ML Lab Endsem

Pulkit Jindal da24b047

November 2025

1 Method of approach

I took a lot at multiple models in the process of finding the best ones and building the best architecture. A lot of these models did not perform too well and thus many are omitted from this discussion. I kept the models that had somewhat of a decent performance in this report as well as the failed attempts at ensembling them and the explanation of the final model, its underlying architecture and why I chose for this to be the final model.

2 Model 1 - Softmax Regression

2.1 Building the Model

The model was built by standard softmax regression where the probability of any class is given by $\frac{e^{\theta_k^T x}}{\sum e^{\theta_j^T x}}$ and the loss function being given by the cross entropy loss. To get the cross entropy loss we essentially compute

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log(\hat{y}_{ik})$$

Here y_i is a one hot encoded vector which consists of zeros besides one value which is one which corresponds to its label, in this case the digit of the number. We simply perform gradient descent on this loss function and use it to update the values of the parameters. In the implementation I have kept the number of iterations as an input parameter. We will obtain the gradient of this loss function to be

$$\nabla L_{\theta_j} = -\sum_{i=1}^{n} y_{ij} x_i + \sum_{i=1}^{n} \frac{x_i e^{\theta_j^T x_i}}{\sum_{l=1}^{K} e^{\theta_l^T x_i}}$$

Here I had to optimize the code, we notice that there are terms that repeat multiple times specifically $e^{\theta_l^T x_i}$. Thus in the for loop when looping through the x_i, I first precompute all the values of $e^{\theta_l^T x_i}$ and store them so that I don't recompute them and save time. This was the most crucial step in speeding up the algorithm.

2.2 Initial Results

Initially the learning rate was quite off. When the learning rate was too less for example $\frac{0.1}{\text{minibatch size}}$ the accuracy was very less around 20-30% which gave me the clue to try and increase the learning rate. At a learning rate between $0.05 - 0.1$ the model seems to perform best with the mini batch size which I used which was to split the entire dataset into 10 parts. This increased the accuracy a lot to around about 88%. Standardizing the values of the pixels also seemed to have a negative impact on the accuracy making it very low, so I decided to not normalize the pixel values.

2.3 Hyperparameter tuning and Further Optimizations

Replacing the dot function which I made with the numpy function increased the efficiency of the code greatly which made me realise the speedup that numpy functions can give. This allowed me to hyperparameter tune more efficiently and 0.05 seemed to be the best learning rate leading to an accuracy of around 89%.

3 Model 2 - K means

3.1 Building the Model

The model I built was similar to ordinary K-Means where I found all the cluster centers and then I labeled those cluster centers by the most frequently occuring labels in their respective clusters. Then for a given point x, its label would be the label corresponding to the closest cluster center. This is a slightly unorthodox method of using K means, but K means has a chance of finding very good clusters in the data and in that case it would perform very well. I used lloyd's algorithm with random point initialization for the model. While K means++ performs better, its computation time is too high, even with just one set of randomly initialized points the time required to run the program was already 3 minutes. Not many optimizations could be done in the algorithm considering it is already quite concise and there is no repeated computations which could be skipped out on like in the previous model.

3.2 Results and Conclusion

The accuracy on first go was around 50% and remained of the same order reaching at most 70% even after changing around of the hyperparameters like number of clusters and considering the accuracy is so low there was not much point in optimizing or trying to find better hyperparameters. Finally when making the value of k very large after numpy optimizations I was able to get a accuracy of around 89 % with k around 300 which is quite good but likely this is working closer to KNN than actually clustering the data.
Likely this is a result of the high dimension number of the data. These methods tend to perform well when the number of dimensions is limited due to the curse of dimensionality,but in this case the number of dimensions is very large. Methods of dimensionality reduction can be tried, but the dimension number is so high that its infeasible to reduce it to the point where K Means would perform well and thus I left this model out of the final Ensemble.

4 Model 3 - Random Forest

4.1 Building the Model

I simply used the code which we had coded in the lab with minor modifications to allow for speedups since the dataset size was very large. The speedups mostly include checking less threshold values to prevent checking every threshold value. Random Forest was still very slow.

4.2 Results and conclusion

Due to the lack of speed of random forest as well as its poor performance yielding about a 76.8% accuracy I decided to not try and refine this model further considering models like KNN perform far better with almost no refinement.

5 Model 4 - Linear Regression

This Model performed very poorly as expected since the task is far too complex and is a classification task not a regression task. The accuracy achieved was around 20% and thus not included in the final result.

6 Model 5 - KNN

6.1 Building the Model

I used KNN because the dataset size was large and the drawings of the numbers are quite simple. It performed surprisingly well. A major speedup was to replace the distance metrics with the numpy functions of the same leading to speedup of 10-50x allowing for better hyperparameter tuning. I also added PCA to try and make sure that the distance metric was as meaningful as possible.

6.2 Hyperparameter tuning and Results

I used a function to check through all values of k from 1 to 100 and obtained k = 5 as the one with the best accuracy on the validation data. Also on hyperparameter tuning on the number of components in PCA I found 50 to perform the best, but in the final model I perform no PCA since it was possible that it was overfitting since the improvement in accuracy was marginal and likely not worth it.

7 Final Summary

Model	Accuracy	Training Time	Observation
Softmax Regression	89%	Fast	Good baseline
K-Means	88%	Medium	Large values of k needed
Random Forest	76.8%	Slow	Overfits
Linear Regression	20%	Fast	Performs very poorly
KNN (k=5)	95%	Fast	Best Performing

Table 1: Comparison of the 4 models.

8 Failed Ensemble of the Models

8.1 KNN Hybrids

Considering KNN had a very high accuracy rate, I tried combining various models with KNN to try and raise its accuracy. The first thing I tried was using softmax regression on points where KNN performed poorly. I classified a 'poor' judgment or hard to judge point by the points which were in low density regions meaning that the average distance of the k closest points was high. In those cases I attempted to use Softmax regression to classify the points. This ultimately led to a marginal drop in the accuracy. This is likely because softmax is quite a bit worse than KNN so in the points with low density softmax still performs worse than KNN and some correctly classified points by KNN may also be present in low density region. Besides softmax, I tried to use leave one out KNN training on the train dataset to identify regions which were incorrectly predicted and built a decision tree to classify those special points. This again dropped the accuracy as many points that would have been classified correctly in the validation dataset started being classified wrongly.
The best KNN accuracy was 95.0%, while the hybrid models achieved around 94.1

8.2 Majority Voting

Using the 3 best models of the ones I had trained, softmax regression, K means and KNN I used the majority vote to see if that would increase the accuracy that KNN produced. This would be helpful if Kmeans and softmax regression happened to be right on the digits where KNN was wrong. However this turned out to still be a decrease on the accuracy of KNN and thus even this approach was scrapped. The Majority Voting resulted in an accuracy of 93.72%.

9 Final Architecture

The final architecture used in the model was all the models placed under a decision tree. I realized that the KNN model is performing far better than the other models and that was the reason for methods such as majority voting not working particularly well,so I decided which model to use based on a decision tree. These are the functions and models in the architecture.

9.1 New Classes Function

This function is designed to label all datapoints in the training with a class. Each class corresponds to a model. Class 1 corresponds to Softmax, class 2 to KMeans and class 3 to KNN. Depending on the priority of classes given in the for loops all the points will be assigned a class based on which model predicts the digits correctly.

9.2 Priority 1 - Model 3 - KNN/LOO KNN

KNN works as the first priority model, if its prediction on the train datapoints is correct then it will be chosen as the label for the datapoint and if all models classify the digit incorrectly it will still be the label since it is the model with the highest accuracy and thus given the highest priority. Here it is clear that since all points have 0 distance with themselves KNN's training performance with the train dataset will be artificially high so a Leave one out KNN function has been created which essentially does KNN and excludes the point itself when performing KNN.

9.3 Priority 2 - Model 2 - KMeans

KMeans is the second priority model meaning it will the label for a datapoint if KNN incorrectly predicits the digit for a given datapoint and KMeans correctly predicts the digit for that datapoint.

9.4 Priority 3 - Model 1 - Softmax

Softmax is the last priority model meaning a datapoint will be assigned its label as 1 corresponding to Model 1 only if both of the first 2 priority models incorrectly predict the digit and Softmax predicts the digit correctly.

9.5 Decision Tree

I train a classifier on the training dataset, but instead of using the digits labels corresponding to the datapoints I use the classes generated by my new classes function. Essentially the decision tree is no longer trying to predict the digit of the datapoint, but the best model to use for this datapoint. This ensures that a datapoint if performing poorly on KNN will be evaluated using a different model.