ELL409 Assignment 1

Harman Singh 2018EE10542 Ayush Srivastava 2018MT10747

***Abstract*—Solutions for Assignment 3 of ELL409.**

I. Q1 SVM on health data

We first ran PCA on the health data which has 3 features and ran SVM to classify into the 2 classes.Then to make the decision boundaries, we changed the problem as suggested on piazza and took the first 2 principal components of the data to. We fit SVM's with multiple kernels and plot the decision boundaries. We also implement the SMO algorithm and compare its performance, training time with libSVM.

We perform our analysis on 2D PCA applied on health data for the purpose of visualization of decision boundaries.

A. *Running SVM with different Kernels*

Different Kernels as mentioned int he Question statement, were used and the optimal values of C were found using cross validation on the training set. For the RBF kernel, optimal values of C and gamma were found using grid search. The optimal hyperparameter values for our randomized dataset are as mentioned in Table XII

Performance metrics obtained on random 70-30 split of the dataset, using optimal hyperparameters is as tabulated in Table I. Linear, RBF nad polynomial of degree 3 are able to traina nd generalize well unlike polynomial of degree 2 and 4 and we believe this is an artifact of the data which is better generalized using Linear or degree 3 kernels.

Kernel	Train Accuracy	Test Accuracy
Linear	0.8630	**0.8436**
RBF	0.8712	**0.8246**
Poly-Deg2	0.6503	0.6872
Poly-Deg3	0.8487	**0.8483**
Poly-Deg4	0.6789	0.6967

TABLE I
Performance metrics, SVM on health data

B. *Decision Boundaries*

Decision boundaries for all kernels (Linear, RBF, polynomia with different degrees) is shown in Figure 1. These figures correspond to the optimal values of 'C' and gamma (in case of RBF) kernel. The yellow dots signify that they are support vectors.

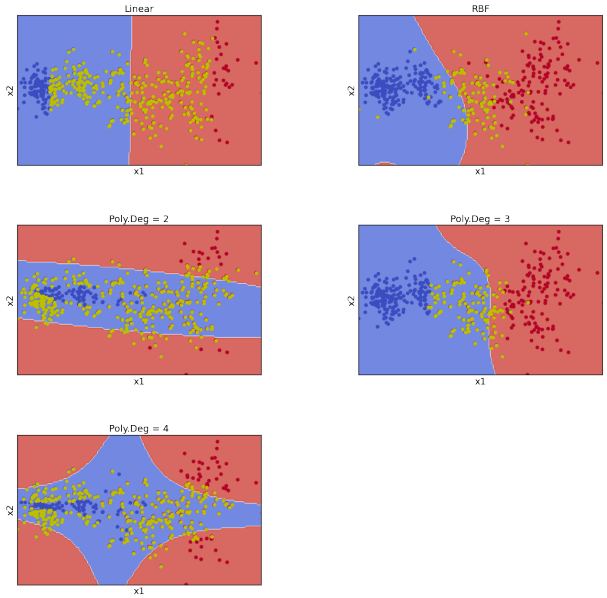

Fig. 1. Decision boundary for various kernels

C. *Decision Boundaries with removed support vectors*

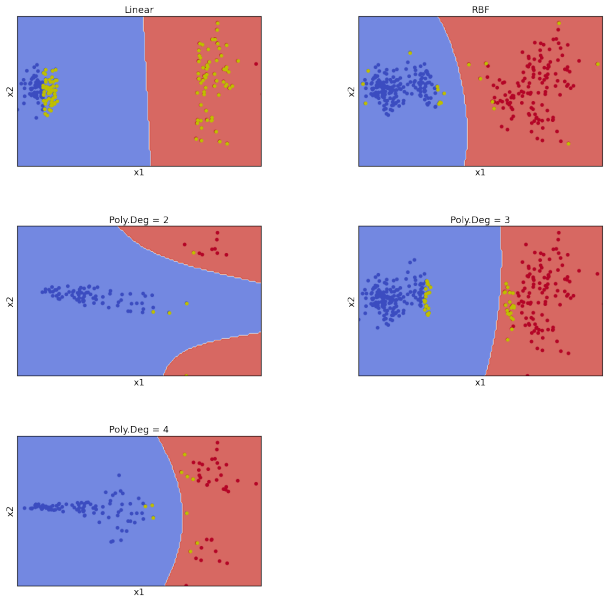

Fig. 2. Decision boundary for various kernels when old support vectors are removed

We removed the old support vectors and then ran the SVM again to fit it to the remaining dataset. Our decision

boundaries are as shown in Figure 2. The yellow marked points are the new support vectors. As we can see these are much smaller in number.

a) Observations:

- Data has become linearly separable when the support vectors are removed
- New support vectors are much smaller in number, this is expected because most of the support vectors earlier may have lied inside the margin which are now removed and the data has become linearly separable so support vectors are only which lie on the margin
- There is no error ($\sum qi$) now in the formulation of the svm, and the svm is simply the maximization of the margin term.
- Also these decision boundaries may not be optimal as they are based on the old dataset where the support vectors are not removed

D. Implementing SMO algorithm (bonus part)

We referred to Platt's original paper, P.S.Sastri's tutorial and CS231 notes and implemented the SMO algorithm in python itself. The performance of our code was comparable to the sklearn's LIBSVM implementation in terms of accuracy but time taken by our code was larger.

- The time taken by our code was more than that of the LIBSVM as shown in Figure 3 a) and it continues to increase as the data scales.
- The accuracy of LIBSVM and our implementation of SMO isplotted and we can see that for a large enough dataset they are nearly equal to each other (Figure 3 b))

We feel that ours is not the most optimal implementation of the algorithm and many more techniques can be used for eg in choosing the 2 vectors at each step and in other parts of the code. We also feel that LIBSVM may have been implemented in C/C++ which makes it much faster than some parts of our code in Python (like loops etc) which cannot be vectorized using numpy arrays and hence have to suffer the slow-ness of python.

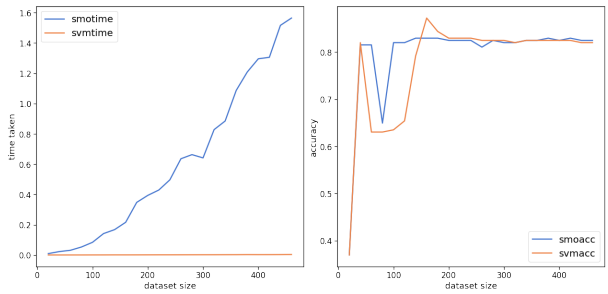

Fig. 3. a)Time taken SMO implementation (smotime) vs Sklearn's LIBSVM (svmtime) and b) accuracy of SMO implemenation (smoacc) and Sklearn's LIBSVM (svmacc)

II. Neural Networks

A. Fully Connected Feed Forward Network

We overfit the MNIST dataset using out self implemented fully connected neural network, the training graph depicting accuracies is as shown in Figure 4

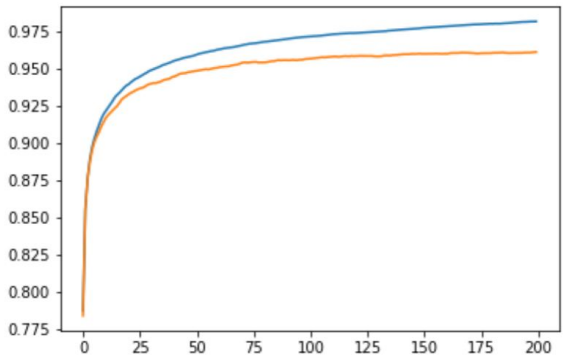

Fig. 4. Accuracies of Train(Blue) and Validation(Orange) data while training

B. Fully Connected Feed Forward Network

Elastic Net, Dropout, Batchnorm regularzations were implemented. The results are tabulated in Table II.Early stopping was not done as the network is not overfitting on MNIST.

Class	Train Accuracy	Test Accuracy
Elastic Net	0.96	0.95
Early Stopping	0.97	0.96
Batchnorm	0.94	0.92
Dropout	0.99	0.97

TABLE II
Regularization stats

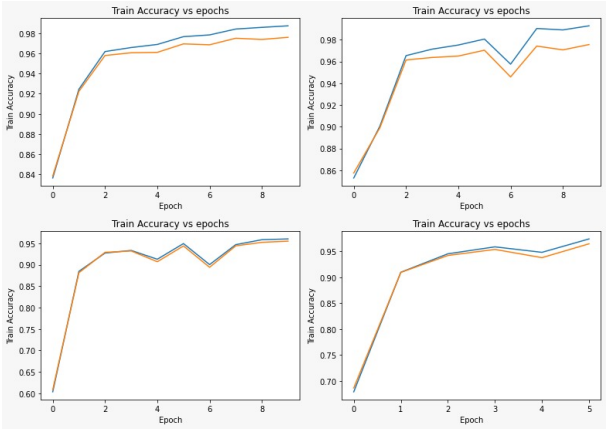

Fig. 5. In cyclic order,graphs for 1.Without regularization, 2.Drpopout, 3.Elastic Net, 4Early Stopping, Train Acc(Blue) Test Acc(Orange)

C. Training Standard Architectures

LeNet, AlexNet were trained on tinyImagenet. We implemented VggNet but were only able to train it for 7 epochs.

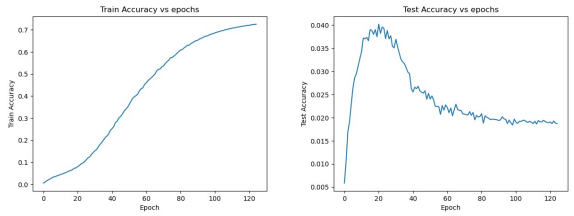

Fig. 6. Train (left) and Test(right) accuracy for LeNet during training phase

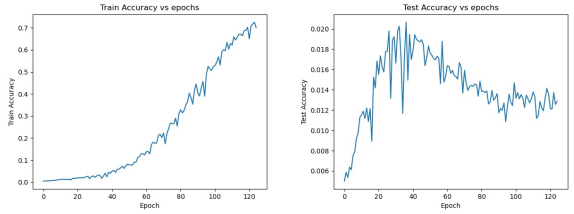

Fig. 7. Train (left) and Test(right) accuracy for AlexNet during training phase

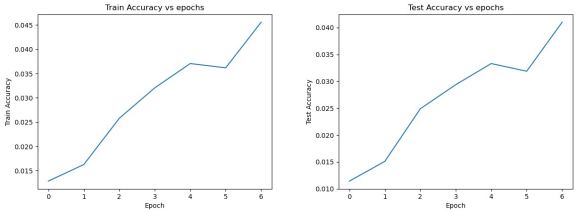

Fig. 8. Train (left) and Test(right) accuracy for VggNet during training phase

D. Comparison between optimizers

We trained a network using SGD, ADAM and sSGD with momentum architectures. The training loss and accuracies are shown in Figure 3

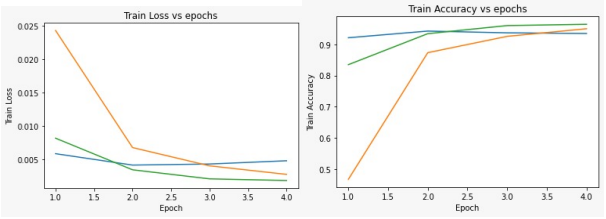

Fig. 9. Train accuracy vs epochs(Left) and Train Loss vs epochs (Right) for SGD(yellow), ADAM(blue) and SGD with momentum(green)

a) Observations:

- ADAM is faster but its convergence is bad as compared to the other 2.
- SGD with momentum is faster than SGD, slower than ADAM but has the best convergence

III. Learning++

Part A to E correspond to the 5 subquestions of Q3

A. Class Imbalance in MNIST

MNIST class 0, 1, 2 were considered in the ratio 70:25:5. We first used a comparitively larger self made (non standard) neural network which gave close to 100%(>99.8 on both train and test datasets) accuracy. This neural network was large for the MNIST dataset and was able to generalize well despite the class imbalance.

We then used a **smaller feedforward neural network with only 2 hidden layers** to first visualize the problems that arose due to class imbalance and then try to solve them. We first tried out the following methods and modifications to our losses.

- Cross entropy loss: We did not modify the loss in this case and the results we obtained are shown in the folloing table having precision recall and F1 score for all three classes. (Table III)

Class	Precision	Recall	F1
0	0.955	0.998	0.976
1	0.955	0.995	0.975
2	**0.995**	**0.908**	**0.949**
Macro F1			**0.967**

TABLE III
Simple Cross entropy loss

Missclassified samples are as follows, these are randomly picked misclassified samples and as we can see most of them are from the class 2 which has the least samples (Figure 10)

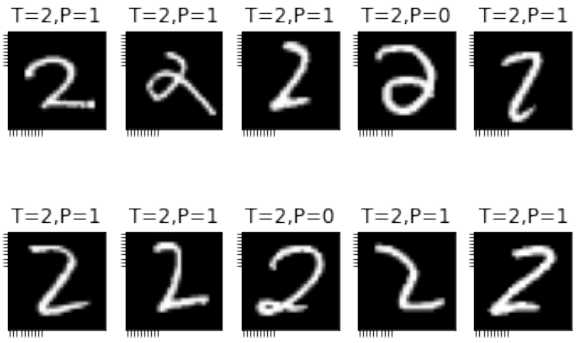

Fig. 10. Missclassified samples, random. T=True, P =Predicted

- Weighted Cross entropy loss: We gave more wight to the class with fewer sample. The weights are inversely proportional to the ratio of classes. We get an increase

in the performance metrics, particularly the F1 score of the class 1 and 2 which have less samples(Table IV)

Class	Precision	Recall	F1
0	0.976	0.996	0.985
1	0.977	0.997	0.986
2	**0.992**	**0.951**	**0.972**
Macro F1			**0.981**

TABLE IV
WEIGHTED CROSS ENTROPY LOSS STATS

- Focal loss: We experiemnted with the focal loss and obtained an increase in performance metrics, when compared with the vanilla cross entropy loss.(Table V)

Class	Precision	Recall	F1
0	0.977	0.995	0.987
1	0.981	0.997	0.988
2	**0.993**	**0.965**	**0.980**
Macro F1			**0.985**

TABLE V
FOCAL LOSS STATS

- MSE loss: Simple MSE loss was tried out, performance is much worse than cross entropy loss.(Table VI)

Class	Precision	Recall	F1
0	0.863	0.997	0.925
1	0.853	0985	0.914
2	**0.996**	**0.685**	**0.811**
Macro F1			**0.8894**

TABLE VI
SIMPLE MSE LOSS STATS

- Weighted MSE: Similar to Weghted Crossentropy but with this time using MSE.(Table VII)

Class	Precision	Recall	F1
0	0.881	0.998	0.936
1	0.862	0.988	0.926
2	**0.994**	**0.718**	**0.833**
Macro F1			**0.896**

TABLE VII
WEIGHTED MSE LOSS STATS

- Focal loss type weighted MSE: We took the ideas of focal loss and used it in MSE to increase the performance of the classification.(Table VIII)

Class	Precision	Recall	F1
0	0.872	0.998	0.930
1	0.859	0.986	0.918
2	**9.993**	**0.703**	**0.823**
Macro F1			**0.890**

TABLE VIII
FOCAL LOSS MSE STATS

B. *KMEANS on SVHN + classification*

KMeans algorithm is computationslly intractable for large datasets like SVHN having hundereds of dimensions, hence we applied KMeans Custering on 10K datasets taken from SVHN. K = 3, 5 ,10, 15 ,20 were used and the clusters were made, and new labels were found and supervised learning was performed using SVM's as classifier.Visulaization of Kmeans clustering for K=10, K=5 is as shown in Figure IX

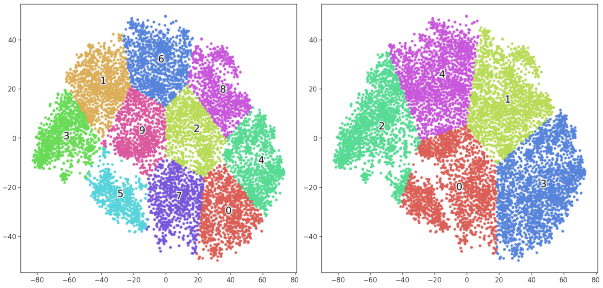

Fig. 11. Kmeans clustering for K =10, 5

We use SVM's for our multi-class classification problem with the "one vs one" approach for classification. We use 8k training points and 2k test points for our classification. The results obtained are tabulated in Table IX

K value	SVM Train Acc	SVM Test Acc
3	0.9959	0.9975
5	0.9918	0.9910
10	**0.9854**	**0.9980**
15	0.9780	0.9811
20	0.9801	0.9750

TABLE IX
SUPERVISED LEARNING RESULTS ON K-MEANS CLUSTERS DATA

1) *Observations:*

- We see that the best test accuracy while using SVM's are obtained for K = 10 clusters. This suggestes that the data can be divided using 10 distributions(gaussian with infinitesimal variance) efficiently and this is in coherence with the fact that there are 10 classes in the svhn dataset.
- High accuracies from SVM classification suggest that tSNE outputs have useful information, and Kmeans followed by classification using SVM is an efficient way to utilize the information for classification.

C. *Domain Adaptation*

1) Target Domain Labels not used: We trained a custom built CNN in pytorch with 2 convolutional layers and 2 hidden layers (details in appendix). We took the input as a 3 channel image so and transformed the MNIST dataset into 3 channels and resized it to 32*32 so that a classifier trained on it can be tested on SVHN. Using this setting we made a classifier using MNIST and trained it on SVHN amd vice versa. The reslts are tabulated in X. We use

Adam optimizer with learning rate 0.001, and we train for 5 epochs everywhere. Loss and Accuracy plots while training are present in Figure 12

Classifier for	MNIST Train	MNIST Test	SVHN Train	SVHN Test
MNIST	99.3	99.25	-	**22.461**
SVHN	-	**57.21**	88.5	88.07

TABLE X
ACCURACIES FOR SUPERVISED LEARNING ON K-MEANS CLUSTERS DATA

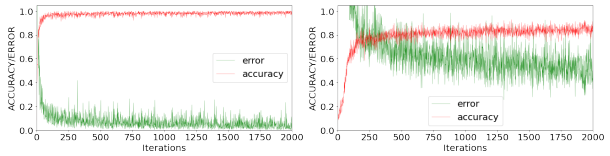

Fig. 12. a)Error and Accuracy for training MNIST classifier and b)Error and Accuracy for training SVHN classifier

Observations: We can cnclude that the classifier trained on SVHN performs decently well on MNIST even when its not trained on it but on the other hand, classifier trained on MNIST doesnot generalize well on SVHN, with only 22 percent accuracy, but nevertheless its double the accuracy we get by random guessing. This is expected because SVHN dataset is more diverse and the network needs to learn much more complex distributions due to varying color if the images, overlapping digits and hence its able to generalize well on MNIST as its a simpler dataset. But the MNIST trained network doesnot have to learn as complex distributions and hence the result is that when its trained on a more complex dataset it performs poorly.

2) Using Target domain labels: We are given samples from source domain $\{Xs, Ys\}$ and samples from another target domain $\{Xt\}$. Using this setting we need to make a classifier that generalizes well on the target domain. We surveyed ,ultiple techniques to do this, including using using CYCLE GAN's to learn a transformation from the sorce to the target domain using generative adversarial networks, and then using the transformed dataset of the source, building a classifier which generalizes well on the target. We also looked at the paper "Maximum Classifier Discrepancy for Unsupervised Domain Adaptation" and tried to implement it.

In this paper we have a Generator G that generates an embedding and 2 Predictors F1 and F2 that classify the 10 digits.There are 3 losss terms 2 of which force the model F1 and F2 to predict classes of source domain correctly while the 3rd loss term maximizes the discrepancy between predictions made by F1 and F2 for the target domain images.We cosulted the repo of the authors as well as kaggle on how to build and train the model.We achieved good results for domain adaptation of SVHN onto MNIST but did not get good results vice versa. We still have to figure out why this is the case. The results of training are shown in Figure 13

We get about **80% Accuracy** when we try to adapt to MNIST domain using SVHN and about **20% Accuracy** when we adapt to SVHN domain using MNIST.

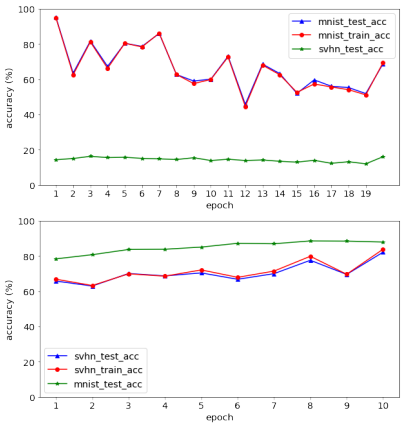

Fig. 13. (Left)MNIST model adaptation to SVHN and (Right)SVHN model adaptation to MNIST

D. Zero-Shot-Learning

Classifier traind on class 0-4 of MNIST was tested on classes 5-10. An accuracy of **23.14% was achieved** which is very close to the random guess accuracy of $100/5 = 20\%$.

a) Approach to increase accuracy:

- Using PCA visualization facts : We see from PCA (Figure 15 (left)), and from common knowledge about digits that some classes are close to each other because they share same features (for eg igits 7,1 look similar, 9,4 look similar). We first choose the most probable class out of the first 5 classes. Then our network classifies the image as 7,9, 8, 6, 5 if the highest activations are for classes 1, 4, 3, 0, 2 respectively and we achieve a **11% accuracy gain** by employing this method. Finally our accuracy was increased to **34%** from the earlier 23%

Method	**Accuracy (Novel Classes)**
Naive	23.14%
PCA-Technique	**34.41%**

TABLE XI
ZERO SHOT LEARNING ON MNIST

E. 2D PCA, tSNE on SVHN dataset

- The result of performing PCA on SVHN is as shown in Figure 14 (left).
- The result of performing tSNE on SVHN is as shown in Figure 14 (right). tSNE was performed on a subset (10k images) of data which took considerable hours on HPC.

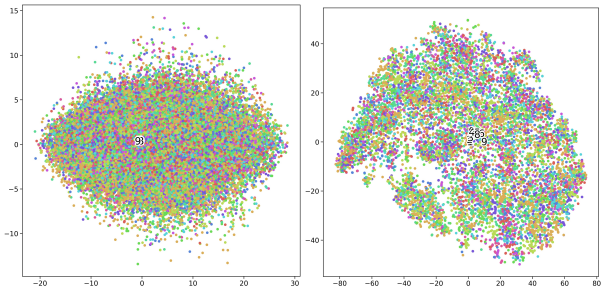

Fig. 14. SVHN 2D PCA(left) and SVHN 2D tSN(right)

- To have a comparison, we went ahead and performed PCA, tSNE on MNIST data as well (Figure 15) and based on all this we present our observations and colnclusions below.

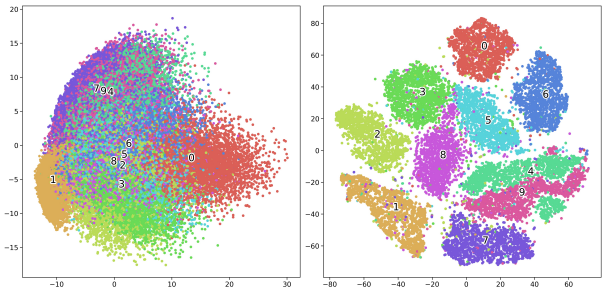

Fig. 15. a)2D PCA and b)2D tSNE MNIST

1) Observations and Conclusions:

- The PCA result on SVHN is very homogeneous, and the classes are not separated at all. We can see that all the class labels are placed at the same point in the figure, ie teh centroid of all the classes is the same and hence wPCA hasn't been able to separate the classes very well in 2 dimesnions.
- tSNE on the other hand isn't alot better but we can ee more clustering than PCA and the centroids of the classes are also displaced from each other if we look closely in the tSNE plot. tSNE being a non linear technique, it is expected that it would perform better than PCA, however in a lot more amount of time due to (O(N^2) complexity)
- Comparing the results with MNIST we get class separationg using PCA on MNIST. Classes 7, 9, 4 and classes 6 ,5, 8, 2 are close to each other which is expected due to the struture of the digits. tSNE is able to completely separate all classes from each other as shown in the tSNE plots. An interesting thing to note is that the digit 4 is separated into 2 groups and one is closer to 9, this is expected because some digits, 4 look very cose to 9.
- We can conclude that due to the noisse, background, overlapping digits and classes, and being in more natural setting, SVHN is difficult to separate using PCA,tSNE but MNIST is highly separable due to the nature of the dataset, having contrast between digits and background, uniform background, no overlapping of labels, classes etc

Appendix

A. Q1 SVM Health Data

- **Optimla Hyperparameter values (see Figure XII)**

Kernel	Optimal C	Optimal gamma
Linear	0.5	-
RBF	18	149
Poly-Deg2,3,4	2, 7, 7	-,-,-

TABLE XII
Optimal Hyperparameters for SVM

- **PCA visualization (see Figure 16)**

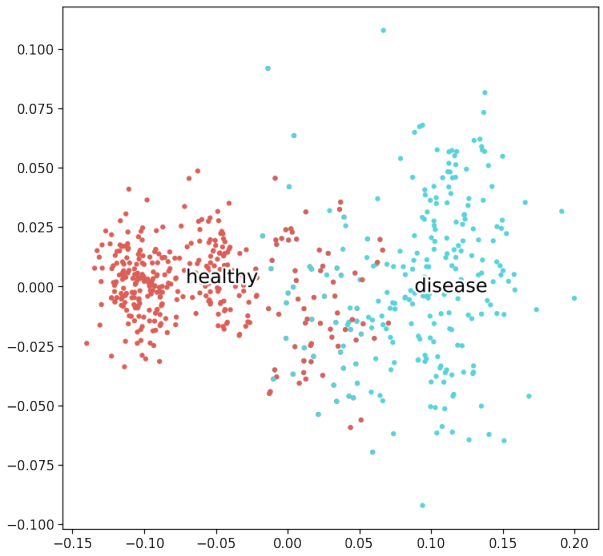

Fig. 16. SVHN 2D PCA

- **Gridsearch for C, gamma**

We took multiple values of C and plot multiple line plots, each having one value of C and multiple values of gamma. Optimal C and gamma were chose to be those values for which the cross validation accuracy was the highest. (Figure 17)

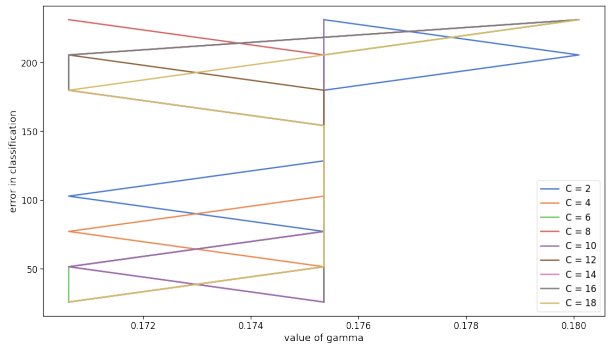

Fig. 17. grid searh for C and gamma

- **Search for C**

searching for SVM hyperparamter C for linear and polynomial kernels was done by plotting accuracies for multiple values of C (Figure 18)

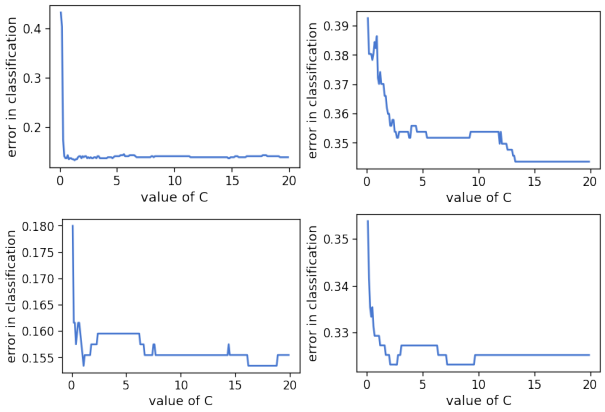

Fig. 18. SVHN 2D PCA

B. Q3.1 Class Imbalanced MNIST confusion Matrices

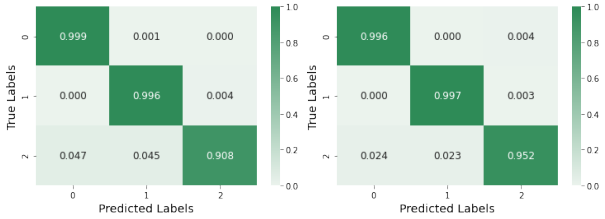

Fig. 19. CrossEntropy Loss (Left)Without weighting and (Right) with proportional weighting of loss

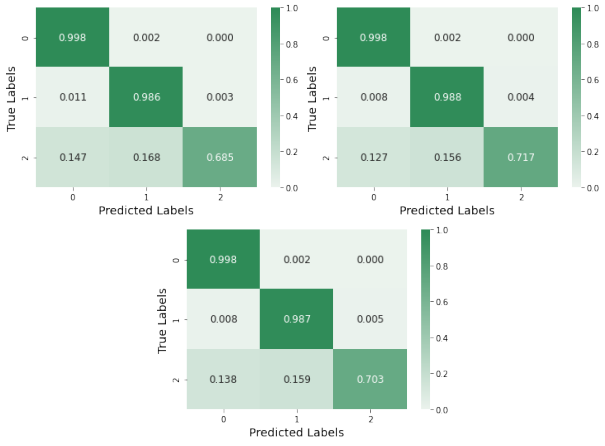

Fig. 20. MSE Loss (Left)Without weighting and (Right)With proportional weighting of loss and (Bottom)MSE with focal loss type weighting

C. Q3.4 Zero Shot Learning

- **Classification of some digits using PCA method (Accuracy 34 %) (Figure 21)**

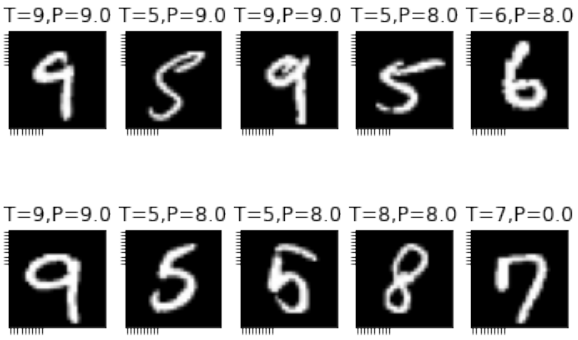

Fig. 21. classification using PCA method

- **Classification of some digits using PCA method (Accuracy 23 %) (Figure 22)**

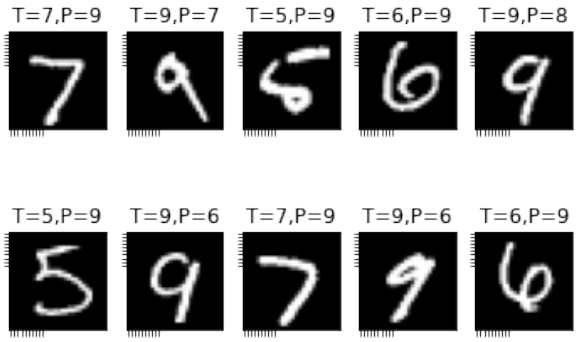

Fig. 22. classification using PCA method