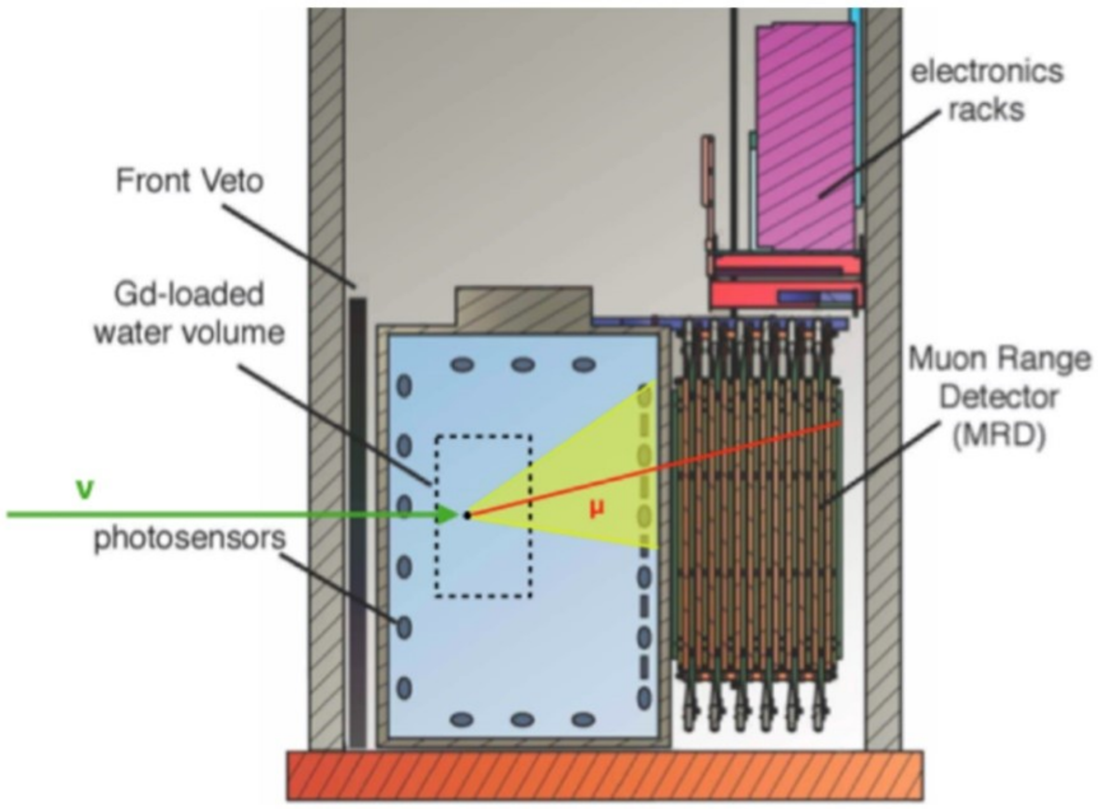


Figure 1: ANNIE Schematic

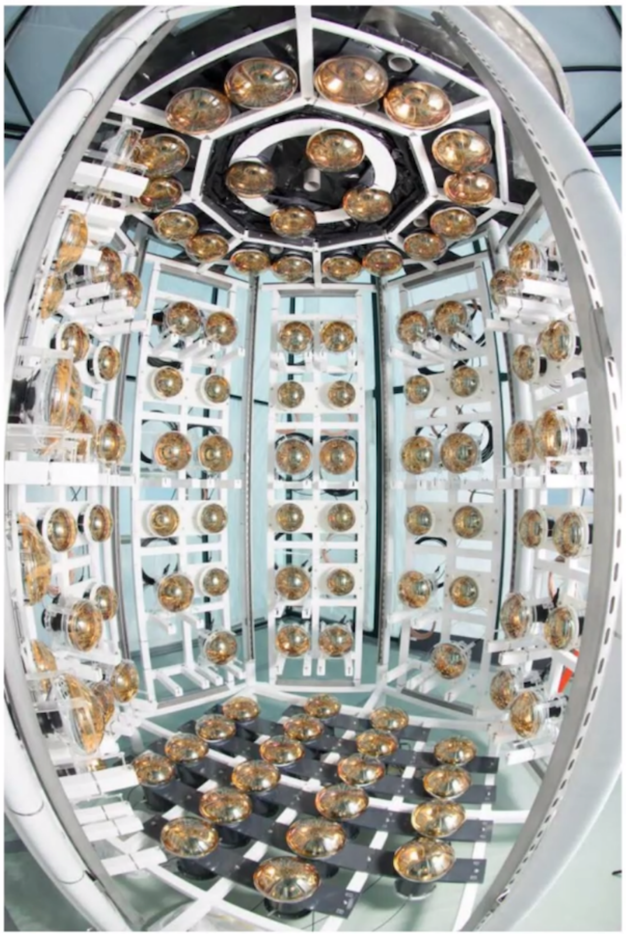


Figure 2: Photo-Multiplier Tubes (PMTs) surrounding ANNIE's water tank

Muons can be created in the rock surrounding the tank. If these muons then enter the tank they can generate Cherenkov radiation using the same mechanisms as the muons created inside the tank. To combat this potential contamination of the data, a front anti-coincidence counter (FACC) or 'front veto' was installed. The purpose of the front veto is to tag incoming muons from upstream of the neutrino beam so that they can then be filtered out of the collected data.

After undergoing recent upgrades, ANNIE began taking measurements again in 2019. However, at the time of first collecting data after the upgrade, the front veto system was not properly calibrated, and so measurements were taken without it installed. It was decided that a potential solution may lie within the data itself. Since the muons produced within the tank and the muons produced in the surrounding rock had difference origins, it was likely that there would be differences in their data. The first aim of this project was to process this data for the two categories of muons separately in order to determine if such differences existed and, if so, what they were.

Developments in the field of machine learning have made it more and more accessible for projects such as this. By identifying variables that could be used to group muons into signal and background muons, data could then be fed to a machine learning algorithm to train it for making these determinations without prior knowledge of a muon's origin. So, the second aim of this project was to investigate whether or not the identified variables could be used to train a clasiffier algorithm.

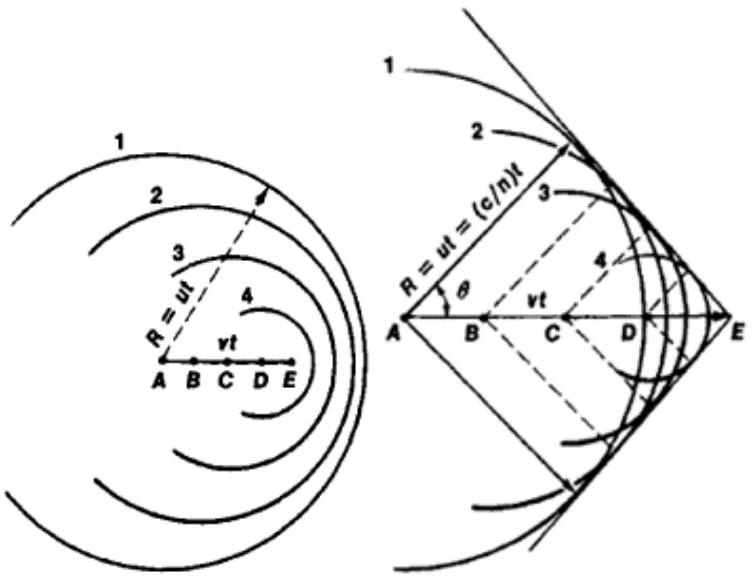
2 Background

As previously mentioned, neutrinos can interact with the nuclei in ANNIE's water tank and generate muons ([H. Gallagher et al., 2011]). The muon that is emitted will be travelling at very high speeds. This creates an effect which will now be explained.

There exists a variation in the speed of light within different media. Since the speed of light in a vacuum is fixed and is the fastest speed at which it is possible to travel, these media invariably reduce the speed of light. It therefore becomes possible to accelerate matter up to speeds which surpass light in the same medium. If the particle is charged and the medium is dielectric, light is emitted in an effect known as Cherenkov radiation, which was discovered in the early 19th century.

The theoretical cause of Cherenkov radiation can be considered as a logical outcome of the Huygen-Fresnel principle. Consider first an electrically charged particle moving at a speed v which is less than the phase velocity of light in the medium, u[1]. Let all points on the particle's trajectory be the source of a secondary wave, generated once the particle passes through that point.

[1]u is given by the speed of light in a vacuum divided by the refractive index of the medium, $u = \frac{c}{n}$



(a) Huygens 1 (b) Huygens 2

Figure 3: Huygens secondary wavelet formations for the two scenarios.

Referring now to figure 3a, when the particle passes through point A, it will emit the spherical wave labelled 1. It does the same thing at points B, C, and D, emitting waves 2, 3, and 4 respectively. When it arrives at point E, it will have travelled a distance vt. The initial wave, 1, will have propagated out such that its radius is now ut. Given that u here us greater than v, the wave labelled 1 will be ahead of the particle when it arrives at E. The same is true for all the other waves emitted by the particle, where each is contained entirely within the one that came before it in an eccentric patter. These secondary wavelets do not overlap in a way that would allow the propagation of a real wavefront. This is because, according to Huygens' principle, the propagating wavefront is determined by the envelope of the secondary wavelets ([P. Enders]). There is no such envelope in this scenario.

If, instead, the velocity of the particle was greater than that of the phase velocity of light in the medium, by the time the particle reached the second point B it would have moved outside of the radius that wave 1 had thus far been able to propagate to. The secondary wavelets now have an envelope tangential to their wavefronts. This can now correctly describe a wave propagating out from the moving particle. The radiation is in the shape of a cone, known as a Cherenkov cone. The angle the cone makes with the particle's trajectory can be determined.

Since the wavefront of the Cherenkov radiation is tangential to the spherical (or circular in the figure) secondary wavelets, a right-angled triangle can be constructed between A, E and the point where the radius of the first wavelet touches the tangent. Using basic trigonometry:

$$\cos\theta = \frac{ut}{vt} = \frac{u}{v}$$

If the refractive index of the medium is n, then this expression can also be written as:

$$\cos\theta = \frac{c}{nv}$$

Here again there is an indication that the radiation only occurs if $v > u$, since $\cos\theta$ is undefined for $\frac{c}{nv} = \frac{u}{v} > 1$.

This description helps to provide an intuitive grasp of the construction of a Cherenkov cone, however it gives no description or explanation of the underlying mechanism. [2]

It is this Cherenkov radiation that is utilised in neutrino detectors to take measurements. By analysing the radiation, it is possible to determine the trajectory of the muon. The interaction that generated the muon is then also made clear.

3 Methods

3.1 Real ANNIE Data Collection Procedure

Data collection was set for periods of 2 microseconds. Since the neutrino beam spills lasted for 1.6 μs each, this ensured that the tank PMTs captured the complete beam window.

Below, a sample of the ANNIE PMT detection times are plotted. The heightened readings show the beam spill taking place. The prompt window is able to capture all events, since the readings have trailed off before measurements stop being taken.

[2]The actual mechanism behind this is beyond the scope of and entirely irrelevant to this project.

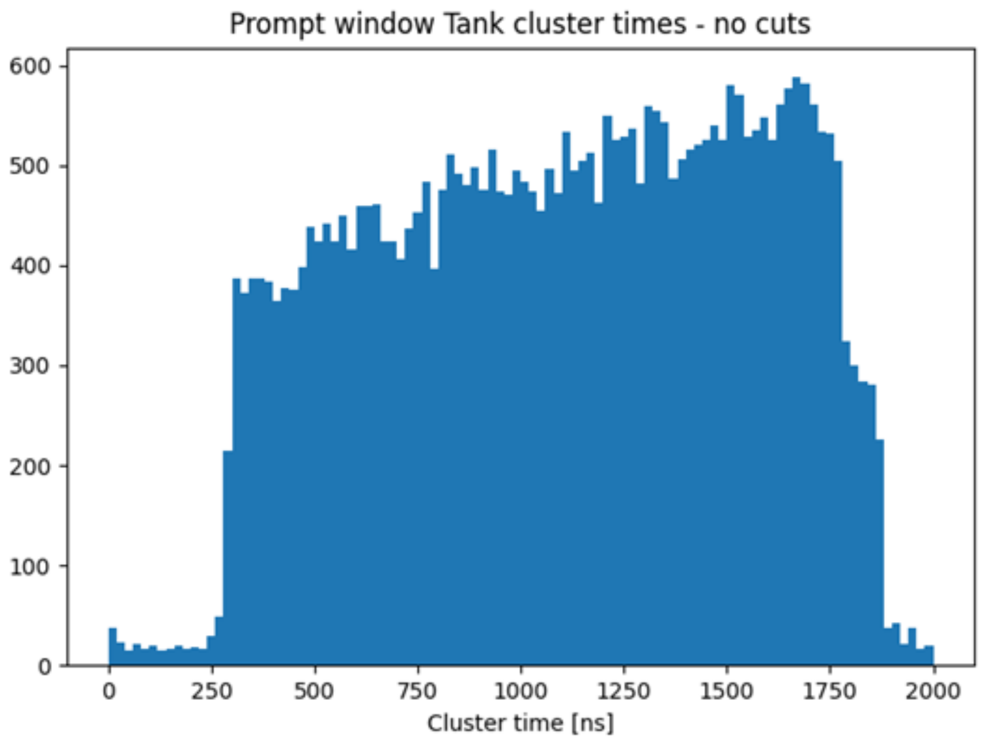


Figure 4: ANNIE prompt events

3.2 Data Simulation

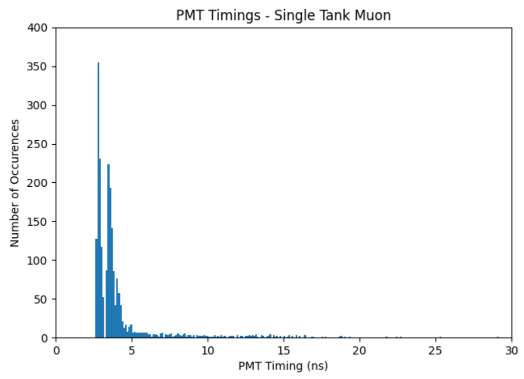
For the purposes of training the machine learning algorithm, data for the muons was simulated. The dirt muons were simulated starting in the rock outside the tank, one metre beyond the veto. They all had their origins within a 2x2x2 volume, although their positions within this were random. Their trajectories were then directed towards the detector, within an angle of around ten or twenty degrees. This ensured that only muons that would actually have interacted with the detector were included. Tank muons were simulated as beginning their trajectories within the ANNIE tank itself, being aimed towards the muon range detector also within a ten to twenty degree range.

3.3 Variable Identification

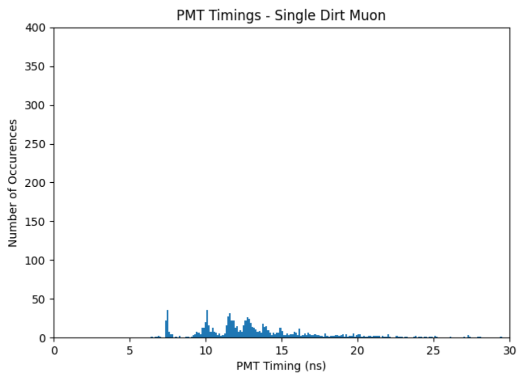
Using simulated muon data, the next task was to identify variables that obviously differed depending on whether they were for a tank or a dirt muon. After a preliminary look at the data, it was noticed that PMT timings had different patterns for dirt muons and for tank muons. This was likely caused by the Cherenkov cones of the dirt and tank muons having a different signature (a direct result of them tending to have a different trajectory).

Due to the difference in typical trajectories that was likely to be present between the tank and the dirt muons, a likely source of differentiation would be found in the PMT timing data. Tank muons could be generated through interactions at any point along the BNB trajectory into the detector. Hence the Cherenkov cones would, on average, be smaller and trigger PMT timing data collection over shorter timeframes.

Indeed, when comparing the PMT timing data for a tank muon and a dirt muon, there are instantly-noticeable differences:



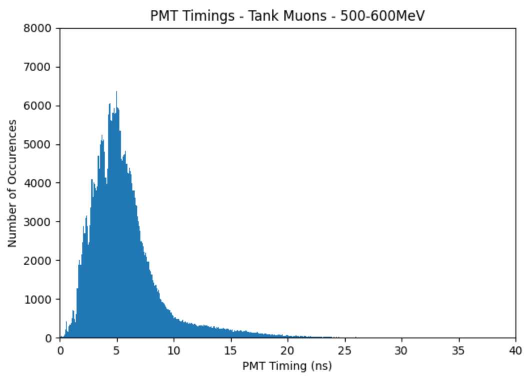
(a) Tank Muon



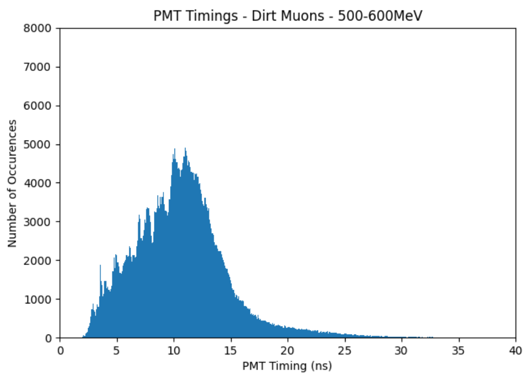
(b) Dirt Muon

Figure 5: PMT timing data for a single muon.

And these differences remain clear when including all PMT timing values for all particles in a given category:



(a) Tank Muons



(b) Dirt Muons

Figure 6: PMT timing data for all muons within an energy range of 500-600MeV

It was also identified that the charges registered by the PMT clusters varied slightly between the muon categories.

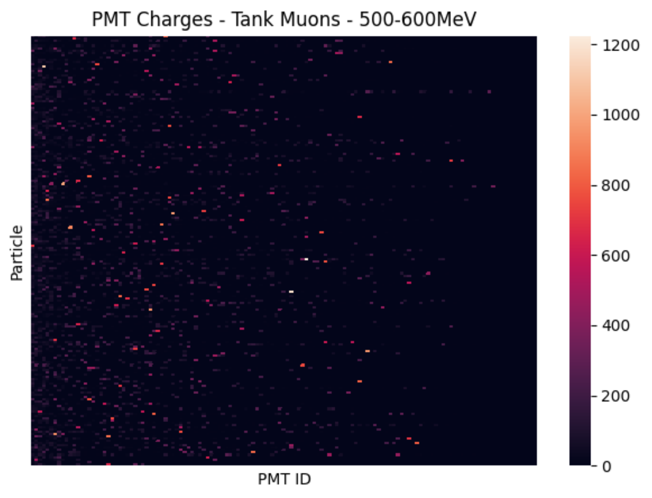


Figure 7: Heatmap of charges for tank muons in a 500-600MeV energy range. Each pixel corresponds to the charge reading of a PMT for a given particle.

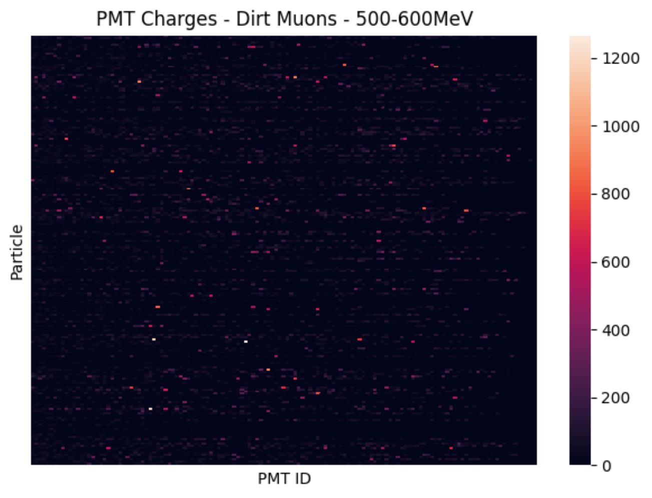
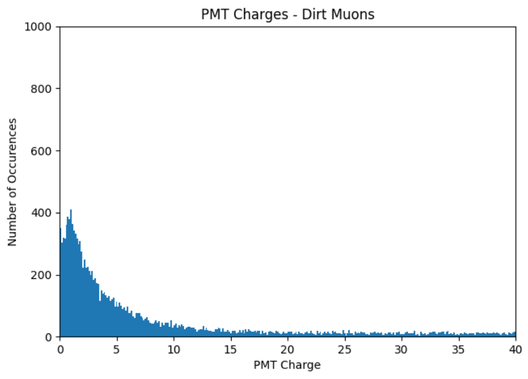
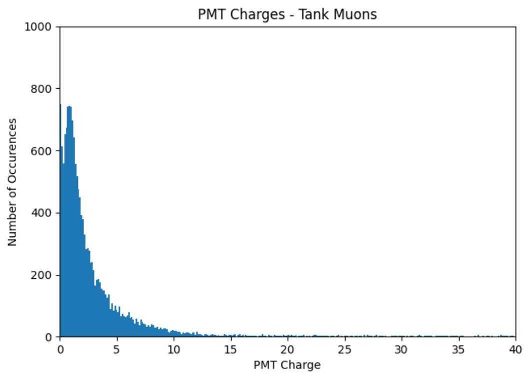


Figure 8: Heatmap of charges for dirt muons in a 500-600MeV energy range. Each pixel corresponds to the charge reading of a PMT for a given particle.



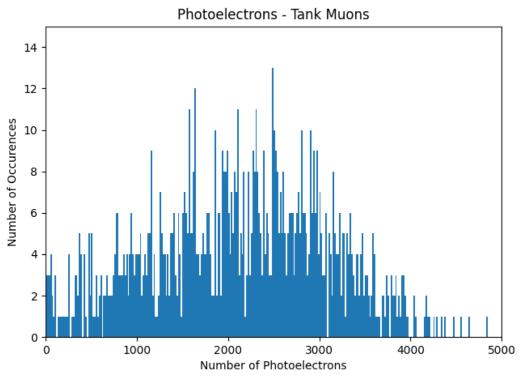
(a) Tank Muons



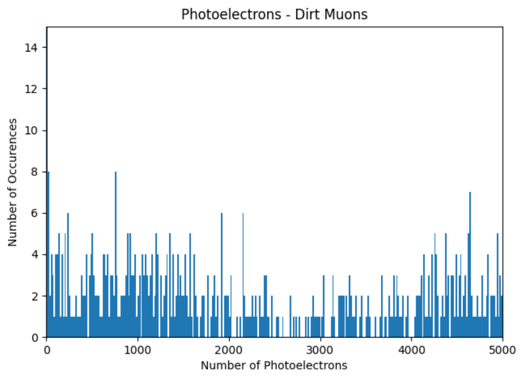
(b) Dirt Muons

Figure 9: Charge data for dirt and tank muons.

Photoelectron count refers to the number of electrons generated in the PMTs by the detection of a Cherenkov photon. A higher photoelectron count for a particle is an indication that it emitted more Cherenkov radiation within the tank. Since the geometry of the Cherenkov cone is one of the key classification parameters, the number of photoelectrons associated with a particular muon is likely to be useful in determining its origin. As seen in the graphs below, the photoelectron count can indeed be used as a distinguishing variable between tank and dirt muons.



(a) Tank Muons



(b) Dirt Muons

Figure 10: Photoelectron counts for tank and dirt muons.

3.4 Algorithms

Having selected the variables that would be used to categorise the muons, an algorithm had to be created that would be able to do this. The desired outcome for such an algorithm was that it would read in the datafiles and generate a binary output for each particle (a 1 for a tank muon and a 0 for a dirt muon). There were 10,000 PMT timing values per particle, each of which acted as a separate input variable. The basis for the algorithm was then that it should read in these input timings and output a 1 or a 0, depending on its prediction.

The algorithm first needed to be trained using the data. 70% of the data was used for training, and the other 30% was reserved for testing. In total there were data for 2000 particles, half of which were tank muons and half of which were dirt muons. These were shuffled before the training began, leaving a random selection of 1400 particles for training and 600 for testing.

Initially, sci-kit learn was used to determine a rough estimate of the potential accuracy of the algorithm. A 10-fold cross-validation was carried out. This means that the data set was first split into 10 subsets. In turn, each subset would then be used to test a model that was trained using the other nine groups. A baseline model was created and given three layers, with 1000, 60, and 1 neurons respectively. The model used a deep learning neural network that was created with TensorFlow (TNN) ([M. Abadi et al., 2015]). It was this model that was then validated using the 10-fold cross-validation.

After this initial proof-test was carried out, the next step was to use the model to make predictions. As stated before, the data was split into 1400 particle data sets for training, and 600 for testing[3].

4 Results

4.1 K-Fold Cross-Validation

The results of the 10-fold cross-validation of the model with different input variables are presented below. They demonstrate the theoretically high accuracy of the model.

[3]The scripts used to create this model and test it can be found in the appendix

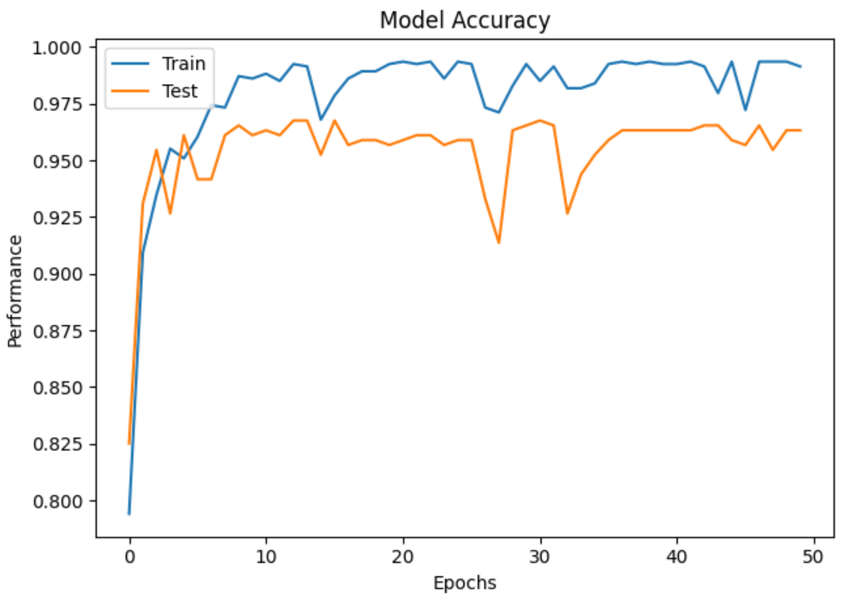


Figure 11: The above graph tracks the accuracy of the model when the only inputs are the PMT timings.

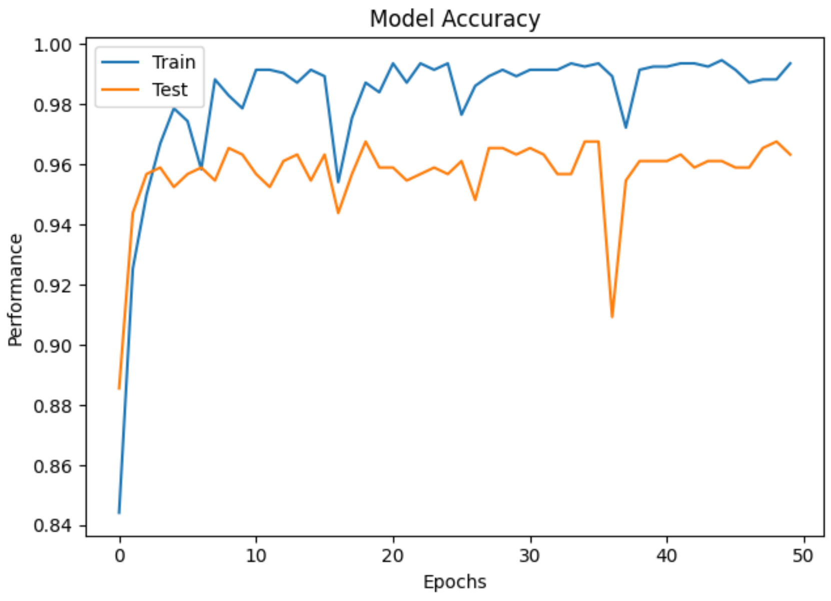


Figure 12: When the additional input parameters of the PMT charges are included, there aren't noticable differences in the 10-fold cross-validation performance.

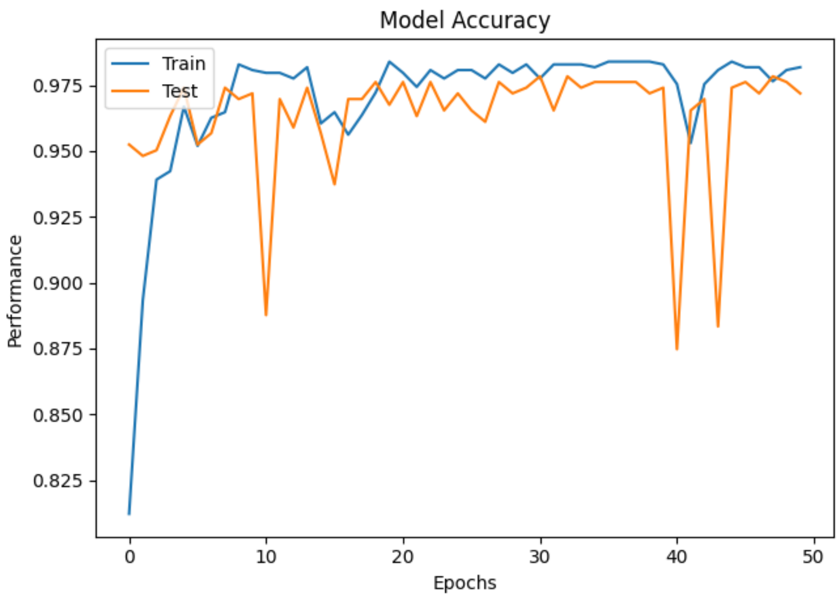


Figure 13: By including the photoelectron counts as an input variable, the k-fold cross-validation estimates that the model should be much more accurate.

4.2 Model Testing

Since four approaches were taken (each with different input variables), the outcomes of each will now be presented separately.

4.2.1 Case 1 - Only PMT Timings

In the case where the 10,000 PMT timing variables were the only inputs into the machine learning algorithm, there was definitely some success. The following Receiver Operating Characteristic (ROC) curve displays that the algorithm was able to successfully categorise the muons with greater accuracy than if it simply guessed at random.

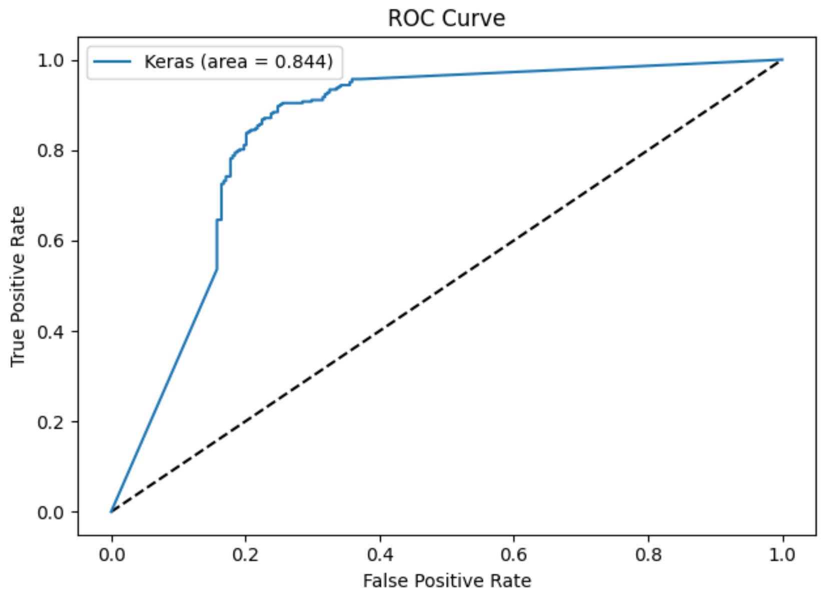
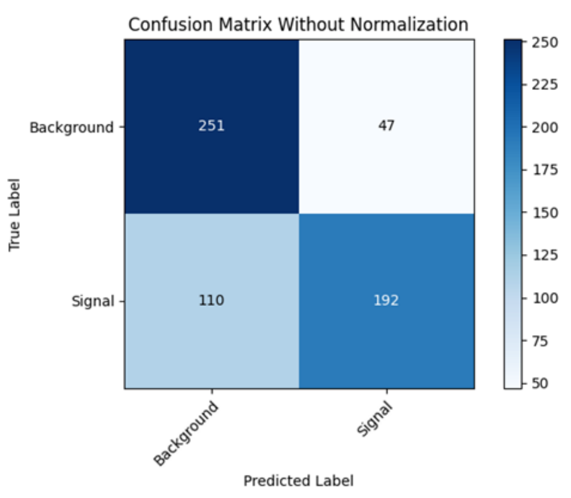


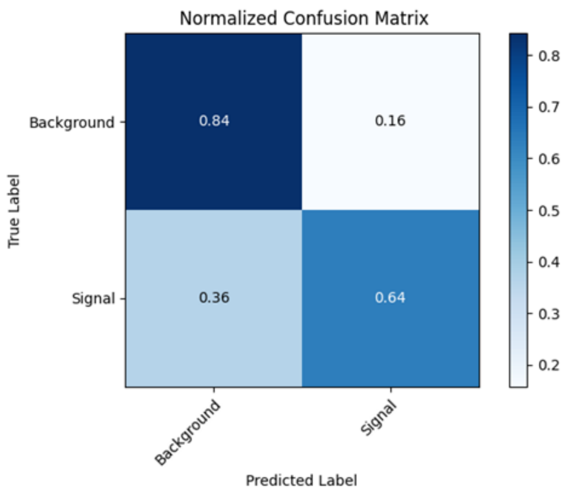
Figure 14: ROC curve for case 1

The diagonal line is a representation of a scenario where the output is merely a 50/50 guess. Curves to the top-left of this are all improvements over guessing; a perfect classifier would be one where the ROC curve went straight from the bottom left, to the top left, to the top right.

The algorithm was run and tested several times, with the results being very consistent. As an example, a given run had the 600 particles with a split of 298 dirt (or 'background') muons, and 302 tank (or 'signal' muons). The classifier algorithm correctly classified 251 of the dirt muons, leaving only 47 with a mis-classification as tank muons. This is an accuracy of 84.2%, significantly better than if no classifier had been used. When it came to classifying the tank muons, the algorithm was less successful. 192 were correctly labelled as tank muons, although 110 were wrongly identified as dirt muons (an accuracy of 63.5%. While this is still a more desired outcome than 50%, it is a significant reduction in the quality of classification that is seen with identifying dirt muons. Overall, the accuracy of this example run was 74%. The following figure displays the confusion matrix that visualises these outcomes.



(a) Absolute Values



(b) Normalized

Figure 15: Confusion matrix for classifier algorithm with only PMT timings for inputs.

4.2.2 Case 2 – PMT Timings and PMT Charge

PMT charge was identified as another variable that displayed differences between tank and dirt muon categories. Each particle had 133 PMT charge data points, expanding the input variables to have 10,133 per particle. Apart from factoring in the slightly larger input sample, not many alterations had to be made to the classifier algorithm. The ROC curve was generated in the same way as before, and is printed below.

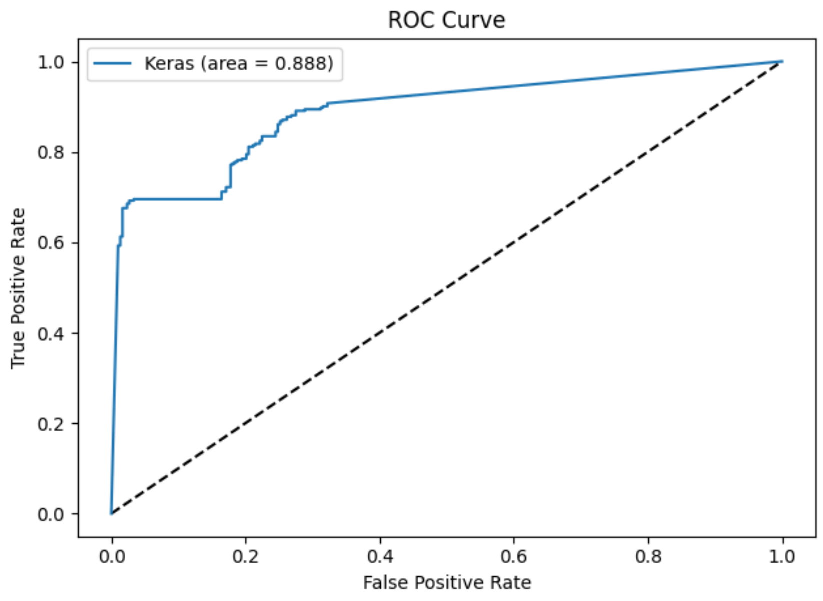
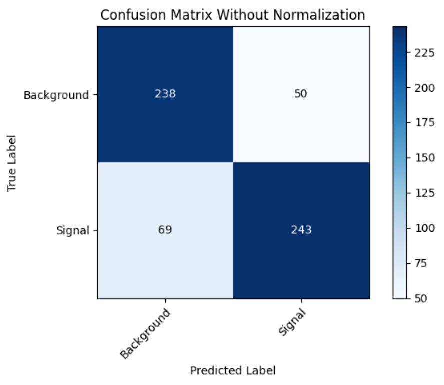


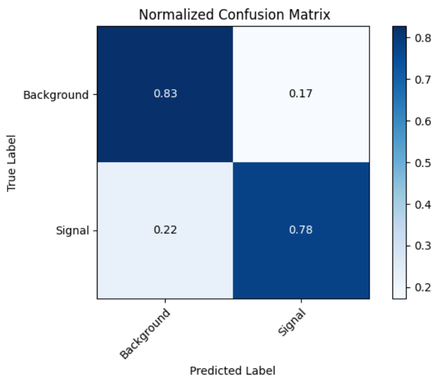
Figure 16: ROC curve for case 2

This curve begins with a much steeper initial gradient. This results in a higher area under the ROC curve, which is typically taken as a sign that the model is better at estimating binary classifications. This would be expected due to the addition of extra input variables.

The exact same procedure was then carried out to determine the accuracy of the model with these input variables. It was noted that the outcome was less consistent, but in all cases had an overall accuracy greater than the cases were charge was not included. Due to the time it took to the run the neural network, not enough data was obtained to make any reliable estimates of the average accuracy of the model. It was therefore decided to use an example in which the model was the least accurate. In this instance, 288 of the muons were dirt muons, and the other 312 were tank muons. The confusion matrices below.



(a) Absolute Values



(b) Normalized

Figure 17: Confusion matrix for classifier algorithm wity PMT timings and PMT charge for inputs.

What is immediately clear is how much more accurate the model is at predicting the classification of the tank muons. This was a trend seen in all instances. In most other cases the accuracy for classifying dirt muons was roughly the same as when only the PMT timings were used as input variables, however in one case the accuracy for this was 98%. Since this result was not consistent, however, it was concluded that this was likely a fluke on the part of the algorithm. The fact that the PMT charge readings causes the tank muon classification accuracy to rise so dramatically is a very positive sign, as it would result in less loss of useful data points when ANNIE runs experiments.

4.2.3 Case 3 - PMT Timings, PMT Charge, and Number of Photoelectrons

Upon the collection of the above data it was then also noticed that photoelectron counts could be another discriminating factor between the tank and the dirt muons. The ROC curve for the algorithm using these inputs is shown below, with the shape justification being as before for case 2.

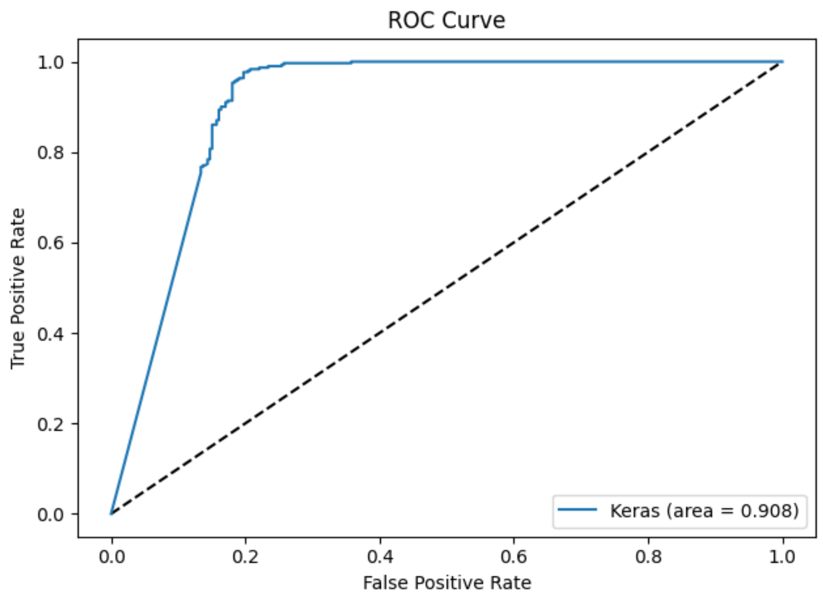
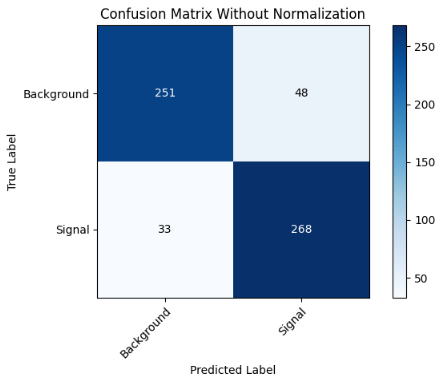
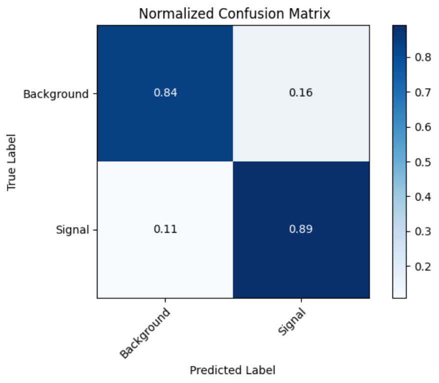


Figure 18: ROC curve for case 3

Due to the lateness with which this variable was selected as an input for the algorithm, only a handful of tests were run. In one of these tests, 299 muons belonged to the dirt muons category, while 301 were tank muons. The model now had three separate physical variables, corresponding to 10,134 variables for the algorithm to use. This certainly was apparent in the data. The classifier now had many ways to assess a muon's origin. The accuracy with which it classified tank muons now even overtook the accuracy for dirt muons, which still remained relatively static. The confusion matrices for this case are shown below.



(a) Absolute Values



(b) Normalized

Figure 19: Confusion matrix for classifier algorithm wity PMT timings, PMT charge, and number of photoelectrons for inputs.

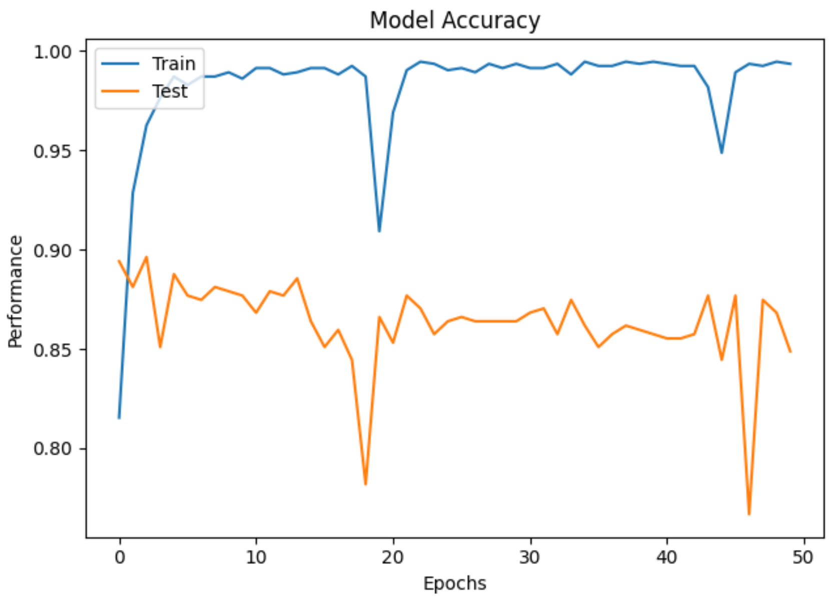
For this example, the classifier had an overall accuracy of 86.5% when using these input variables.

4.2.4 Case 4 - Only PMT Charge and Number of Photoelectrons

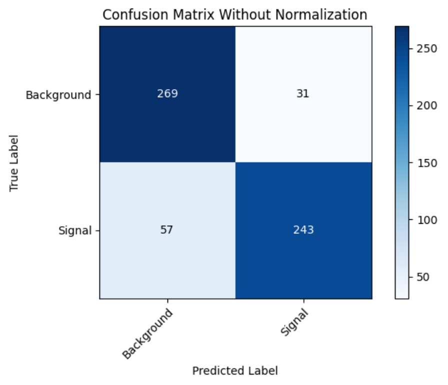
The data used to simulate the tank and the dirt muons has thus far given the algorithm a distinct advantage over what happen in a real-life scenario. The PMT timings are recorded as the time between from the particle's inception and the detection of a photon from the Cherenkov cone. This means that it is possible to generate histograms as seen in Figures 5 and 6. These histograms demonstrate the typical times at which the PMTs receive hits during the particle's lifetime in the simulation.

In real data collecting, PMT timing data would not be so useful. The timings would be start to be recorded moments before the BNB beam spill, and would take place over the following microseconds. This is much more similar to the case of the histogram seen in Figure 4. By training the algorithm in the way it has been trained, the PMT timing data would yield no value as an input due to this difference.

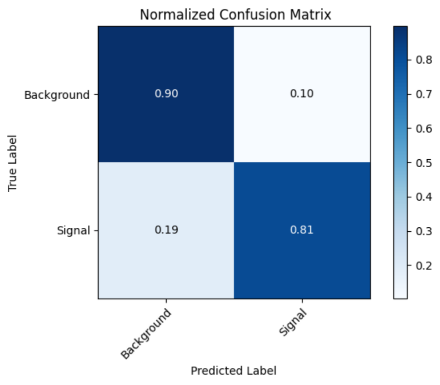
It was therefore decided to run a test on the model in which only PMT charge and photoelectron numbers are used as input variables. This removes 10,000 of the input variables and leaves 134. The 10-fold cross-validation graph is shown below.



This appears to show a drop in the expected accuracy of the model, which is expected with a reduced number of input variables. Despite this, however, the classifier performed extremely well, being able to correctly classify 90% of the dirt muons and 81% of the tank muons. This results in an overall accuracy of 85.3%. The confusion matrices are shown below, along with the ROC curve for this test run.



(a) Absolute Values



(b) Normalized

Figure 20: Confusion matrix for classifier algorithm with only PMT charge, and number of photoelectrons for inputs.

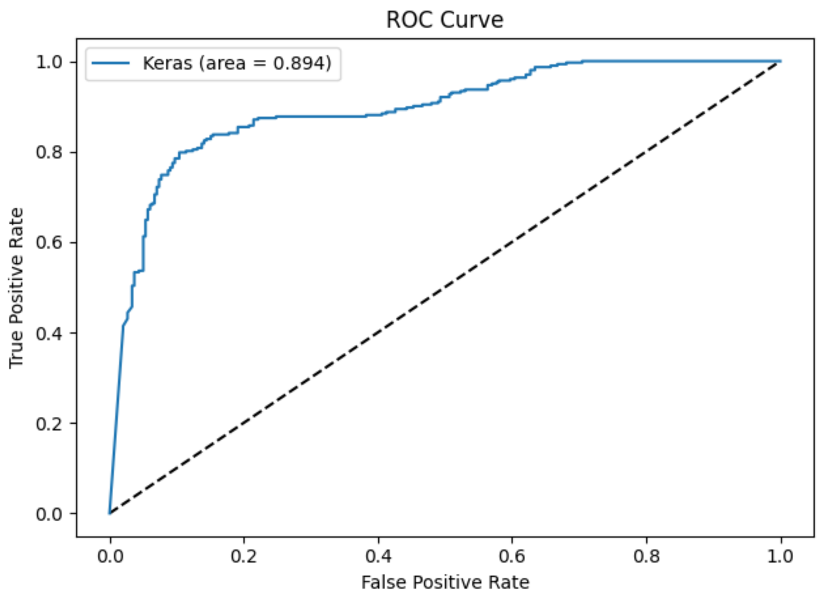


Figure 21: ROC curve for case 4

It seems counter-intuitive at first that this configuration of input variables was by far the best at classifying the tank muons. There must be something within the construction of the data that leads to less accuracy with the inclusion of PMT timings. Since the PMT timings for the tank muons tend to fit a much sharper curve (refer to Figure 5a), it is possible that the algorithm prefers to classify muons that don't quite fit this as dirt muons. This is something that would require further investigation, however, and is not within the scope of the project. Machine learning is a deep and complex subject; the specific reasoning behind this supposed anomaly may take a large amount of work to uncover.

5 Conclusions and Future Outlook

This project aimed to determine whether or not it was feasible to reliably clean data from the ANNIE detector whilst the FACC component was not installed. By simulating typical particle trajectories and using it to train and algorithm to classify muons, it was shown that it was indeed possible and effective at filtering background particles.

The case where just the PMT charge and the photoelectron counts were used to train the classifier provided staggering accuracy considering the massive reduction in input variables. However, it is possible that the vast number of PMT timing variables caused the model to be less accurate. By removing these, the model could potentially have had more clarity, particularly when classifying the tank muons.

An interesting next step would be to use an algorithm, such as the one written for this report, in conjunction with real ANNIE data-taking. By comparing the model's predictions with the tagging of muons by the FACC, it would perhaps be possible to further refine the accuracy that it is capable of achieving. Prior to this, however, it would be useful to expand upon the data simulation, so that a dataset more like that of reality could be used.

6 Acknowledgements

My supervisors have been endlessly helpful and have been available to answer my questions at any time of day (or night); I would like to thank them for their support with this project and for introducing me to such an interesting and exciting topic.

7 References

[H. Gallagher et al., 2011] H. Gallagher, G. Garvey, G.P. Zeller (2011). Neutrino-Nucleus Interactions. *The Annual Review of Nuclear and Particle Science,* 61:355–78.

[P. Enders] P. Enders. Huygens' Principle and the Modelling of Propagation. *European Journal of Physics,*17, 4

[M. Abadi et al., 2015] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, Software available from tensorflow.org, (2015).] via the Keras [Keras: The Python Deep Learning library, keras.io.] library in python

8 Appendix

Please note that in places, a single line of code has needed to be written across two lines.

8.1 Data Plotting Script

```
import pandas as pd
import matplotlib.pyplot as plt
import scipy.optimize as scp
import numpy as np
import seaborn as sns

Tdf = pd.read_csv("tank_muons_data_1000.csv")
Ddf = pd.read_csv("dirt_muons_data_1000.csv")

T_rows = len(Tdf.index)
D_rows = len(Ddf.index)
Total_rows = T_rows + D_rows

Tank_1 = [1]*T_rows
Dirt_0 = [0]*D_rows
Tdf.insert(1, "T or D", Tank_1, True)
Ddf.insert(1, "T or D", Dirt_0, True)

frames = [Tdf, Ddf]
df = pd.concat(frames, ignore_index = True)



print (df.head())
#print("All columns are: ", df.columns.values.tolist())
print("")
#print(df[:10])
#df.to_csv("test.csv",index=False) #write dataframe to csv file

 #---- My Plots:
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

print("Input data number of columns = " + str(len(df.columns)))
total_rows = len(df.index)
print("Input data frame rows = " + str(total_rows))
print("")

lower_limit = int(input("Input lower energy limit: "))
upper_limit = int(input("Input upper energy limit: "))
print("")
```

```
#pmt—timings plots
SingleTankTimes = []
SingleTankTimings = Tdf.loc[0, "T_0":"T_9999"]
for i in range(10000):
        column = "T_" + str(i)
        if SingleTankTimings.loc[column] != 0:
            SingleTankTimes.append(SingleTankTimings.loc[column])

plt.hist(SingleTankTimes, bins=350, range=(0,40))
plt.title("PMT Timings - Single Tank Muon")
plt.xlabel("PMT Timing (ns)")
plt.ylabel("Number of Occurences")
plt.axis([0, 30, 0, 400])
plt.show()

SingleDirtTimes = []
SingleDirtTimings = Ddf.loc[0, "T_0":"T_9999"]
for i in range(10000):
        column = "T_" + str(i)
        if SingleDirtTimings.loc[column] != 0:
            SingleDirtTimes.append(SingleDirtTimings.loc[column])

plt.hist(SingleDirtTimes, bins=350, range=(0,40))
plt.title("PMT Timings - Single Dirt Muon")
plt.xlabel("PMT Timing (ns)")
plt.ylabel("Number of Occurences")
plt.axis([0, 30, 0, 400])
plt.show()


relevant_rows = 0

#Identify number of  rows within given energy range
for n in range(len(df.index)):
    if df.loc[n, "trueKE"] <= upper_limit:
        if df.loc[n, "trueKE"] >= lower_limit:
            relevant_rows = relevant_rows + 1
```

```
array = np.zeros(shape = (relevant_rows, len(df.columns)))

#Update array entries with rows from dataframe that fit energy range

n = 0
for i in range(len(df.index)):
    if df.loc[i, "trueKE"] <= upper_limit:
        if df.loc[i, "trueKE"] >= lower_limit:
            array[n] = df.loc[i]
            n = n+1

#Convert array into dataframe

Edf = pd.DataFrame(data = array, columns = df.columns)
print(Edf.head())

TEdf = Edf.loc[Edf["T or D"] == 1]
print(len(TEdf.index))
print(TEdf.head(20))

DEdf = Edf.loc[Edf["T or D"] == 0]
DEdf = DEdf.reset_index(drop=True)
print(len(DEdf.index))
print(DEdf.head(20))

#Create plot for timings

Ttotal_timings = []
counter = 0

for n in range(len(TEdf.index)):
    Ttimings = TEdf.loc[n, "T_0":"T_9999"]
    for i in range(len(Ttimings)):
        column = "T_" + str(i)
        if Ttimings.loc[column] != 0:
            Ttotal_timings.append(Ttimings.loc[column])



plt.hist(Ttotal_timings, bins=500, range=(0,40))
plt.title("PMT Timings - Tank Muons - " + str(lower_limit) + "-"
+ str(upper_limit) +"MeV")
```

```
plt.xlabel("PMT Timing (ns)")
plt.ylabel("Number of Occurences")
plt.axis([0, 40, 0, 8000])
plt.show()

Dtotal_timings = []
counter = 0

for n in range(len(DEdf.index)):
    Dtimings = DEdf.loc[n, "T_0":"T_9999"]
    for i in range(len(Dtimings)):
        column = "T_" + str(i)
        if Dtimings.loc[column] != 0:
            Dtotal_timings.append(Dtimings.loc[column])



plt.hist(Dtotal_timings, bins=500, range=(0,40))
plt.title("PMT Timings - Dirt Muons - " + str(lower_limit) + "-"
+ str(upper_limit) +"MeV")
plt.xlabel("PMT Timing (ns)")
plt.ylabel("Number of Occurences")
plt.axis([0, 40, 0, 8000])
plt.show()

#plot of pmt charge vs ID

Tcharges = TEdf.loc[0:len(TEdf.index), "pmt_charge_0":
"pmt_charge_132"]
ax = sns.heatmap(Tcharges)
plt.title("PMT Charges - Tank Muons - " + str(lower_limit) + "-"
+ str(upper_limit) +"MeV")
plt.xlabel("PMT ID")
plt.ylabel("Particle")
ax.axes.xaxis.set_ticks([])
ax.axes.yaxis.set_ticks([])
plt.show()

Dcharges = DEdf.loc[0:len(DEdf.index), "pmt_charge_0":
"pmt_charge_132"]
ax = sns.heatmap(Dcharges)
plt.title("PMT Charges - Dirt Muons - " + str(lower_limit) + "-"
+ str(upper_limit) +"MeV")
plt.xlabel("PMT ID")
plt.ylabel("Particle")
ax.axes.xaxis.set_ticks([])
ax.axes.yaxis.set_ticks([])
plt.show()

Ttotal_charges = []
counter = 0

for n in range(len(TEdf.index)):
    Tcharges = TEdf.loc[n, "pmt_charge_0":"pmt_charge_132"]
    for i in range(len(Tcharges)):
        column = "pmt_charge_" + str(i)
        if Tcharges.loc[column] != 0:
            Ttotal_charges.append(Tcharges.loc[column])

Dtotal_charges = []
counter = 0

for n in range(len(DEdf.index)):
     Dcharges = DEdf.loc[n, "pmt_charge_0":"pmt_charge_132"]
     for i in range(len(Dcharges)):
        column = "pmt_charge_" + str(i)
        if Dcharges.loc[column] != 0:
            Dtotal_charges.append(Dcharges.loc[column])

print(Dtotal_charges)
print(Ttotal_charges)
plt.hist(Ttotal_charges, bins=300, range=(0,40))
plt.title("PMT Charges - Tank Muons")
plt.xlabel("PMT Charge")
plt.ylabel("Number of Occurences")
plt.axis([0, 40, 0, 1000])
plt.show()

plt.hist(Dtotal_charges, bins=300, range=(0,40))
plt.title("PMT Charges - Dirt Muons")
plt.xlabel("PMT Charge")
plt.ylabel("Number of Occurences")
plt.axis([0, 40, 0, 1000])
plt.show()
```

```
TPEs = []
for n in range(len(Tdf.index)):
    TPEs.append(Tdf.loc[n, "numberofPEs"])


plt.hist(TPEs, bins=300, range=(0,5000))
plt.title("Photoelectrons - Tank Muons")
plt.xlabel("Number of Photoelectrons")
plt.ylabel("Number of Occurences")
plt.axis([0, 5000, 0, 15])
plt.show()

DPEs = []
for n in range(len(Ddf.index)):
    DPEs.append(Ddf.loc[n, "numberofPEs"])

plt.hist(DPEs, bins=300, range=(0,5000))
plt.title("Photoelectrons - Dirt Muons")
plt.xlabel("Number of Photoelectrons")
plt.ylabel("Number of Occurences")
plt.axis([0, 5000, 0, 15])
plt.show()
```

8.2 Classifier Algorithm

```
import sys
import glob
import numpy as np
import pandas as pd
import tensorflow as tf
import tempfile
import random
import csv
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from array import array
from sklearn import datasets
from sklearn import metrics
from sklearn import model_selection
from sklearn import preprocessing
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import
ModelCheckpoint
from tensorflow.keras.wrappers.scikit_learn
import KerasClassifier

from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.utils import shuffle

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels



#Create dataframe of both tank and dirt muon data

Tdf = pd.read_csv("tank_muons_data_1000.csv")
Ddf = pd.read_csv("dirt_muons_data_1000.csv")

T_rows = len(Tdf.index)
D_rows = len(Ddf.index)
Total_rows = T_rows + D_rows

Tank_1 = [1]*T_rows
Dirt_0 = [0]*D_rows
Tdf.insert(1, "T or D", Tank_1, True)
Ddf.insert(1, "T or D", Dirt_0, True)

frames = [Tdf, Ddf]
df = pd.concat(frames, ignore_index = True)
df = df.sample(frac=1)
rows = int(len(df.index))
print(df.head())
```

```
#Isolate input and output variables for algorithm
#data = df.values
Input_data0 = df.loc[:,"pmt_charge_0":
"pmt_charge_132"]
Input_data1 = df.loc[:,"T_0":"T_9999"]
Input_data2 = df.loc[:,"numberofPEs"]
Input_data_frames = [Input_data0, Input_data1,
Input_data2]
Input_data = pd.concat(Input_data_frames, axis = 1)
print(Input_data)
Output_data = df.loc[:,"T or D"]
print("Defined input and output data")
print(Output_data)

#Train and Test Splitter
trainX = np.array(Input_data[:1400])
testX = np.array(Input_data[1400:])

dataY0 = np.array(Output_data[:1400])
print(dataY0)
print(len(dataY0))
Y = dataY0
print("before Y.shape ",Y.shape)
Y=Y.reshape(-1)
print("after: ",Y.shape)
print("Y: ",Y[0])

Ytest = np.array(Output_data[1400:])
Ytest= Ytest.reshape(-1)
print(Ytest)
print("Ytest length: " + str(len(Ytest)))

# Scale data (training set) to 0 mean and
 unit standard deviation.
scaler = preprocessing.StandardScaler()
X = scaler.fit_transform(trainX)
print("X.shape: ", X.shape)
print("types: ",type(X), " ",type(Y))
Xtest = scaler.fit_transform(testX)
print("testX.shape: ", testX.shape," Ytest.shape ",
Ytest.shape)
```

```
encoder = LabelEncoder()
encoder.fit(Output_data[:1400])
encoded_Y = encoder.transform(Output_data[:1400])
print("encoded")

def create_model():
    # create model
    model = Sequential()
    model.add(Dense(1000, input_dim=10134, activation='relu'))
    model.add(Dense(60, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam',
 metrics=['accuracy'])
    return model

estimator = KerasClassifier(build_fn=create_model, epochs=100,
batch_size=5, verbose=0)
estimator.fit(X,encoded_Y,verbose=0)

print("Model defined")
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=create_model,
 epochs=100, batch_size=5, verbose=0)))
print("1")
pipeline = Pipeline(estimators)
print("2")
kfold = StratifiedKFold(n_splits=10, shuffle=True)
print("3")
#results = cross_val_score(pipeline, X, encoded_Y, cv=kfold)
#print("Standardized: %.2f%% (%.2f%%)" %
 (results.mean()*100, results.std()*100))

# checkpoint
filepath="weights_bets.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy',
verbose=1, save_best_only=True, save_weights_only=True,
mode='auto')
callbacks_list = [checkpoint]
# Fit the model
```

```
history = estimator.fit(X, encoded_Y, validation_split=0.33,
epochs=50, batch_size=5, callbacks=callbacks_list , verbose=0)

# summarize history for loss
f, ax2 = plt.subplots(1,1)
ax2.plot(history.history['accuracy'])
ax2.plot(history.history['val_accuracy'])
ax2.set_title('Model Accuracy')
ax2.set_ylabel('Performance')
ax2.set_xlabel('Epochs')
#ax2.set_xlim(0.,10.)
ax2.legend(['Train', 'Test'], loc='upper left')
plt.savefig("keras_train_test.pdf")

#ROC curve:
from sklearn.metrics import roc_curve

Ypred = estimator.predict(Xtest).ravel()
print("Length of Ypred: " + str(len(Ypred)))
Ypred =np.vstack(Ypred)
Ytest=np.vstack(Ytest)
#store predictions to csv:
df1 = pd.DataFrame(data=Ytest,columns=['Yvalues'])
df2 = pd.DataFrame(data=Ypred,columns=['Prediction'])
print("check df head: ",df1.head()," ", df2.head())
df = pd.concat([df1,df2], axis=1)
print(df.head())
df.to_csv("predictionsKeras.csv")

Y_probs = estimator.predict_proba(Xtest)
#print("Ytest: ",Ytest[:10]," and predicts ", Ypred[:10])
#print(type(Xtest)," ",type(Ytest)," Ytest.shape ",Ytest.shape,
" Ypred.shape ",Ypred.shape)
fpr_keras, tpr_keras, thresholds_keras =
 roc_curve(Ytest, Y_probs[:, 1]) #Ypred)
#AUC:
from sklearn.metrics import auc

auc_keras = auc(fpr_keras, tpr_keras)

f1, ax1 = plt.subplots(1,1)
ax1.plot([0, 1], [0, 1], 'k--')
```

```
ax1.plot(fpr_keras, tpr_keras, label=
'Keras (area = {:.3f})'.format(auc_keras))
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('ROC Curve')
ax1.legend(loc='best')
plt.savefig("ROC_curve.pdf")
plt.show()

a = 0
b = 0
for i in range(len(df)):
    if df.loc[i, "Yvalues"] == df.loc[i, "Prediction"]:
        a = a + 1
    else:
        b = b + 1

SuccessRate = a/(a+b)
print(SuccessRate)

data = pd.read_csv("predictionsKeras.csv")
print(data.head())
class_names=["Background","Signal"] #= data0["11"].values
print("class_names: ",class_names)
class_types=["Background","Signal"]

positives = data.loc[(data['Yvalues']==1)]
print("P: ", data.loc[(data['Yvalues']==1)].shape)
print("Pred P (TP): ", positives.loc[(data['Prediction']==1)].shape)
print("False N: ",positives.loc[(data['Prediction']==0)].shape)
negatives= data.loc[(data['Yvalues']==0)]
print("N: ", data.loc[(data['Yvalues']==0)].shape)
print("Pred N (TN): ", negatives.loc[(data['Prediction']==0)].shape)
print("False P: ",negatives.loc[(data['Prediction']==1)].shape)

#convert strings to numbers:
#data1 = data0.replace("muon", 0)
#data = data1.replace("electron", 1)
#print(data.head())

def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
"""
This function prints and plots the confusion matrix.
Normalization can be applied by setting `normalize=True`.
"""
if not title:
    if normalize:
        title = 'Normalized confusion matrix'
    else:
        title = 'Confusion matrix, without normalization'

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)
# Only use the labels that appear in the data
print(unique_labels(y_true,y_pred))
#classes2 = class_types[unique_labels(y_true, y_pred)]
#print(classes2)
if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
else:
    print('Confusion Matrix, without normalization')

print(cm)

fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       # ... and label them with the respective list entries
       xticklabels=class_types, yticklabels=class_types,
       title=title,
       ylabel='True Label',
       xlabel='Predicted Label')

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")
    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax


np.set_printoptions(precision=2)

#print("data[11]:")
#print(data["11"])
#print("data[Prediction]:")
#print(data["Prediction"])
#print(class_names[0])
#print(class_names[1])
#print(class_names[2])
#print(class_names[3])
#print(class_names[4])
#print(class_names[5])
# Plot non-normalized confusion matrix
plot_confusion_matrix(data["Yvalues"], data["Prediction"],
classes=class_names,
                      title='Confusion Matrix Without Normalization')

# Plot normalized confusion matrix
plot_confusion_matrix(data["Yvalues"], data["Prediction"],
 classes=class_names, normalize=True,
                      title=' Normalized Confusion Matrix')

plt.show()
```