PR: SVM

November 17, 2022

S.M. Erfan Moosavi M.

- ***Parsing Data (0_parser.ipynb)***

For starter, I wrote a simple parser to parse input data.

This parser reads binary files which was downloaded from the provided link (http://yann.lecun.com/exdb/mnist/) and then convert them to CSV datasets. In this way we can use NumPy built-in functions to read data.

So, let's talk about parser. When you open up the parser notebook, first you should set up some parameters, these parameters are the path to binary datasets and save folder for CSV dataset.

There are 4 cells related to data parsing and 2 cells for checking parsed test and train images.

You can run all the cells altogether or you can run them one by one. In testing cells, you can pass the index to image and see it's visualization along with its label.

This code could have been integrated with other codes but I preferred to make the code modular so I would have more control over each module.

- ***Preprocessing (1_data_preprocess.ipynb)***

In this module, we are going to preprocess the CSV data that was generated by previous module in two steps.

In the first step, we're going to normalize the data. For this task we're going to use min-max normalization. This normalization applies the following formula on each sample:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$ from Wikipedia

The normalization is usually done when the dataset consists of features with different ranges, but for our case, all our features are pixel values between 0 to 255. But empirically I found out that normalized features tend to be faster when it comes to model training and it doesn't affect the accuracy of the models so much. So, I decided to do the normalization.

I used built-in MinMaxScalar of sklearn library.

The second preprocessing step is to run PCA on dataset. Before this step, I concatenated the test and train data so they could be feature extracted together. Basically, if we do PCA over test and train separately we usually end up with different features, even if the number of features be the same. So, we can do two things, first is to use the extracted matrix from PCA operations to extract new features from new samples, the second thing is to concatenate test and train data and do one PCA over them, I did the latter and I kept a Boolean mask vector which keeps track of test and train samples.

We use PCA with two feature counts. Keeping 30 and 100 features.

So, we end up with 3 different set of data. First is the data that is not fed to PCA and have all 784 features, but it got scaled. Second one is the set of data that is scaled and fed to PCA30 and the third one is the one that is scaled and fed to PCA100. We shall call them Full set, PCA30 and PCA100.

Then we stored these sets along with output set (let’s call it y, and this output is a concatenation of test and train output but with the same order of inputs). And we also saved the mask vector.

- ***Full Set Evaluation (2_intro_full.ipynb)***

Introduction of questions sheet asks us to train three different models corresponds to three different sets. In this section and next two sections we are talking about evaluations that are done over every one of these sets.

The first test is done over full dataset. We loaded the preprocessed datasets and then using the mask vector, separated the test and train sets. After that, using LinearSVC class of sklearn, we trained a multiclass SVM classifier using One VS Rest technique. To solve the SVM optimization there are two options, first is to solve the dual problem and the second option is to solve the primal problem. We choose to solve the primal because empirically the dual option causes the algorithm to not converge. Also, it is not suggested by sklearn documentation since n_samples > n_features. Then, using the trained SVM classifier, we predicted labels for our train and test samples. Here are the results:

Model: Linear SVM (LinearSVM class from sklearn)

Train Accuracy: 92.63%

Test Accuracy: 91.81%

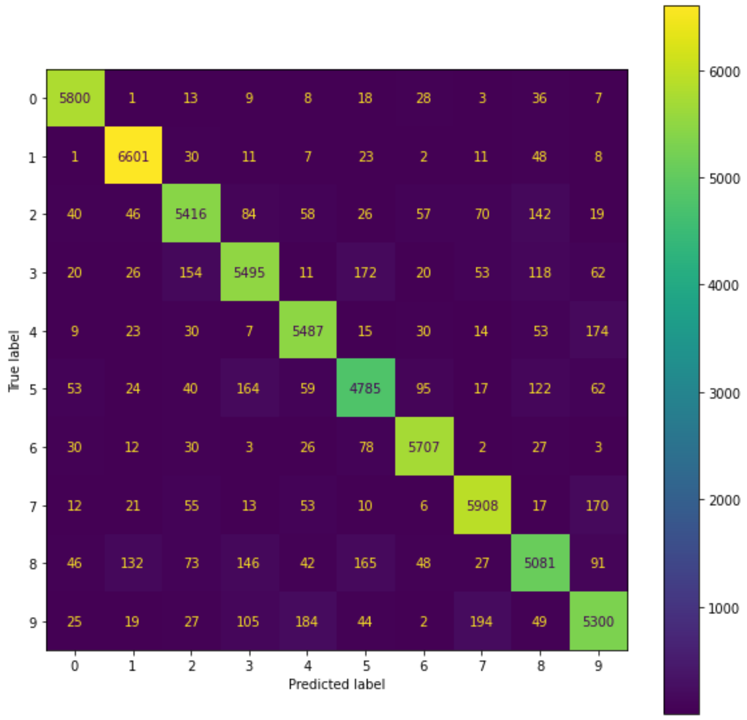
Train Precision and Recall:

CLASSES	0	1	2	3	4	5	6	7	8	9
PRECISION (TRAIN)	0.96	0.95	0.92	0.91	0.92	0.89	0.95	0.93	0.89	0.89
RECALL (TRAIN)	0.97	0.97	0.90	0.89	0.93	0.88	0.96	0.94	0.86	0.89

Test Precision and Recall:

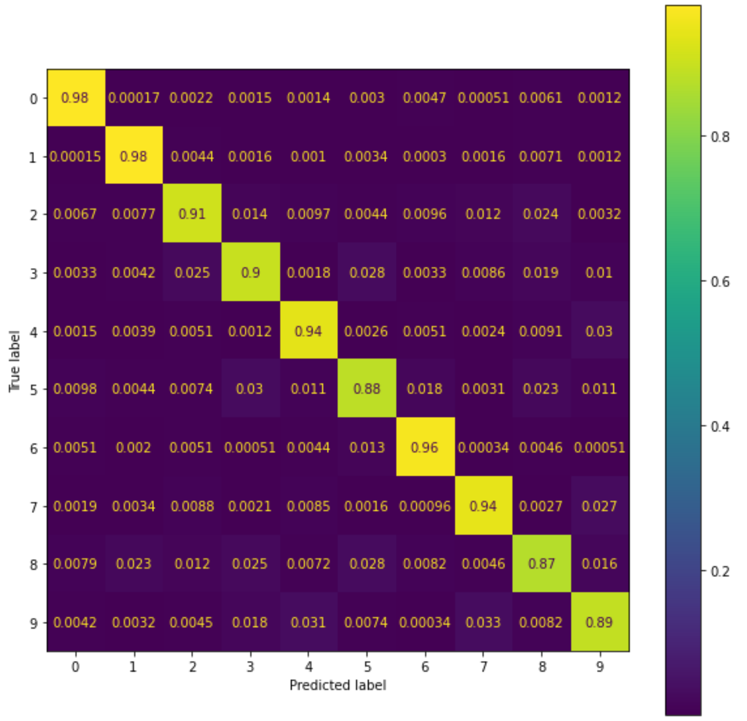
CLASSES	0	1	2	3	4	5	6	7	8	9
PRECISION (TEST)	0.94	0.95	0.92	0.89	0.91	0.88	0.93	0.92	0.87	0.89
RECALL (TEST)	0.98	0.97	0.88	0.90	0.92	0.85	0.94	0.92	0.86	0.88

Train Confusion Matrix:



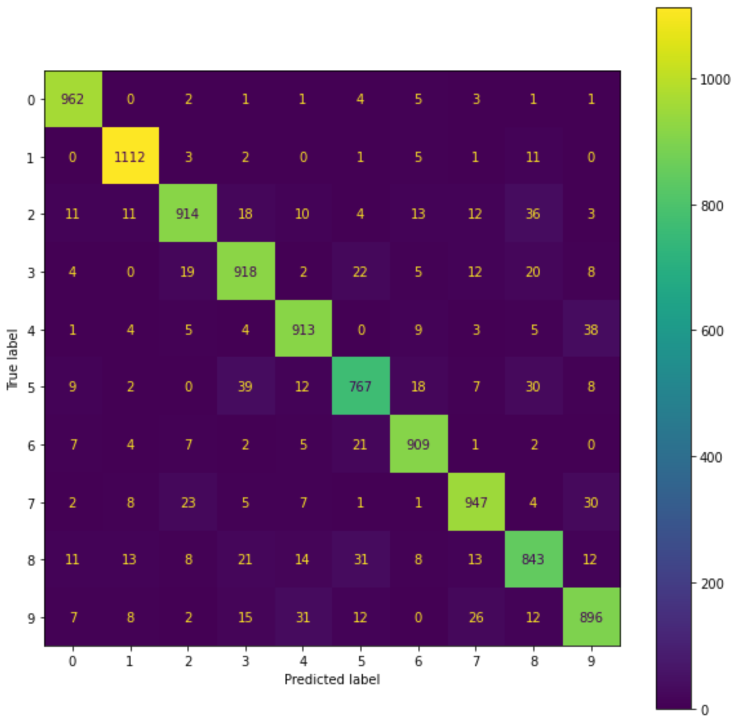
	0	1	2	3	4	5	6	7	8	9
0	5800	1	13	9	8	18	28	3	36	7
1	1	6601	30	11	7	23	2	11	48	8
2	40	46	5416	84	58	26	57	70	142	19
3	20	26	154	5495	11	172	20	53	118	62
4	9	23	30	7	5487	15	30	14	53	174
5	53	24	40	164	59	4785	95	17	122	62
6	30	12	30	3	26	78	5707	2	27	3
7	12	21	55	13	53	10	6	5908	17	170
8	46	132	73	146	42	165	48	27	5081	91
9	25	19	27	105	184	44	2	194	49	5300

Normalized Train Confusion Matrix:



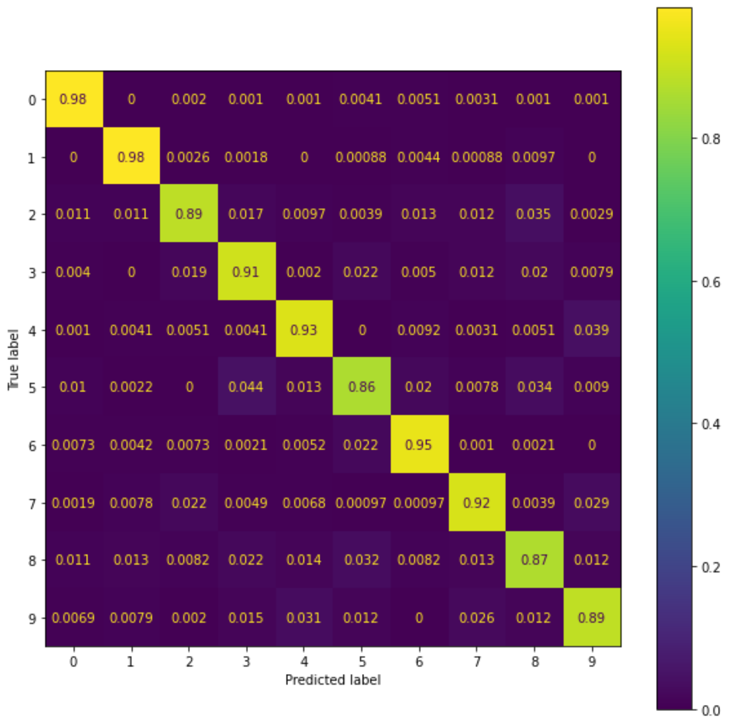
	0	1	2	3	4	5	6	7	8	9
0	0.98	0.00017	0.0022	0.0015	0.0014	0.003	0.0047	0.00051	0.0061	0.0012
1	0.00015	0.98	0.0044	0.0016	0.001	0.0034	0.0003	0.0016	0.0071	0.0012
2	0.0067	0.0077	0.91	0.014	0.0097	0.0044	0.0096	0.012	0.024	0.0032
3	0.0033	0.0042	0.025	0.9	0.0018	0.028	0.0033	0.0086	0.019	0.01
4	0.0015	0.0039	0.0051	0.0012	0.94	0.0026	0.0051	0.0024	0.0091	0.03
5	0.0098	0.0044	0.0074	0.03	0.011	0.88	0.018	0.0031	0.023	0.011
6	0.0051	0.002	0.0051	0.00051	0.0044	0.013	0.96	0.00034	0.0046	0.00051
7	0.0019	0.0034	0.0088	0.0021	0.0085	0.0016	0.00096	0.94	0.0027	0.027
8	0.0079	0.023	0.012	0.025	0.0072	0.028	0.0082	0.0046	0.87	0.016
9	0.0042	0.0032	0.0045	0.018	0.031	0.0074	0.00034	0.033	0.0082	0.89

Test Confusion Matrix:



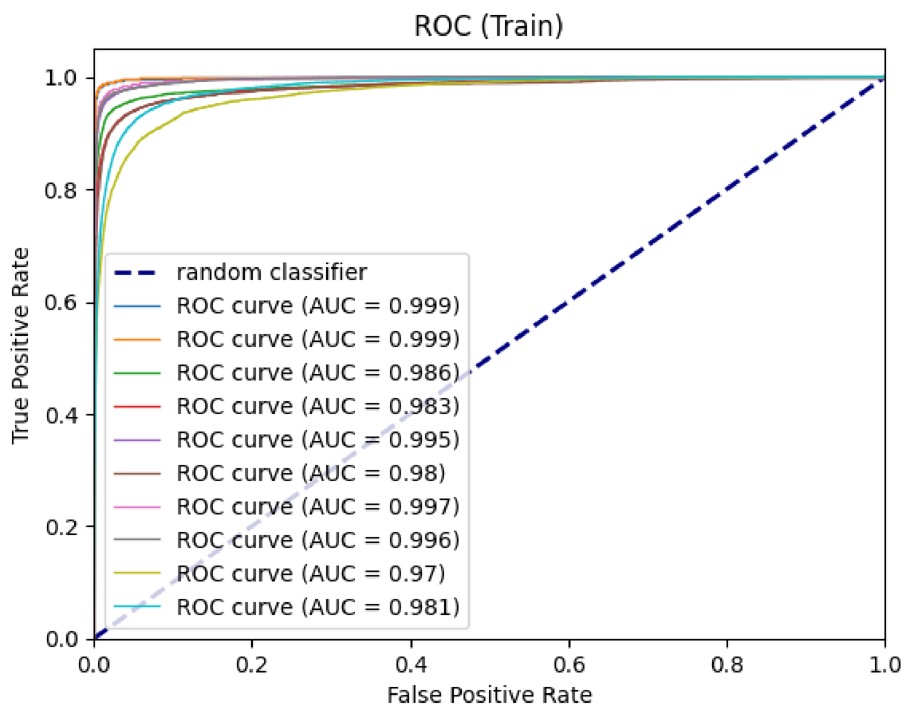
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	962	0	2	1	1	4	5	3	1	1
1	0	1112	3	2	0	1	5	1	11	0
2	11	11	914	18	10	4	13	12	36	3
3	4	0	19	918	2	22	5	12	20	8
4	1	4	5	4	913	0	9	3	5	38
5	9	2	0	39	12	767	18	7	30	8
6	7	4	7	2	5	21	909	1	2	0
7	2	8	23	5	7	1	1	947	4	30
8	11	13	8	21	14	31	8	13	843	12
9	7	8	2	15	31	12	0	26	12	896

Normalized Test Confusion Matrix:

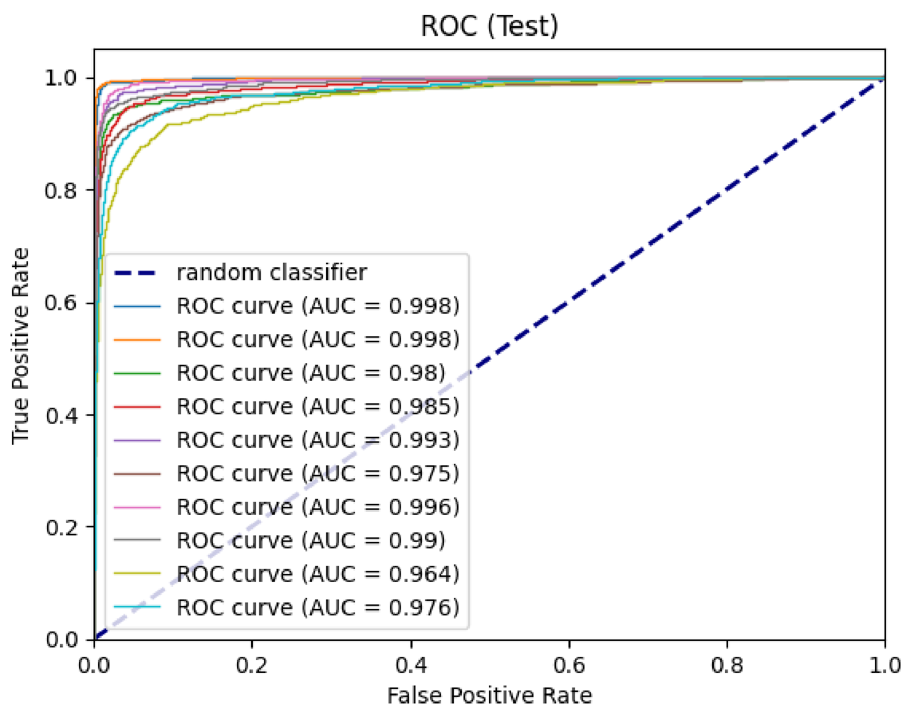


True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	0.98	0	0.002	0.001	0.001	0.0041	0.0051	0.0031	0.001	0.001
1	0	0.98	0.0026	0.0018	0	0.00088	0.0044	0.00088	0.0097	0
2	0.011	0.011	0.89	0.017	0.0097	0.0039	0.013	0.012	0.035	0.0029
3	0.004	0	0.019	0.91	0.002	0.022	0.005	0.012	0.02	0.0079
4	0.001	0.0041	0.0051	0.0041	0.93	0	0.0092	0.0031	0.0051	0.039
5	0.01	0.0022	0	0.044	0.013	0.86	0.02	0.0078	0.034	0.009
6	0.0073	0.0042	0.0073	0.0021	0.0052	0.022	0.95	0.001	0.0021	0
7	0.0019	0.0078	0.022	0.0049	0.0068	0.00097	0.00097	0.92	0.0039	0.029
8	0.011	0.013	0.0082	0.022	0.014	0.032	0.0082	0.013	0.87	0.012
9	0.0069	0.0079	0.002	0.015	0.031	0.012	0	0.026	0.012	0.89

Train ROC Plot & AUC Values:



Train ROC Plot & AUC Values:



- ***PCA30 Set Evaluation (3_intro_pca30.ipynb)***

The second test is done over PCA30 dataset. We loaded the preprocessed datasets (Scaled and PCA30) and then using the mask vector, separated the test and train sets. After that, using LinearSVC class of sklearn, we trained a multiclass SVM classifier using One VS Rest technique. To solve the SVM optimization there are two options, first is to solve the dual problem and the second option is to solve the primal problem. We choose to solve the primal because empirically the dual option causes the algorithm to not converge. Also, it is not suggested by sklearn documentation since n_samples > n_features. Then, using the trained SVM classifier, we predicted labels for our train and test samples. Here are the results:

Model: Linear SVM (LinearSVM class from sklearn)

Train Accuracy: 87.66%

Test Accuracy: 88.38% (seems the test set was easier to classify)

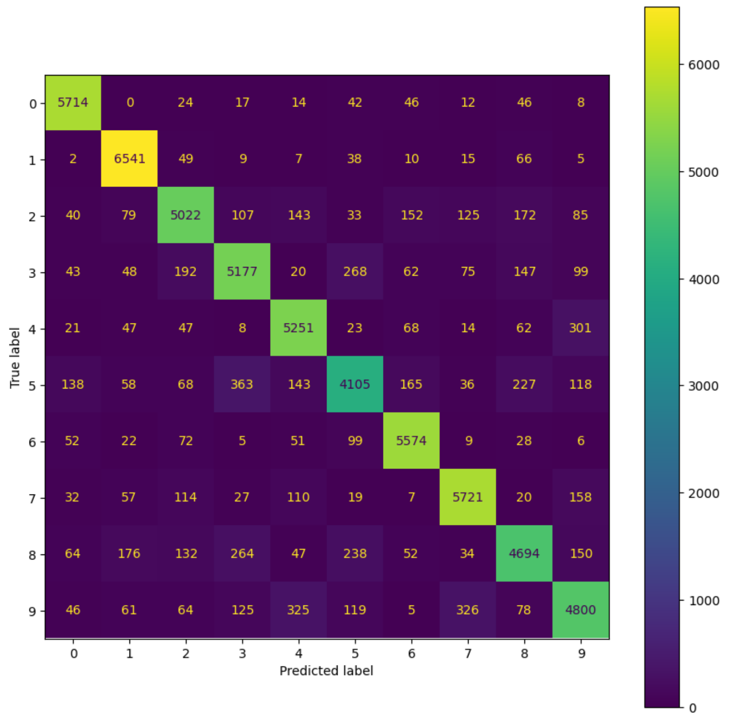
Train Precision and Recall:

CLASSES	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
PRECISION (TRAIN)	0.92	0.92	0.86	0.84	0.85	0.82	0.90	0.89	0.84	0.83
RECALL (TRAIN)	0.96	0.97	0.84	0.84	0.89	0.75	0.94	0.91	0.80	0.80

Test Precision and Recall:

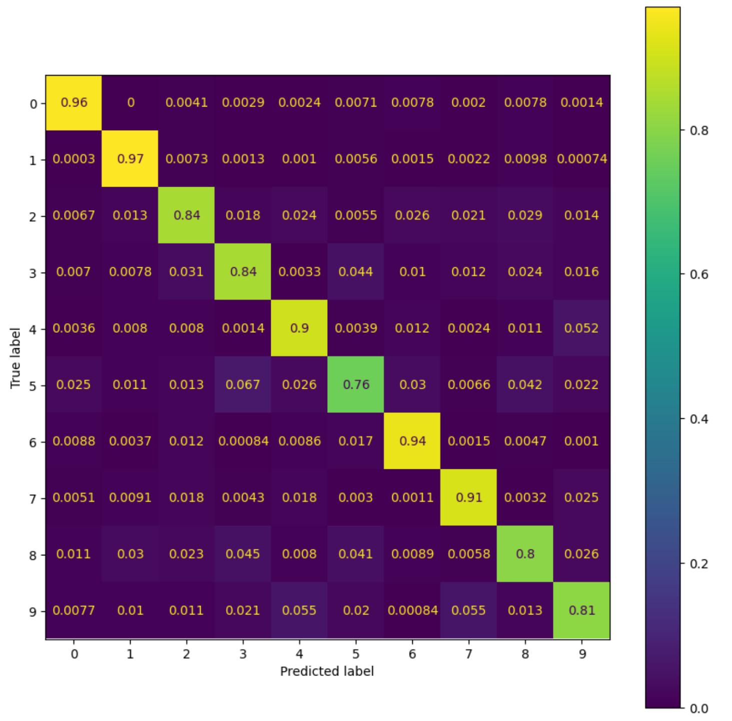
CLASSES	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
PRECISION (TEST)	0.91	0.94	0.88	0.85	0.85	0.83	0.91	0.90	0.85	0.85
RECALL (TEST)	0.97	0.97	0.84	0.86	0.90	0.77	0.93	0.90	0.82	0.81

Train Confusion Matrix:



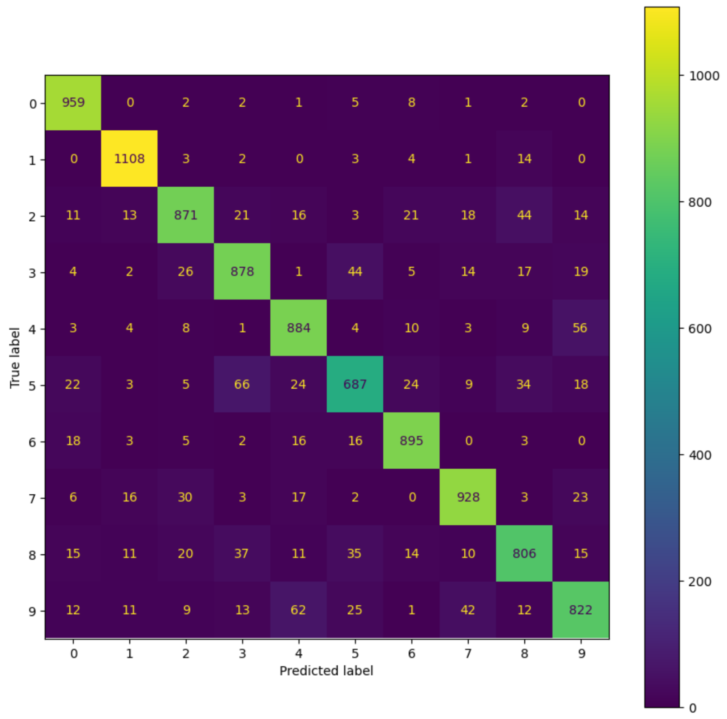
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	5714	0	24	17	14	42	46	12	46	8
1	2	6541	49	9	7	38	10	15	66	5
2	40	79	5022	107	143	33	152	125	172	85
3	43	48	192	5177	20	268	62	75	147	99
4	21	47	47	8	5251	23	68	14	62	301
5	138	58	68	363	143	4105	165	36	227	118
6	52	22	72	5	51	99	5574	9	28	6
7	32	57	114	27	110	19	7	5721	20	158
8	64	176	132	264	47	238	52	34	4694	150
9	46	61	64	125	325	119	5	326	78	4800

Normalized Train Confusion Matrix:



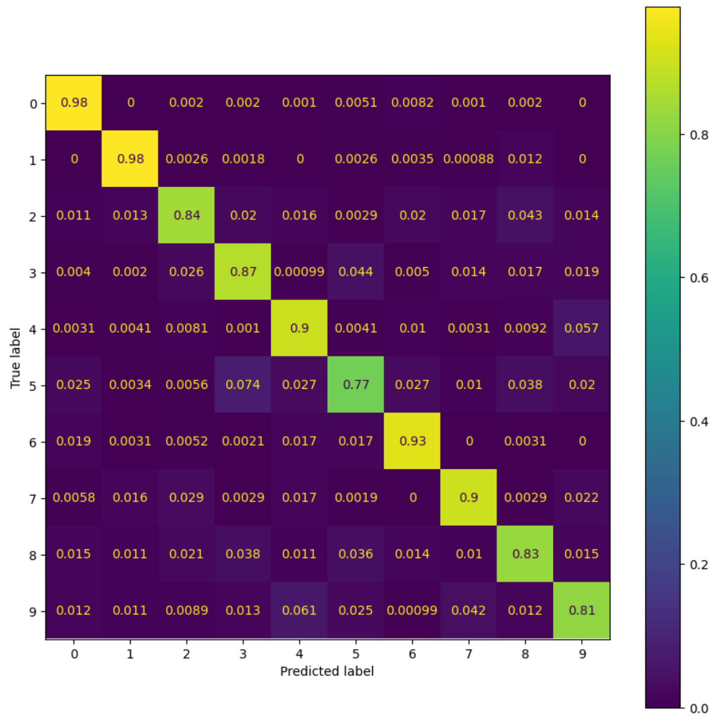
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	0.96	0	0.0041	0.0029	0.0024	0.0071	0.0078	0.002	0.0078	0.0014
1	0.0003	0.97	0.0073	0.0013	0.001	0.0056	0.0015	0.0022	0.0098	0.00074
2	0.0067	0.013	0.84	0.018	0.024	0.0055	0.026	0.021	0.029	0.014
3	0.007	0.0078	0.031	0.84	0.0033	0.044	0.01	0.012	0.024	0.016
4	0.0036	0.008	0.008	0.0014	0.9	0.0039	0.012	0.0024	0.011	0.052
5	0.025	0.011	0.013	0.067	0.026	0.76	0.03	0.0066	0.042	0.022
6	0.0088	0.0037	0.012	0.00084	0.0086	0.017	0.94	0.0015	0.0047	0.001
7	0.0051	0.0091	0.018	0.0043	0.018	0.003	0.0011	0.91	0.0032	0.025
8	0.011	0.03	0.023	0.045	0.008	0.041	0.0089	0.0058	0.8	0.026
9	0.0077	0.01	0.011	0.021	0.055	0.02	0.00084	0.055	0.013	0.81

Test Confusion Matrix:



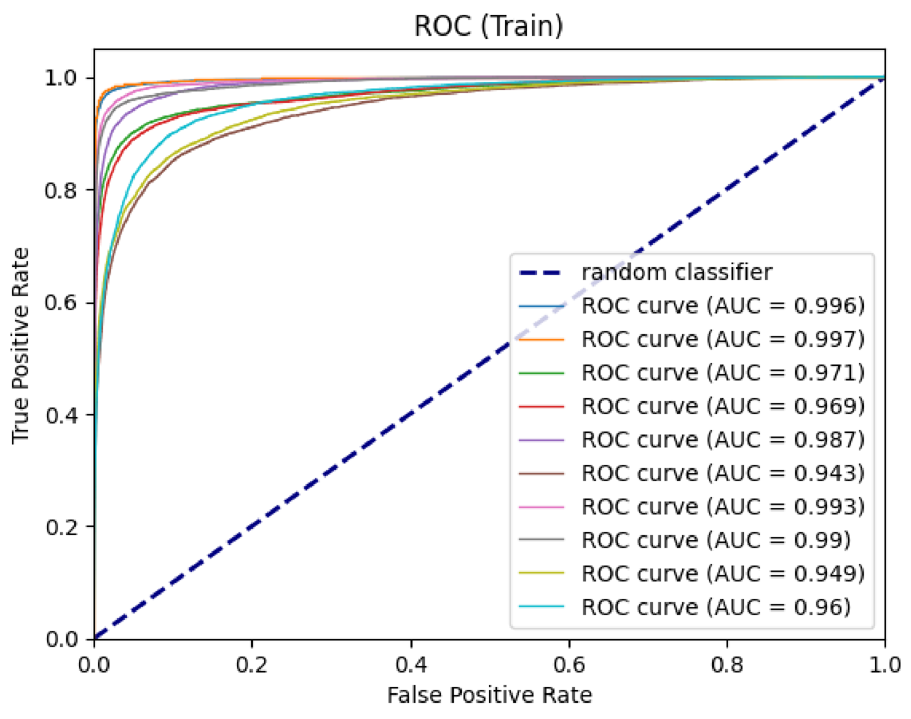
	0	1	2	3	4	5	6	7	8	9
0	959	0	2	2	1	5	8	1	2	0
1	0	1108	3	2	0	3	4	1	14	0
2	11	13	871	21	16	3	21	18	44	14
3	4	2	26	878	1	44	5	14	17	19
4	3	4	8	1	884	4	10	3	9	56
5	22	3	5	66	24	687	24	9	34	18
6	18	3	5	2	16	16	895	0	3	0
7	6	16	30	3	17	2	0	928	3	23
8	15	11	20	37	11	35	14	10	806	15
9	12	11	9	13	62	25	1	42	12	822

Normalized Test Confusion Matrix:

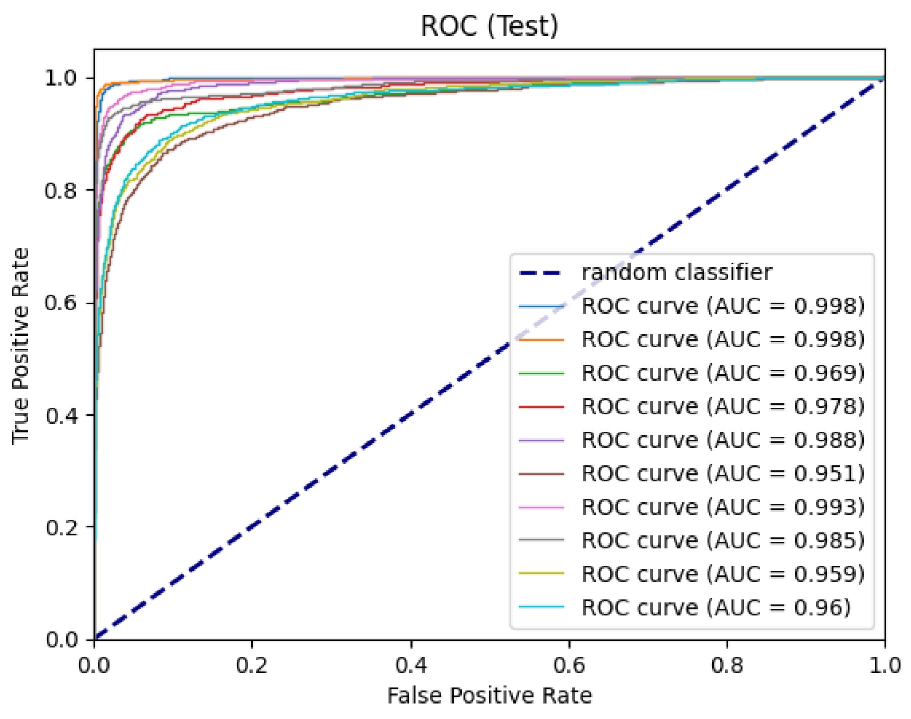


	0	1	2	3	4	5	6	7	8	9
0	0.98	0	0.002	0.002	0.001	0.0051	0.0082	0.001	0.002	0
1	0	0.98	0.0026	0.0018	0	0.0026	0.0035	0.00088	0.012	0
2	0.011	0.013	0.84	0.02	0.016	0.0029	0.02	0.017	0.043	0.014
3	0.004	0.002	0.026	0.87	0.00099	0.044	0.005	0.014	0.017	0.019
4	0.0031	0.0041	0.0081	0.001	0.9	0.0041	0.01	0.0031	0.0092	0.057
5	0.025	0.0034	0.0056	0.074	0.027	0.77	0.027	0.01	0.038	0.02
6	0.019	0.0031	0.0052	0.0021	0.017	0.017	0.93	0	0.0031	0
7	0.0058	0.016	0.029	0.0029	0.017	0.0019	0	0.9	0.0029	0.022
8	0.015	0.011	0.021	0.038	0.011	0.036	0.014	0.01	0.83	0.015
9	0.012	0.011	0.0089	0.013	0.061	0.025	0.00099	0.042	0.012	0.81

Train ROC Plot & AUC Values:



Train ROC Plot & AUC Values:



- ***PCA100 Set Evaluation (4_intro_pca100.ipynb)***

The third test is done over PCA100 dataset. We loaded the preprocessed datasets (Scaled and PCA100) and then using the mask vector, separated the test and train sets. After that, using LinearSVC class of sklearn, we trained a multiclass SVM classifier using One VS Rest technique. To solve the SVM optimization there are two options, first is to solve the dual problem and the second option is to solve the primal problem. We choose to solve the primal because empirically the dual option causes the algorithm to not converge. Also, it is not suggested by sklearn documentation since n_samples > n_features. Then, using the trained SVM classifier, we predicted labels for our train and test samples. Here are the results:

Model: Linear SVM (LinearSVM class from sklearn)

Train Accuracy: 90.84%

Test Accuracy: 91.3% (again, it seems the test set was easier to classify)

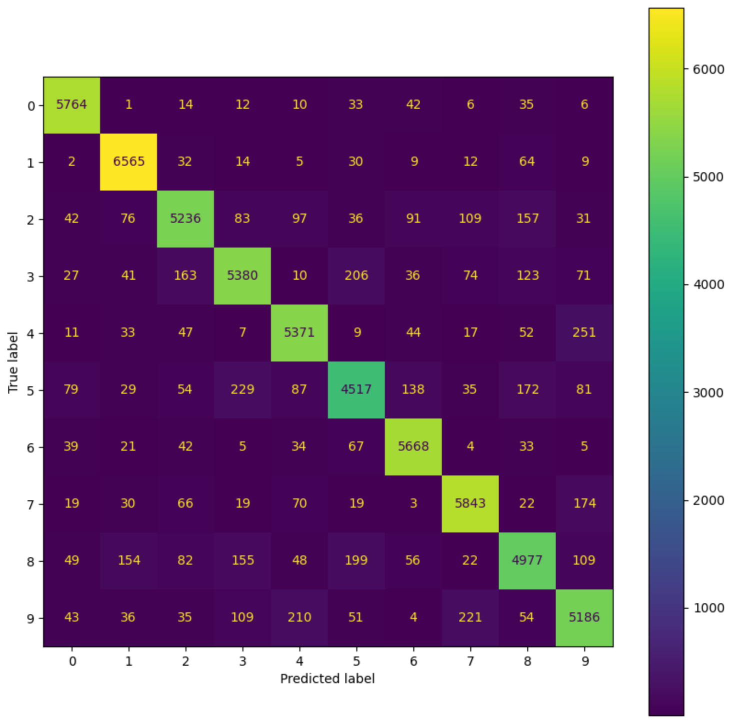
Train Precision and Recall:

CLASSES	0	1	2	3	4	5	6	7	8	9
PRECISION (TRAIN)	0.94	0.93	0.90	0.89	0.90	0.87	0.93	0.92	0.87	0.87
RECALL (TRAIN)	0.97	0.97	0.87	0.87	0.91	0.83	0.95	0.93	0.85	0.87

Test Precision and Recall:

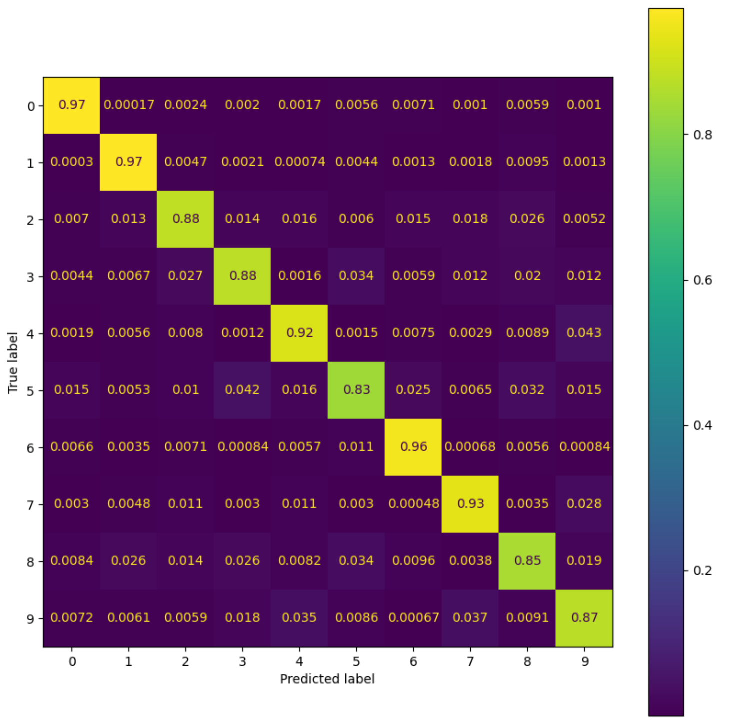
CLASSES	0	1	2	3	4	5	6	7	8	9
PRECISION (TEST)	0.94	0.95	0.92	0.90	0.90	0.87	0.92	0.91	0.87	0.88
RECALL (TEST)	0.97	0.97	0.88	0.90	0.91	0.84	0.95	0.92	0.85	0.87

Train Confusion Matrix:



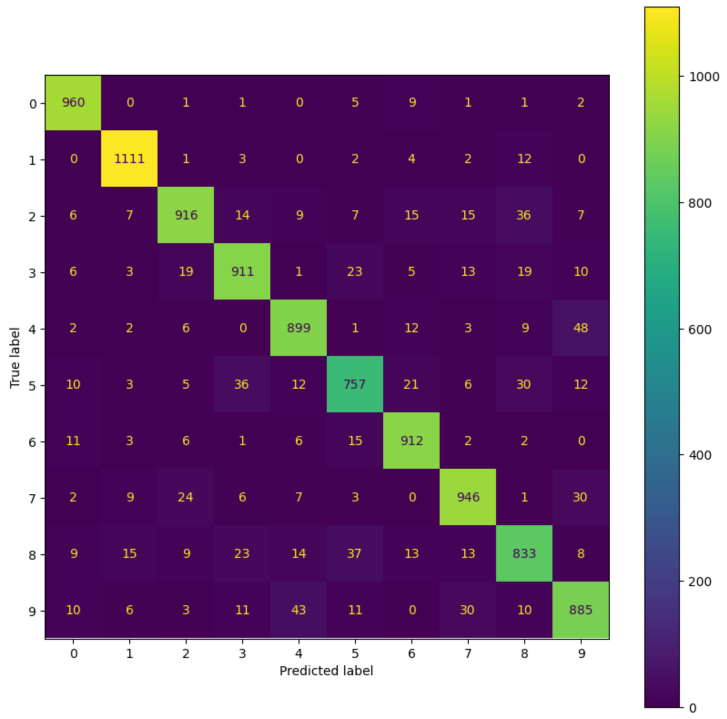
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	5764	1	14	12	10	33	42	6	35	6
1	2	6565	32	14	5	30	9	12	64	9
2	42	76	5236	83	97	36	91	109	157	31
3	27	41	163	5380	10	206	36	74	123	71
4	11	33	47	7	5371	9	44	17	52	251
5	79	29	54	229	87	4517	138	35	172	81
6	39	21	42	5	34	67	5668	4	33	5
7	19	30	66	19	70	19	3	5843	22	174
8	49	154	82	155	48	199	56	22	4977	109
9	43	36	35	109	210	51	4	221	54	5186

Normalized Train Confusion Matrix:



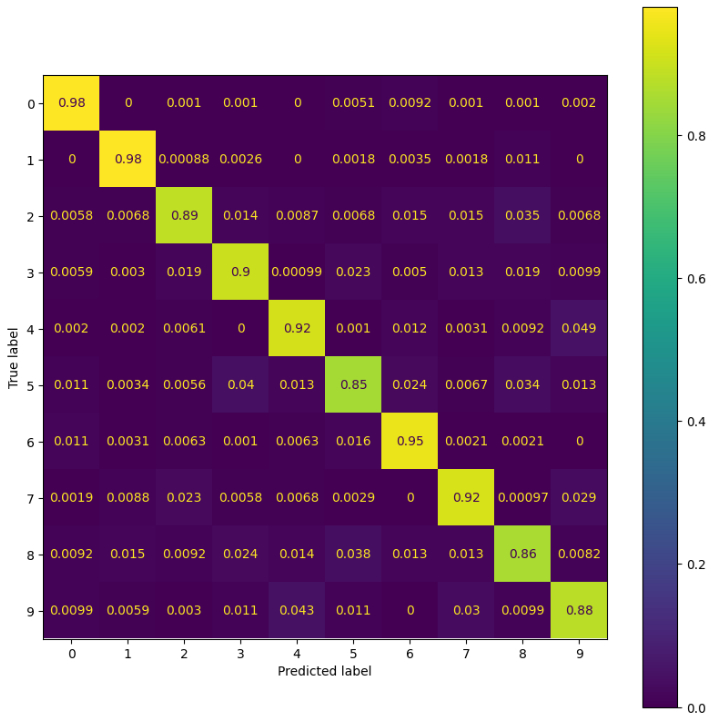
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	0.97	0.00017	0.0024	0.002	0.0017	0.0056	0.0071	0.001	0.0059	0.001
1	0.0003	0.97	0.0047	0.0021	0.00074	0.0044	0.0013	0.0018	0.0095	0.0013
2	0.007	0.013	0.88	0.014	0.016	0.006	0.015	0.018	0.026	0.0052
3	0.0044	0.0067	0.027	0.88	0.0016	0.034	0.0059	0.012	0.02	0.012
4	0.0019	0.0056	0.008	0.0012	0.92	0.0015	0.0075	0.0029	0.0089	0.043
5	0.015	0.0053	0.01	0.042	0.016	0.83	0.025	0.0065	0.032	0.015
6	0.0066	0.0035	0.0071	0.00084	0.0057	0.011	0.96	0.00068	0.0056	0.00084
7	0.003	0.0048	0.011	0.003	0.011	0.003	0.00048	0.93	0.0035	0.028
8	0.0084	0.026	0.014	0.026	0.0082	0.034	0.0096	0.0038	0.85	0.019
9	0.0072	0.0061	0.0059	0.018	0.035	0.0086	0.00067	0.037	0.0091	0.87

Test Confusion Matrix:



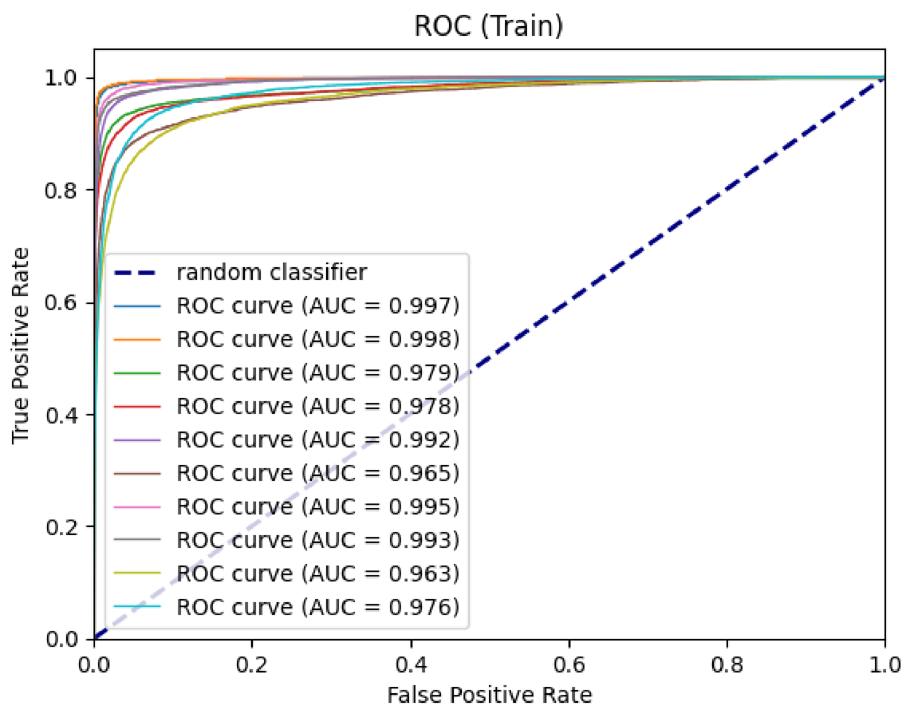
	0	1	2	3	4	5	6	7	8	9
0	960	0	1	1	0	5	9	1	1	2
1	0	1111	1	3	0	2	4	2	12	0
2	6	7	916	14	9	7	15	15	36	7
3	6	3	19	911	1	23	5	13	19	10
4	2	2	6	0	899	1	12	3	9	48
5	10	3	5	36	12	757	21	6	30	12
6	11	3	6	1	6	15	912	2	2	0
7	2	9	24	6	7	3	0	946	1	30
8	9	15	9	23	14	37	13	13	833	8
9	10	6	3	11	43	11	0	30	10	885

Normalized Test Confusion Matrix:

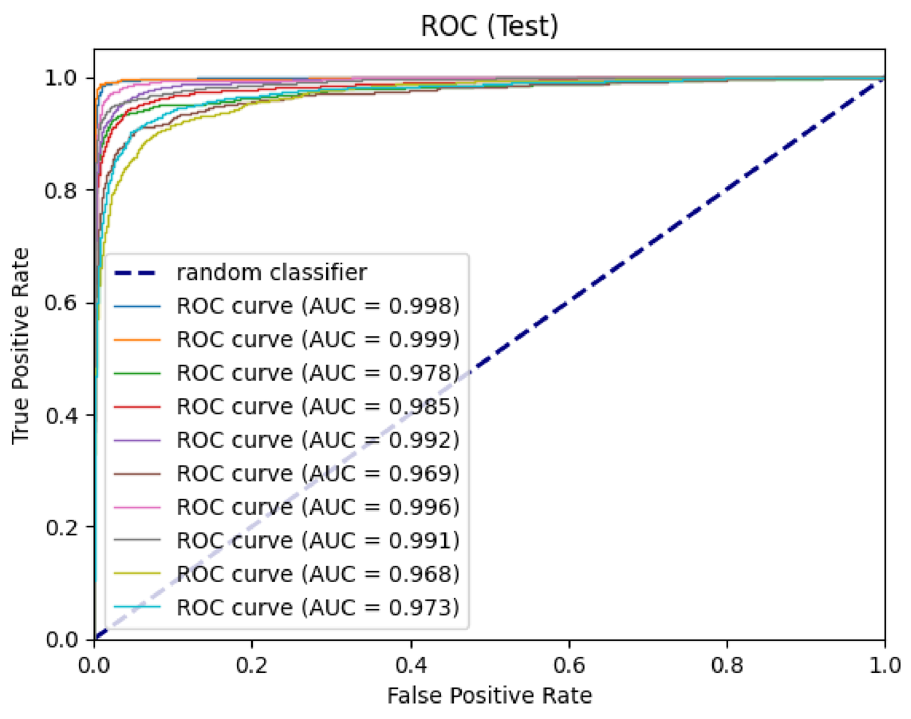


	0	1	2	3	4	5	6	7	8	9
0	0.98	0	0.001	0.001	0	0.0051	0.0092	0.001	0.001	0.002
1	0	0.98	0.00088	0.0026	0	0.0018	0.0035	0.0018	0.011	0
2	0.0058	0.0068	0.89	0.014	0.0087	0.0068	0.015	0.015	0.035	0.0068
3	0.0059	0.003	0.019	0.9	0.00099	0.023	0.005	0.013	0.019	0.0099
4	0.002	0.002	0.0061	0	0.92	0.001	0.012	0.0031	0.0092	0.049
5	0.011	0.0034	0.0056	0.04	0.013	0.85	0.024	0.0067	0.034	0.013
6	0.011	0.0031	0.0063	0.001	0.0063	0.016	0.95	0.0021	0.0021	0
7	0.0019	0.0088	0.023	0.0058	0.0068	0.0029	0	0.92	0.00097	0.029
8	0.0092	0.015	0.0092	0.024	0.014	0.038	0.013	0.013	0.86	0.0082
9	0.0099	0.0059	0.003	0.011	0.043	0.011	0	0.03	0.0099	0.88

Train ROC Plot & AUC Values:



Train ROC Plot & AUC Values:



- ***Analysis and Comparison of Datasets***

The first thing that caught my eyes was that the test accuracy over PCA30 & PCA100 sets are bigger than train accuracy. But this is not the case for Full set.

The next interesting thing is that the classification quality of one and zero is higher than others (Higher precision and recall). Which may happen because of visually distinguishable features of these two digits which make their separation from others easier. Digit 5 has the lowest precision, and from its confusions we can see it mostly mistaken as 8. Which is expected because of these two digits similarities in writing.

We can observe that all the classes got classified with good qualities. But this is no surprise since MNIST is a toy dataset 😌.

- ***Cost Parameter Effect (5_models_pca30.ipynb)***

First thing first, we loaded the preprocessed dataset, then we separated the test and train data.

By defining a list of C parameters, we can iterate over the list and train a different SVM with each of those Cs in every iteration.

Here, we used the LinearSVC class from sklearn library. We trained this SVM model in an OneVsRest mode, which means it's going to train a single SVM for each class. For example, in case of MNIST dataset, it's going to train 10 different SVM models, one for every class. Every SVM model trained in this mode should predict each corresponding class vs every other class. (It should differentiate between that class and others).

There are two ways that we can incorporate when it comes to training our SVM model. Primal and Dual. The documentation of sklearn provided a simple criterion which is going to help to decide which method to use. The criteria are as follows:

If the number of samples are bigger than the number of features, you should use primal method, other wise use dual method.

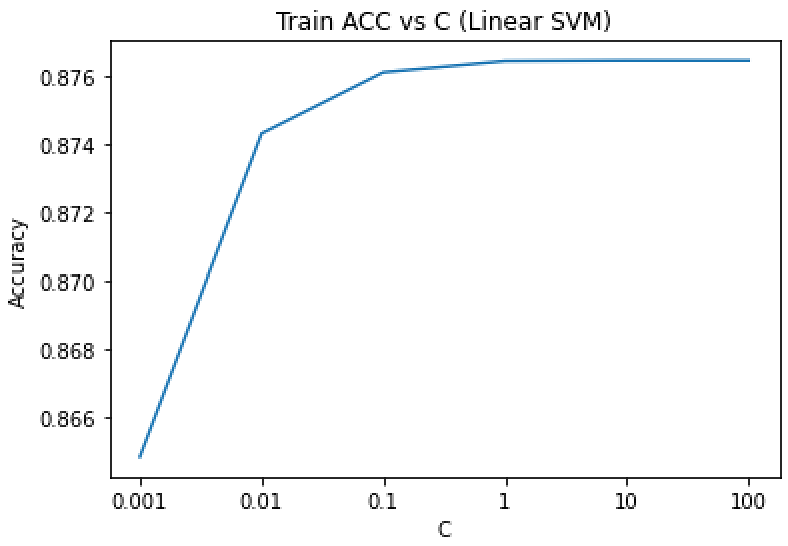
In our case since we know the number of samples are bigger than number of features, we turn dual option off.

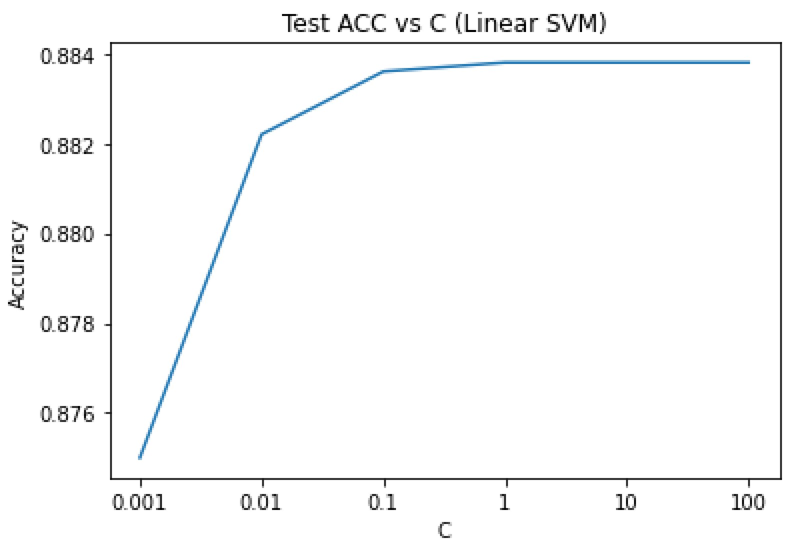
After training each model, the training and test accuracy calculated and stored to be compared with others.

So, here’s the result of this experiment:

	0.001	**0.010**	**0.100**	**1.000**	**10.000**	**100.000**
TRAIN ACC	0.864833	0.874333	0.876133	0.876467	0.876483	0.876483
TEST ACC	0.875	0.8822	0.8836	0.8838	0.8838	0.8838

We can see that by increasing the C parameter, the accuracy of train and test got increased too. To see this effect more clearly, we plotted the train and test accuracy against the C parameter separately:





We can observe from these plots that from one point (C=0.1) the effect of C parameters is negligible, hence it increased the training time a lot.

- ***Gamma Parameter Effect***

Again, for this assignment, we loaded the preprocessed training and test data, then we trained 3 different SVM models with RBF kernels and each time we used a different gamma value.

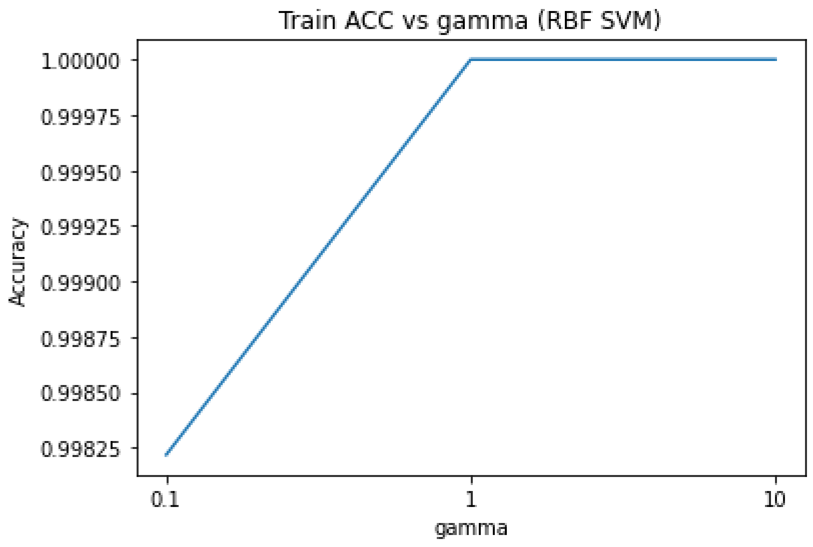
All the models, used the same C=1 value and OneVsRest method for handling the multiclass problem.

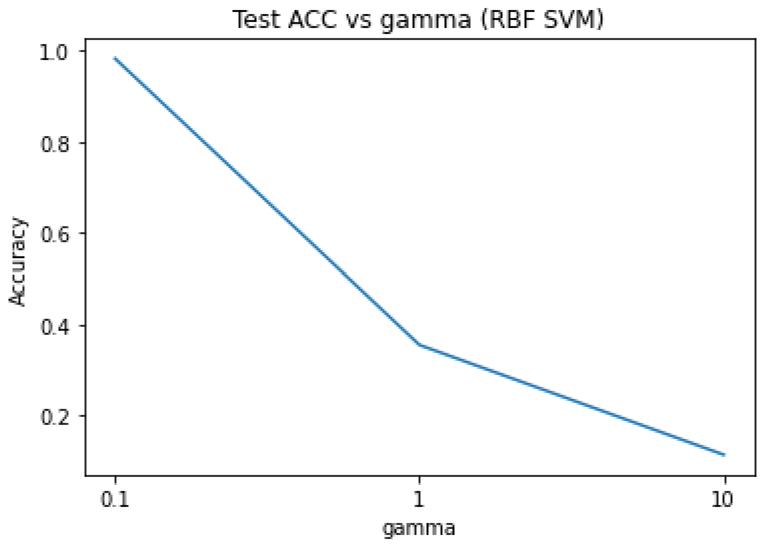
Here's the result of this experiment:

	0.1	**1.0**	**10.0**
TRAIN ACC	0.998217	1.0	1.0
TEST ACC	0.9833	0.3549	0.1135

Here's we can observe a direct relation between gamma parameter and overfitting. Which means by increasing the gamma value, the accuracy over training data is maximized but the test accuracy reduced.

Here's some plot for accuracy against gamma value:





- ***Crammer & Singer method (6_crammer_singer_pca100.ipynb)***

Traditional SVM is one of the most used and most famous among classifiers but the problem with this classifier is it's limitations when it comes to multiclass problems. Traditional SVM can only separate two classes which means it's suitable for binary classification problems.

People usually use some tricks to get SVM works on multiclass problems, for example they are going to train different independent SVM models for each class. The problem with this method is its lack of ability when it comes to capturing inter-class correlations.

To address this issue, crammer and singer proposed a new method of multiclass SVM in which all classes are trained together.

Basically, what Crammer and Singer proposed is to transform the problem to an optimization problem and now the task is to solve this optimization problem.

This is the optimization problem:

$$\begin{array}{ll} \text{maximize} & -(1/2)\mathbf{Tr}(U^TQU) + \mathbf{Tr}(E^TU) \\ \text{subject to} & U \preceq \gamma E \\ & U\mathbf{1}_m = 0 \end{array}$$

There’s an option to use C&S method in LinearSVC of sklearn library.

Here’s the result of the algorithm:

Train Accuracy: 92.56%

Test Accuracy: 92.6%

Head to the notebook to see Confusion Matrix and ROC, AUC.

- ***Crammer and Singer Implementation (7_c&s_svm_final.ipynb)***

For this part of the assignment, we implemented and corrected an implementation of Crammer and Singer method for training Multiclass SVM.

Test Accuracy: 92.642