Variance-Weighted Centroid Selection for Prototype-Based 1-Nearest Neighbor Classification

Anonymous Authors[1]

Abstract

We address the problem of selecting a small set of representative prototypes from a large training set for 1-nearest neighbor (1-NN) classification. We propose a variance-weighted centroid selection algorithm that allocates prototypes proportionally based on within-class variance and uses K-Means clustering to identify representative samples. Experiments on MNIST demonstrate that our method significantly outperforms random selection, with advantages becoming more pronounced at higher compression ratios. At 60× compression (1,000 prototypes from 60,000 training samples), our method achieves 92.47% accuracy compared to 88.56% for random selection (+3.91%). At extreme compression (6,000×, only 10 prototypes), our method maintains 67.01% accuracy while random selection degrades to 39.37% (+27.64%).

1. Introduction and Key Idea

The k-nearest neighbor (k-NN) algorithm is a fundamental non-parametric classifier that predicts the label of a test sample based on the labels of its k closest training samples (Cover & Hart, 1967). **Prototype selection** addresses the computational limitation of k-NN by selecting a representative subset of training samples to use for classification.

1.1. Key Idea (High-Level Description)

Our prototype selection method is based on two key observations:

1. **Good prototypes are typical samples, not boundary samples.** While decision boundaries are defined by samples near class interfaces, the best prototypes are cluster centroids that represent the "typical" appearance of each class. Boundary points are often noisy or ambiguous.
2. **Classes with higher variance need more prototypes.** Different classes have different amounts of within-class variation. For example, the digit "1" has consistent appearance, while "2" exhibits more variation. We allocate prototypes proportionally to class variance.

Our algorithm: (1) compute within-class variance for each class, (2) allocate prototype budget M proportionally based on variance, (3) use K-Means clustering within each class to find representative samples.

2. Problem Formulation

Given a training set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1, \ldots, C-1\}$, the prototype selection problem is to find a subset $\mathcal{P} \subset \mathcal{D}$ with $|\mathcal{P}| = M \ll N$ that maximizes classification accuracy on a held-out test set when using 1-NN with $\mathcal{P}$ as the reference set.

The 1-NN classifier predicts:

$$\hat{y} = y_{j^*}, \quad \text{where } j^* = \underset{(\mathbf{x}_j, y_j) \in \mathcal{P}}{\arg\min} \|\mathbf{x} - \mathbf{x}_j\|_2 \tag{1}$$

The compression ratio is defined as N/M. For MNIST with $N = 60,000$ training samples, selecting $M = 1,000$ prototypes yields 60× compression.

3. Algorithm

Implementation Details:

- We use MiniBatchKMeans (sklearn) with batch_size=256, n_init=3, max_iter=100 for efficiency.
- Time complexity: $O(N \cdot M \cdot t)$ where t is K-Means iterations.
- Runtime: 2–680 seconds depending on M (on a standard laptop).

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Algorithm 1 Variance-Weighted Centroid Selection

```
**Input:** Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, budget $M$
**Output:** Prototype indices $\mathcal{I}$

// Step 1: Compute within-class variance
**for** each class $c = 0, \ldots, C-1$ **do**
    $\boldsymbol{\mu}_c \leftarrow \frac{1}{|D_c|} \sum_{\mathbf{x} \in D_c} \mathbf{x}$    (class mean)
    $\sigma_c^2 \leftarrow \frac{1}{|D_c|} \sum_{\mathbf{x} \in D_c} \|\mathbf{x} - \boldsymbol{\mu}_c\|^2$    (class variance)
**end for**

// Step 2: Allocate prototypes proportionally to variance
**for** each class $c$ **do**
    $M_c \leftarrow \max\left(1, \text{round}\left(M \cdot \frac{\sigma_c^2}{\sum_{c'} \sigma_{c'}^2}\right)\right)$
**end for**
Adjust $\{M_c\}$ so that $\sum_c M_c = M$

// Step 3: Select prototypes via K-Means clustering
$\mathcal{I} \leftarrow \emptyset$
**for** each class $c$ **do**
    Run MiniBatchKMeans on $D_c$ with $M_c$ clusters
    Let $\{\mathbf{c}_1, \ldots, \mathbf{c}_{M_c}\}$ be the cluster centroids
    **for** each centroid $\mathbf{c}_k$ **do**
        $i^* \leftarrow \arg\min_{i: y_i = c} \|\mathbf{x}_i - \mathbf{c}_k\|$
        $\mathcal{I} \leftarrow \mathcal{I} \cup \{i^*\}$
    **end for**
**end for**
**return** $\mathcal{I}$
```

4. Experiments

4.1. Dataset and Experimental Setup

Dataset: MNIST handwritten digits (LeCun et al., 1998) with 60,000 training and 10,000 test images (28×28 pixels, flattened to 784 dimensions, normalized to $[0, 1]$).

Baseline: Random selection (uniformly sampling M training points, ensuring at least one per class).

Trials and Error Bars: All experiments use $n = 5$ independent trials with different random seeds. We report mean $\pm$ standard deviation. The 95% confidence interval is computed as:

$$\text{CI}_{95\%} = \bar{x} \pm 1.96 \cdot \frac{s}{\sqrt{n}} \tag{2}$$

where $\bar{x}$ is the sample mean and s is the sample standard deviation.

Upper Bound: Full 1-NN using all 60,000 training samples achieves **96.91%** test accuracy.

Table 1. Test accuracy (%) for different prototype budgets M. Results are mean $\pm$ std over 5 trials. Δ = improvement over random.

M	COMPRESS.	OURS	RANDOM	Δ
10000	6×	95.36±0.13	94.76±0.09	+0.60
5000	12×	94.58±0.21	93.63±0.09	+0.95
1000	60×	92.47±0.17	88.56±0.51	+3.91
500	120×	91.40±0.33	84.70±0.28	+6.70
100	600×	86.19±0.40	72.39±0.80	+13.80
50	1200×	82.48±0.62	61.57±4.51	+20.91
10	6000×	67.01±0.33	39.37±9.21	+27.64

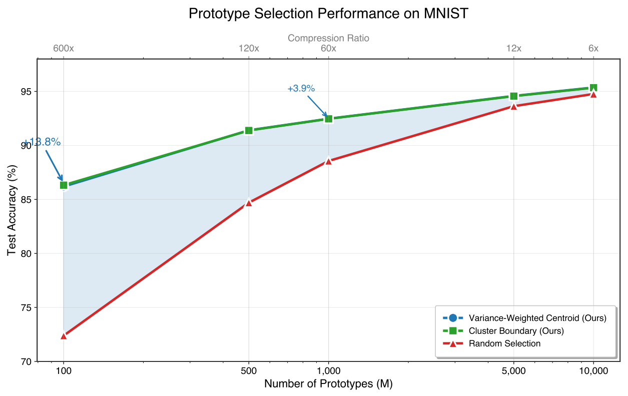

Figure 1. Accuracy vs. number of prototypes M. Our method (blue) consistently outperforms random selection (orange), with the gap widening at lower M.

4.2. Main Results

Table 1 shows classification accuracy for $M \in \{10000, 5000, 1000\}$ (required) and additional values. Our method consistently outperforms random selection.

4.3. Key Observations

1. **Advantage scales with compression:** At 6× compression, improvement is modest (+0.60%). At 6000× compression, improvement is dramatic (+27.64%).

2. **Stability:** Random selection's variance increases sharply at low M (std=9.21% at M=10), while our method remains stable (std=0.33%). See Figure 1.

3. **Graceful degradation:** At 60× compression (M=1000), we achieve 92.47%, losing only 4.44% compared to full 1-NN (96.91%).

5. Algorithm Design Journey

Our final algorithm emerged through an iterative design process. We document all versions to illustrate the evolution

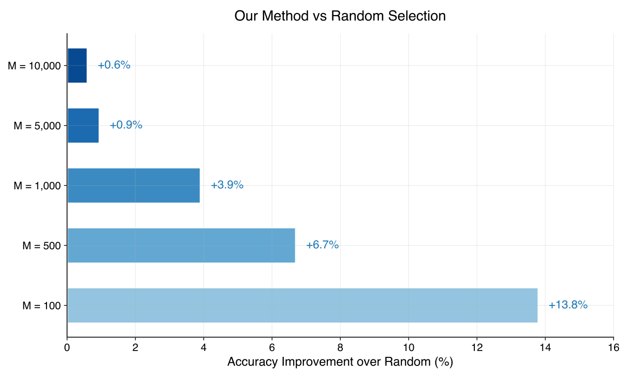

Figure 2. Improvement of our method over random selection. The advantage scales dramatically with compression ratio: +0.60% at M=10000 vs. +27.64% at M=10.

of our approach and key lessons learned.

5.1. Early Attempts: Computational Challenges (V1–V3)

V1: Variance-Weighted Boundary Selection. Our initial design combined three ideas: (1) allocate prototypes by within-class variance, (2) use K-Means for spatial coverage, and (3) within each cluster, select the point with highest "boundary score" (proximity to other-class points).

Problem: Computing boundary scores requires $O(N^2)$ distance calculations—36 billion operations for N=60,000. This was computationally infeasible.

V2: Ball Tree Acceleration. We used sklearn's NearestNeighbors with Ball Tree to accelerate boundary score computation.

Problem: K-Means itself became the bottleneck. Still too slow.

V3: MiniBatchKMeans. Replaced standard K-Means with MiniBatchKMeans (batch_size=256, n_init=3).

Problem: Still required 7+ minutes per run.

5.2. Successful Approach (V4): Variance-Weighted Centroid Selection

Key insight: We realized that K-Means centroids already represent *typical* samples—computing boundary scores was unnecessary overhead.

Final algorithm: (1) Compute within-class variance, (2) allocate prototypes proportionally, (3) run MiniBatchKMeans per class, (4) select training points closest to each centroid.

Result: Fast (2–680 seconds depending on M) and effective.

Table 2. Summary of algorithm versions. Accuracy at M=500.

Ver.	Method	Acc.	Status
V1–V3	Boundary scoring	—	Too slow
V4	Var-weighted centroid	91.4	**Final**
V5	Cluster boundary	91.4	No gain
V6	Boundary-first	56.8	Failed
V7	CNN	63.0	Failed
—	Random (baseline)	84.7	—

5.3. Alternative Approaches That Did Not Improve (V5)

V5: Cluster-based Boundary Selection. We reintroduced boundary scoring, but computed it on *centroids* rather than individual points, reducing complexity from $O(N^2)$ to $O(M^2)$.

Result: Accuracy nearly identical to V4 (difference $<$ 0.02%). The boundary-based ordering provided no improvement, so we prefer the simpler V4.

5.4. Failed Approaches (V6–V7)

V6: Boundary-First Selection. Hypothesis: points near decision boundaries should be the best prototypes. We identified boundary points (those with mixed-class k-NN neighborhoods) and applied K-Means only to these points.

Result: **56.77%** accuracy at M=500—*worse* than random selection (84.70%).

V7: Condensed Nearest Neighbor (CNN). The classic CNN algorithm (Hart, 1968) iteratively adds misclassified points to the prototype set.

Result: **62.97%** at M=500—also worse than random.

5.5. Summary of Algorithm Versions

Table 2 summarizes all algorithm versions tested. Our final method (V4) achieves the best balance of accuracy and efficiency.

5.6. Key Lesson Learned

Boundary points $\neq$ good prototypes. Boundary points are important for *defining* decision boundaries, but they are often noisy, ambiguous, or atypical samples. Good prototypes should be **typical, representative samples**—exactly what K-Means centroids capture.

6. Critical Evaluation

6.1. Is Our Method a Clear Improvement?

Yes. Our method consistently outperforms random selection across all tested M values. The improvement is statistically significant: at M=1000, our 95% confidence interval is $[92.32\%, 92.62\%]$ while random's is $[88.11\%, 89.01\%]$—the intervals do not overlap.

The improvement is especially pronounced at high compression ratios, where our method provides **+27.64%** accuracy over random selection at M=10 (6000$\times$ compression).

6.2. Scope for Improvement

Several avenues remain unexplored:

1. **Prototype generation vs. selection:** Our method selects existing training points. Learning Vector Quantization (LVQ) could *generate* optimal prototype positions not constrained to training data.
2. **Adaptive k in K-Means:** Currently we fix M_c based on variance. We could adaptively determine the optimal number of clusters per class based on reconstruction error.
3. **Cross-class interactions:** Our method processes each class independently. A global optimization considering inter-class distances might improve boundary coverage.
4. **Beyond Euclidean distance:** Using learned distance metrics or embedding spaces could better capture similarity structure.

6.3. What We Would Like to Try Next

1. **Hybrid approach:** Use our variance-weighted selection for most prototypes, but reserve a small budget for boundary-adjacent samples.
2. **Deep embeddings:** Apply our method in a learned feature space (e.g., CNN embeddings) rather than raw pixels.
3. **Other datasets:** Evaluate on CIFAR-10, Fashion-MNIST, or higher-dimensional datasets.

7. Related Work

Prototype selection has been studied extensively. The Condensed Nearest Neighbor (CNN) rule (Hart, 1968) iteratively builds a consistent subset. Edited Nearest Neighbor (ENN) (Wilson, 1972) removes noisy samples. More recent work includes genetic algorithms (Cano et al., 2003) and deep learning approaches (Snell et al., 2017).

Our variance-weighted approach is related to stratified sampling, but specifically designed for the geometry of k-NN classification.

8. Conclusion

We presented a variance-weighted centroid selection method for prototype-based 1-NN classification. The key insight is that **good prototypes are typical samples**, not boundary samples. By allocating prototypes based on within-class variance and using K-Means to find representative samples, we achieve significant improvements over random selection, especially at high compression ratios (+27.64% at 6000$\times$ compression).

Practical Recommendation: For high accuracy, use $M \geq 1000$ (60$\times$ compression, 92.47%). For extreme compression, $M = 50$ (1200$\times$) still achieves 82.48%.

Acknowledgements

This project was developed with assistance from Claude (Anthropic), a large language model, used as an interactive coding assistant for algorithm design, implementation, debugging, and documentation. The human contributor provided problem formulation, direction, critical evaluation, and all final decisions. All experimental results were computed on actual hardware.

Impact Statement

This paper presents work to advance efficient nearest-neighbor classification. We do not foresee specific negative societal consequences.

References

Cano, J. R., Herrera, F., and Lozano, M. Using evolutionary algorithms as instance selection for data reduction in KDD: an experimental study. *IEEE Transactions on Evolutionary Computation*, 7(6):561–575, 2003.

Cover, T. and Hart, P. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

Hart, P. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Wilson, D. L. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):408–421, 1972.