SMAI Assignment 1 by Naila Fatima (201530154)

<u>Question 1:</u>

a) <u>Single sample perceptron without margin</u>

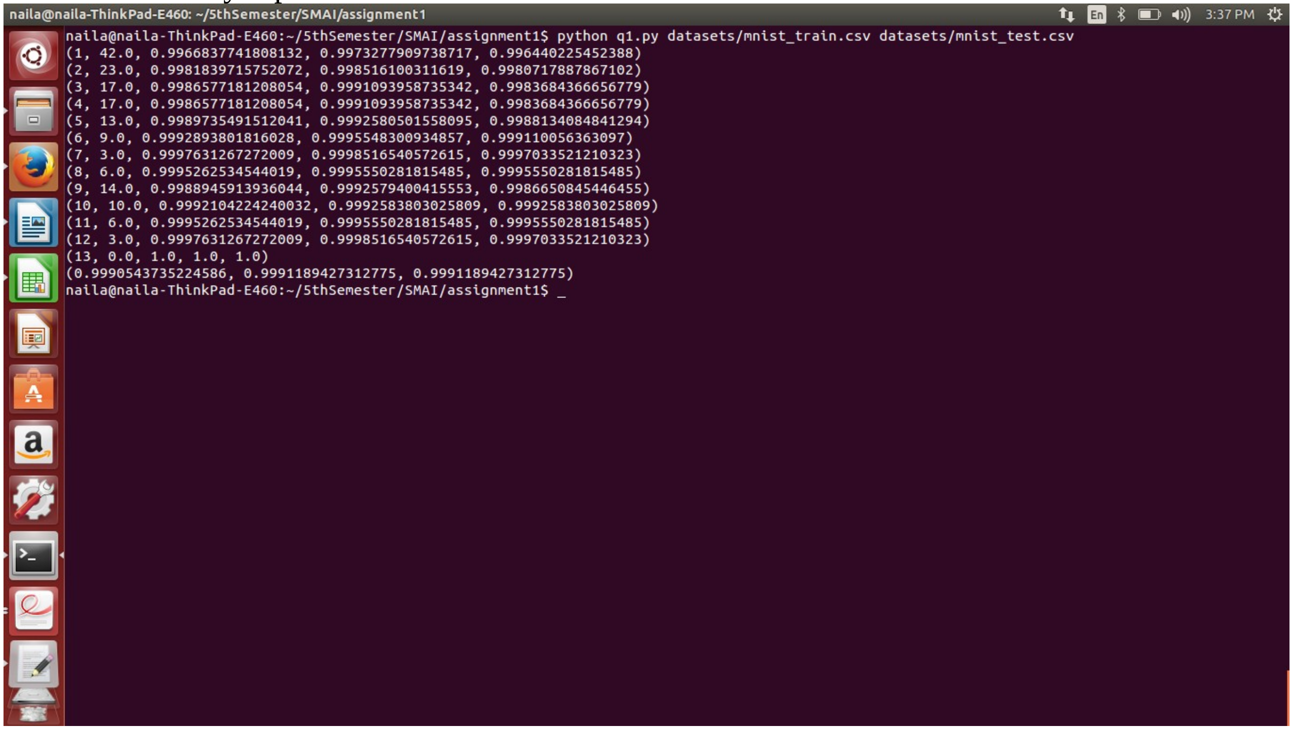
This algorithm was implemented by using functions to estimate weights and predict the class label. The weights were first initialized to be 0 and they were updated as long as both the error was greater than 0 and the maximum number of epochs was not yet reached. For each row in the training data, the label was noted (first column) and a class was predicted by using the weights. This was done by taking the dot product of the row and weights and if the dot product was above or equal to the margin (0 in this case), the class was predicted to be 1 else 0. If the predicted class and actual class were not same, the weights vector was updated by adding the row if the correct class label was 1 and subtracting it if the correct class label was 0. This was done by making y = 1 if correct class label is 1 and y = -1 if correct class label is 0. The update is given as:

weights = weights + y*row   (if predicted label is not equal to correct label)

Note that the weights were updated once a misclassified row was seen.

The accuracy ((TN+TP)/Total) and recall (TP/(TP+FN)) for this case were 0.9990543735224586 and 0.9991189427312775, respectively. Out of 2115 cases, 2113 were reported correctly in the test data.

13 epochs were needed to train data and the number of errors converged to 0 showing that the data is linearly seperable.



```
naila@naila-ThinkPad-E460: ~/5thSemester/SMAI/assignment1
naila@naila-ThinkPad-E460:~/5thSemester/SMAI/assignment1$ python q1.py datasets/mnist_train.csv datasets/mnist_test.csv
(1, 42.0, 0.9966837741808132, 0.9973277909738717, 0.996440225452388)
(2, 23.0, 0.9981839715752072, 0.998516100311619, 0.9980717887867102)
(3, 17.0, 0.9986577181208054, 0.9991093958735342, 0.9983684366656779)
(4, 17.0, 0.9986577181208054, 0.9991093958735342, 0.9983684366656779)
(5, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(6, 9.0, 0.9992893801816028, 0.9995548300934857, 0.999110056363097)
(7, 3.0, 0.9997631267272009, 0.9998516540572615, 0.9997033521210323)
(8, 6.0, 0.9995262534544019, 0.9995550281815485, 0.9995550281815485)
(9, 14.0, 0.9988945913936044, 0.9992579400415553, 0.9986650845446455)
(10, 10.0, 0.9992104224240032, 0.9992583803025809, 0.9992583803025809)
(11, 6.0, 0.9995262534544019, 0.9995550281815485, 0.9995550281815485)
(12, 3.0, 0.9997631267272009, 0.9998516540572615, 0.9997033521210323)
(13, 0.0, 1.0, 1.0, 1.0)
(0.9990543735224586, 0.9991189427312775, 0.9991189427312775)
naila@naila-ThinkPad-E460:~/5thSemester/SMAI/assignment1$ _
```

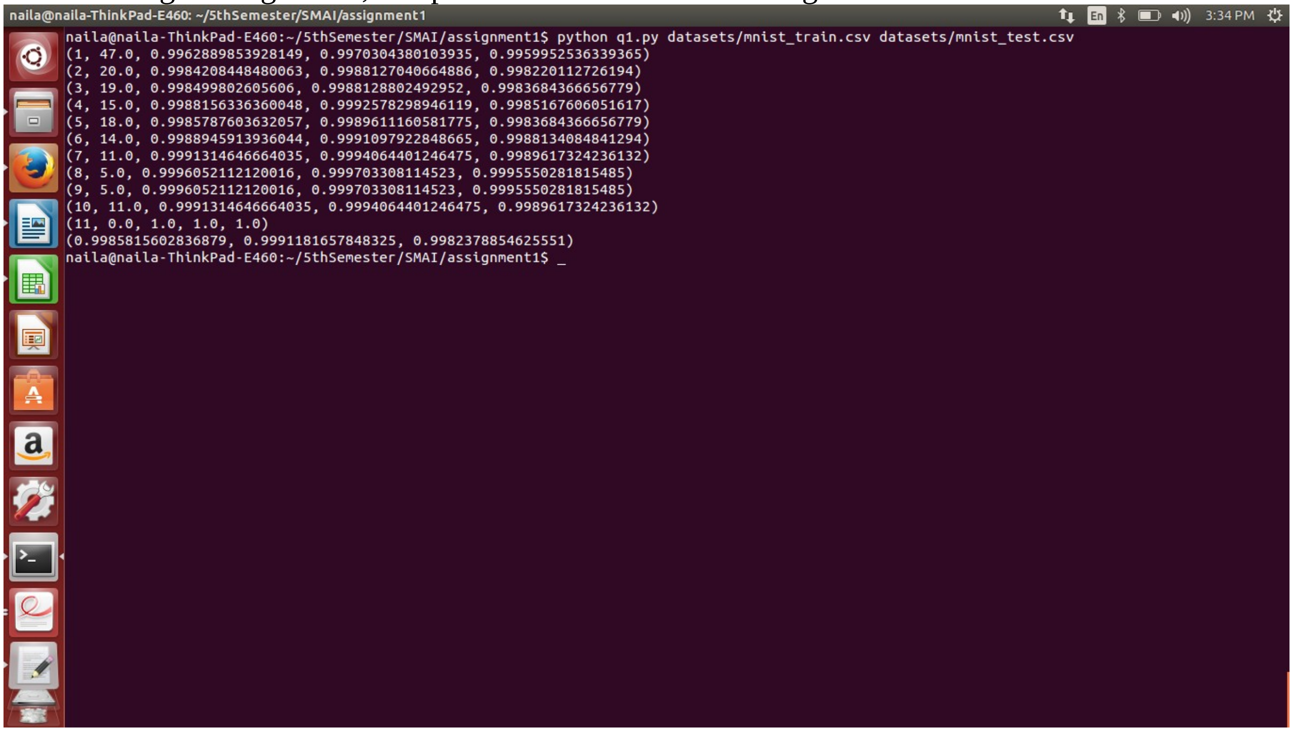
The above screenshot shows the epoch number, the number of errors for that particular epoch, accuracy for that epoch, precision for that epoch and recall for that epoch. We can see that the number of errors reduced from 42 to 6 in 8 epochs but increased to 14 (in 9th epoch) after which it continued to decreased until it converged. Also, up till the 9h epoch, the accuracy, precision and recall continue to increase and at the 9h epoch, all fall and then continue to increase until they become 1 at the 13th epoch when the algorithm converges. These values are for the training data. The last line shows the accuracy, precision and recall for the <u>test</u> data which is around 0.999 each. Note that the last line is for test data whereas the other ones are for the training data.

b) <u>Single sample perceptron with margin</u>

This algorithm was implemented similarly to the above one. The weights were first initialized to be 0 and then a class label was predicted. If the predicted label was wrong, the weights were updated such that

weights = weights + y*row where y is 1 if correct class label is 1 and y is -1 if correct class label is 0. The weights were updated once a misclassified row was seen. The difference in this method lies in the prediction part. If the dot product of the row and the weights is greater than or equal to the margin provided, it is predicted to be class 1 else it is predicted to be class 0. I have used a margin of 1.8.

Using this algorithm, 11 epochs were needed to converge.



```
naila@naila-ThinkPad-E460: ~/5thSemester/SMAI/assignment1
naila@naila-ThinkPad-E460:~/5thSemester/SMAI/assignment1$ python q1.py datasets/mnist_train.csv datasets/mnist_test.csv
(1, 47.0, 0.9962889853928149, 0.9970304380103935, 0.9959952536339365)
(2, 20.0, 0.9984208448480063, 0.9988127040664886, 0.998220112726194)
(3, 19.0, 0.998499802605606, 0.9988128802492952, 0.9983684366656779)
(4, 15.0, 0.9988156336360048, 0.9992578298946119, 0.9985167606051617)
(5, 18.0, 0.9985787603632057, 0.9989611160581775, 0.9983684366656779)
(6, 14.0, 0.9988945913936044, 0.9991097922848665, 0.9988134084841294)
(7, 11.0, 0.9991314646664035, 0.9994064401246475, 0.9989617324236132)
(8, 5.0, 0.9996052112120016, 0.999703308114523, 0.9995550281815485)
(9, 5.0, 0.9996052112120016, 0.999703308114523, 0.9995550281815485)
(10, 11.0, 0.9991314646664035, 0.9994064401246475, 0.9989617324236132)
(11, 0.0, 1.0, 1.0, 1.0)
(0.9985815602836879, 0.9991181657848325, 0.9982378854625551)
naila@naila-ThinkPad-E460:~/5thSemester/SMAI/assignment1$ _
```

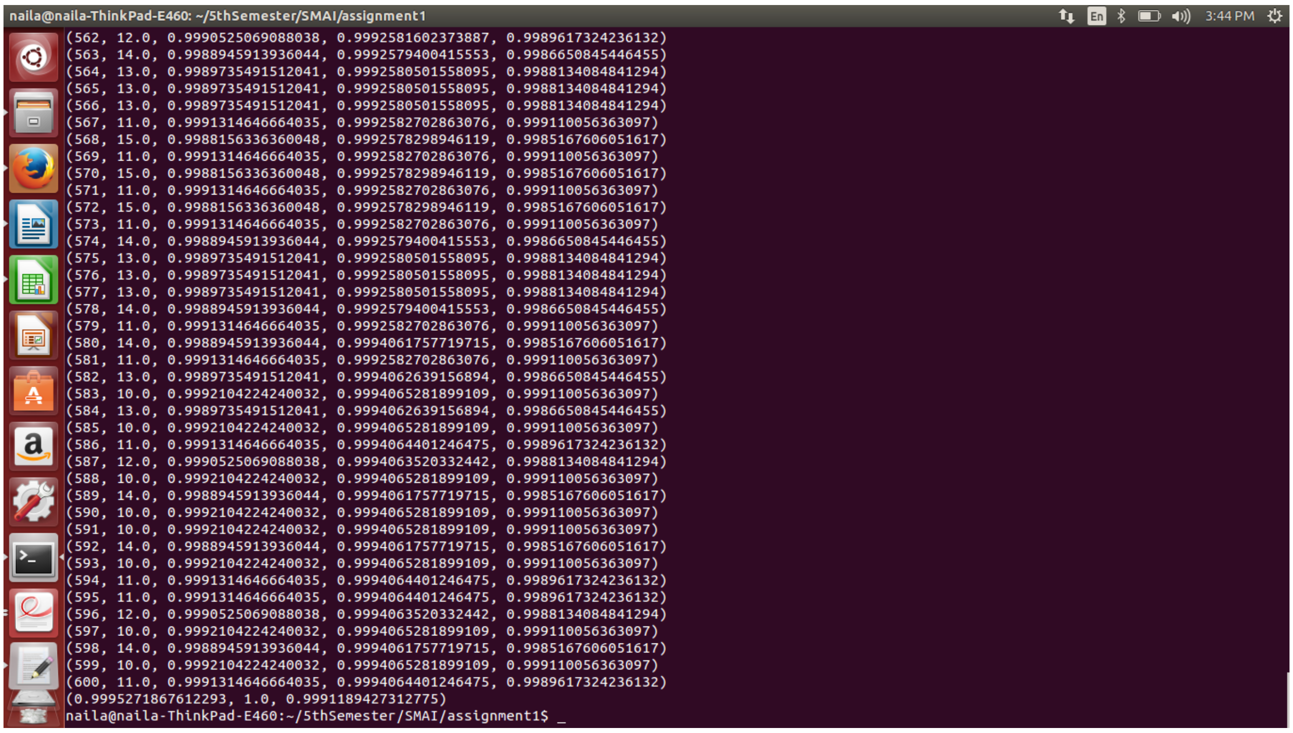
The above screenshot shows the epoch number, number of errors accuracy, precision and recall for that epoch. We can see that the accuracy, precision and recall continue to increase up until the 9th epoch where they are constant and then decrease for the 10th epoch (since number of errors rises up to 11) and then increase to 1 when the algorithm converges. These values were for the training data. The last line shows the accuracy, precision and recall on the test data. We can see that since they are around 0.998, they can be considered to be satisfactory. We can see that previous algorithm performed a bit better in terms of accuracy, precision and recall.

c) Batch perceptron without margin

In this algorithm, the weights were first initialized to be 0. In an epoch, a class was predicted for each row by taking the dot product and classifying it as class 1 if the dot product was greater than or equal to 0 else class 0. All the misclassified rows were stored and at the end of each epoch, the weights were updated. This is different from the single perceptron algorithm since in that case weights were updated for each misclassified row (so there were multiple updates in a single epoch) whereas in this case, we are updating the weights once for each epoch. This is done by storing all the misclassified rows and at the end of the epoch, doing:

weights = weights + y*misclassifiedrow(i)

where y is 1 for correct class label 1 and -1 for correct class label 0. The update occurs for each misclassified row in that epoch.



```
(562, 12.0, 0.9990525069088038, 0.9992581602373887, 0.9989617324236132)
(563, 14.0, 0.9988945913936044, 0.9992579400415553, 0.9986650845446455)
(564, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(565, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(566, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(567, 11.0, 0.999131464664035, 0.9992582702863076, 0.999110056363097)
(568, 15.0, 0.9988156336360048, 0.9992578298946119, 0.9985167606051617)
(569, 11.0, 0.999131464664035, 0.9992582702863076, 0.999110056363097)
(570, 15.0, 0.9988156336360048, 0.9992578298946119, 0.9985167606051617)
(571, 11.0, 0.999131464664035, 0.9992582702863076, 0.999110056363097)
(572, 15.0, 0.9988156336360048, 0.9992578298946119, 0.9985167606051617)
(573, 11.0, 0.999131464664035, 0.9992582702863076, 0.999110056363097)
(574, 14.0, 0.9988945913936044, 0.9992579400415553, 0.9986650845446455)
(575, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(576, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(577, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(578, 14.0, 0.9988945913936044, 0.9992579400415553, 0.9986650845446455)
(579, 11.0, 0.999131464664035, 0.9992582702863076, 0.999110056363097)
(580, 14.0, 0.9988945913936044, 0.9994061757719715, 0.9985167606051617)
(581, 11.0, 0.999131464664035, 0.9992582702863076, 0.999110056363097)
(582, 13.0, 0.9989735491512041, 0.9994062639156894, 0.9986650845446455)
(583, 10.0, 0.999210422424032, 0.9994065281899109, 0.999110056363097)
(584, 13.0, 0.9989735491512041, 0.9994062639156894, 0.9986650845446455)
(585, 10.0, 0.999210422424032, 0.9994065281899109, 0.999110056363097)
(586, 11.0, 0.999131464664035, 0.9994064401246475, 0.9989617324236132)
(587, 12.0, 0.9990525069088038, 0.9994063520332442, 0.9988134084841294)
(588, 10.0, 0.999210422424032, 0.9994065281899109, 0.999110056363097)
(589, 14.0, 0.9988945913936044, 0.9994061757719715, 0.9985167606051617)
(590, 10.0, 0.999210422424032, 0.9994065281899109, 0.999110056363097)
(591, 10.0, 0.999210422424032, 0.9994065281899109, 0.999110056363097)
(592, 14.0, 0.9988945913936044, 0.9994061757719715, 0.9985167606051617)
(593, 10.0, 0.999210422424032, 0.9994065281899109, 0.999110056363097)
(594, 11.0, 0.999131464664035, 0.9994064401246475, 0.9989617324236132)
(595, 11.0, 0.999131464664035, 0.9994064401246475, 0.9989617324236132)
(596, 12.0, 0.9990525069088038, 0.9994063520332442, 0.9988134084841294)
(597, 10.0, 0.999210422424032, 0.9994065281899109, 0.999110056363097)
(598, 14.0, 0.9988945913936044, 0.9994061757719715, 0.9985167606051617)
(599, 10.0, 0.999210422424032, 0.9994065281899109, 0.999110056363097)
(600, 11.0, 0.999131464664035, 0.9994064401246475, 0.9989617324236132)
(0.9995271867612293, 1.0, 0.9991189427312775)
naila@naila-ThinkPad-E460:~/5thSemester/SMAI/assignment1$ _
```

Each line shows the epoch number, number of errors, accuracy, precision and recall for that epoch while training. We can see that the values change very little and take several epochs to show significant change. I ran the algorithm for 600 epochs to get an accuracy, precision and recall around 0.999. The algorithm does converge but it requires around 2000 epochs. Since I limited the number of epochs, we can see that the accuracy and recall are around 0.999 whereas precision is 1 for the test data. If the algorithm was trained until it converged, both the accuracy and recall might have been a bit higher.

We can notice that this algorithm did not perform as well as the single perceptron without margin and it took way more time.

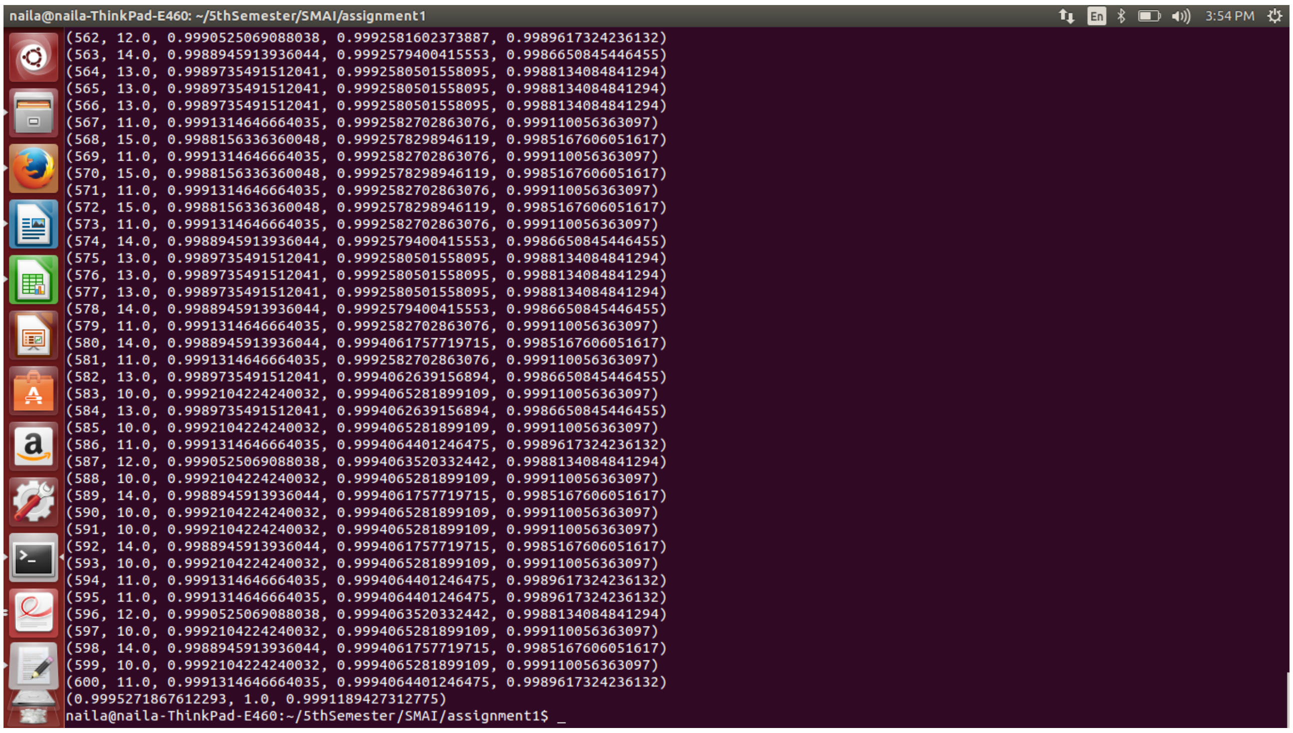
d) Batch perceptron with margin

This algorithm is similar to the above one. The weights are initialized to 0 and in each epoch a class is predicted for each row. All misclassified rows are stored and at the end of each epoch, the weights are updated as:

weights = weights + y*misclassifiedrow(i)

where y is 1 for correct class label 1 and -1 for correct class label 0. The update occurs at the end of each epoch for each misclassified row.

The only difference between this and the previous algorithm is in the way the classes are predicted. In this case, to predict the class of a row, the dot product of the row and the weights is taken and if it is greater than or equal to the margin (1.8 in my case), it is predicted to be class 1, else it is predicted to be class 0.



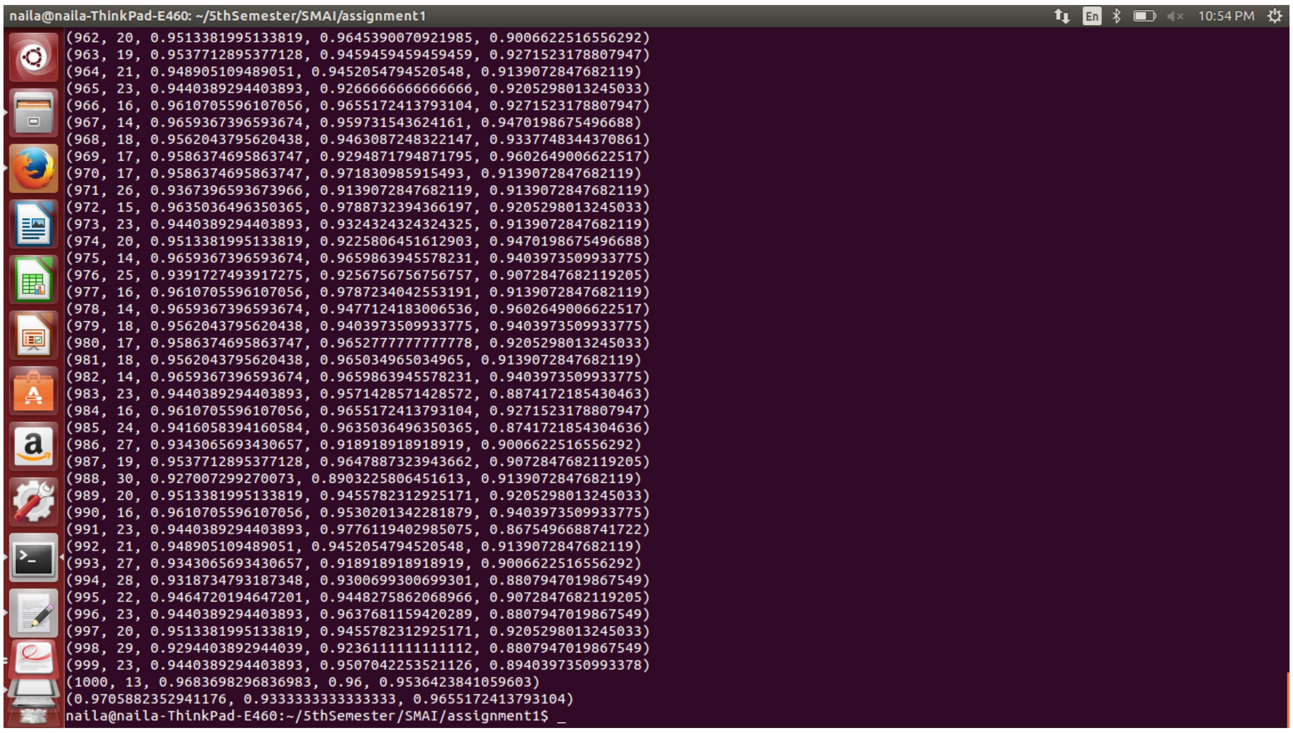
```
naila@naila-ThinkPad-E460: ~/5thSemester/SMAI/assignment1
(562, 12.0, 0.9990525069088038, 0.9992581602373887, 0.9989617324236132)
(563, 14.0, 0.9988945913936044, 0.9992579400415553, 0.9986650845446455)
(564, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(565, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(566, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(567, 11.0, 0.9991314646664035, 0.9992582702863076, 0.999110056363097)
(568, 15.0, 0.9988156336360048, 0.9992578298946119, 0.9985167606051617)
(569, 11.0, 0.9991314646664035, 0.9992582702863076, 0.999110056363097)
(570, 15.0, 0.9988156336360048, 0.9992578298946119, 0.9985167606051617)
(571, 11.0, 0.9991314646664035, 0.9992582702863076, 0.999110056363097)
(572, 15.0, 0.9988156336360048, 0.9992578298946119, 0.9985167606051617)
(573, 11.0, 0.9991314646664035, 0.9992582702863076, 0.999110056363097)
(574, 14.0, 0.9988945913936044, 0.9992579400415553, 0.9986650845446455)
(575, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(576, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(577, 13.0, 0.9989735491512041, 0.9992580501558095, 0.9988134084841294)
(578, 14.0, 0.9988945913936044, 0.9992579400415553, 0.9986650845446455)
(579, 11.0, 0.9991314646664035, 0.9992582702863076, 0.999110056363097)
(580, 14.0, 0.9988945913936044, 0.9994061757719715, 0.9985167606051617)
(581, 11.0, 0.9991314646664035, 0.9992582702863076, 0.999110056363097)
(582, 13.0, 0.9989735491512041, 0.9994062639156894, 0.9986650845446455)
(583, 10.0, 0.9992104224240032, 0.9994065281899109, 0.999110056363097)
(584, 13.0, 0.9989735491512041, 0.9994062639156894, 0.9986650845446455)
(585, 10.0, 0.9992104224240032, 0.9994065281899109, 0.999110056363097)
(586, 11.0, 0.9991314646664035, 0.9994064401246475, 0.9989617324236132)
(587, 12.0, 0.9990525069088038, 0.9994063520332442, 0.9988134084841294)
(588, 10.0, 0.9992104224240032, 0.9994065281899109, 0.999110056363097)
(589, 14.0, 0.9988945913936044, 0.9994061757719715, 0.9985167606051617)
(590, 10.0, 0.9992104224240032, 0.9994065281899109, 0.999110056363097)
(591, 10.0, 0.9992104224240032, 0.9994065281899109, 0.999110056363097)
(592, 14.0, 0.9988945913936044, 0.9994061757719715, 0.9985167606051617)
(593, 10.0, 0.9992104224240032, 0.9994065281899109, 0.999110056363097)
(594, 11.0, 0.9991314646664035, 0.9994064401246475, 0.9989617324236132)
(595, 11.0, 0.9991314646664035, 0.9994064401246475, 0.9989617324236132)
(596, 12.0, 0.9990525069088038, 0.9994063520332442, 0.9988134084841294)
(597, 10.0, 0.9992104224240032, 0.9994065281899109, 0.999110056363097)
(598, 14.0, 0.9988945913936044, 0.9994061757719715, 0.9985167606051617)
(599, 10.0, 0.9992104224240032, 0.9994065281899109, 0.999110056363097)
(600, 11.0, 0.9991314646664035, 0.9994064401246475, 0.9989617324236132)
(0.9995271867612293, 1.0, 0.9991189427312775)
naila@naila-ThinkPad-E460:~/5thSemester/SMAI/assignment1$ _
```

From the screenshot, we can see the number of epoch, number of errors, accuracy, precision and recall for that particular epoch for the training data. We can see that the algorithm takes around 600 epochs for 11 errors and accuracy around 0.999. The algorithm converges for more than 2000 epochs but I had limited this to 600. Since I had limited it, the test data shows an accuracy and recall around 0.999 and precision of 1 (in the last line). We can note that the single perceptron is better since it is faster and gives better values of accuracy, precision and recall.

Question 2:

a) Relaxation Algorithm

This was done by first initializing the weights to 0 and then updating them as follows:
weights = weights + eta*(margin-dot)*row/(norm of row)^2
where eta refers to the learning rate kept at 0.098, the number of epochs being 1000 and the margin being equal to 75 and dot being the dotproduct of row and weights. The update was done until the maximum number of epochs was reached or the number of errors reduced to 0 (algorithm converged).



```
naila@naila-ThinkPad-E460: ~/5thSemester/SMAI/assignment1
(962, 20, 0.9513381995133819, 0.9645390070921985, 0.9006622516556292)
(963, 19, 0.9537712895377128, 0.9459459459459459, 0.9271523178807947)
(964, 21, 0.948905109489051, 0.9452054794520548, 0.9139072847682119)
(965, 23, 0.9440389294403893, 0.9266666666666666, 0.9205298013245033)
(966, 16, 0.9610705596107056, 0.9655172413793104, 0.9271523178807947)
(967, 14, 0.9659367396593674, 0.959731543624161, 0.9470198675496688)
(968, 18, 0.9562043795620438, 0.9463087248322147, 0.9337748344370861)
(969, 17, 0.9586374695863747, 0.9294871794871795, 0.9602649006622517)
(970, 17, 0.9586374695863747, 0.971830985915493, 0.9139072847682119)
(971, 26, 0.9367396593673966, 0.9139072847682119, 0.9139072847682119)
(972, 15, 0.9635036496350365, 0.9788732394366197, 0.9205298013245033)
(973, 23, 0.9440389294403893, 0.9324324324324325, 0.9139072847682119)
(974, 20, 0.9513381995133819, 0.9225806451612903, 0.9470198675496688)
(975, 14, 0.9659367396593674, 0.9659863945578231, 0.9403973509933775)
(976, 25, 0.9391727493917275, 0.9256756756756757, 0.9072847682119205)
(977, 16, 0.9610705596107056, 0.9787234042553191, 0.9139072847682119)
(978, 14, 0.9659367396593674, 0.9477124183006536, 0.9602649006622517)
(979, 18, 0.9562043795620438, 0.9403973509933775, 0.9403973509933775)
(980, 17, 0.9586374695863747, 0.9652777777777778, 0.9205298013245033)
(981, 18, 0.9562043795620438, 0.965034965034965, 0.9139072847682119)
(982, 14, 0.9659367396593674, 0.9659863945578231, 0.9403973509933775)
(983, 23, 0.9440389294403893, 0.9571428571428572, 0.8874172185430463)
(984, 16, 0.9610705596107056, 0.9655172413793104, 0.9271523178807947)
(985, 24, 0.9416058394160584, 0.9635036496350365, 0.8741721854304636)
(986, 27, 0.9343065693430657, 0.918918918918919, 0.9006622516556292)
(987, 19, 0.9537712895377128, 0.9647887323943662, 0.9072847682119205)
(988, 30, 0.927007299270073, 0.8903225806451613, 0.9139072847682119)
(989, 20, 0.9513381995133819, 0.9455782312925171, 0.9205298013245033)
(990, 16, 0.9610705596107056, 0.9530201342281879, 0.9403973509933775)
(991, 23, 0.9440389294403893, 0.9776119402985075, 0.8675496688741722)
(992, 21, 0.948905109489051, 0.9452054794520548, 0.9139072847682119)
(993, 27, 0.9343065693430657, 0.918918918918919, 0.9006622516556292)
(994, 28, 0.9318734793187348, 0.9300699300699301, 0.8807947019867549)
(995, 22, 0.9464720194647201, 0.9448275862068966, 0.9072847682119205)
(996, 23, 0.9440389294403893, 0.9637681159420289, 0.8807947019867549)
(997, 20, 0.9513381995133819, 0.9455782312925171, 0.9205298013245033)
(998, 29, 0.9294403892944039, 0.9236111111111112, 0.8807947019867549)
(999, 23, 0.9440389294403893, 0.9507042253521126, 0.8940397350993378)
(1000, 13, 0.9683698296836983, 0.96, 0.9536423841059603)
(0.9705882352941176, 0.9333333333333333, 0.9655172413793104)
naila@naila-ThinkPad-E460:~/5thSemester/SMAI/assignment1$ _
```

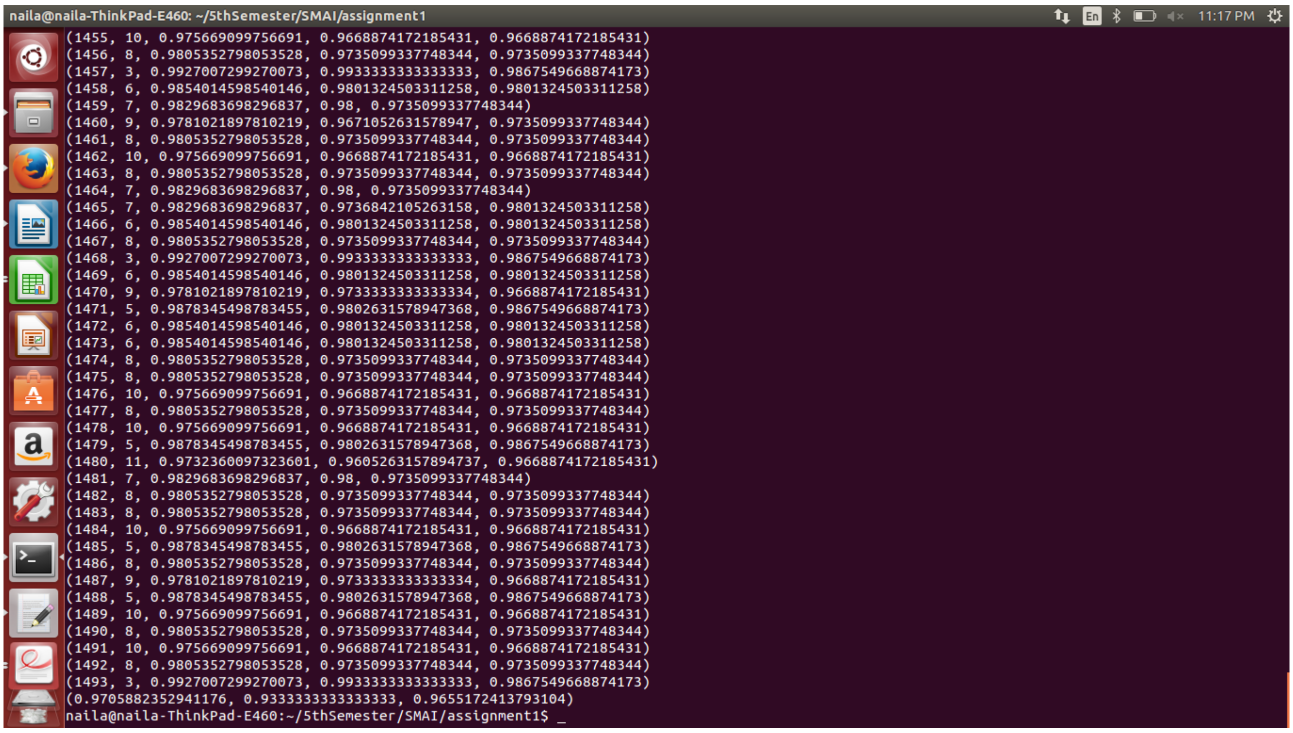
In the screenshot above, the lines with 5 parameters are for the training data. The parameters are the epoch number, number of errors, accuracy, precision and recall for that particular epoch. I have observed that the precision, accuracy and recall all lie in the range of 0.92 to 0.97. The number of errors slowly decreases to 13 on the 1000th epoch while the accuracy, precision and recall for it are around 0.96 each. I have noticed that keeping the eta (learning rate) around 0.1 gives the best performance whereas the value of margin ranging in 60-80 gives a slight improvement in performance. The last line containing 3 parameters denotes the accuracy, precision and recall for the validation data. Since no test data was provided, I had used a 80-20 training/validation split and had crosschecked my results with the validation set. We can see that the accuracy is around 0.98 whereas the precision is around 0.93 and the recall around 0.96. The performance is decent enough.

b) <u>Voted Perceptron algorithm</u>

In this algorithm, I initialized my weight vector to be 0. I would keep track of the number of times a weight vector would give me correct label without being updated. On each updation of the weight vector, I would store the number of times the previous vector had given correct labels as well as the vector itself. The updation would be done in the same manner as before:

new weight vector = old weight vector + y*x where y is 1 if correct label is 4 else it is -1 and x refers to the row in training data for which the old weight vector misclassified. The prediction was however different, as I predicted a row to have class label 4 if summation{ c(i)*sign(w(i),row)} was greater than or equal to 0. We can see that I am weighing each of the previously computed weight vectors w(i) with the number of times they predicted correctly c(i). If the above summation is less than 0, the class is predicted to be 2.

I ran the algorithm for 1493 epochs.



```
naila@naila-ThinkPad-E460: ~/5thSemester/SMAI/assignment1
(1455, 10, 0.975669099756691, 0.9668874172185431, 0.9668874172185431)
(1456, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1457, 3, 0.9927007299270073, 0.9933333333333333, 0.9867549668874173)
(1458, 6, 0.9854014598540146, 0.9801324503311258, 0.9801324503311258)
(1459, 7, 0.9829683698296837, 0.98, 0.9735099337748344)
(1460, 9, 0.9781021897810219, 0.9671052631578947, 0.9735099337748344)
(1461, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1462, 10, 0.975669099756691, 0.9668874172185431, 0.9668874172185431)
(1463, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1464, 7, 0.9829683698296837, 0.98, 0.9735099337748344)
(1465, 7, 0.9829683698296837, 0.9736842105263158, 0.9801324503311258)
(1466, 6, 0.9854014598540146, 0.9801324503311258, 0.9801324503311258)
(1467, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1468, 3, 0.9927007299270073, 0.9933333333333333, 0.9867549668874173)
(1469, 6, 0.9854014598540146, 0.9801324503311258, 0.9801324503311258)
(1470, 9, 0.9781021897810219, 0.9733333333333334, 0.9668874172185431)
(1471, 5, 0.9878345498783455, 0.9802631578947368, 0.9867549668874173)
(1472, 6, 0.9854014598540146, 0.9801324503311258, 0.9801324503311258)
(1473, 6, 0.9854014598540146, 0.9801324503311258, 0.9801324503311258)
(1474, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1475, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1476, 10, 0.975669099756691, 0.9668874172185431, 0.9668874172185431)
(1477, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1478, 10, 0.975669099756691, 0.9668874172185431, 0.9668874172185431)
(1479, 5, 0.9878345498783455, 0.9802631578947368, 0.9867549668874173)
(1480, 11, 0.9732360097323601, 0.9605263157894737, 0.9668874172185431)
(1481, 7, 0.9829683698296837, 0.98, 0.9735099337748344)
(1482, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1483, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1484, 10, 0.975669099756691, 0.9668874172185431, 0.9668874172185431)
(1485, 5, 0.9878345498783455, 0.9802631578947368, 0.9867549668874173)
(1486, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1487, 9, 0.9781021897810219, 0.9733333333333334, 0.9668874172185431)
(1488, 5, 0.9878345498783455, 0.9802631578947368, 0.9867549668874173)
(1489, 10, 0.975669099756691, 0.9668874172185431, 0.9668874172185431)
(1490, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1491, 10, 0.975669099756691, 0.9668874172185431, 0.9668874172185431)
(1492, 8, 0.9805352798053528, 0.9735099337748344, 0.9735099337748344)
(1493, 3, 0.9927007299270073, 0.9933333333333333, 0.9867549668874173)
(0.9705882352941176, 0.9333333333333333, 0.9655172413793104)
naila@naila-ThinkPad-E460:~/5thSemester/SMAI/assignment1$ _
```

The lines with 5 parameters refer to the epoch number, number of errors, accuracy, precision and recall for that particular epoch. I used a 80-20 train/validation split. We can see that the number of errors reduces to 3 in the last epoch with the accuracy, precision and recall reaching around 0.99. The number of errors has slowly reduced with the number of epochs. The last line is for the validation set, we can see that the 3 parameters refer to the accuracy, precision and recall for the validation data. The accuracy is around 0.98 whereas the precision is around 0.93 and recall around 0.97.