CSE251A Project 1 Write-up

Junwei Chen [1]

Abstract

In this project we implement 5 methods to select prototypes from MNIST for 1-NN (Nearest Neighbor) classification problems. 4 of the methods are based on the common KMeans clustering, while the remaining 1 is based on Locality Sensitive Hashing. We evaluate the 5 ideas and compared their performance to random selection as a baseline.

1. Method Description

In this section we discuss our development of the 5 ideas for prototype selection that shrinks the training set down to size M, which we specify. Methods 1 & 2 are closely related, while methods 3 & 4 are closely related. Therefore, we put the pseudocode of 1 & 2, 3 & 4 together in Sec. 2.

1.1. Method 0: Baseline with Random Selection

We use random selection as a baseline method. It shrinks the training set by randomly selecting M datapoints.

1.2. Method 1: KMeans with Majority Vote

A natural way to select prototypes is to group points into clusters and use cluster centroids as prototypes. If a query is closest to a centroid, it is close to all the cluster's datapoints. Therefore, we use KMeans on all of the training data to cluster them into M clusters. Then we can create a compressed training data with the M cluster centroids. To label the centroid, we use a majority vote inside each cluster and break ties by using the label with smaller numeric value.

1.3. Method 2: KMeans with Purified Cluster

We improve on our idea for Method 1. After the majority vote inside each cluster that determines the label, it makes sense to purify the cluster by removing all the points with labels other than the majority label and recompute the cluster centroid. Therefore, the cluster centroid can better represent the features of the majority label and be free of interference from data with other labels.

1.4. Method 3: KMeans with Label Buckets

Instead of using the entire training set for KMeans clustering and then purify the clusters, why not separate the training set first by different labels and then do KMeans separately on each partition which is pure in labels? We first split the training data of MNIST into 10 buckets by its label. Then we do KMeans clustering on each bucket to obtain $M/10$ clusters for each bucket, which give $M/10$ cluster centroids. We label each centroid by its bucket's label. Putting all centroids of the 10 buckets together gives us M prototypes.

1.5. Method 4: KMeans with 1NN of Bucket Centroids

In all the methods above, we use cluster centroids obtained from KMeans clustering, which are the mean of all the points in each cluster. These are "virtual points" that do not exist in the training data. If we what to sample a subset of M prototypes from the training data instead of constructing "virtual points", we can take the 1-Nearest-Neighbor of the cluster centroids we obtained in Method 3 in the training data. In this way, we can have a good approximation to the cluster centroids.

1.6. Method 5: Locality Sensitive Hashing

During literature search, we encountered a method called Locality Sensitive Hashing (LSH). The idea is to use a similarity-preserving hashing method to hash the raw data into binary code of several bits. The Hamming distances between the binary codes preserve the distance between original data points. Moreover, data that are close together are more likely to have the same binary code. We exploit this property of LSH and cluster data of the same binary code. In other words, LSH replaces KMeans as the clustering algorithm in Method 3 (we also use label buckets).

2. Pseudocode

We put the pseudocode of Method 1 and 2 together, method 3 and 4 together, method 5 by itself in Algorithm 1, 2, 3.

[1]UC San Diego, CSE Department. Correspondence to: Junwei Chen <juc005@ucsd.edu>, Taylor Berg-Kirkpatrick <tberg@ucsd.edu>.

Algorithm 1 Method 1 & 2

Input: data X, compressed size M
$prototypeSet$ = []
k = $KMeans(M).fit(X)$
for cluster $c \in k$ **do**
  label = $MajorityVote(c)$
  (for Method 2) $Purify$(c, label)
  $c.centroid.label$ = label
  $prototypeSet.append(c.centroid)$
end for
return $prototypeSet$

Algorithm 2 Method 3 & 4

Input: data X, compressed size M
$prototypeSet$ = []
for $i = 1$ **to** 10 **do**
  B = $labelEquals(X, i)$
  k = $KMeans(M).fit(B)$
  for cluster $c \in k$ **do**
    (for M4) $c.centroid$ = $oneNN(c.centroid, B)$
    $c.centroid.label$ = i
    $prototypeSet.append(c.centroid)$
  end for
end for
return $prototypeSet$

3. Experimental Results

3.1. Methods

We use the standard library sklearn to implement the 1NN and KMeans. The distance metric and accuracy computation are default used in sklearn.

We compute the 1NN 10 times to measure the average runtime.

For Method 5, we use the public GitHub repository LSHash by user loretoparisi to implement Locality Sensitive Hashing (LSH). We cannot specify the number of hash buckets in LSH. We can only specify the length of the binary code after hashing. Therefore, we pick a code length that gives us an M close to 10000 and evaluate all our methods based on that M.

Algorithm 3 Method 5

Input: data X, compressed size M
$prototypeSet$ = []
for $i = 1$ **to** 10 **do**
  B = $labelEquals(X, i)$
  lsh = $LocalitySensitiveHashing(\log_2 M).fit(B)$
  for hash entry $h \in lsh.hashTable$ **do**
    $h.centroid.label$ = i
    $prototypeSet.append(h.centroid)$
  end for
end for
return $prototypeSet$

Table 1. M, accuracy, runtime of Baseline v.s. Methods 1-4

METHOD	M	ACCURACY(%)	RUNTIME(S)
BASELINE	10000	94.92	2.86
METHOD 1	10000	96.38	2.77
METHOD 2	10000	96.78	2.78
METHOD 3	10000	97.08	2.79
METHOD 4	10000	95.68	2.80
BASELINE	5000	93.75	1.46
METHOD 1	5000	95.74	1.35
METHOD 2	5000	96.28	1.48
METHOD 3	5000	97.01	1.39
METHOD 4	5000	94.97	1.36
BASELINE	1000	87.70	0.54
METHOD 1	1000	94.38	0.54
METHOD 2	1000	95.20	0.54
METHOD 3	1000	96.00	0.53
METHOD 4	1000	92.55	0.55

Table 2. M, accuracy, runtime of Baseline v.s. Methods 1-5

METHOD	M	ACCURACY(%)	RUNTIME(S)
BASELINE	9472	94.84	2.51
METHOD 1	9472	96.40	2.60
METHOD 2	9472	96.72	2.67
METHOD 3	9472	97.03	2.63
METHOD 4	9472	95.67	2.75
METHOD 5	9472	94.91	2.61

3.2. Results

We summarize our results in two tables Table 1 and 2. Table 1 contains the results from Methods 1-4 with M being 10000, 5000, 1000. Table 2 has the results from Method 5 compared to other methods, with M=9472 close to 10000.

4. Evaluation

4.1. Comparison

From Table 1 and 2, we see all our 5 methods achieve better results than the baseline, which is random selection. The runtime difference is negligible, may be due to randomness. This makes sense because all of them are performing 1NN on the same number of datapoints. Under all M, Method 3 achieves the best result. All the methods that use virtual points derived from KMeans (Methods 1-3) are better than Method 4, which uses the closest actual point to the centroid in the training data. Method 5 performs the worst in all 5

methods, offering only a slight improvement over baseline, suggesting that using Locality Sensitive Hashing as a clustering method may be a bad choice compared to KMeans, as it is not better than random selection.

4.2. Discussion

If time permits, I would like to try to combine Method 2 and 3, that is, first clean the dataset by KMeans on all data and purify, then separate the data into buckets and do KMeans on each. This way, we can remove outliers of a certain label.