

**Project 2 Report**

**Students:**

**Ahmed Elsharkawy 201309020**

**Ibrahim Hamed 201518019**

**Abstract:**

In this project. Our main aim was to apply the perceptron algorithm on a data set that was given to us and it consisted of 5 classes. We took a different approach in testing our data. We chose our optimal hyper-parameters based on each class and not on the whole data. And in the end we reported all the details that was required from us and mentioned whether it is for one class or for the whole data.

**Steps that lead to starting the experiments:**

1. **Getting the data:**

we extracted the data from the file using function **getDataFromFile**

1. **Splitting our data with equal proportions:**

To split the data to training, validation and testing we used the function **splitData(data)** that takes our data and split the data. First what we did was shuffling the data. Then we called a function **getdataClasses(data)** that simple returns the data divided between 5 classes arrays that each belong to one class (1,2,3,4,5). This will keep the size of our data but divided depending on the class. Then we start splitting the data. Testing always got 20% of our data. For each class we chose different percentage for the training and validation between (5% and 20% for validation). The reason for this was to give the validation a good proportion of the data so that we can test on it later. The function return our training, validation and testing data

1. **Reducing the classes to OVA:**

In here we use **getBinaryData(data).** Data represents any of the training, validation or testing depends on what we send it. We use this to make 5 classes array that each has the data for one class with label 1 against the other classes with label 0. We do this for all training, validation and testing so that we can apply the perceptron algorithm on them later.

1. **Applying the perceptron algorithm on training and validation:**

**FindOptimalHyperParameters(TchosenClass, VchosenClass, NoOfEpochs, learningRate)** takes 4 parameters. Chosen class training data and validation data (we get these from **getBinaryData(data)**) number of epochs which we specify (in here we are using 5 as a standard) and the learning rate. This function gets the true label for our class training and validation data we start with very small weights (0, -0.1, 0.1). Then we start training on the training data. For each example we check if we got a correct classification by using the **classifyexample(weights, testExample)** that simply takes the current weights and the example. Apply some logic and return 1 if positive and 0 if negative. Then if this predicted label was not equal to the true label. We update the weights using **updateWeights(n, weights, example, predictedLabel)** that uses the weight updating logic mentioned in the slides and return the new weights that we use in the next example in our data and we keep doing this until the number of examples is over and then we take those final weights we got and test them on the validation data we have. We do the same process here except that we calculate the accuracy for each epoch. We keep doing this until the number of epochs is done. After we try different learning rates we picked the best one and the best number of epochs so that we can use them later for the testing data which is in another function.

**Other functions our program has:**

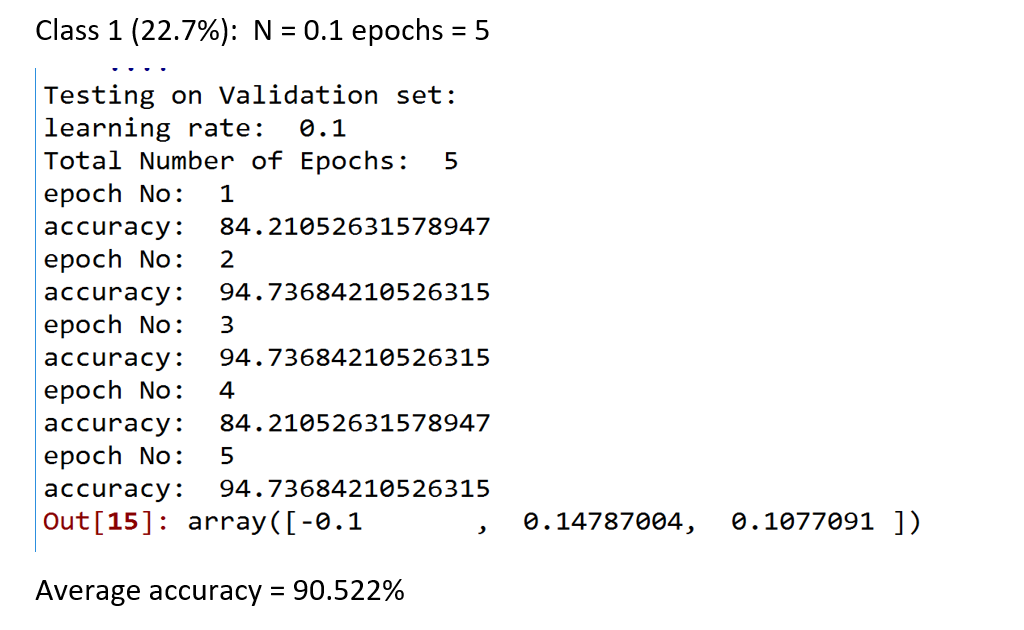
We have functions that are important to apply the experiment like getting the accuracy function. Some other functions are mainly used for the results we got from the testing. Like calculating the recall, precision. Macro recall etc. we have two functions that we made but did not really have to use. Oversample and undersample. Both works perfectly.

More comments about the code and the functionality can be found in the code file.

**The part where the magic happens:**

1. **Choosing the optimal data:**

What we did in choosing our optimal data was: we ran a lot of tests using different learning rates and the same number of epochs (5). And we checked the accuracy after every epoch for the validation set. One thing that we did notice is having the same accuracy between epochs for one learning rate. And having the same accuracy sometimes with different learning rates. The learning rates we tested with were between 0.1 and 0.5. we also tried some few trials with lower learning rate like 0.01 and 0.00. the accuracy can be different sometimes but it all happened about the same numbers and followed the same accuracy logic we encountered before. That is why we stuck to our 0.1 to 0.5 range. More analysis about our assumptions of why this is happening can be found in the project notes at the end (1). In any case. Since we had these tests and we saw their results. We decided to hand pick our optimal data. We looked at the accuracy at the end of each epoch. Compared the highest accuracy and checked what weights did we have when it happened. To determine which learning rate was best it was simply a comparison between the highest accuracy average between every epoch. For example:



This is a sample of what happened: we averaged the accuracies for each learning rate. Compared them with different learning rates and chose the best one for **the chosen class**.

Our chosen learning rate goes as follows:

Class 1: N= 0.1 (accuracy = 90.522%)

Class 2: N= 0.5 (accuracy = 77.888%)

Class 3: N= 0.1 (accuracy = 83.156%)

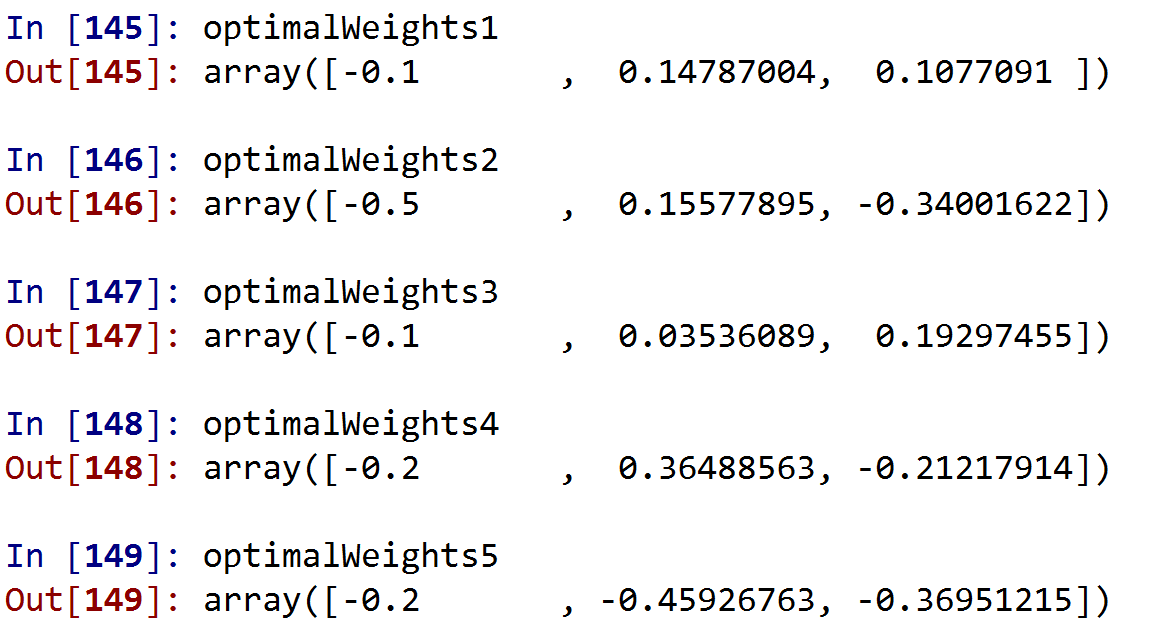
Class 4: N= 0.1 (accuracy = 94.732%)

Class 5: N= 0.2 (accuracy = 87.364%)

Then we average the accuracies after each epoch and compare them to know which epoch had the highest accuracy and choose it as an optimal solution. And then return the optimal epochs. In our case we went for 5 epochs as the optimal number. We did not do the method we mentioned above. The reason will be more explained in the notes and assumption below (2). Due to our accuracies. I thought it is better if we do not oversample or undersample due to the high accuracy in all classes.

1. **Training the optimal on training + validation:**

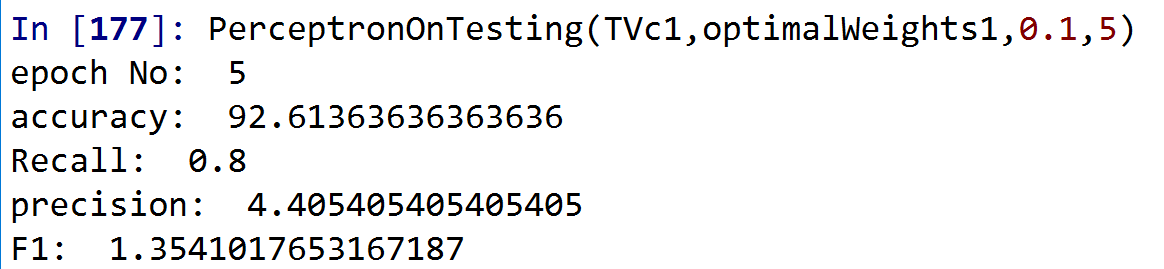
Then we call a new function called trainOptimalOnData() which does pretty much the same thing as FindOptimalHyperParameters() except that it returns new weights. We are going to send the optimal hyper pararmeter and initialize our weights to 0, -0.1, and 0.1 and then get the final weights for each class and use them for testing later. The weights we got:



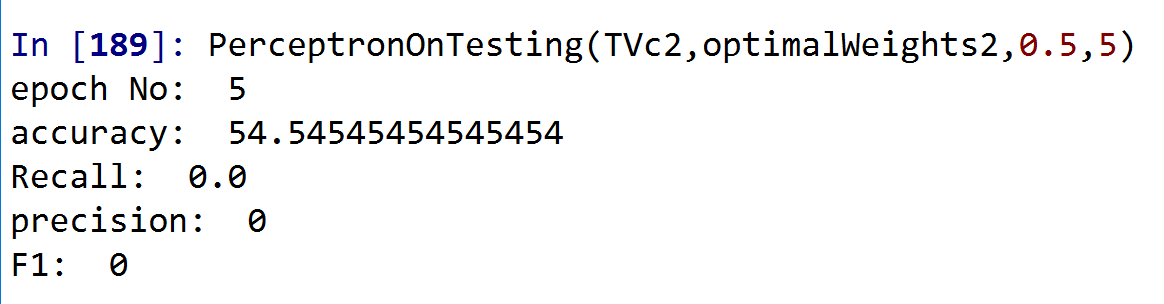
1. **Finding out the results on testing:**

We use the new weights to we find in step 2 to carry on with our experiment on the testing data. On the testing data we calculate the accuracy, precision, recall and F1 for every class and the we use the macro methods to average them out afterwards. Our tests gave these results:

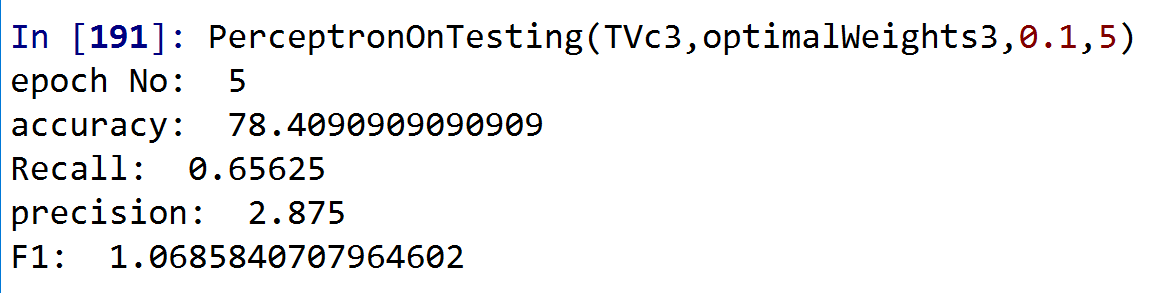
Class 1:



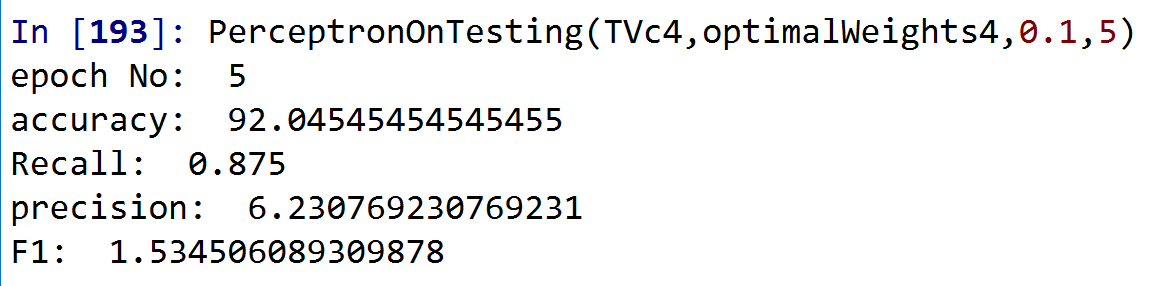
Class 2:



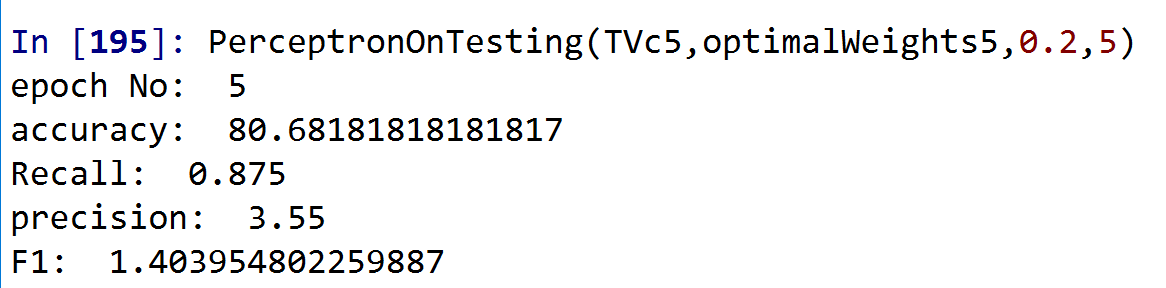
Class 3:



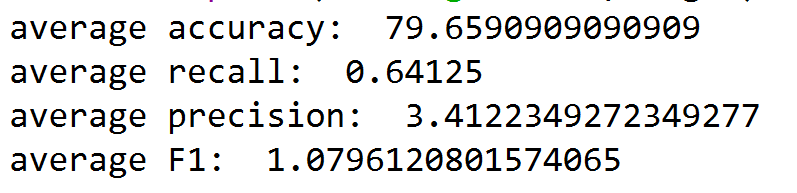
Class 4:



Class 5:



Macro Everything:

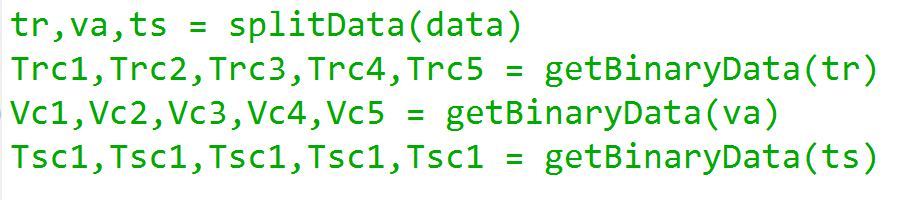


By this. Our experiment has come to an end. We had some surprises that we tried our best to work with and this is the final result that we got.

**Project notes and assumptions:**

1. One thing I tried to do to fix the accuracy was changing the learning rate to a lower rate like 0.01 or 0.001 because it was probably converging too fast. but we did not see that there was this much difference in accuracy and the same equal accuracy occurrences were happening so I kept my original range. i tried testing if my data is reaching correctly to the functions and I found out it was. The weights were always updating and changing with made this harder to know the reason behind it. So at the end we just decided to work on it and see. We got good accuracies which is good enough for what we have.
2. After I took a while calculating the accuracies to know the learning rates. One thing I noticed that lowereing the epochs number after deciding the number of epochs will affect my choice on the learning rate since we will have less learning rates accuracies to consider. And since I did not want to go over all the calculations again. I assumed our optimal epochs number will be 5 since this is the number I used to find out the optimal learning rate and weights, but I found out the epochs first it would have been easier to find the optimal learning rate.
3. The different approach we had in this project: instead of assuming and checking which class are we testing on. We already did that when we divide it our classes and data into training, validation and testing. This let us test on each class separately.

Tr = training. Va = validation. Ts = testing. c = class



1. More info and comments can be found in the code.
2. Final note is that we were two people in this project. It was very hard to consider all possibilities and cases. Never the less we did try to assume, test, debug and apply as much possibilities as possible. Trying different routes and approaches to find out what was best.