PA1 - Prototype Selection for Nearest Neighbor

Jorge Avila
A14226029

January 2021

1 Least Likely Nearest Neighbor

The Least Likely Nearest Neighbor (LLNN) prototype selection method finds the feature vectors that are the nearest neighbor (NN) of some other feature vector in the training set, the least number of times. From 1, it is evident that MNIST is highly clustered but also noisy. Finding the feature vectors with the most nearest neighbors resulted in highly unbalanced prototypes. In LLNN, the nearest nieghbor is computed for all feature vectors, excluding the point itself, and the number of hits for each training point are collected and sorted in ascending order. The number of prototypes, M, required are then taken from the first M entries of the sorted list.

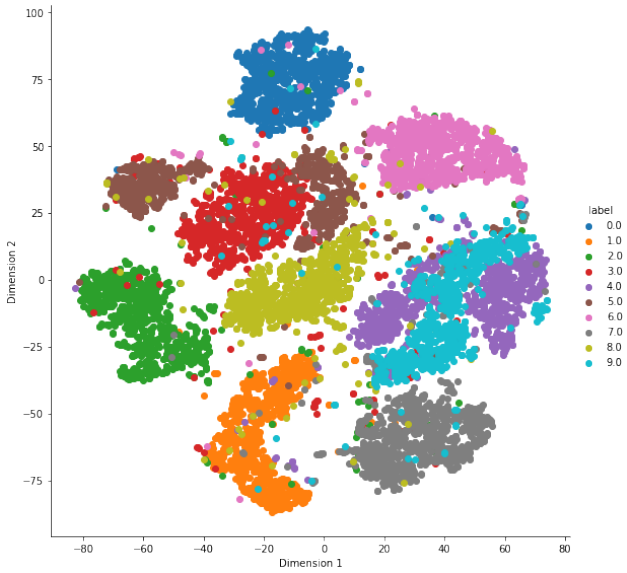

Figure 1: T-SNE visualization of 10,000 samples of MNIST. The feature vectors were preprocessed using PCA to improve runtime.

The Least Likely Nearest Neighbor (LLNN) prototype selection method came about through the experimental results from established prototype selection methods - including KMeans, Learning Vector Quantization (LVQ), and adaptations of these methods [1, 3, 4, 5]. In all previous attempts, the prototypes were either drawn or generated near clusters of points with matching labels. This consistently resulted in low test accuracy and LLNN was developed based on these results. It should be noted that LLNN requires thorough testing on other datasets to determine if performance is specific to MNIST.

2 Algorithm

Algorithm 1: LeastLikelyNN($A = \{(x_1, y_1), ..., (x_n, y_n)\}, M$)

$\Delta \leftarrow \emptyset$;
$\Phi \leftarrow \emptyset$;
foreach $x_i \in A$ **do**
  foreach $z_i \in A \setminus x_i$ **do**
    $\Delta \leftarrow \text{argmin}_i \, d(z_i, x_i)$
  end
end
$\Delta \leftarrow \textbf{sort}(\Delta, \textbf{ascending} = False)$;
$\Phi \leftarrow \{\Delta_1, \ldots, \Delta_M\}$;
return Φ;

Algorithm 1 shows the simplicity of LLNN. We simply compute the distance from every point to every other point in the training set and keep the nearest neighbor. Then we sort in descending order and take the first M feature vectors as the prototypes.

Despite it's simplicity, finding the nearest neighbor between every pair of points in the dataset is extremely computationally expensive. Computing these distances necessitated the use of *PyKeOps* [2] to drastically reduce the runtime for LLNN and KNN.

Algorithm 2: RandomPrototype($A = \{(x_1, y_1), ..., (x_n, y_n)\}, M$)

$I \leftarrow \textbf{shuffle}(\{1, \ldots, |A|\})$;
$\Phi \leftarrow \{A_{I_1}, \ldots, A_{I_M}\}$;
return Φ

3 Experimental Results

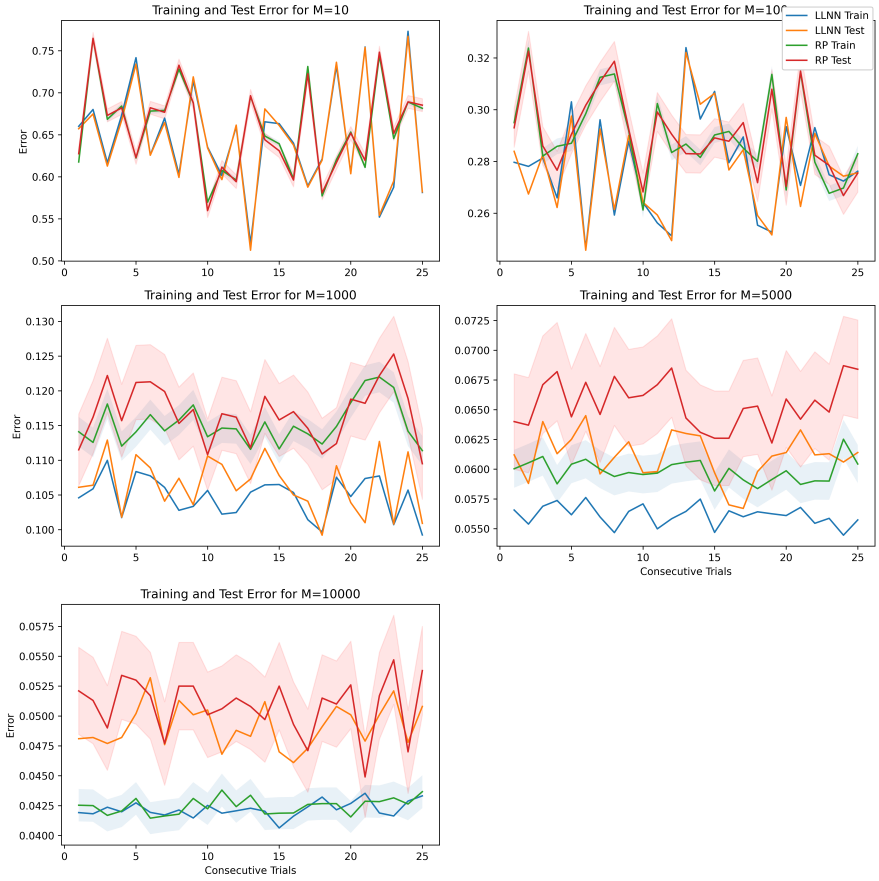

Figure 2: Training and test error of 25 trials for each configuration of M. Confidence intervals were computed for the baseline prototype selection model.

From 2 we can see that LLNN outperformed the baseline model for M values greater than 100. Although, the baseline model performed unexpectedly well. Previous experiments made use of KMeans and LVQ but the baseline model outperformed them by a wide margin making them unfit for prototype selection on MNIST. Understandably, lower values of M produced mixed results.

Due to the randomness of the baseline prototype selection model confidence intervals were computed for each trial using the following equation.

$$c = z\sqrt{\frac{\gamma(1-\gamma)}{n}} \tag{1}$$

Here $z = 1.64$ for 90% confidence intervals, γ is the training or test error, and n is the number of samples.

For each configuration of M, 25 trials were run to compute the prototypes and evaluate performance on KNN - where $K = 3$ for every trial. The dataset was also shuffled for every trial and the $70,000$ MNIST samples were split $10,000$ for testing and $60,000$ for training.

4 Critical Evaluation

LLNN was a clear improvement over random selection. Established prototype selection methods were not able to outperform random selection in previous experiments on MNIST. In LVQ and KMeans initial prototypes were generated with random values and it seems this does suit MNIST. LLNN is a heuristic that was designed based on these results and would likely not benefit from further improvements. Future experiments would likely introduce different methods for initializing prototypes in LVQ. Since it seems this greatly affects performance on this dataset, LVQ would likely benefit from initialization techniques that use some offset of the training data.

References

[1] Jacob Bien and Robert Tibshirani. Prototype selection for interpretable classification. *The Annals of Applied Statistics*, 5(4):2403–2424, Dec 2011.

[2] Benjamin Charlier, Jean Feydy, Joan Alexis Glaunès, François-David Collin, and Ghislain Durif. Kernel operations on the GPU, with autodiff, without memory overflows. *arXiv preprint arXiv:2004.11127*, 2020.

[3] S. Garcia, J. Derrac, J. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435, 2012.

[4] Hemavati, V. Susheela Devi, Seba Ann Kuruvilla, and R. Aparna. Prototype selection and dimensionality reduction on multi-label data. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, CoDS COMAD 2020, page 195–199, New York, NY, USA, 2020. Association for Computing Machinery.

[5] David Nova and Pablo A. Estévez. A review of learning vector quantization classifiers. *Neural Computing and Applications*, 25(3-4):511–524, Dec 2013.