Can Medipix Analysis Algorithms be improved with Machine Learning?

Toby Freeland
Langton Star Centre

January 13, 2016

Abstract

The purpose of this EPQ is to increase the accuracy of the algorithms used to identify radiation particles from the tracks they create on Medipix detectors. It explores the use of Machine Learning, Computer Vision and geometry to make new and improved programs that can be benchmarked against pre-existing algorithms and humans. Medipix detectors are used extensively in many research projects that utilise the chips in a variety of ways, by increasing the accuracy of the analysis, the scientific value of the research will be improved as uncertainty in measurements will be reduced.

Contents

1 Introduction **3**

2 The Medipix Detector **4**
2.1 Blobs . . . 4
2.2 Classification . . . 5

3 The Algorithms **7**
3.1 Blob Identification . . . 7
3.1.1 Definition . . . 7
3.1.2 Connectivity . . . 7
3.1.3 Connected Component Labelling . . . 8
3.2 Blob Properties . . . 9
3.2.1 Pixel Count . . . 9
3.2.2 Circle Density . . . 9
3.2.3 Length . . . 11
3.2.4 Rectangle Area and Aspect Ratio . . . 11
3.2.5 Polynomial Fitting . . . 12
3.2.6 Convex Hull Sides . . . 13
3.2.7 Energy . . . 14
3.2.8 Tabulated Properties . . . 14

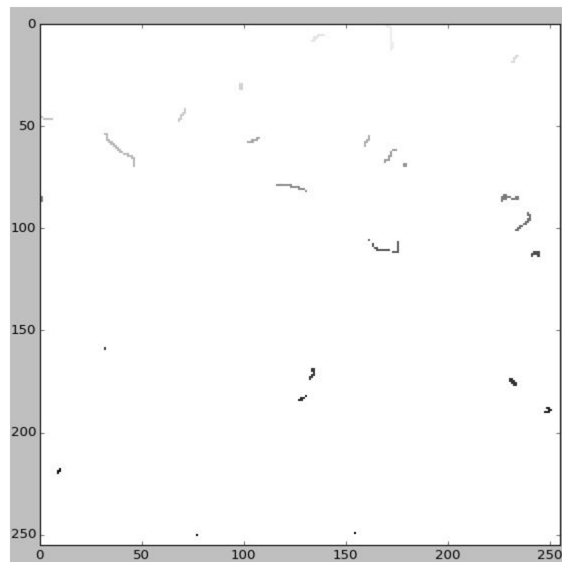
4 Machine Learning **15**
4.1 Supervised Learning 15
4.1.1 K-Nearest Neighbour 16
4.1.2 Support Vector Machine 17
4.2 Unsupervised Learning 18
5 Computer Coding **20**
6 Results **21**
7 Conclusion **22**
A Kernels **23**
References **25**

1 Introduction

A large amount of Radiation Physics research is being done at the Langton Star Centre and other Academic Facilities as well as commercially with NASA using Medipix detectors (and its derivatives). These are detectors developed by the CERN Microelectronics Group that use similar technology to those used at the Large Hadron Collider for detecting subatomic particles. The detectors rely on analysis algorithms to convert the visual output of the detectors to quantitative data. By improving these algorithms I hope to directly help reduce the uncertainty of all these studies and therefore improve the scientific benefit as well as the human benefit as the uses of Medipix include medical imaging.

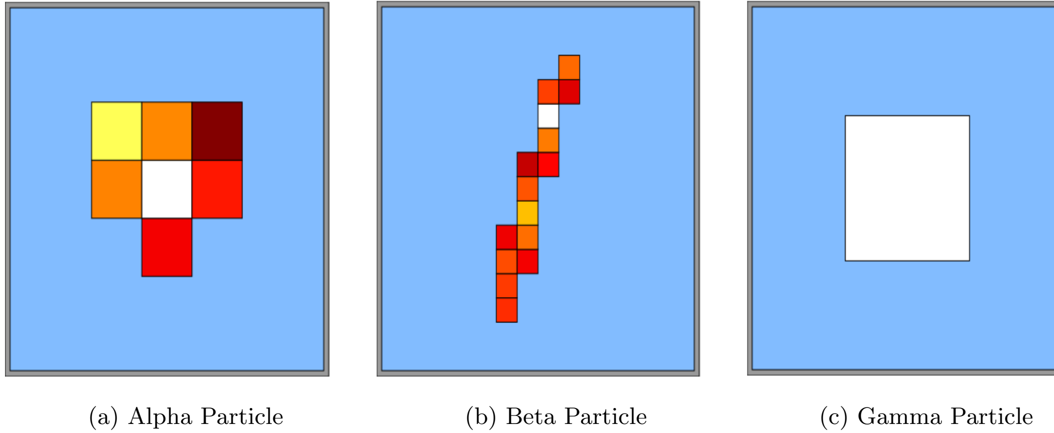
2 The Medipix Detector

Medipix chips have 65,536 individual pixels. When an ionising particle interacts with the chip it deposits charge in the silicon of the detector. This charge is slowly removed by a voltage across the detector. Each of the pixels records the time that it takes for that charge to be reduced below a threshold, so we call the output Time over Threshold or ToT. The more energy the particle had, the more charge deposited, the longer it takes to bring below the threshold so a higher ToT. The chip outputs a file in a format known as XYC: the X co-ordinate, Y co-ordinate and 'charge' referring to ToT. We can use this to create visual representation of each capture that the detector makes, we call this a frame. Any pixel in a frame that has a zero ToT is ignored, but pixels with a nonzero ToT are normally displayed on a grid with their colour scaled to their ToT, as shown with the example particles in Section 2.1. However for reasons explained in Section 3.2.7 I will not use ToT so I have coloured the frame to highlight connected areas. [1]



2.1 Blobs

One will quickly notice that patterns emerge in the frames. In general pixels that are touching were activated due to the same particle depositing charge. I will refer to a set of touching pixels as a blob. Blobs come in all shapes and sizes however due to the way different particles interact with the silicon in the detector, blobs will often have similar shapes and sizes when they have been caused by the same type of particle. The 3 main ionising radiation particles detectable on earths surface are Alpha, Beta and Gamma so I will mostly concentrate on these during my EPQ - however for research in diffrent environments such as space as on the ISS or LUCID experiment, adaptations may have to be made. I have included 3 images of the typical blobs that these particles create.



(a) Alpha Particle (b) Beta Particle (c) Gamma Particle

Figure 1: Comparison of Alpha, Beta and Gamma tracks on a Medipix chip (Note the difference in scaling, one square is one pixel on the frame)

2.2 Classification

While these blobs make pretty pictures, one has to remember their purpose is for science and valuable information can be extracted from blobs. For an example let me give you the data requirement for my own project CERN@sea that utilises Medipix Chips:

- Take frames at 0.1 Hz rate for 6 months along with other meta-data.
- Identify every particle in each frame.
- Generate statistics on particles and their relationship to locations and times.

To quickly understand this task, one must approximate the time taken to analyse this data by hand. For simplicity's sake I will say it would take a person trained on an efficient analysis interface 10 seconds to manually identify every particle per frame. Therefore it would take 6 months (as 10 x 0.1 cancel out) of constant work to manually analyse all the frames. This is obviously far too big a task to undertake. So instead computer methods have been employed to do this task. There have been many successful attempts at creating an analysis platform.

I often use the analogy of trying to get a biologist to identify an animal that you have a photo of over the phone. So the set of steps or algorithm that would be used to identify my animal would be:

1. Identify where the animal (or animals) are in the photo
2. Find out some properties of that animal e.g. Number of Legs, Size, Colour...
3. Tell these to the biologist and they will classify based on the properties and their knowledge

However in our case it is much more simple as there are only around a dozen possible ionizing radiation particles rather than the millions of different animal species. Also over the phone the biologist would ask questions based on the answers of previous ones, following a classification flow chart - this would lead to a quicker classification because time would not need to be wasted on redundant properties, for instance if the biologist determined its a fish - they don't need to waste time by discussing numbers and positions of legs. However for this EPQ I will not consider computation time (which this is analogous to). So if we apply this methodology to particle recognition, the task of a computer analysis software can be sub-divided into 3 sections:

1. Blob Identification
2. Computation of Blob Properties
3. Classification of Blob based on those Properties

For each step I will evaluate how other classification methods approached it and then how I chose to implement it.

3 The Algorithms

In this section I will proceed to discuss individual parts of the algorithm. I will discuss how I first researched what I thought was the best method (sometimes correctly, often completely incorrectly) and then compared my solution to that of other programmers who have attempted the same task.

3.1 Blob Identification

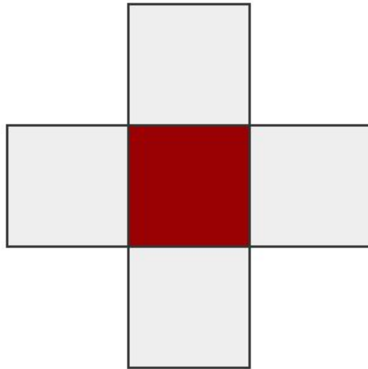
3.1.1 Definition

When processing a frame it is crucial that blobs are identified. We can mathematically define a blob as a set of pixels which are activated and adjacent. With this definition a few problems arise:

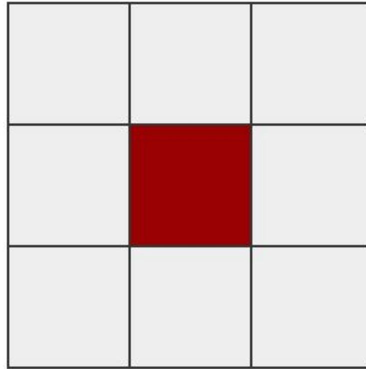
- When tracks cross over, it will only find one big blob.
- Tracks which have missing pixels in will be identified as separate blobs.

3.1.2 Connectivity

A quick note must be made in how we define two pixels in a frame to be connected. In a 2D problem such as is the Medipix frames, there are really only 2 options (though I will explain later the possibility of using others): 4 or 8 connectivity.



(a) 4-Way Connectivity



(b) 8-Way Connectivity

To decide between these two, one simply needs to refer to a long beta track that was diagonally across the frame. If we take Figure 3 as an example, a human would identify all the activated pixels in the figure as a blob (even though they are not all touching). A 4-Way connectivity set up would find 4 different blobs and an 8-Way connectivity would find 3.

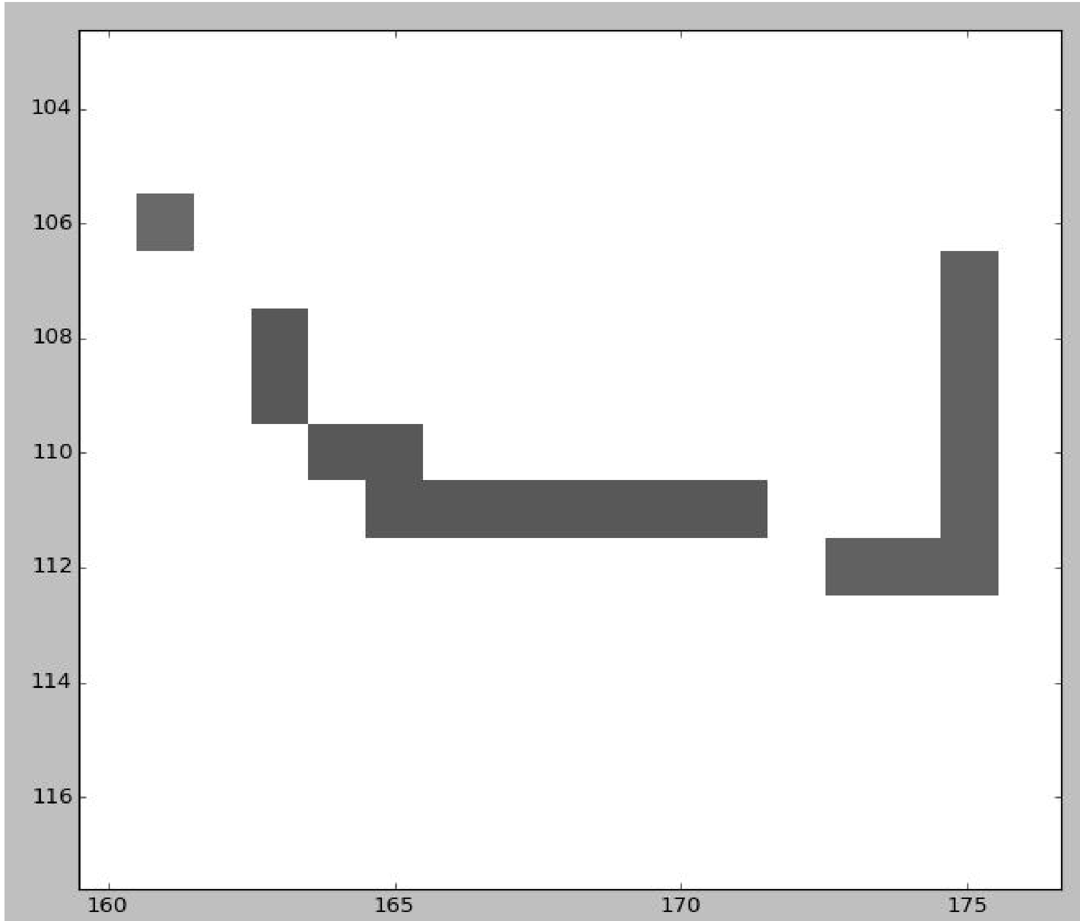


Figure 3: An obvious case for 8 way connectivity

While I only suggested two types of connectivity, one could suggest more neighbours such as 20. This would enable blobs that seem to have missing pixels as in the frame above to be classified as one big blob. However this has the obvious disadvantage of classifying blobs that run close to each other on the frame as the same which quite often they're not.

Clearly there has to be a trade-off here, I chose to implement an 8-Way Connectivity model. However even if I developed this further to maybe have a more advanced and intelligent labeller, there will always be a debate to if pixels should be included in a blob or not. For instance, how many blobs do you think there are in Figure 3?

3.1.3 Connected Component Labelling

I chose to implement a One-Pass Connected Component Labeller for its simplicity to understand. The algorithm follows the steps:

1. Scan through frame's pixels from left to right, top to bottom
2. If pixel is activated (ToT NOT 0) then label it with the current blob label (start at 0)

3. Add all of its activated and unlabelled neighbours to a queue

4. While the queue isn't empty, serve[1] it and repeat step 3.

5. Now queue is empty, increase current blob label by one. Go back to Step 1

Every time the list is empty, a blob has been completely discovered and each pixel in the blob has been labelled. It is then an easy task to put all pixels with the same label in a list that can be passed to the Blob Property algorithms that are defined below.

3.2 Blob Properties

Now we have identified all the blobs on a given frame and the pixels in each blob, we now need to extract data from the blobs so that they can be run through the machine learning algorithms. For each of the properties defined that I actually used, I have given a table of example values for the properties in Section 3.2.8.

3.2.1 Pixel Count

Definition: The number of pixels in a blob
Pixel Count was one of the first properties I worked on and implemented due to its simplicity and impressive classification ability [7] because in general:

$$\textit{PixelCount(Gamma)} < \textit{PixelCount(Alpha)} < \textit{PixelCount(Beta)} \tag{1}$$

This is a very simple quantity to calculate and can be defined as the size of the set that contains all the pixels in the blob. In my research all other implementations worked in a similar way so was very simple to program myself:

```
self.pixelCount = len(self.pixels)
```

3.2.2 Circle Density

Definition: The Pixel Count divided by area of the smallest circle that fits around the blob
I originally began working with circles and blobs because I wanted to find an algorithm for evaluating the property: 'Radius' as a substitute for length however during my research and implementation of circles I decided against this idea and looked at length in its own right which is discussed later. Not wanting to waste my hours of work on circles I instead used them to evaluate Circle Density.

When I started my implementation I used the idea that a circle needed a radius and a centre and it would be good enough for all my calculations and diagrams so the method I used was:

1. Find the centre of the blob (By 'centre of mass' - averaging all coordinates)

[1] Take the first item that was added to the queue because a queue is a First In First Out (FIFO) data structure.

2. Find the furthest away pixel of the blob from the centre, this distance is the radius

I implemented this algorithm and because I now have a centre and a radius I can plot the circles on top of a frame. The result wasn't as expected:

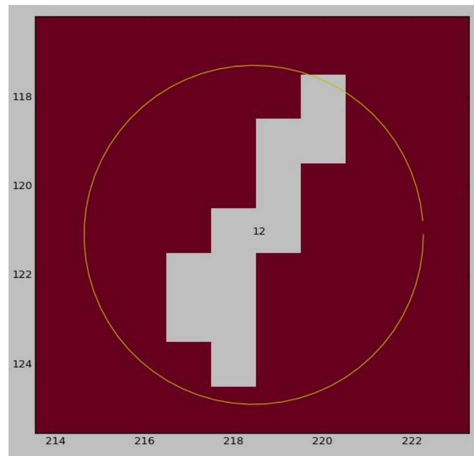


Figure 4: Poorly fitted circle

You will notice that the circle does not fit the blob correctly and is "off centre", this is because the blob would have to have two pixels that are opposite the center at the equal maximum distant for the circle to fit properly. This obviously does not fit our definition, so what has gone wrong? It is all because the algorithm for center I used, corresponds to the center of mass and not the center of the circle. So I re-wrote those original steps for finding the radius to read:

1. Find the 2 pixels that are furthest away

2. The centre is midway between these points. The radius is half this distance

Again I plotted the circle on a frame and found this curious Alpha:

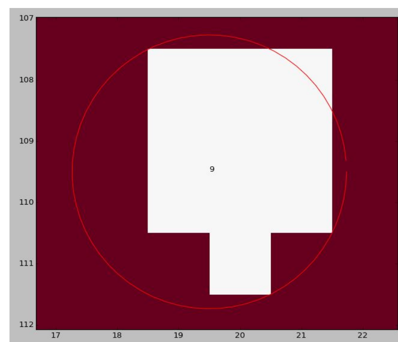


Figure 5: Another poorly fitted circle

At this point I realised how naive it was to assume that I would only need to look at the two furthest points, because you need 3 points to define a circle (as with less, there are an infinity of circlines[2]). At this point I began researching the problem and found its name: Smallest Enclosing Circle Problem. I used an implementation that worked by starting with one point,creating the smallest circle that lay on that point, and adding points outside of the circle one by one, knowing they must be on the circumference. This performed very well:

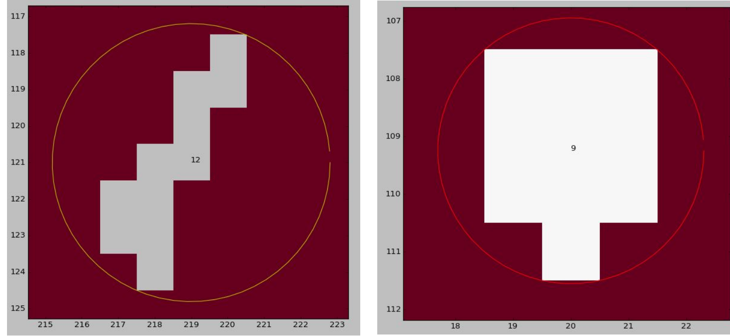


Figure 6: Well fitted circles

Now that I had successfully found the smallest circle that enclosed all the pixels in the blob, I can calculate the density with the simple equation:

$$\text{density} = \frac{\text{pixelcount}}{\pi \text{radius}^2} \tag{2}$$

This gives a number between 0: Impossible, but 0 is approached with a long straight line such as a muon and 1: a circular blob.

3.2.3 Length

Definition: The furthest possible distance between any of two pixels in the blob

I wanted a property that describes the length of a blob on the page. There are many ways of representing length such as the diameter of the smallest fitting circle of a blob or the line of best fit length. However for simplicity and ease of programming I took the furthest distance between any two points in the blob.

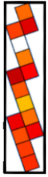
This means I had to check all possible pairs of pixels in a blob of which there are $n^2 - n$ where n is the number of pixels in the blob. For each pair I calculated the straight line distance using Pythagoras' Theorem: $d = \sqrt{(\Delta x)^2 + (\Delta y)^2}$.

3.2.4 Rectangle Area and Aspect Ratio

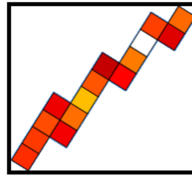
Definition: Rectangle Area - The Area of the smallest rectangle that fits around the blob. Aspect Ratio - The ratio of the longest side to smallest side of the rectangle

[2] Circline: Maths speak for circle or line - because in geometry its not such a silly idea to think of a straight line as being a circle with an infinite radius

In discussion with various people during my research, people would discuss particles with adjectives such as wide or short. To use those adjectives you have to compare the lengths of the blob in 2 perpendicular directions. Therefore to make this comparison I needed to calculate the effective height and width of a blob - this is effectively approximating the blob as a rectangle. I could do this by finding the pixels with smallest and largest x and y values then finding the rectangle with these 4 points as the corner. Unfortunately this has some issues as the height and width dimensions are fixed to the axis of the frame, not to the blob. This makes the property orientation dependant - this is not ideal because particles should not be classified differently just because they cross the frame at a different angle. See the figure below for a graphical demonstration of this, it uses the same beta but with different rotations:



(a) Vertical Beta: Height/Width = 4.2



(b) Vertical Beta: Height/Width = 1.1

As you can see this is not a good enough method for classification as the Height/Width value changes dramatically with orientation. I used code written by David Butterworth from the University of Queensland, which implements an algorithm for finding a 'minimum area bounding rectangle'. With this I could calculate the height/width or Aspect Ratio of each blob. I also thought that rectangle area would give an alternative to that of the circular area calculations in Section 3.2.2, so included this in the database.

3.2.5 Polynomial Fitting

Definition: How well does a polynomial line fit the blob

When discussing blobs, its 'straightness' was often discussed as high energy particles would often leave long straight blobs in the detector compared to alphas were very non-linear. I could have used the Least Squares Regression Line, however the NumPy module in Python comes with a function 'polyfit' which fits a polynomial to a dataset. A straight line is a polynomial of order 1, so by giving the polyfit function the order argument '1', the function will find a straight line which fits the data the best.

$$\text{It does this by minimising: } \sum_{j=0}^{k}(p(x_j) - y_j)^2 \quad (3)$$

Where $p(x)$ is a polynomial of x, k is pixel count and x and y are the coordinates of pixel j.

This got me thinking and I tried using order 2 polynomials which take the form: $y = Ax^2 + Bx + C$ and are known as quadratic functions. This was also useful as many blobs have slight curves to them. Here is an example of a quadratic and linear equations produced by the NumPy function on a long beta particle.

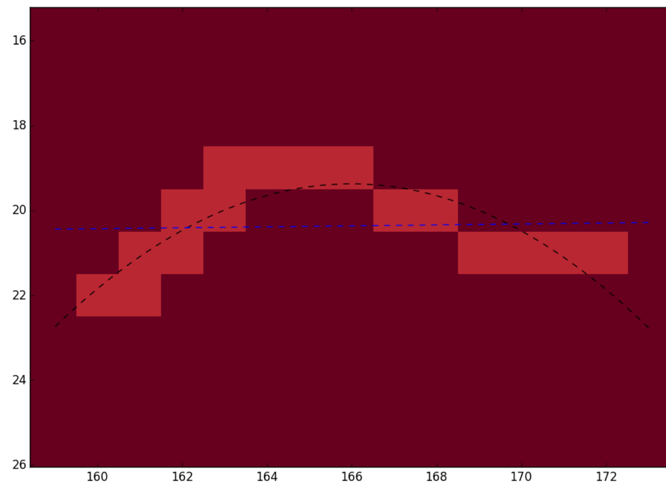


Figure 8: Polynomial lines on a beta: Quadratic and Linear

While this looks very effective on the visual output, my machine learning algorithm needs numerical values so I simply took Equation 3 which returns a large value for a badly fit polynomial and a lower number for a well fit one. So this number I called Non-Linear and Non-Quadratic.

3.2.6 Convex Hull Sides

Definition: The number of sides that the convex hull of the blob has

When I generated the rectangles for Section 3.2.4 I had to find the angle to rotate the rectangle so it would form the minimum enclosing rectangle. My original plan was just to rotate the rectangle through small increments and arrive at an approximation for the rectangle[3] however during my research into the minimum enclosing rectangle I found the following statment online [3]:

> The smallest-area enclosing rectangle of a polygon has a side collinear with one of the edges of its convex hull. [4]

The convex hull of a set of points can be best of thought as if all the points were pegs on a pegboard and you stretched an elastic band around the pegs, forming a shape that encloses all the pegs. This shape is the convex hull of those points.w This lead me to investigate the convex hull further for its uses in solving problems with the rectangles. After implementing the algorithm for determining the convex hull for a blob. I wondered what other uses I could make with it. In the end I just took the number of sides in the convex hull and used this as a property in my database.

[3]This of course would have issues with local minimums and could be horribly incorrect if the optimum solution wasn't steadily approached but instead existed suddenly at a specific angle not in the set of angles being iterated through

3.2.7 Energy

Definition: The sum of energy (or its equivalent)

Each pixel has a Time Over Threshold (ToT) value in the detector chip output. This can be used to calculate the energy disapated in the pixel, provided you know the bias voltage[4] and the Threshold value[5]. In general $Energy \propto Charge \propto ToT$. By summing the energy for each pixel in a blob, a value for the amount of energy deposited in the chip by the ionising radiation can be calculated and used to compare blobs. However the constants used to convert from ToT to Energy aren't the same for every pixel on a frame, this means the chip must be calibrated by finding the constants for each pixel before energy can be a reliable property. Unfortunately calibration is an expensive and time consuming process and isn't available to all detector chips (i.e. ones orbiting on a satellite). So for these reasons I will not use energy or the sum of ToT in my algorithm. But given more time it would be a very interesting area to research further as often particles of the same type will have similar energy so if a blob was found with an integer multiple of that energy then it could suggest two particles of the same type have overlapped.

3.2.8 Tabulated Properties

To give a feel of typical of values that these algorithms produce, I ran them on the 3 example blobs in Section 2.2 and produced these results:

Particle Type	Pixel Count	Length	Circle Density	Rectangle Area	Aspect Ratio	Non-Linear	Non-Quadratic	Convex Hull Sides
Alpha	7	3.61	0.62	9.00	1.00	0.57	2.00	6
Beta	15	11.71	0.14	28.87	4.72	0.33	1.48	8
Gamma	1	1.41	0.64	1.00	1.00	0.00	0.00	4

Only 3 pieces of data would never be enough to classify and test with. So I prepared a SQL database using the SQLite3 module in Python to interface to a database with similar headers as the table above. This allowed would allow me to store large numbers of blobs separately from Python as I was unsure until I began the machine learning algorithms how many blobs I would need. For instance when I was following a tutorial to learn how to use the python module: Sci-Kit Learn, numbers of 10,000 training sets were quoted. [5] I did not want to implement storage for over 90,000 (9 columns x 10,000 rows) pieces of data in Python myself, so using SQL's powerful databasing tools was a massive help.

[4] potential difference across silicon - this determines how fast the charge is removed

[5] How high the charge has to be in the pixel to let the timer run

4 Machine Learning

Using the property algorithms defined in Section 3.2, I created a dataset of 30,830 blobs from 1,122 frames provided by CERN@school projects and elsewhere. In Machine Learning there are two ways a machine can classify data either by Supervised Learning or Unsupervised Learning. In Supervised Learning, the machine is given labelled training data so for our purposes the machine would receive a large number of blobs alongside the desired output known as the target. For example:

$$(7, 3.61, 0.62, 3, 1, 0.57, 2, 6),(\text{"Alpha"}) \tag{4}$$

The first set is the properties of the blob (pixelCount, length, density etc.) and the second is a string referring to the output we expect from the algorithm. This is analogous to saying to the computer "Here is a blob, it is an Alpha particle" over and over again 30 thousand times! Now the algorithm has hopefully "learnt" from the dataset, we can give it new unseen blobs and test if it can correctly recognize and classify them.

The alternative, Unsupervised Learning, just gives that first set without the target and is like saying "Here is a blob" over and over again instead - you must also tell the algorithm how many types of particle you expect there to be so we might enter 3 for example (referring to Alpha, Beta, Gamma). This algorithm instead tries to split the data given to it into 3 different types by finding 3 groups of similar particles.

None of these algorithms ever know or care about what each value represents, which means in general they treat each value with equal weighting. This can be a problem when different properties have very different ranges for example in my database Pixel Count varies between 1 and 58 whereas Circle Density varies between 0.04 and 0.79. I read about potential issues comparing properties with different ranges [2] so I chose to make use of SciKit-Learn's Preprocessing functions . With these I scaled every property so to have an average value of 0 and I also changed the distribution so the Standard Deviation of each property was 1. This made every property more comparable by the algorithms and in some cases made the algorithms around 10% more accurate.

4.1 Supervised Learning

Before I discuss the supervised learning methods in more depth, I will visually represent this data for you to 'learn from' so you can understand the task that the computer has to do. Unfortunately the computer has 8 variables to plot for each blob, this is far too many dimensions for the brain to process, so I have only plotted Density against Pixel Count.

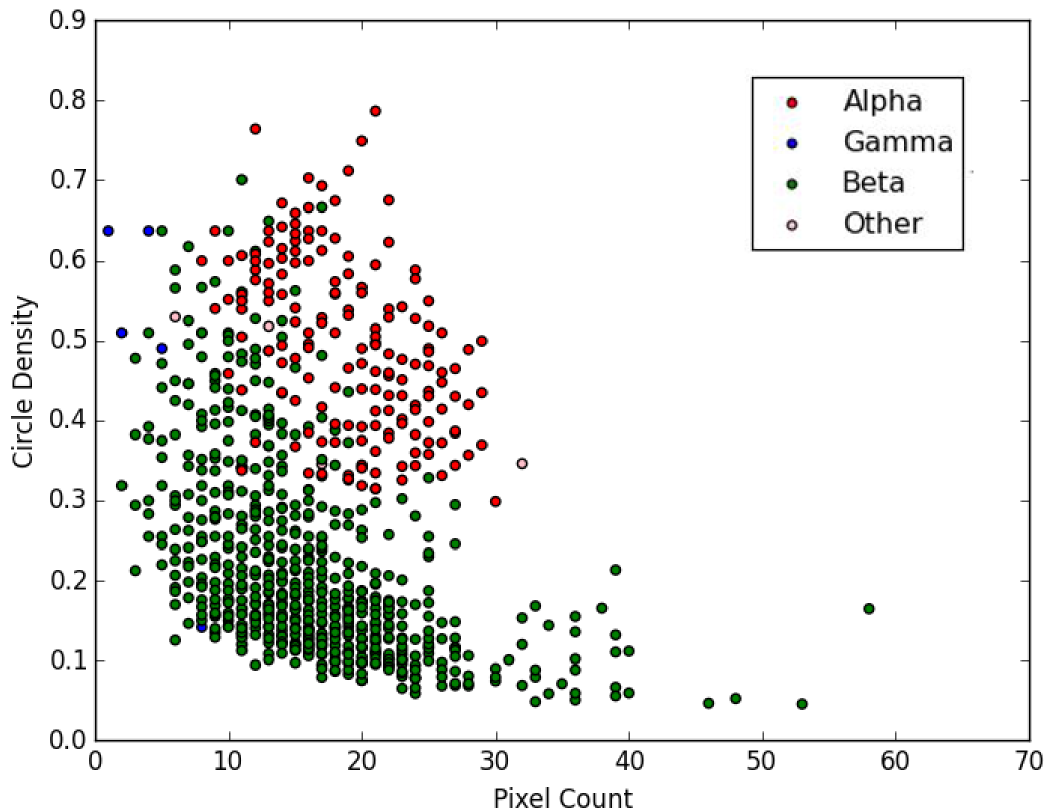


Figure 9: 2D scatter plot of unique values in database

I looked at 2 Supervised Learning Algorithms: K-Nearest Neighbour (KNN) and Support Vector Model (SVM).

4.1.1 K-Nearest Neighbour

This algorithm is often thought of as the simplest of all machine learning algorithms. You can think of it as plotting all the training data in the database onto a set of axes (although this may be in 8 dimensional hyperspace for my usage but for ease of mental processing - we will use the 2D graph in Figure 9). This is all the machine does in terms of learning so we call it 'lazy learning' because it now waits for an unknown blob to be given to it for it to classify.

When a blob is given to it that needs to be classified - it finds the position where it would have been plotted on the graph and then looks outward from that position to the nearest k neighbours. Now we have defined a "neighbourhood" we have different options to what we do with the neighbours. The simplest is to take the majority vote of all those neighbours and whichever wins is the output of the classifier however you could weight the closer neighbours (give them more votes) more than the further

away ones. I did this by using the inverse of the distance from the neighbour to the test point. In my research I had no idea which method would achieve the best results nor did I know what values of k would work the best so I plotted this graph:

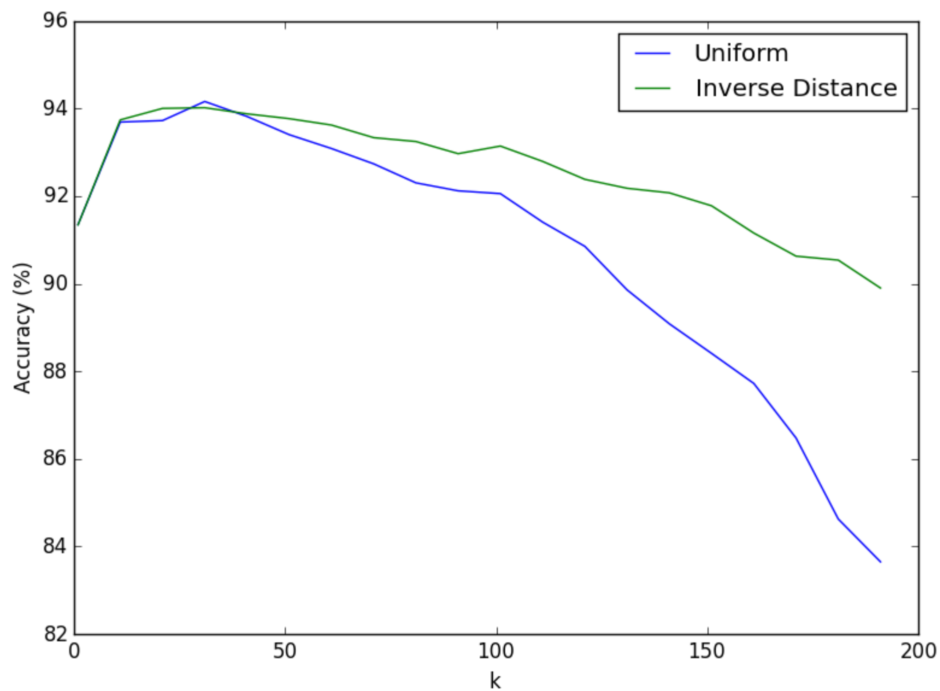


Figure 10: A graph to show how the accuracy changes with the number of neighbours in neighbourhood (k) and with weighting. 500 repeats per data point and 500 random training examples from database

This clearly shows optimum values for both a ”uniform” and ”distance” weighting but not much in it between the different weighting methods. For comparison with other methods - I used a Uniform Weighting and a k value of 28.

4.1.2 Support Vector Machine

An SVM starts of in a similar way to a K-Nearest Neighbour algorithm by plotting all points in higher dimensional space. However instead of stopping here as with the lazy KNN, it now constructs a set of hyperplanes (like a multi-dimensional line on our 2D graph) that divide the plot of training data as best they can into a number of sections, where the number is equal to the number of classifications that are possible. Now it waits for a training data to be given to it, where it simply finds which section the training data would have been plotted in and returns that section. There are some fancy tricks involving kernels (See Appendix A) which are useful if a section would form a 'ring' around another section. This is impossible for a linear hyperplane to deal with correctly so kernels can be used for this purpose. However because I didn't feel this would apply to my dataset - I used a linear SVM.

I used this to test my database when I first generated it and was curious on the effect of the amount of training data on the accuracy of the classifier. If I found the classifier only became accurate with 1000 or more training examples then I would have to get more data as my database only contained 768 unique blobs. So I ran this test which shows how accuracy is dependant on training set size. For each size I tested I did 200 repeats and took 100 blobs not used in the training set from the database and tested if the classifier outputted the same answer as stored in the database for that specific blob. The percentage of these that were correct is labelled on the graph as Accuracy.

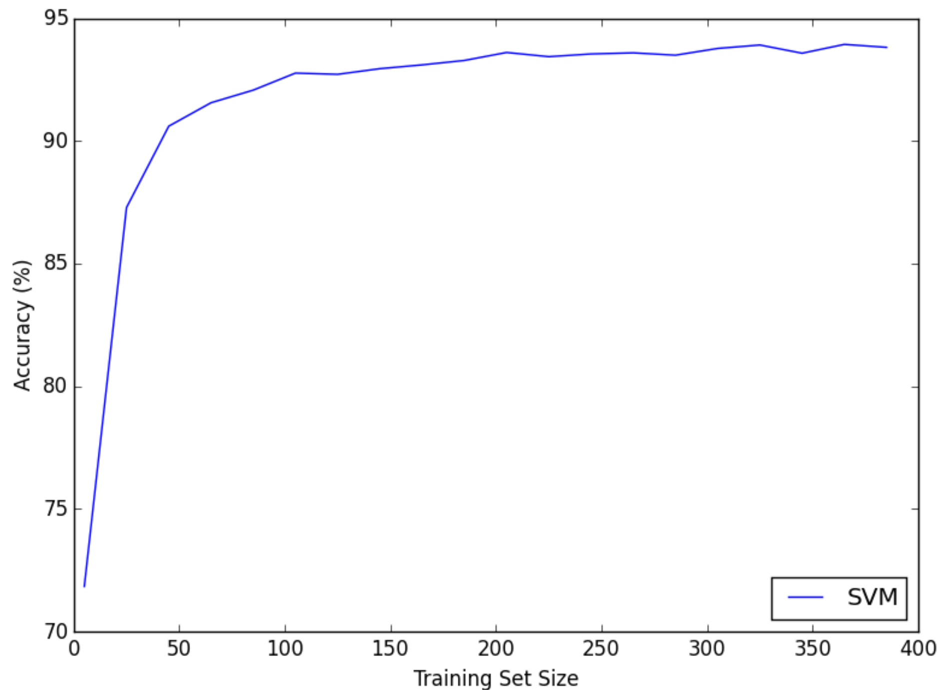


Figure 11: A graph showing how the accuracy of a classifier is dependant on the training set size.

There are 2 points that really surprised me when I generated this graph. Firstly it is still reasonably accurate with only 5 training blobs with an accuracy of 71%. Secondly that the accuracy rose above 92% with only 80 learning blobs. This surprised me as in my research vast numbers of hundreds of thousands were quoted as good training set sizes. [2]

4.2 Unsupervised Learning

Supervised learning relies on giving a fully classified training set, the classifier will only ever classify new blobs into one of the pre-existing classes. However lets say a new cosmic particle interacts with our detector, we may never know it hit the detector if it gets automatically classified as a beta lets say and if it falls below a certain threshold then reject the classification and flag it up to a human expert or to a citizen science programme similar to Galaxy Zoo. The alternative is to let the classifier classify all

the data into as many groups as it feels necessary to distinguish blobs for instance 4 groups for Alpha, Beta, Gamma and Crossed Blobs. I implemented an Unsupervised Learning known as K-means. It works in a similar way to K-Nearest Neighbour but instead works with 'mean points' of groups that it iterates through and adjusts the means to better fit the data. As much as I tried to make this work I never got an accuracy higher than around 65%. This was also hard to actually quantify as the K-Means classifier labels the groups with a numeric value between 0 and K. I originally had to map these to the ["a","b","g"][6] I have stored in the database by hand. That got tedious quickly so I went through all the different possible mapping of the two sets of which there are K![7] and took the one with the highest accuracy.

[6]When K = 3, this set was used

[7]! means Factorial. A mathematical operation such that: $n! = 1 \times 2 \times 3 \times ... \times n$

5 Computer Coding

I programmed, tested and implemented all of these algorithms in Python 2.7, this is a widely used programming language with very readable code which made it quick for me to learn. I used a variety of modules such as NumPy, MatPlotLib, SciKit-Learn, Collections, OS and SQLite3. These are all powerful pre-written pieces of code used by programmers to speed up their development time. They enabled me to plot graphs, run machine learning algorithms, build databases and deal with complex data structures without the extremely time-consuming process of writing it myself - most of these modules have been developed for years.

I wrote 2 main programs AnalysisTools.py and MachineLearning.py. The Analysis Tools implemented all the algorithms in Section 3. It used Object Oriented Programming (OOP) by having a Frame Class and a Blob Class. This was the first time I had used OOP in a major project and it became very useful as it kept my code tidier.

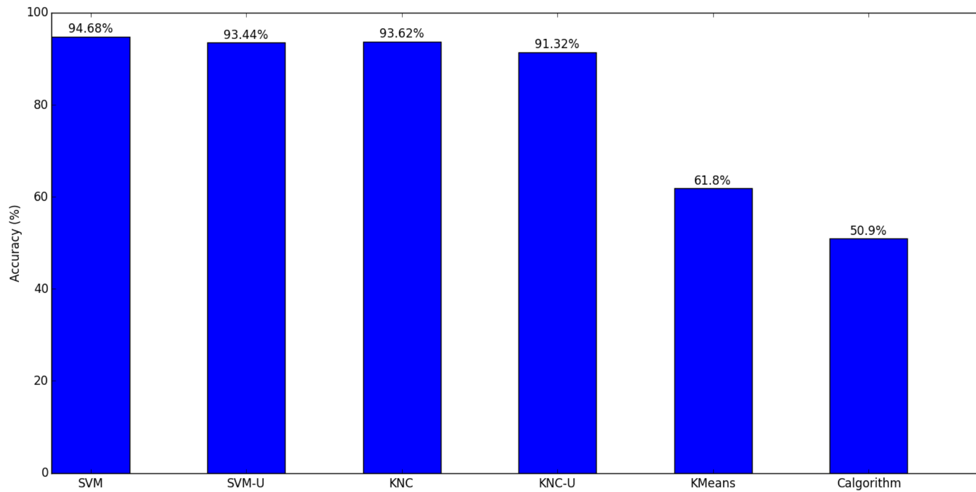
The Machine Learning code read from the SQL database and heavily used the SciKit-Learn and MatPlotLib modules to run machine learning tasks over the database and output graphical representations as seen in Section 4.

I defined over 55 functions in the code, most of which took many attempts to implement, as detailed in previous sections. Functions are small contained programs that take inputs, process them and give an output. I defined functions so that I could re-use code all over the two programs. For instance a function which displayed the frame could be called by many different parts of the code when I wanted. This saves repeating large amounts of code. In total I wrote over 1,100 lines of code however all the functionality could be used in very few lines, for example:

```
1  #Import all the code from the two files.
2  from AnalysisTools import *
3  from MLTools import *
4
5  #Establish a new frame.
6  newFrame = Frame("...Path to a XYC file...")
7
8  #Display the frame with the circles and convex hull.
9  newFrame.plot(hulls= True,circles=True,)
10
11 #Update the SQL database with any new files in folder.
12 updateSQLdb()
13
14 #Print the average accuracy of a Support Vector Machine algorithm over 20 repeats
15 #with a training set of a 100 randomised processed blobs.
16 print(SVM(100,20,process=True))
17
18 #Plots graph of accuracy of algorithms in code.
19 compareAlgorithmAccuracy()
```

6 Results

To compare how well all the individual machine learning algorithms compared I ran each one with a large randomized training set size and the remaining blobs from the database as the test set. I used both the pre-processed data and the unprocessed data[8] to compare their effects on the algorithms, I gave the unprocessed versions the tag "-U". I also ran the algorithm currently used by the CERN@school data storage and analysis platform: TAP@S. This algorithm was developed by Tom Whyntie and re-written for my Python by Cal Hewitt so was affectionately known as the 'Calgorithm'. This has since been improved with a new decision tree and I look forward to testing this again also.



The Calgorithm can be called a 1R or 1-rule decision tree. These often yield very good results as was the subject of a paper [7] that showed how only using one property can classify very well - this was the premise for the hypothesis in Equation 1. The values used in the calgorithm have been approximated by hand and also account for a far larger range of particles than just alpha, beta and gamma. I believe that with minor adjustments to the algorithm a reasonable improvement could be made to the accuracy.

However overall both the K-Nearest Neighbour (4.1.1) and the Support Vector Machine (4.1.2) algorithms scored admirably. With the processed data input increasing the accuracy of both by 1% or 2%.

[8]See Section 4

7 Conclusion

In this EPQ I worked out how to process the raw data from Medipix chips. Then find all the tracks on the frame and work out properties of each track. I designed a database to contain a large number of example tracks and their properties. A subsection of these could then be put into a certain breed of algorithms known as machine learning algorithms. I tested multiple different types of machine learning algorithms against each other. While there were only 2 that achieved high results, the accuracy was way into 90%s with the highest accuracy of 94.68% over 50 repeats.

During this EPQ I encountered and developed a huge number of computer vision and machine learning methods, most of which I did not have enough time to develop and investigate further. These ideas include:

- Loading blobs visually into specialised machine learning algorithms. This could be done by converting the set of pixels in a blob to an 2D array the size of the blob. However care would have to be taken to standardize the input. For example have the same center and orientation.
- Look at multiple tracks crossing over to form one blob - by using blob templates or expected values.
- Look at calculating 'endpoints' of a track to determine how many particles (or if any nuclear event took place during interaction) as a beta has 2 endpoints, if 4 are detected it would imply a 2 beta crossover.
- Implementing a "confidence" value for every classification so that particles which do not seem to fit can be dealt with Citizen Science or other classifiers.

A Kernels

Kernels are functions which compute another layer of processing onto the data to make them much more simple for the machine to understand. For instance a Radial Based Function (RBF) Kernel moves data from circular domains to linear ones this is much easier for a machine to learn from (We can teach an 11 year old the equation for a line but it will take them 5 more years to learn the equation for a circle!). For instance, if we are trying to separate the red and blue points in Figure 12, they can be separated with a straight line:

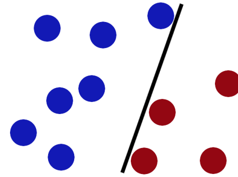


Figure 12

However some sets of data, while its clear are part of some pattern, can not be separated by a straight line.

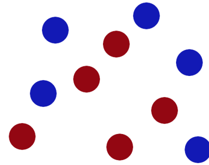


Figure 13

So we can use a kernel, this moves all the pixels from the 'Input Space' to the 'Feature Space'. In this case this has been made possible by using a third dimension. It is now possible to separate the red and blue points with a plane (because that is a line in 3-dimensions). If we use the plane as the input to the inverse of the kernel, we can transfer it back to the original 2D problem. You can see this is in the black line on the 'Input Space'

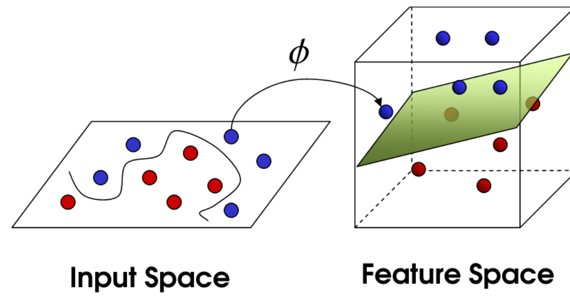


Figure 14

We have now successfully separated the red and blue dots by using a kernel.

All diagrams from a Reddit Post by user "copperking". [8]

References

[1] Llopart, Xavier and Ballabriga, Rafael and Campbell, Michael and Tlustos, Lukas and Wong, Winnie
Timepix, a 65k programmable pixel readout chip for arrival time, energy and/or photon counting measurements
Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 2007

[2] Ian H. Witten, Eibe Frank, Mark A. Hall
Data Mining: Practical Machine Learning Tools and Techniques
Third Edition, 2011

[3] David Geier
Computing oriented minimum bounding boxes in 2D
`https://geidav.wordpress.com/2014/01/23/computing-oriented-minimum-bounding-boxes-in-2d/`
2014

[4] H. Freeman, R. Shapira.
Determining the minimum-area encasing rectangle for an arbitrary closed curve.
Communications of the ACM, 18 Issue 7, 1975

[5] Scikit-learn developers
Choosing the right estimator
`http://scikit-learn.org/stable/tutorial/machine_learning_map/`
2014

[6] Scikit-learn developers
Scikit-learn documentation
`http://scikit-learn.org/stable/documentation.html`
Accessed 2015

[7] R. C. Holte
Very simple classification rules perform well on most commonly used datasets
Machine Learning
1993

[8] copperking (Reddit username)
`https://www.reddit.com/r/MachineLearning/comments/15zrpp/please_explain_support_vector_machines_svm_like_i`
Posted 2013, Accessed 2015