Statistic Machine Learning

Digit Recognition

Presented By: Nadav Gover

<u>Part A</u>

In this part we used a multi-class perceptron classifier. A multi-class perceptron classifier is the extension of the Perceptron Learning Algorithm (PLA) for binary classification that was developed in class.
We used one-vs-all strategy which means we trained each class to identify if the input is from its class or not (binary classification).
The input, in this case (so as in part B), was the MNIST data set which is constructed out of 70,000 images of hand written digits between 0-9. This means that for our multi-class classifier we trained 10 different classes (1 for each digit).
To then predict the input image correctly we predict the label that gets the highest confidence, like so:

$$\hat{y} = argmax_{y \in \{0,\ldots,9\}} (w^y)^T x$$

Where w^y are the weights of class y and x is the input image.

The Perceptron Learning Algorithm for binary classification in the linearly separable case:

- Initialization: w = w0

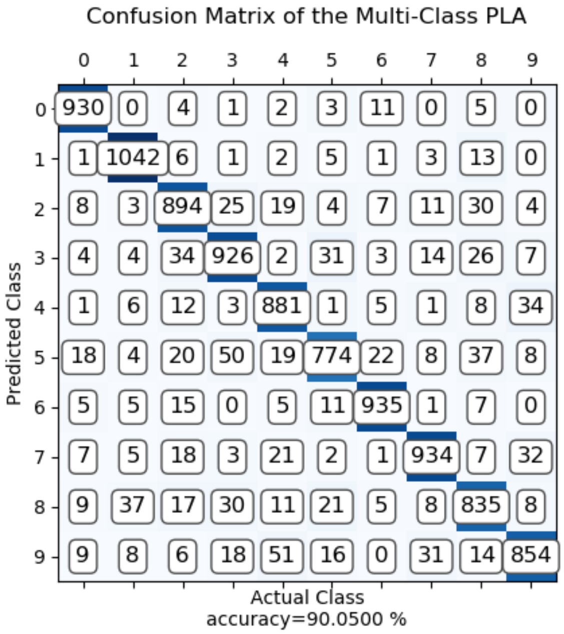
- Pick a misclassified example and denote it as $(x(t), y(t))$
i.e., $y(t) \neq h(x(t)) = sign(w^T(t)x(t))$

- Update the weights $w(t) + y(t)x(t) \rightarrow w(t + 1)$

-Continue iterating until there are no misclassified examples in the training set.

Since the input data is not linearly separable, PLA will never terminate, and its behavior can become quite unstable, and can jump from a good perceptron to a very bad one within one update. To overcome this, we used the pocket algorithm as described in the assignment.

Following are the resulting confusion matrix and the confusion table of each class. In each confusion table the sensitivity is written and in the confusion matrix the accuracy is written.

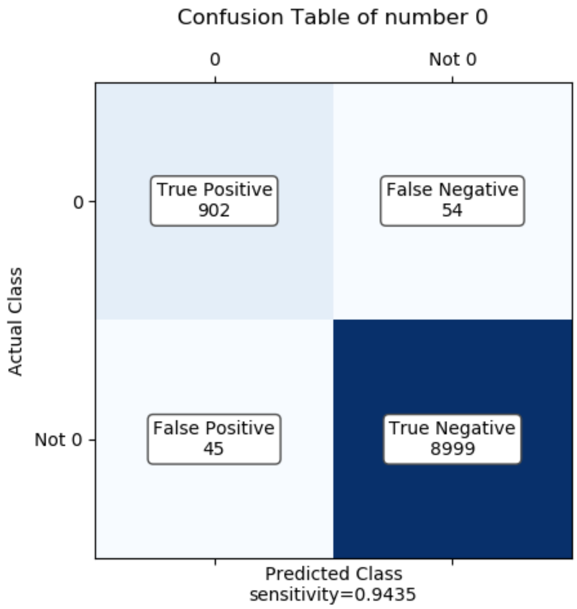


Confusion Matrix of the Multi-Class PLA

Predicted Class

	0	1	2	3	4	5	6	7	8	9
0	930	0	4	1	2	3	11	0	5	0
1	1	1042	6	1	2	5	1	3	13	0
2	8	3	894	25	19	4	7	11	30	4
3	4	4	34	926	2	31	3	14	26	7
4	1	6	12	3	881	1	5	1	8	34
5	18	4	20	50	19	774	22	8	37	8
6	5	5	15	0	5	11	935	1	7	0
7	7	5	18	3	21	2	1	934	7	32
8	9	37	17	30	11	21	5	8	835	8
9	9	8	6	18	51	16	0	31	14	854

Actual Class
accuracy=90.0500 %

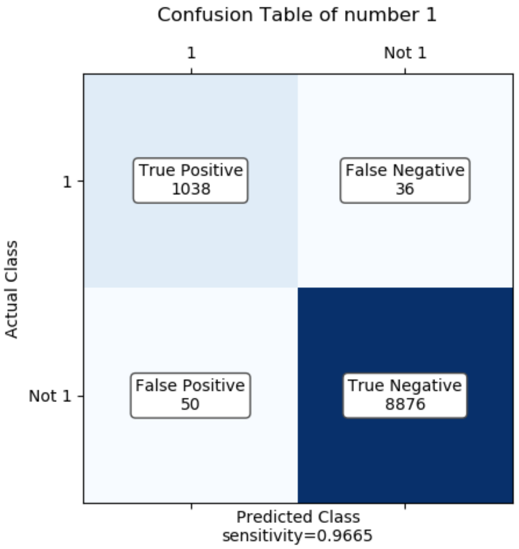


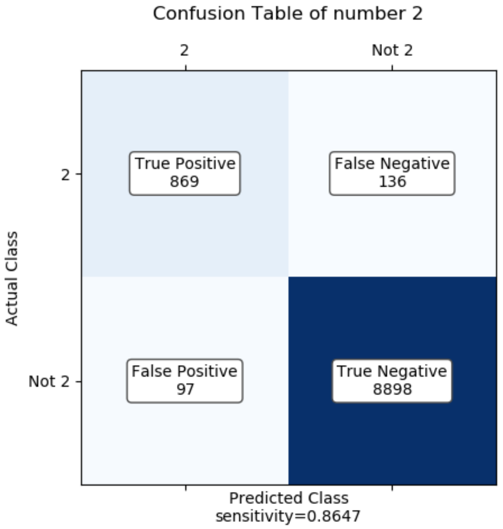
Confusion Table of number 0

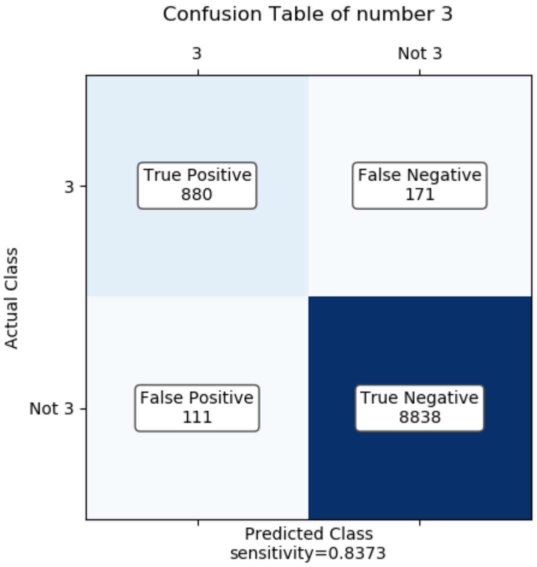
Actual Class

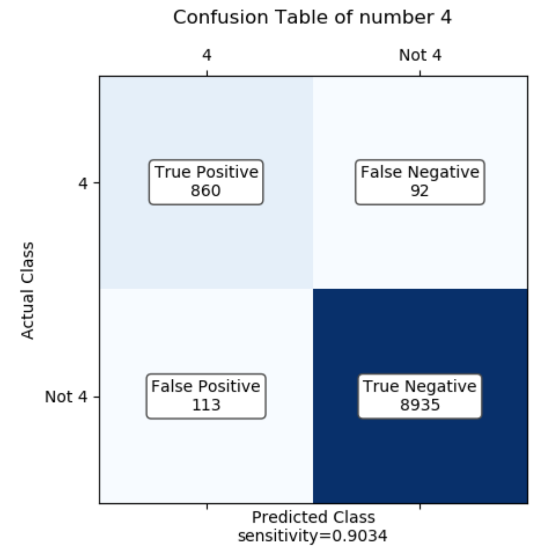
	0	Not 0
0	True Positive 902	False Negative 54
Not 0	False Positive 45	True Negative 8999

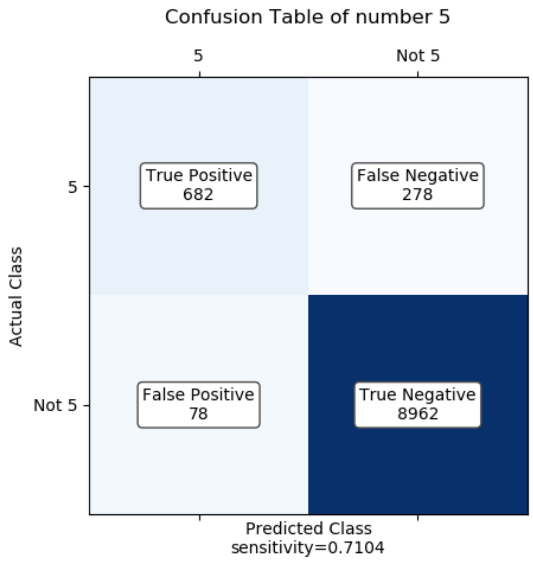
Predicted Class
sensitivity=0.9435

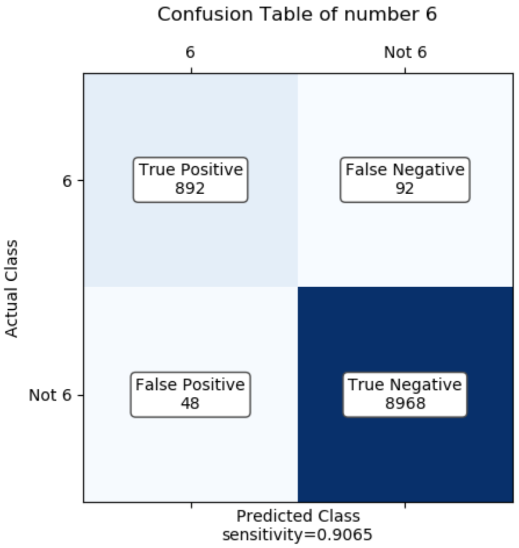

Confusion Table of number 1
1
Not 1
1
Not 1
Actual Class
True Positive
1038
False Negative
36
False Positive
50
True Negative
8876
Predicted Class
sensitivity=0.9665

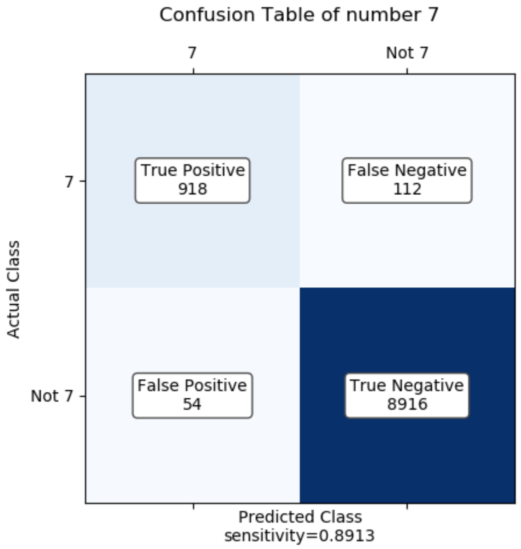

Confusion Table of number 2
2
Not 2
2
Not 2
Actual Class
True Positive
869
False Negative
136
False Positive
97
True Negative
8898
Predicted Class
sensitivity=0.8647

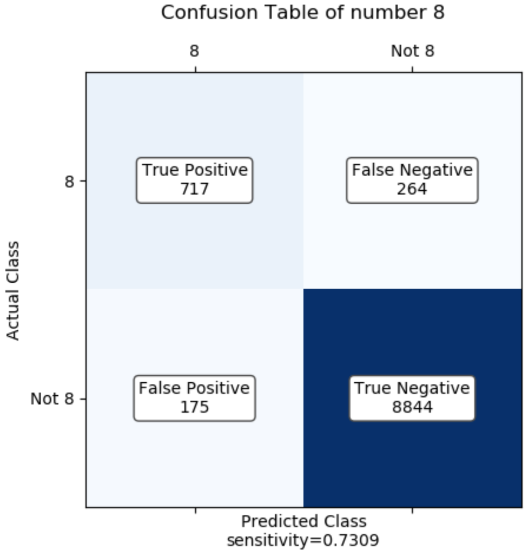

Confusion Table of number 3
3
Not 3
3
Not 3
Actual Class
True Positive
880
False Negative
171
False Positive
111
True Negative
8838
Predicted Class
sensitivity=0.8373

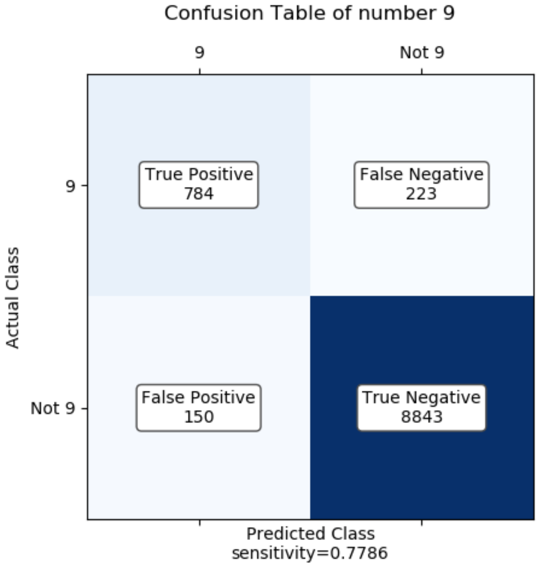

Confusion Table of number 4
4
Not 4
Actual Class
4
Not 4
True Positive
860
False Negative
92
False Positive
113
True Negative
8935
Predicted Class
sensitivity=0.9034


Confusion Table of number 5
5
Not 5
Actual Class
5
Not 5
True Positive
682
False Negative
278
False Positive
78
True Negative
8962
Predicted Class
sensitivity=0.7104


Confusion Table of number 6
6
Not 6
Actual Class
6
Not 6
True Positive
892
False Negative
92
False Positive
48
True Negative
8968
Predicted Class
sensitivity=0.9065


Confusion Table of number 7


Confusion Table of number 8


Confusion Table of number 9

Conclusions from the above results

First, we will discuss the sensitivity. Sensitivity or True Positive Rate (TPR), is the proportion of inputs that tested positive and are positive (True Positive, TP) of all the inputs that actually are positive (TP + FN). With higher sensitivity, fewer actual cases of misclassification go undetected. So the higher the sensitivity the better.

Looking in the results above we can see that classes of number 0, 1, 2, 3, 4, 6, 7 have sensitivity greater than 0.8 and is good enough for us.

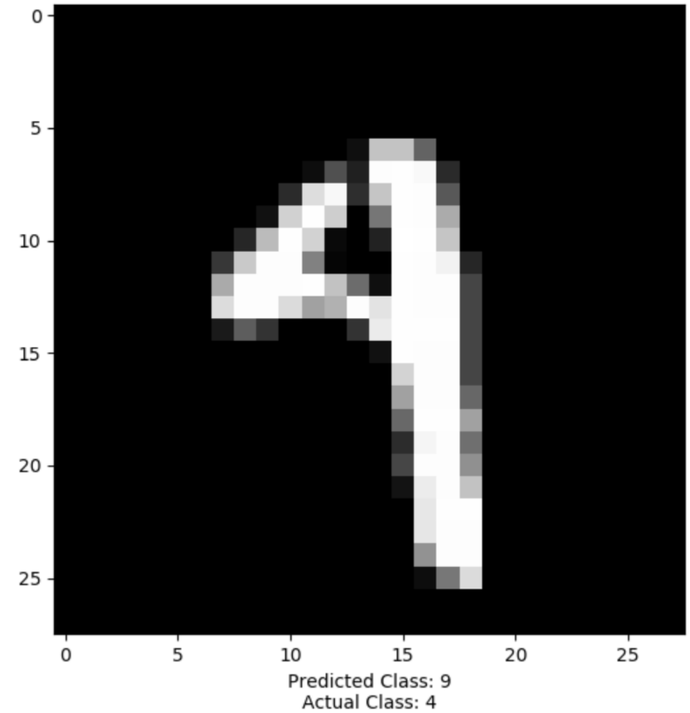
For the other classes (classes 5, 8, 9) have sensitivity less than 0.8. To understand that lets look in the confusion matrix.

In the confusion matrix of class 5 we can see that the multi-class PLA has trouble distinguishing mostly between 5 and 3 and 8 (when the predicted class is 5, the actual class is more than 30 times 3 or 8). The reason for that might be that the actual input is blurry and even hardly recognizable by humans. The digits 5, 3, 8 might be similar in some handwritings.

Same goes to classes 8, 9.
In class 8 we can see from the confusion matrix that the multi-class PLA has trouble distinguishing mostly between 8 and 1 and 3 (when the predicted class is 8, the actual class is more than 30 times 1 or 3).

In class 9 we can see from the confusion matrix that the multi-class PLA has trouble distinguishing mostly between 9 and 1 and 3 (when the predicted class is 9, the actual class is more than 30 times 4 or 7).

Just to demonstrate how similar 9 can be to 4 for example, look at the following image.



In the above image there is an example of misclassification. The predicted class is 9 and the actual class is 4. This image would most likely be interpreted for a human reader as either a 4 or a 9 (perhaps even tending more to 9 as classified) and maybe the reader will need a context to understand. This image comes to demonstrate the difficulty of a human reader to distinguish between these two classes so all the more so to a computer and all the more so to a computer without a context.

Now we come to discuss the accuracy. As seen in the confusion matrix the accuracy is approximately 90%.

Accuracy is not a reliable metric for the real performance of a classifier, because it will yield misleading results if the data set is unbalanced (that is, when the numbers of observations in different classes vary greatly). For example, if there were 95 cats and only 5 dogs in the data, a classifier might classify all the observations as cats. The overall accuracy would be 95%, but in more detail the classifier would have a 100% recognition rate for the cat class but a 0% recognition rate for the dog class.

In our case though, we used the MNIST data set which is a balanced data set, so the above issue is not valid.

This accuracy was achieved using 1 epoch, using more epochs did not yield any significant better results. This means the weights were updated enough times to preform a good classifier. Using 1 epoch has an advantage of less training time.

Let's compare our accuracy of 90% to historical benchmarks of the MNIST data set. As can be seen here an accuracy of 92.4% using linear classifiers (as we did) is considered an historical benchmark. So this 90% accuracy is good indeed.

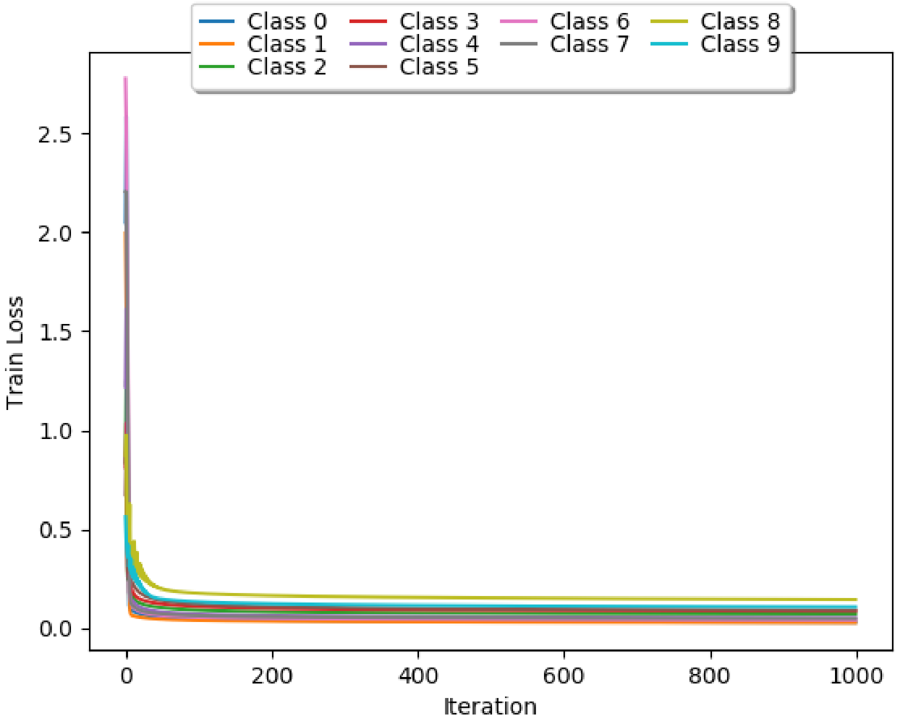
Part B

In this part we applied a Softmax Regression on the MNIST dataset. Softmax regression (or multinomial logistic regression) is a generalization of logistic regression, that was developed in class, to the case where we want to handle multiple classes, i.e., $y_n \in \{1, \dots, K\}$ where K is the number of classes.

The given cost function is the cross-entropy error that was developed in class but generalized to K classes. So in the code you can see the cross-entropy was used to each class and the corresponding gradient, and in the end it was all summed to create the cost function given in the assignment.

The composition function is the sigmoid function. Since zero valued inputs can artificially kill weight updates, the inputs were scaled to be between 0.01 to 1 and not 0 to 1.

In this part there was a need to find the optimal number of iterations for training the model. Optimal in the sense of good accuracy and short run-time. For this need, the train loss came to help.
Looking at the train loss of all 10 classes one can see where the loss converges to a minimum.



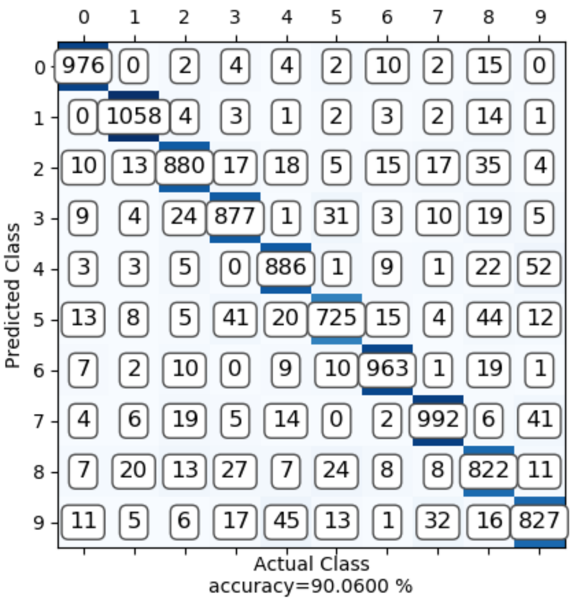
Above is the train loss of all classes as a function of the iteration number.

One thing to notice here is that in all the classes the train loss is converging to its minimum approximately at the same time. Another thing is that it converges rather quickly (around 200 iterations) and after that there is no significant difference in the loss. The loss is correlated with the accuracy, so we can deduce that the optimal number of iterations is 200-300.

On another note, we can see that the minimum of class 8 is the highest so we can expect that its sensitivity will be the lowest (we will test this hypothesis later).

Following are the resulting confusion matrix and the confusion table of each class. In each confusion table the sensitivity is written and in the confusion matrix the accuracy is written.

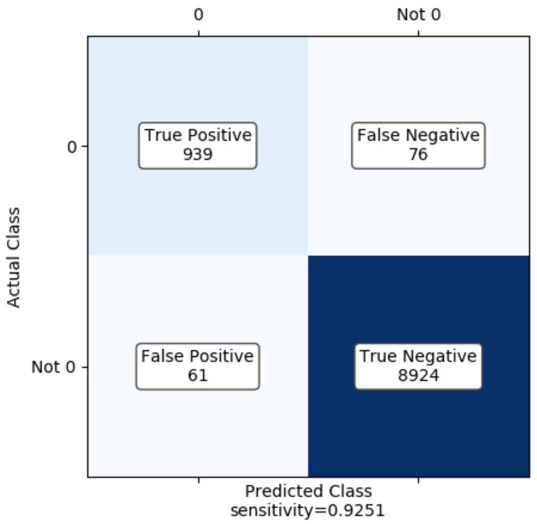
Confusion Matrix of the Multi-Class Linear Regression



Predicted Class \ Actual Class	0	1	2	3	4	5	6	7	8	9
0	976	0	2	4	4	2	10	2	15	0
1	0	1058	4	3	1	2	3	2	14	1
2	10	13	880	17	18	5	15	17	35	4
3	9	4	24	877	1	31	3	10	19	5
4	3	3	5	0	886	1	9	1	22	52
5	13	8	5	41	20	725	15	4	44	12
6	7	2	10	0	9	10	963	1	19	1
7	4	6	19	5	14	0	2	992	6	41
8	7	20	13	27	7	24	8	8	822	11
9	11	5	6	17	45	13	1	32	16	827

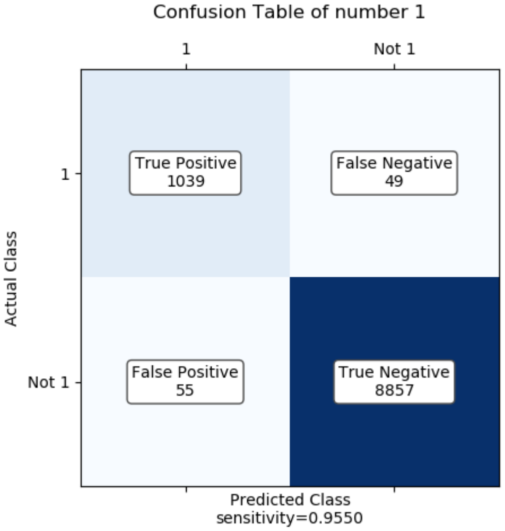
accuracy=90.0600 %

Confusion Table of number 0



Actual Class \ Predicted Class	0	Not 0
0	True Positive 939	False Negative 76
Not 0	False Positive 61	True Negative 8924

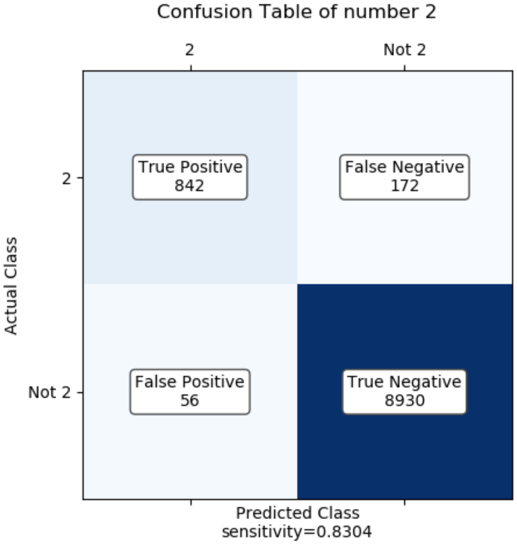
sensitivity=0.9251



Confusion Table of number 1

Actual Class \ Predicted Class	1	Not 1
1	True Positive 1039	False Negative 49
Not 1	False Positive 55	True Negative 8857

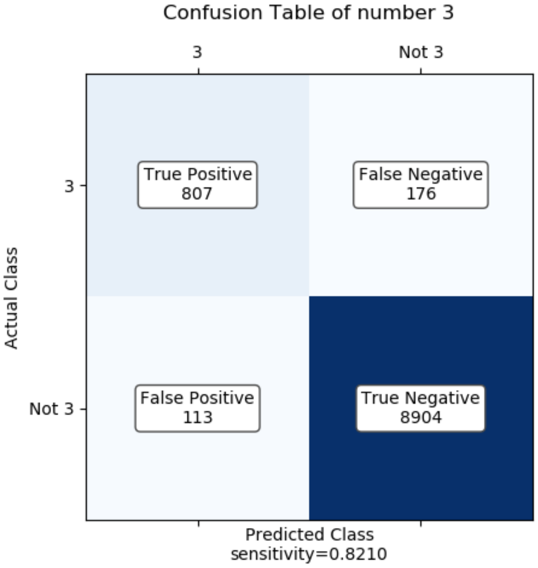
Predicted Class
sensitivity=0.9550



Confusion Table of number 2

Actual Class \ Predicted Class	2	Not 2
2	True Positive 842	False Negative 172
Not 2	False Positive 56	True Negative 8930

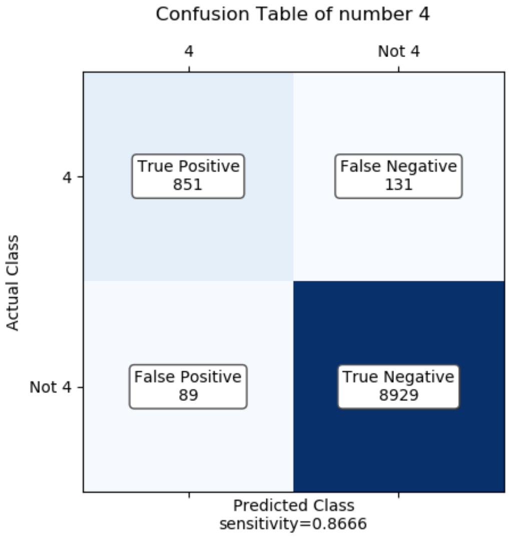
Predicted Class
sensitivity=0.8304

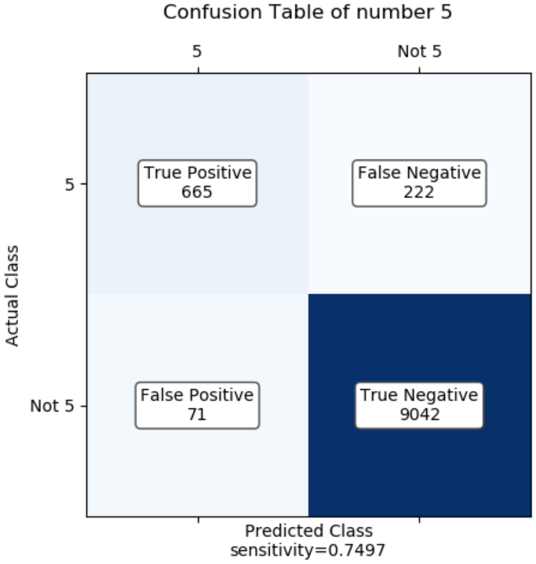


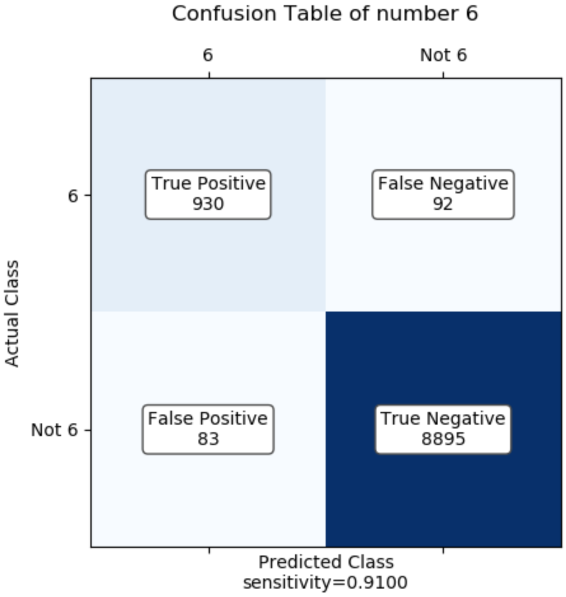
Confusion Table of number 3

Actual Class \ Predicted Class	3	Not 3
3	True Positive 807	False Negative 176
Not 3	False Positive 113	True Negative 8904

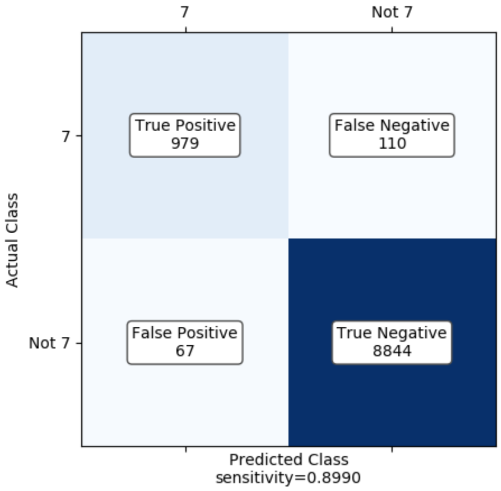
Predicted Class
sensitivity=0.8210


Confusion Table of number 4
4
Not 4
4
Not 4
Actual Class
True Positive
851
False Negative
131
False Positive
89
True Negative
8929
Predicted Class
sensitivity=0.8666


Confusion Table of number 5
5
Not 5
5
Not 5
Actual Class
True Positive
665
False Negative
222
False Positive
71
True Negative
9042
Predicted Class
sensitivity=0.7497


Confusion Table of number 6
6
Not 6
6
Not 6
Actual Class
True Positive
930
False Negative
92
False Positive
83
True Negative
8895
Predicted Class
sensitivity=0.9100

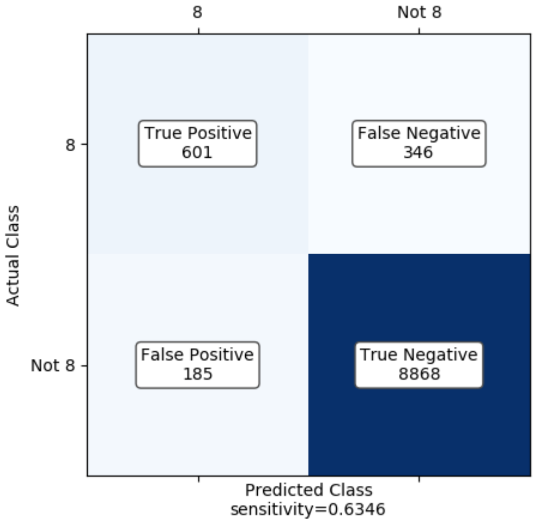
Confusion Table of number 7



	7	Not 7
7	True Positive 979	False Negative 110
Not 7	False Positive 67	True Negative 8844

sensitivity=0.8990

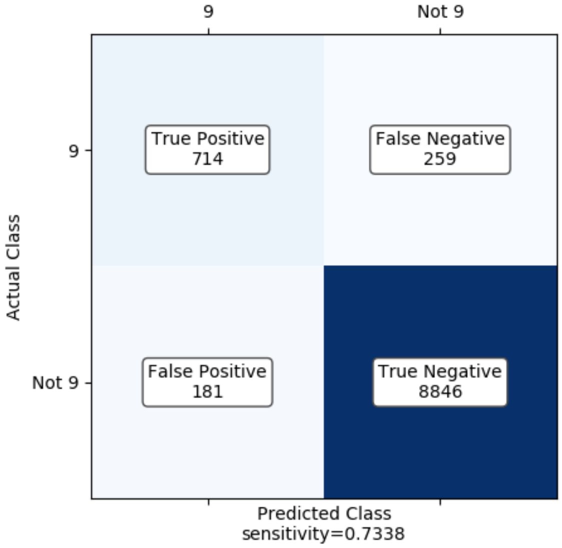
Confusion Table of number 8



	8	Not 8
8	True Positive 601	False Negative 346
Not 8	False Positive 185	True Negative 8868

sensitivity=0.6346

Confusion Table of number 9



	9	Not 9
9	True Positive 714	False Negative 259
Not 9	False Positive 181	True Negative 8846

sensitivity=0.7338

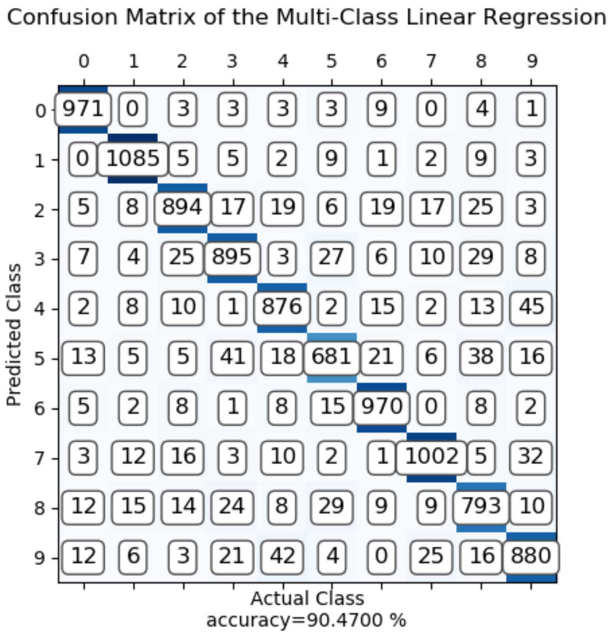
Conclusions from the above results

The sensitivity has similar explanations as in part A. There we saw it is even difficult for humans to distinguish between the digits so all the more so computers.

Recall the hypothesis from above that class 8 will have the lowest sensitivity, from the above charts we can see that this hypothesis is correct and that it is not a coincidence.

The confusion matrix was generated with 300 iterations (above we explained the reason to the number 300). We can see that the accuracy is 90.06%.

Just to justify the number of iterations, following is the confusion matrix generated with 1,000 iterations:



	0	1	2	3	4	5	6	7	8	9
0	971	0	3	3	3	3	9	0	4	1
1	0	1085	5	5	2	9	1	2	9	3
2	5	8	894	17	19	6	19	17	25	3
3	7	4	25	895	3	27	6	10	29	8
4	2	8	10	1	876	2	15	2	13	45
5	13	5	5	41	18	681	21	6	38	16
6	5	2	8	1	8	15	970	0	8	2
7	3	12	16	3	10	2	1	1002	5	32
8	12	15	14	24	8	29	9	9	793	10
9	12	6	3	21	42	4	0	25	16	880

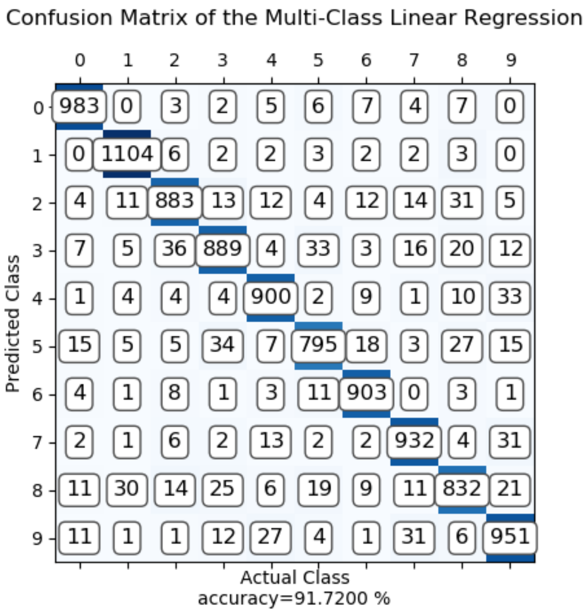
---Confusion matrix generated with 1,000 iterations---

We can note that the accuracy here is 90.47%.

This means that the difference in accuracies between the 300 and 1,000 iterations is $\approx 0.4\%$. Some may argue this is significant, but we omitted a very important detail, the time it took to generate this result.

The former confusion matrix was generated in 3 minutes and the latter in 10. This is a very important detail and the compensations between time over accuracy is an ongoing dilemma in the world of machine learning.

Just to emphasize this point even more, following is a confusion matrix generated with 10,000 iterations (one order of scale larger than 1,000).



	0	1	2	3	4	5	6	7	8	9
0	983	0	3	2	5	6	7	4	7	0
1	0	1104	6	2	2	3	2	2	3	0
2	4	11	883	13	12	4	12	14	31	5
3	7	5	36	889	4	33	3	16	20	12
4	1	4	4	4	900	2	9	1	10	33
5	15	5	5	34	7	795	18	3	27	15
6	4	1	8	1	3	11	903	0	3	1
7	2	1	6	2	13	2	2	932	4	31
8	11	30	14	25	6	19	9	11	832	21
9	11	1	1	12	27	4	1	31	6	951

---Confusion matrix generated with 10,000 iterations---

Note that the accuracy is 91.72% and the difference between this accuracy and the 1,000 iterations accuracy is 1.25%. This accuracy took 95 minutes to achieve. You can see for yourselves the big timing difference and small accuracy difference.

Of course, one will prefer the more accurate model over the less accurate one, but due to timing limitations in this assignment, 300 iterations were chosen as a compromise.

Final Conclusions

The model in both parts preformed well in accuracy terms and sensitivity terms. We saw that the model is not far from historical benchmarks and thus is successful.