ELL409 Assignment 2

Shubham Mittal
Electrical Engineering
IIT Delhi
ee1180957@iitd.ac.in

I. Introduction

In this report the transition from kernel methods (SVMs) to modern deep learning methods is shown. Even though the work is done by the author alone, the report is written as 'we'.

II. SVM and Kernel Machines

The discussion in this section is in the order of the problem statement - first the best kernel is found on the binary health dataset (data dependent kernel), second the decision boundary is visualized, third the relevance of support vectors in the dataset is empirically shown, and lastly the time complexity of SMO implemented from scratch and LibSVM (sklearn) is compared. The health dataset visualization in 3D is shown in Appendix (Fig 19). Note that since the dataset is randomly shuffled, the results may not be consistent every-time with code execution.

A. Kernel Comparison

For the linear kernel SVM, the best penalty constant is obtained as shown in Figure 1 (top). It shows the number of support vectors, train and test accuracy with C. Hence, we conclude that $C = 1$ gives the best generalization over the dataset. Figure 1 (bottom) shows the effect of polynomial degree on the SVM's performance and the number of support vectors. Hence, it is evident that for degree greater than 1 there is no significant improvement in the performance. It is interesting to note that the number of support vectors don't monotonically increase with the degree of polynomial kernel.

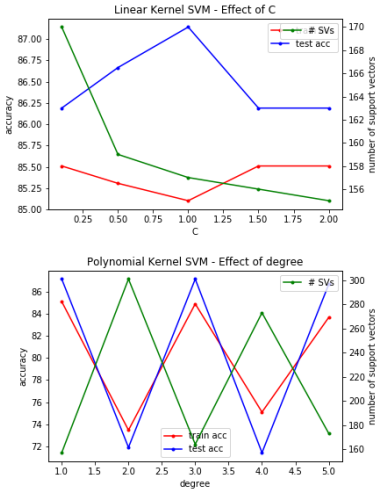


Fig. 1. Linear and Polynomial Kernel SVM

Further, we perform a grid search over C and γ in the RBF kernel SVM. From Table I, we conclude that $C = 2$ and $\gamma = 0.01$ gives the best performance in RBF SVM. The train accuracies in the grid search is given Appendix (Table V)

	0.01	**0.1**	**0.5**	**1.0**	**2.0**
0.01	51.43	87.14	84.29	51.43	51.43
0.1	86.19	87.14	87.14	87.62	86.67
0.5	87.62	86.19	84.76	85.24	84.76
1.0	87.14	85.71	85.24	85.24	84.76
2.0	87.14	85.24	85.71	85.71	82.38

TABLE I
Test Accuracies in Grid Search for RBF Kernel SVM

The comparison between linear kernel SVM and RBF kernel SVM is summarized in Table II. The polynomial kernel results were shown in Figure 1 (bottom) suggesting that linear SVM is better. Since both the linear and RBF kernel shows same test performance, we visualize the decision boundary for the latter to investigate the degree of non-linearity in the decision boundary. Also, since the random shuffling of data changes the results, the RBF kernel was showing slightly better performance than linear kernel when the experiments were run multiple times, hence the RBF SVM decision boundary is visualized.

	Train Acc	**Test Acc**	**F1 score**	**# of SVs**
Linear	85.1	87.14	0.87	157.0
RBF	85.1	87.14	0.86	185.0

TABLE II
Comparison of various kernels in SVM

B. Decision Boundary of Kernel SVMs

In order to visualize the decision boundary, we use PCA to reduce to 2 dimensions. Figure 2 shows that the first two components contains 94.7% variance.

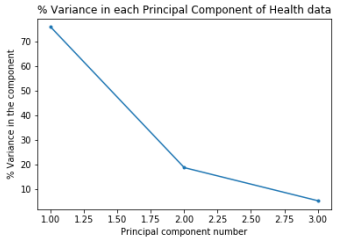


Fig. 2. PCA on Health Dataset

The RBF kernel SVM is trained on this two dimensional dataset. Figure 3 (left) shows the decision boundary obtained using train set of size 490. The support vectors are marked with green cross. The decision boundary is more or less linear as we would expect from Table II. The decision boundary obtained after removing 185 support vectors is shown in Figure 3 (right). It allows us to make following inferences:

1) The decision boundary before and after removing support vectors is approximately same. This suggests that most of the original SVs in the train set were the points causing margin error, and even if we remove them, the optimal margin remains approximately same.
2) After removing SVs, most of the new SVs lie on the supporting hyperplanes ($\lambda \in (0, C)$) and very few cause margin error ($\lambda = C$).
3) The test accuracy remains same which also supports the fact that w^* remained almost same.

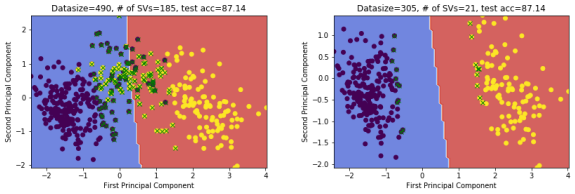


Fig. 3. RBF Kernel SVM: left :- with SVs; right :- without SVs

C. Sequential Minimal Optimization

Figure 4 shows the comparison of time between SMO (implemented from scratch [3]) and the Sklearn LibSVM for a soft margin linear kernel SVM. This shows that the sklearn LibSVM is highly optimized and efficient in terms of its computation in the optimization algorithm even though both use the same optimization algorithm, SMO.

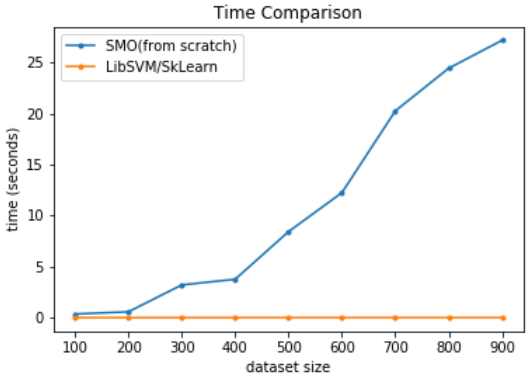


Fig. 4. SMO (from scratch) and LibSVM (sklearn) comparison

D. Support vectors and dataset size

We conclude the discussion on kernel methods by showing the relationship between the number of support vectors and the dataset size. Figure 5 shows that as the dataset size increases, the number of support vectors also increases.

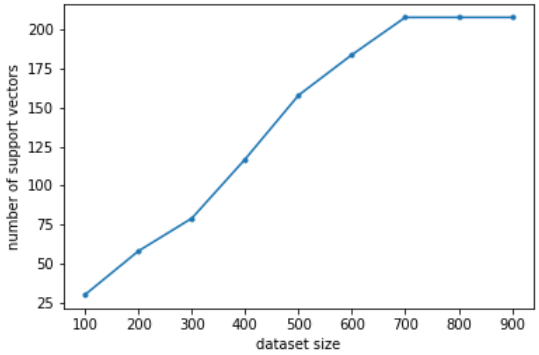


Fig. 5. Number of Support Vectors with Dataset Size

III. Neural Networks

In this section we first present the results obtained on MNIST dataset using a multi layer perceptron (hard coded from scratch in python). The regularizers given in problem statement are also implemented from scratch. We conclude this section with the experiments using CNNs implemented in pytorch.

A. Feed-forward Fully Connected Neural Network

Experimental details- A three layer neural network with 512 and 128 neurons in the hidden layer is used. The MNIST dataset comprising of 784 dimensional $60k$: $10k$ train and test images. $tanh()$ activation function is used and SGD with batch size of 128 on cross entropy loss.

Figure 6 shows the neural network trained on MNIST without any regularization. It is visible that the test curve has plateaued but the model is continuing to memorize or learn from the train set.

Upon increasing the depth or width of the neural network, there was no sign of overfitting (when the test loss starts to rise up). Thus we can say that the MNIST classification problem is not challenging in the sense that test and train set are not very different and even if the model memorizes the train set, it will still do good (like more than 97% accuracy) on the test set.

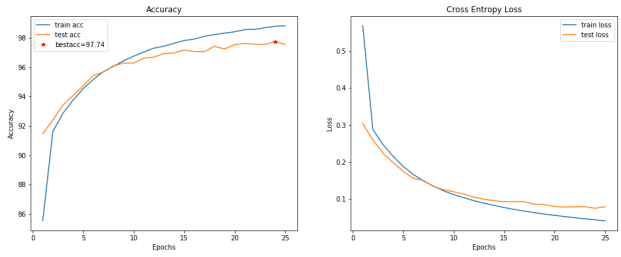


Fig. 6. Fully Connected Neural Network on MNIST

Points of learning from neural network training:

1) The hyper-parameters like learning rate, batch size, etc play a critical role in the training time and model convergence.
2) The random shuffle of the training set during SGD showed improvement in the accuracy. This possibly stems from the fact that if the mini-batches have only one/two class data-points then the model parameters go in one particular direction because of the gradient computed from the mini-batch.

The effect of learning rate is shown in Figure 7. Here L2 regularization is also used with $\alpha = 5 \times 10^{-4}$. Thus, learning rate of 0.1 shows the best model convergence. It was empirically observed that using L1 regularizer also gives same test accuracy like L2.

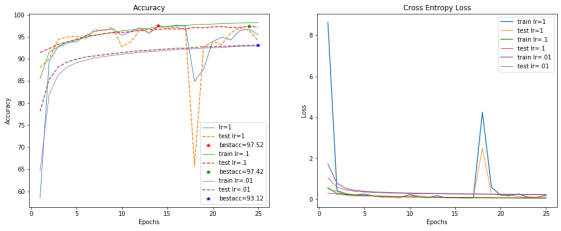


Fig. 7. Effect of learning rate in NN training

Figure 8 shows the effect of dropout, batch-norm and dropout-batchnorm together applied on the neural network. Since the model could not overfit or give the point where the test loss rises up, early stopping is not used. However its implementation is pretty straightforward and is done in the CNNs (next subsection). Here we make following inferences regarding dropout and batch normalization

1) Using a dropout layer, with $p = 0.5$, after every layer (except output layer) in the NN shows that it reduces the training time and improves the generalization capability. It is evident that the test accuracy is higher than the train accuracy by approx. 5%.
2) Using batch normalization after the hidden layers (before activation function) shows that its performance is almost same as without any regularization.
3) Using both of them together shows that the accuracy increased relative to dropout alone. We can say that adding batch norm layer improved the performance.
4) We can say that, here, the dropout layer is worsening off the model but it also reduces the training time. Since we are using small dataset and small model, we restrict our conclusions to
    a) dropout reduces the training time and improves generalization error
    b) batch-norm helps in avoiding exploding/vanishing gradient problem and also improves the performance.

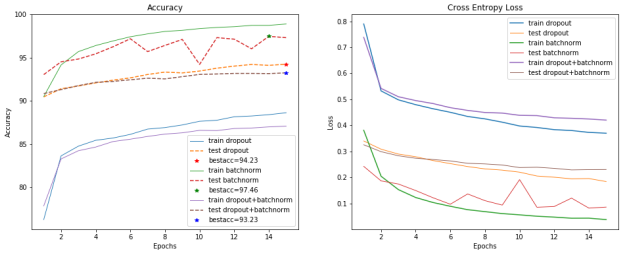


Fig. 8. Regularization in NN

B. Convolutional Neural Networks

Experimental details- LeNet, VGG16 [4] and AlexNet [2] models are used to solve 200 class classification problem on Tiny ImageNet. The dataset comprises of approximately 80k:20k train and test rgb images. The model training is done with data augmentation (another form of regularization) like random crop and random flip. Since the models are designed for different datasets in their respective papers, slight changes are made in the first or last layer to let it train on $3 \times 64 \times 64$ images. The model architectures used are given in Appendix (Figure 20-22)

Figure 9 summarizes the results obtained using the three CNNs. Following inferences and learnings from the experiments were drawn:

1) Performance wise (test accuracy) LeNet and VGG-16 are nearly same and AlexNet is better than both of them by almost 4%. Number of trainable parameters are $0.35, 14.8, 21.1$ M in the three models respectively.
2) The VGG-16 not performing better than LeNet most likely stems from the degradation problem as discussed in [1].
3) AlexNet is not a deeper but a wider network, and so it is working better than LeNet. However it doesn't mean with increase in width of the NN, the performance increases. There is a tradeoff between the depth and the width of the NN.
4) Empirical results showed that using a softmax layer at the end (in pytorch) is degrading the performance. This is counterintuitive and upon investigation it was found out that using a cross entropy loss, pytorch inherently uses softmax in its implementation. Thus no extra softmax layer is put in the code.

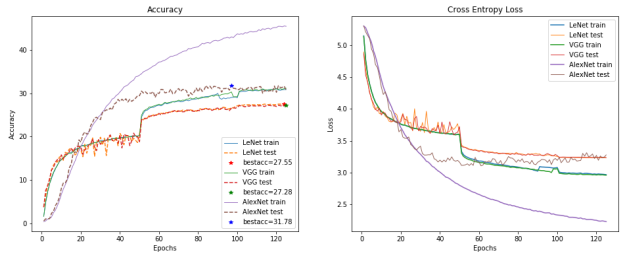


Fig. 9. CNNs on Tiny Image Net

The effect of different optimizers is summarized in Figure 10. Here, LeNet model is used with cross entropy loss (like before). Following inferences are drawn:

1) Momentum converges faster than SGD - it is evident in the plot that with learning rate 0.1 at the start of training, momentum plateaus near 20 epochs whereas SGD does near 70 epochs.
2) Adam gives relatively easier convergence in the sense that scheduling the learning rate was not much required. This is likely to be due to the learning adaptation in the adam optimizer.
3) However, since SGD is giving best test accuracy, there is no motivating reason to switch to different optimizers like Adam.

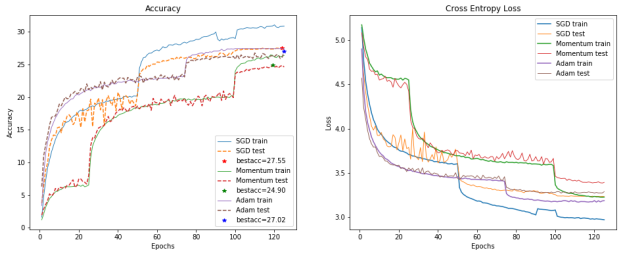


Fig. 10. LeNet with Different Optimizers

IV. Learning ++

A. PCA and tSNE on MNIST and SVHN

From Table III we can see that the MNIST data variance is spread across its all 28×28 components whereas the SVHN data variance is mostly in the first component.

	1	2	3	4	5	6	7	8
MNIST	9.7	7.1	6.17	5.39	4.87	4.31	3.27	2.88
SVHN	57.7	5.61	5.36	4.01	2.22	1.82	1.52	1.51

TABLE III
% Variance in first 8 principle components

Taking the first two principle components, the dataset is plotted in Figure 11. The MNIST results show that tSNE does better clustering (in respective class labels) than PCA. Whether or not, there is better clustering in SVHN using tSNE, is investigated in IV-C (supervised learning on SVHN after K-Means). It was empirically observed that tSNE time complexity is exponential with respect to that of PCA. Due to this, 20% data was used for tSNE on SVHN unlike 100% for MNIST.

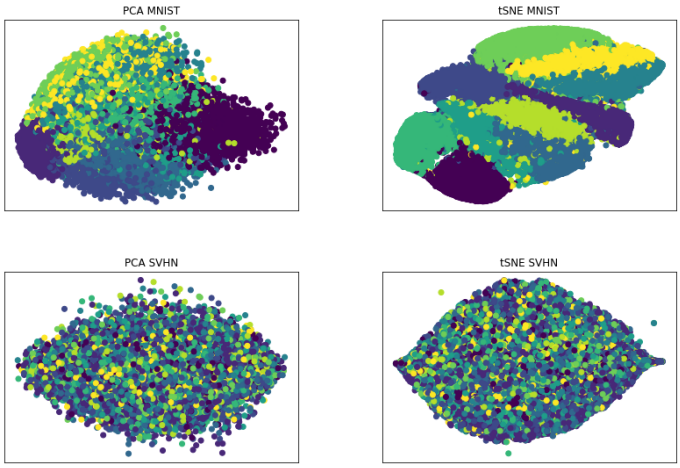


Fig. 11. Data Visualization

B. Q3(1) - Skewed Dataset Classification Problem

Experimental details- 5k samples from MNIST dataset are taken with $70 : 25 : 5$ ratio for class labels $0, 1$ and 2. The labels are wisely chosen so that the digits are not 'confusing' and the model learns to distinguish from the skewed dataset. Few layers are removed from LeNet model and the architecture used is given in Figure 23. Figure 12 summarizes the results. The results show that the model is able to learn and distinguish between the digits even the dataset is very skewed.

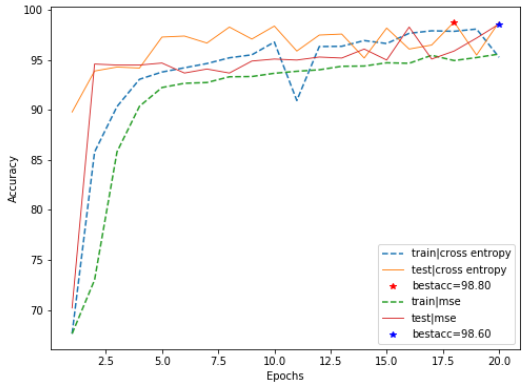


Fig. 12. Smaller LeNet on Skewed MNIST

C. Q3(2) Supervised Learning on SVHN after K-Means clustering

First K-Means algorithm is studied in detail. Figure 13 shows the clustering on SVHN for two values of k obtained using Sklearn KMeans. Clustering obtained using KMeans implemented from scratch is shown in Appendix (Figure 24). The goodness of clustering in KMeans is determined by (not only) the initialization strategy. The k-means are randomly initialized from the dataset in our implementation.

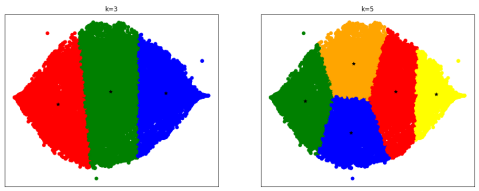


Fig. 13. K-Means Clustering on SVHN

Next, we do supervised learning on the 32×32 SVHN images with labels obtained from KMeans. Since the number of classes is determined by k, the last fully connected layer (output layer) has k neurons. Since we have 20k samples (obtained using tSNE), an even smaller LeNet model is used (model architecture given in Appendix(Figure 25)). Figure 14 summarizes the results. As k increases, the test accuracy decreases. From Figure 11, we see that there is no good clustering in the data through tSNE and now after using KMeans, the model is forced to distinguish between the clusters. Since each cluster contains images mapping to different digits (original labels), this not-good data representation using tSNE only seems to be the possible reason for the model not performing for $k = 10$ (total original class labels).

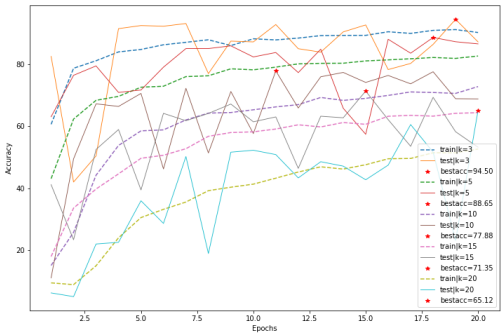


Fig. 14. Supervised Learning on SVHN with labels from K-Means

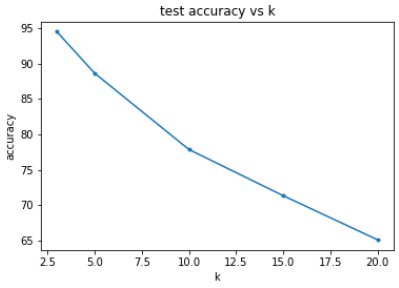


Fig. 15. Test accuracy and K in Supervised Learning & KMeans

Future work- Since the tSNE over MNIST gave good clustering (visualizable by humans), we expect that this approach of supervised learning using kmeans will give anticipated results.

D. Q3(3) Semi Supervised Learning Problem

Experimental details- A smaller LeNet model architecture is used (given in Appendix Figure 23). SVHN is 32×32 rgb images which is transformed to 28×28 grayscale images so that same architecture can be used for both the datasets. Figure 16 shows few sample images.

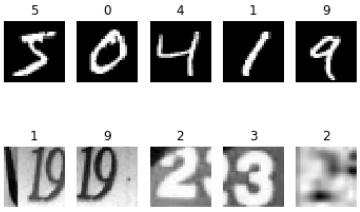


Fig. 16. MNIST (top) and SVHN in grayscale (bottom)

The model is trained on one dataset and simultaneously tested on the other dataset during SGD. Figure 17 summarizes the results. We can see that the model trained on MNIST doesn't perform well on SVHN (test accuracy $\leqslant 16\%$) but the model trained on SVHN still performs well over MNIST (test accuracy $= 50\%$). This is inline with what we expect from the datasets (Figure 16) - MNIST dataset is easy to learn since it consists of only (single) digits and no background whereas SVHN also has multiple digits and more noise than MNIST. Hence the model is able to learn or generalize more from SVHN and becomes more robust as compared to learning from MNIST.

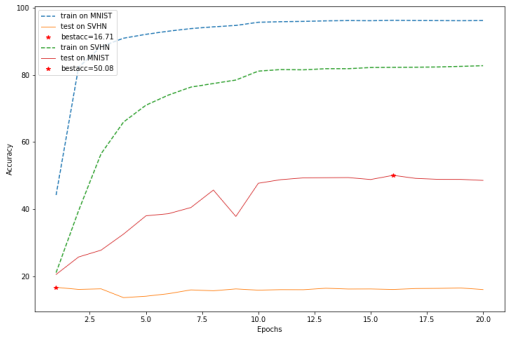


Fig. 17. LeNet trained on MNIST and tested on SVHN and vice-versa

Future work- After doing literature review, we infer the given problem statement as a semi-supervised learning problem. We intend to solve it using Noisy Student Training as discussed in [5]. This algorithm improves the accuracy and robustness (out-of-distribution generalization). We empirically

also saw a small version of out-of-domain generalization in above results where the model trained on SVHN is performing decently on MNIST (however this is not strictly out-of-domain generalization).

E. Q3(4) Methods to perform on Novel Classes

Experimental details- here the same, smaller, LeNet model is used (Figure 23) for 10-class classification problem. The model is trained on MNIST dataset comprising of 30k samples of classes $0, 1, ..., 4$. The testing is done complete test set of size 10k with all samples in $0, 1, ..., 9$ class labels.

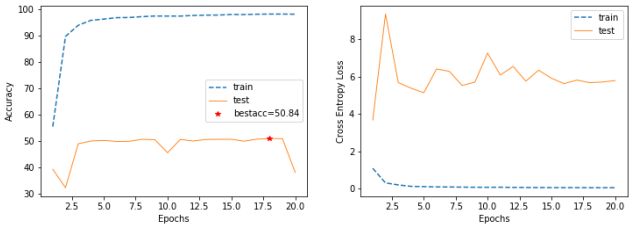


Fig. 18. LeNet trained on 5-class MNIST and tested on all 10 classes

Figure 18 and Table IV summarizes the results. It is evident that the model is not taking any chance to predict $5, 6, ..., 9$ classes over which it was not trained.

	Precision	Recall	F1
0	91.34	86.12	0.89
1	26.14	100.0	0.41
2	62.19	39.05	0.48
3	36.17	44.26	0.4
4	34.11	98.98	0.51
5	0.0	0.0	0.0
6	0.0	0.0	0.0
7	0.0	0.0	0.0
8	0.0	0.0	0.0
9	0.0	0.0	0.0

TABLE IV
PER-CLASS TEST METRICS

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.

[3] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods-Support Vector Learning*, 208, 07 1998.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[5] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V. Le. Self-training with noisy student improves imagenet classification, 2020.

APPENDIX

A. SVM and Kernel Machines - Question 1 : Health Dataset Visualization

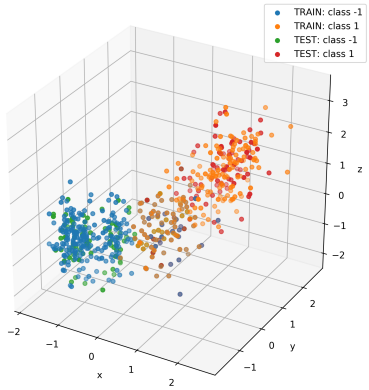


Fig. 19. Health Data

B. SVM and Kernel Machines - Question 1: Grid Search RBF SVM

	0.01	0.1	0.5	1.0	2.0
0.01	59.59	82.86	81.02	59.59	59.59
0.1	85.51	85.1	85.1	84.9	85.31
0.5	84.9	85.51	86.12	85.92	86.12
1.0	85.1	85.71	85.92	85.92	86.12
2.0	85.1	85.92	85.71	85.71	87.14

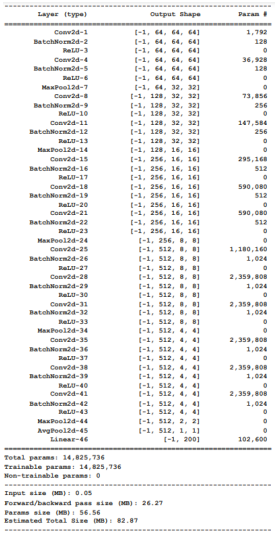
TABLE V
TRAIN ACCURACIES IN GRID SEARCH FOR RBF KERNEL SVM

C. LeNet Model Architecture

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 60, 60]             456
            Conv2d-2           [-1, 16, 26, 26]           2,416
            Linear-3                 [-1, 120]         324,600
            Linear-4                  [-1, 84]          10,164
            Linear-5                 [-1, 200]          17,000
================================================================
Total params: 354,636
Trainable params: 354,636
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 0.25
Params size (MB): 1.35
Estimated Total Size (MB): 1.65
----------------------------------------------------------------
```

Fig. 20.

D. VGG-16 Model Architecture



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 64, 64]	1,792
BatchNorm2d-2	[-1, 64, 64, 64]	128
ReLU-3	[-1, 64, 64, 64]	0
Conv2d-4	[-1, 64, 64, 64]	36,928
BatchNorm2d-5	[-1, 64, 64, 64]	128
ReLU-6	[-1, 64, 64, 64]	0
MaxPool2d-7	[-1, 64, 32, 32]	0
Conv2d-8	[-1, 128, 32, 32]	73,856
BatchNorm2d-9	[-1, 128, 32, 32]	256
ReLU-10	[-1, 128, 32, 32]	0
Conv2d-11	[-1, 128, 32, 32]	147,584
BatchNorm2d-12	[-1, 128, 32, 32]	256
ReLU-13	[-1, 128, 32, 32]	0
MaxPool2d-14	[-1, 128, 16, 16]	0
Conv2d-15	[-1, 256, 16, 16]	295,168
BatchNorm2d-16	[-1, 256, 16, 16]	512
ReLU-17	[-1, 256, 16, 16]	0
Conv2d-18	[-1, 256, 16, 16]	590,080
BatchNorm2d-19	[-1, 256, 16, 16]	512
ReLU-20	[-1, 256, 16, 16]	0
Conv2d-21	[-1, 256, 16, 16]	590,080
BatchNorm2d-22	[-1, 256, 16, 16]	512
ReLU-23	[-1, 256, 16, 16]	0
MaxPool2d-24	[-1, 256, 8, 8]	0
Conv2d-25	[-1, 512, 8, 8]	1,180,160
BatchNorm2d-26	[-1, 512, 8, 8]	1,024
ReLU-27	[-1, 512, 8, 8]	0
Conv2d-28	[-1, 512, 8, 8]	2,359,808
BatchNorm2d-29	[-1, 512, 8, 8]	1,024
ReLU-30	[-1, 512, 8, 8]	0
Conv2d-31	[-1, 512, 8, 8]	2,359,808
BatchNorm2d-32	[-1, 512, 8, 8]	1,024
ReLU-33	[-1, 512, 8, 8]	0
MaxPool2d-34	[-1, 512, 4, 4]	0
Conv2d-35	[-1, 512, 4, 4]	2,359,808
BatchNorm2d-36	[-1, 512, 4, 4]	1,024
ReLU-37	[-1, 512, 4, 4]	0
Conv2d-38	[-1, 512, 4, 4]	2,359,808
BatchNorm2d-39	[-1, 512, 4, 4]	1,024
ReLU-40	[-1, 512, 4, 4]	0
Conv2d-41	[-1, 512, 4, 4]	2,359,808
BatchNorm2d-42	[-1, 512, 4, 4]	1,024
ReLU-43	[-1, 512, 4, 4]	0
MaxPool2d-44	[-1, 512, 2, 2]	0
AvgPool2d-45	[-1, 512, 1, 1]	0
Linear-46	[-1, 200]	102,600

Total params: 14,825,736
Trainable params: 14,825,736
Non-trainable params: 0

Input size (MB): 0.05
Forward/backward pass size (MB): 26.27
Params size (MB): 56.56
Estimated Total Size (MB): 82.87

Fig. 21.

E. AlexNet Model Architecture

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 15, 15]	23,296
ReLU-2	[-1, 64, 15, 15]	0
MaxPool2d-3	[-1, 64, 7, 7]	0
Conv2d-4	[-1, 192, 7, 7]	307,392
ReLU-5	[-1, 192, 7, 7]	0
MaxPool2d-6	[-1, 192, 3, 3]	0
Conv2d-7	[-1, 384, 3, 3]	663,936
ReLU-8	[-1, 384, 3, 3]	0
Conv2d-9	[-1, 256, 3, 3]	884,992
ReLU-10	[-1, 256, 3, 3]	0
Conv2d-11	[-1, 256, 3, 3]	590,080
ReLU-12	[-1, 256, 3, 3]	0
MaxPool2d-13	[-1, 256, 1, 1]	0
Dropout-14	[-1, 256]	0
Linear-15	[-1, 4096]	1,052,672
ReLU-16	[-1, 4096]	0
Dropout-17	[-1, 4096]	0
Linear-18	[-1, 4096]	16,781,312
ReLU-19	[-1, 4096]	0
Linear-20	[-1, 200]	819,400

Total params: 21,123,080
Trainable params: 21,123,080
Non-trainable params: 0

Input size (MB): 0.05
Forward/backward pass size (MB): 0.69
Params size (MB): 80.58
Estimated Total Size (MB): 81.31

Fig. 22.

F. Question 3 part (1) Model Architecture

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 6, 28, 28]	60
Conv2d-2	[-1, 16, 14, 14]	880
Linear-3	[-1, 120]	94,200
Linear-4	[-1, 3]	363

Total params: 95,503
Trainable params: 95,503
Non-trainable params: 0

Input size (MB): 0.00
Forward/backward pass size (MB): 0.06
Params size (MB): 0.36
Estimated Total Size (MB): 0.43

Fig. 23. Smaller LeNet for Skewed MNIST

G. Question 3 part (2) K-Means from scratch

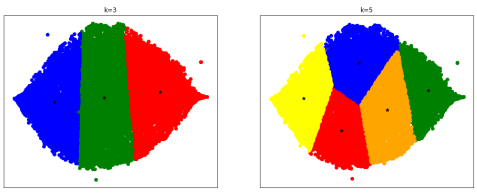


Fig. 24. K-Means Clustering on SVHN (from scratch)

H. Question 3 part (2) Model Architecture

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 6, 32, 32]	168
Conv2d-2	[-1, 16, 16, 16]	880
Linear-3	[-1, 10]	10,250

Total params: 11,298
Trainable params: 11,298
Non-trainable params: 0

Input size (MB): 0.01
Forward/backward pass size (MB): 0.08
Params size (MB): 0.04
Estimated Total Size (MB): 0.13

Fig. 25. Model architecture for $k = 10$