Known Issues :

First and foremost, I want to point out some strange unexpected behaviour that I ran into from the program. There is a function called selectHyperParams which depending on the function name that we pass to it, does cross validation over 5 passes, and one epoch, with that function.
After choosing the parameter, I run the training experiment with that parameter as the passed hyper parameter.

For some reason, **simply calling the selectHyperParams function changes the result shown by the second call to the training perceptron.** Even when I'm not passing the hyper parameter selected by the previous function. So while **without running the function, it shows number of mistakes to be around 60, after calling the function it shows number of mistakes as about 260.**
ALL my variables other than numRuns have been defined inside the context of a function.

As a work around, I have commented the selectHyperParams function in the second part of the experiment (Although, thankfully as per the language, it felt more like we could try out a few values and plug in what we thought best)

I also did the experiment while commenting out this particular function in the rest of the code with hard coded hyper parameters, and I think I got better results. If you have any insight, do let me know.

Experiment 1: Sanity Check

The sanity check works out. The weight vector shows the most significant attribute as **X4 with a weight of more than 6** for a rate of 3. Far more than any of the other vectors.

Experiment 2: One Pass for Normal & Margin Perceptron

Three functions are used here.

1. trainPerceptron() : Accuracy on **test** set : **1 training** set : **0.969**
2. trainMarginPerceptron(): Accuracy on **test** set : **0.999 training** set : **0.912**
3. testAnyPerceptron() : Tests the perceptron. Takes weight and records as params

For both, by default, if no weight vector is passed, the function generates it randomly. However a particular initial weight vector can also be passed to the function. I tried a number of values for the hyper parameters, and left the ones that worked best. An issue I came across has been mentioned above.
As per the algorithm of the perceptron, both the functions call 2 methods the dotProd which returns the dot product of the 2 passed vectors (Xvector and Weight vector), as well as the weight vector. The weight vector can be updated using the updateWvec function which multiplies the Xvector with the label, and rate value and adds it to the old weight vector value.

While for the perceptron, this change is made when y*dotProduct < 0, for margin it is for <Mu where Mu is a margin value.

Experiment 3: 10 Epochs for all data sets

Five functions are used here.

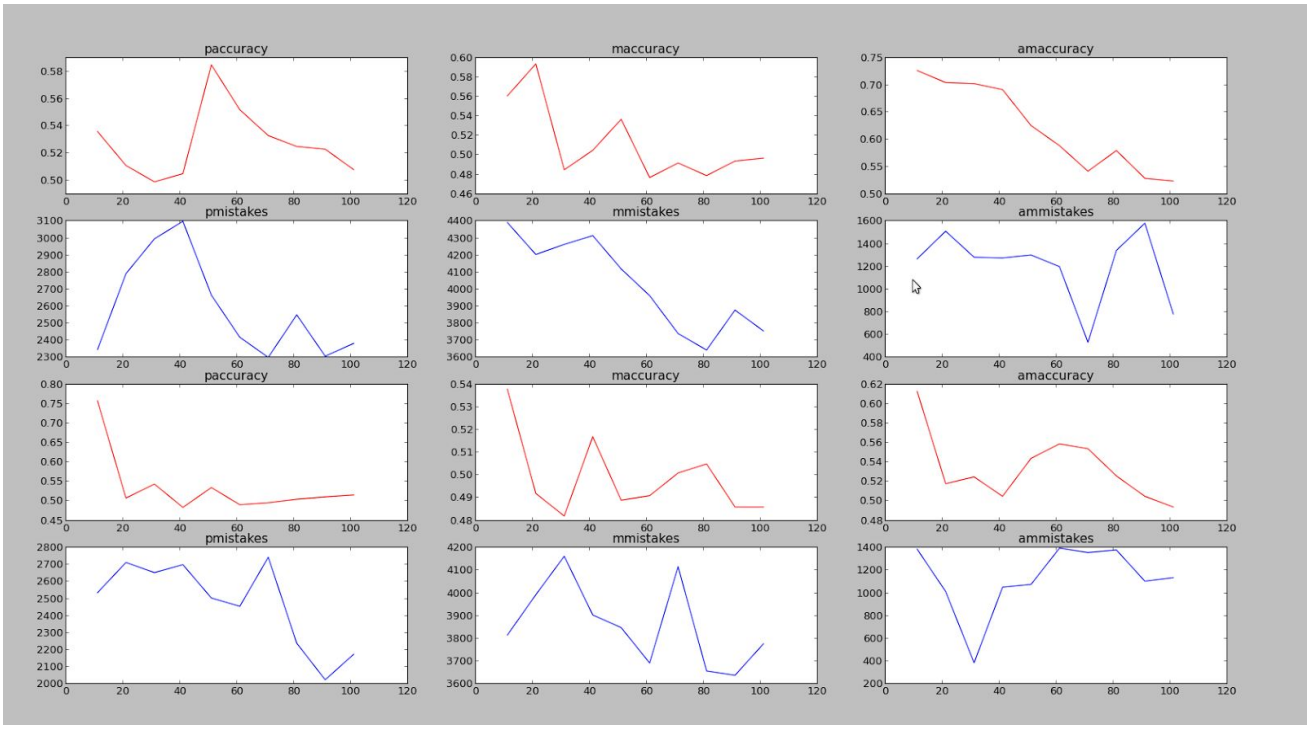
1. trainPerceptron()
2. trainMarginPerceptron()
3. trainAggressiveMarginPerceptron()
4. testAnyPerceptron
5. selectHyperParams()

The functioning of the first 2 perceptrons is already described above. For the aggressive perceptron, rate is not a passed parameter but is calculated based on the formula in the assignment. The behaviour of functions 4 and 5 is described above.
This experiment is carried out over 10 epochs from each function, coupled with a 5 fold validation while selecting the hyper parameters.

I have also tried to make the listing out of results a readable process, and the program produces graphs at the end of the computation with any values you wish to tweak.

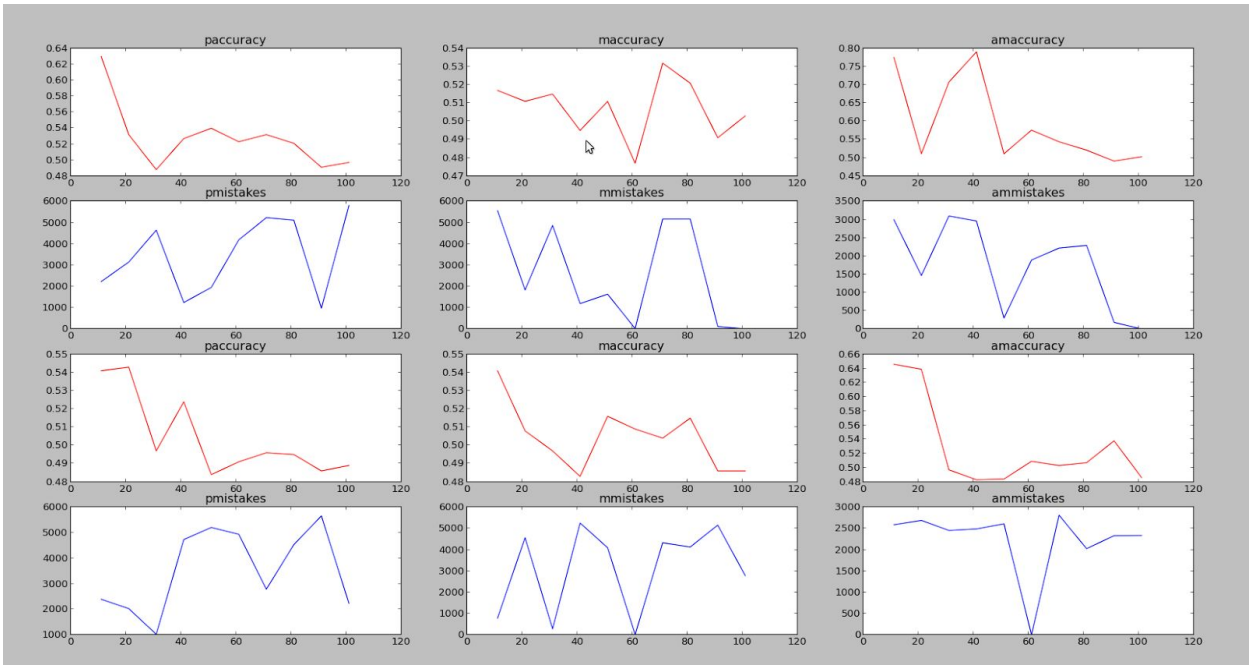
Result

It is quite clear that while the accuracy of the regular perceptron decreases with increasing dimensionality, the margin and aggressive margin perceptron perform better in comparison.
The first 6 images are from data0, and next 6 from data1



The below was the recorded performance from the same experiment, with a few tweaks.

One of which was switching on the selectHyperParams function. The data seems consistent save a few outliers with low accuracy.



The reason for stating those issues was more out of curiosity if you have any ideas about what could be causing that. I suppose in retrospect they don't affect the program that badly since the later data seems to be fairly consistent with expected results save a few outliers.