DE GRUYTER OPEN

JAISCR, 2017, Vol. 7, No. 3, pp. 155 – 169
10.1515/jaiscr-2017-0011

PARALLEL MCNN (PMCNN) WITH APPLICATION TO PROTOTYPE SELECTION ON LARGE AND STREAMING DATA

V. Susheela Devi and Lakhpat Meena

Department of Computer Science and Automation,
Indian Institute of Science, Bangalore, India

Submitted: 1st January 2016; accepted: 4th July 2016

Abstract

The Modified Condensed Nearest Neighbour (MCNN) algorithm for prototype selection is order-independent, unlike the Condensed Nearest Neighbour (CNN) algorithm. Though MCNN gives better performance, the time requirement is much higher than for CNN. To mitigate this, we propose a distributed approach called Parallel MCNN (pMCNN) which cuts down the time drastically while maintaining good performance. We have proposed two incremental algorithms using MCNN to carry out prototype selection on large and streaming data. The results of these algorithms using MCNN and pMCNN have been compared with an existing algorithm for streaming data.

Keywords: prototype selection, one-pass algorithm, streaming data, distributed algorithm.

1 Introduction

Prototype selection is the process of finding the subset of the training dataset which gives the best results when used instead of the training set for classification [2]. If we have a training dataset X=$(x_1,\theta_1),(x_2,\theta_2),...,(x_n,\theta_n)$, prototype selection choses m examples from X where $m < n$. Prototype selection is used when the training dataset is very large and/or has a large number of features. To obtain saving in space and time, m patterns are used instead of n. This may lead to some reduction in the performance of the classifier. If the training dataset has redundant patterns or outliers, then the reduced set may give better performance than the complete training dataset. This paper uses the Modified Condensed Nearest Neighbour (MCNN) [6] algorithm to carry out prototype selection. This is an order independent algorithm (unlike the Condensed Nearest Neighbour algorithm) which gives good performance. A parallel implementation of MCNN called pMCNN has been developed to mitigate the time complexity of MCNN on large datasets. We have applied this technique to carry out prototype selection on large and streaming data.

In this work, we propose an incremental prototype set selection algorithm for streaming data and large data sets, where prototypes are added in an incremental fashion to the condensed set. The algorithm produces a training set consistent subset for classification, which gives good results. After using a sequential approach with MCNN, we have used a distributed approach using a parallel implementation of MCNN called pMCNN.

The Condensed nearest neighbor rule (CNN rule) [4] is the first and the most popular prototype selection algorithm. CNN first puts one point from the training set T into a condensed set S. The other points are considered one at a time and if it does not have the same class as its nearest neighbor in C, it is added to C. This process is repeated a number of times on T till C correctly classifies all the patterns in T. CNN is *order-dependent* as the points in the condensed set S depends on selection of initial sample point and order in which data is presented to the algorithm. The Modified CNN algorithm (MCNN) first includes a representative data point from the patterns in T for each class. This may be the medoid of all the points in the dataset. Any other representative pattern can also be used. In each step, T is classified using C. A representative point is found for the misclassified points from each class and added to C. This is repeated till C classifies all the points in T correctly. Unlike CNN, MCNN is *order-independent* which means it computes the same condensed set independently of the order of data points in the training set. The Fast CNN rule[7] computes representative data points using the Voronoi cell. The Voronoi cell of point $p \in S$ set is denoted by

$$Vor(p,S,T) = \{q \in T | \forall p' \in S, d(p,q) \leq d(p',q)\}, \quad (1)$$

which means it is the set of all points of T that are closer to p than any other point p' in set S. Voronoi enemies of $p \in S$ is denoted by

$$Voren(p,S,T) = \{q \in Vor(p,S,T) | l(q) \neq l(p)\}. \quad (2)$$

Like MCNN, condensed set S is initialized with the centroid of each class. During each iteration, for each point $p \in S$, representative data point in $Voren(p,S,T)$ is selected and inserted to condensed set S. The representative data point of p in $Voren(p,S,T)$ is denoted as

$$rep(p, Voren(p,S,T)) = nn(p, Voren(p,S,T)), \quad (3)$$

which means nearest neighbor of point p is its Voronoi enemies cell. The FCNN algorithm terminates when for each data point $p \in S$, $Voren(p,S,T)$ is empty. Like MCNN, FCNN rule is also order-independent. The FCNN rule requires more iterations than CNN to compute training set consistent subset but less than the MCNN rule. They also proposed a distributed approach of this algorithm for large data sets[8]. The Structural Risk Minimization using the NN rule (NNSRM)[9] is costly as its complexity is $O(n^3)$. The Reduced nearest neighbor (RNN) rule [5] starts with the condensed set C_r having all the patterns in C as obtained by CNN. Every instance in C_r is removed if this deletion does not cause a misclassification of another instance in T. A set cover optimization problem is solved in [15] to find the prototypes. A fuzzy rough approach is used in [17] to carry out prototype selection. Multi-objective optimization and partitioning is used to carry out prototype selection in [18]. The various methods for prototype selection have been described and classified in [13]. [16] is a survey paper of prototype selection algorithms for kNN classification.

Streaming and large data sets have large space complexity and time complexity as huge data sets have to be processed. For this reason, prototype selection algorithms for large datasets and for streaming data should run in linear time and should be basically incremental algorithms. In [14], the algorithm WECU has been proposed which is an ensemble classifier used for classification of data streams. Each incoming chunk of data is used to induce an ensemble classifier. An adaptive Nearest Neighbor classification algorithm for data streams is proposed in [10]. The authors propose an incremental adaptive nearest neighbor classifier based on multi-resolution data representation to compute nearest neighbors of a point. Instance-based learning on data streams is used in [11]. An instance is simply a pattern or data with a class label. Let χ denote the instance space, where instance corresponds to the description x of an object. A labelled instance is denoted as $\langle x, \lambda_x \rangle$, where λ_x is the label of x. The method uses subset D of complete instance space χ and $\langle x_i, \lambda_{x_i} \rangle, 1 \leq i \leq n = |D|$. Then D is considered as the training set for operations on the given query. So, for a given new query x_0, it uses *k-nearest neighbor* algorithm on set D to estimate class label λ_{x_0} of x_0. The algorithm uses some deletion strategies to update D, whenever adding new instances leads to the upper bound being exceeded. A volume prototype selection algorithm for streaming data [12] uses the acceptance region to update prototypes. If the new sample data point falls into the acceptance region, then this point is added to update the prototypes. One sample point can be used to update more than one prototype. Finally, from $\acute{L}$ prototypes, it selects greedily L prototypes, where $L \leq \acute{L}$. The algorithm works in two phases. The first phase is used to compute the initial prototypes. The initial prototypes are computed according to M

times random permutation of data samples. This phase requires multiple scans of initial N data samples to get initial condensed set. The second phase requires only one-time scan of remaining data samples. By taking random permutation of samples, we can see that initial prototypes depend on different order of samples. So, initial prototypes may be different each time we apply this algorithm on the same data samples.

In this paper, prototype selection is carried out to get the condensed set for streaming data sets and large data sets. When there is streaming labeled data, at any instance, we have a condensed set depending on the data which has come in till now. This condensed set can be used to classify a new pattern, if it is required to do so at that point. As the labeled data keeps coming, the condensed set has to be modified to reflect the data coming in. This should be done in an incremental manner without having to run the entire prototype selection algorithm again on the entire training set. Also, as the data stream comes in, stays for some time and goes away, we can sample the data only once and we need a one-pass algorithm to incrementally obtain the condensed set.

Carrying out prototype selection for streaming data is an interesting and challenging problem. Our algorithm is a one-pass algorithm and computation time is linear. Unlike volume prototype selection, we do point prototype selection. Our algorithm can be used for both streaming data and large datasets. In the case of large datasets, our method helps to reduce the space and time complexity. After discussing this algorithm, it has been applied to streaming and large datasets using the serial MCNN and the distributed MCNN (pMCNN).

This paper is organized as follows. In Section 2, the methodology used in this paper for streaming and large datasets is explained. In Section 3, we explain the distributed approach of MCNN i.e. pMCNN. Section 4 gives the results of using our streaming algorithm to obtain prototypes using the MCNN and the pMCNN algorithms. Section 5 gives the conclusion.

2 Methodology

This algorithm [1] is a modification of the MCNN algorithm for prototype selection which takes into account large and streaming data. The algorithm first starts with a small amout of data points m where $m << n$. The initial condensed set is obtained by applying MCNN algorithm on these m patterns. As the data comes in as a stream, the data is handled one at a time, incrementally updating the condensed set using MCNN. A one pass algorithm is proposed by us for handling the streaming data.

2.1 MCNN Rule

MCNN rule partitions the region of each class into non-overlapping Vornoi regions. Each region has a representative pattern in the condensed set. It computes prototype samples in an incremental manner. MCNN rule starts with taking representative data points of each class of the training set into the condensed set. We can use different approaches to compute representative data point of a class. When we have Gaussian distribution, we use the centroid as the representative data point of a class. As described in Section 1, MCNN rule is order-independent. It always computes the same condensed set with any order of data points in the data set. We use MCNN with a small subset of stream points to get the initial condensed set.

Algorithm 1 Algorithm of MCNN Rule for initial phase

```
Input: $D$= training set for first phase, $cond = \phi$.
1. Initialize $typical = \phi$, $S_t = D$.
2. Compute representative point for each class of $S_t$ and add to $typical$.
3. Set $S_m = \phi$, $S_c = \phi$.
4. Classify $S_t$ with $typical$ set.
5. Add misclassified data points to $S_m$ and correctly classified to $S_c$.
if $S_m \neq \phi$ then
    Set $S_t = S_c$.
    Set $typical = \phi$.
    Go to step 3.
end if
6. Set $cond = cond \cup typical$.
7. Set $S_m = \phi$, $S_c = \phi$.
8. Classify training set $D$ with $cond$.
9. Add misclassified data points to $S_m$ and correctly classified to $S_c$.
if $S_m \neq \phi$ then
    Set $S_t = S_m$.
    Set $typical = \phi$.
    Go to step 3.
    end if
  10. Stop and $cond$ is final condensed set.
```

Algorithm 1 gives the algorithm for MCNN. Steps 3-6 iteratively computes representative samples for each class in the initial training set *D*. It uses two sets S_m and S_c, where it adds misclassified data points to set S_m and classified data points to set S_c. Steps 7-9 are used for checking classification of the training set using the current condensed set *cond*. The algorithm computes centroids as representative samples for each class of the data set. It uses nearest-neighbor rule to classify data samples using the condensed set. The initial training set *D* is much smaller in size compared to training set *T*. Here *T* is a data stream or big data set, whereas *D* is a small subset of *T*. The algorithm terminates when all data points of the training set *D* are correctly classified with the reduced prototype set *cond* which is the final condensed set. In each iteration, the algorithm incrementally adds prototypes to *cond* set.

The final condensed set of MCNN (just like for CNN) correctly classifies the complete training dataset. The condensed set, therefore, gives 100% accuracy on the training set.The MCNN algorithm taken from [6], is reproduced in Algorithm 1 for easy reference.

2.2 Prototype Selection on Streaming Data Instances

This phase of the algorithm deals with the streaming data coming one at a time. It requires only one-time scan of incoming data samples. In this phase, two methods have been used for updating the condensed set. In Method 1, data points which are misclassified by the condensed set are added to the condensed set. In Method 2, misclassified samples are collected together and every now and then, MCNN is run on these samples to get a condensed set which is added on to the original condensed set.

2.3 Algorithm

2.3.1 Method 1:

The algorithm computes the condensed set in two phases. The input for the first phase is a small subset *D* of complete training set *T*. The final condensed set computed by the initial phase is used as input in the second phase. The second phase is used to carry prototype set selection on remaining data points. In this phase, the algorithm considers each incoming data sample and incrementally adds misclassified data samples to the current condensed set.

Algorithm 2 Method 1 for Prototype selection on Streaming data set

```
  Input: $T$ = training set, $cond = \phi$.
  1. Initialize set $D=\{(x_1,c_1),(x_2,c_2),\ldots,(x_m,c_m)\}$
     dataset for first phase.
  2. Initialize set $R = T \setminus D$ remaining data set.
  3. Set $cond$ = mcnn($D$) .
  4. Classify remaining data samples.
  for all $x_i$ in $R$ do
     find neigh = nearest-neighbor($x_i$ , $cond$) .
     if $label(x_i) \neq label(neigh)$ then
        update $cond$ by adding $x_i$ sample into $cond$.
     end if
  end for
  5. Stop and $cond$ is final condensed set.
```

It requires only one-pass to get the remaining prototypes of the condensed set from the remaining data stream. It will terminate after a full scan of the training data set *T*. The algorithm of Method 1 is shown in Algorithm 2. In this algorithm, the initial subset of training set *D* is input to MCNN to compute initial condensed set. *R* is the set containing all remaining data samples after *D*. Let size of *D* be *m*, which means it contains *m* initial data samples of the data stream. So all remaining data samples x_i, where $i > m$, are data samples belonging to *R*. The condensed set *cond* computed by MCNN rule and *R* are used in the second phase of the algorithm.

In step 2, it creates the initial subset *D* of size m and set *R* is the remaining data set. In step 3, the initial condensed set using MCNN rule is found. Step 4 is used to select prototypes from set *R*. When new data sample comes, it finds its nearest-neighbor in the current *cond* set. For all data samples, if class label of data sample is different from the class label of computed neighbor data point, then it is added into *cond*. For streaming data, at any time, the condensed set existing at that point can be used for classification. For big data sets, it will terminate after a full scan of training data set T. While more number of prototypes maybe addeded to the condensed set when the streaming data is just starting, as more streaming data comes in, the number of new condensed patterns being added will decrease and will be minimal after a while.

The algorithm also computes the condensed set for big data sets in only a one-pass scan. The prototype selection algorithms for static data sets are used when we have the whole training data set. The algorithm in this paper computes the condensed set without the availability of the complete data set at the same time. To use this algorithm on big data sets, we can handle the big data set as a data streams. In this approach, data samples of the big data set will come as a sequence of data samples. It can, therefore, be seen that it handles both memory management and execution time requirements in the case of big data.

2.3.2 Method 2:

The algorithm described in this Section results in a smaller condensed set than Method 1. On the other hand, it takes more time to compute the final condensed set than Method 1. As the data stream comes in, if a data point is misclassified using *cond* these are noted. Every now and then, MCNN is used on this set of points. The condensed set that results is added on to *cond*. The algorithm of Method 2 is shown in Algorithm 3.

Algorithm 3 2 for Prototype selection on Streaming data set

```
Input: $T$= training set, $cond = \phi$.
1. Initialize set $D$=$\{(x_1,c_1),(x_2,c_2),\ldots,(x_m,c_m)\}$
   dataset for first phase.
2. Initialize set $R = T \setminus D$ remaining data set.
3. Set $cond$ = mcnn($D$) .
4. Set $counter = 0$.
5. Classify remaining data samples.
for all $x_i$ in $R$ do
    Increment $counter$ by one.
    Find neigh = nearest-neighbor($x_i$ , $cond$) .
    if $label(x_i) \neq label(neigh)$ then
        Update typical by adding $x_i$ sample into typical.
    end if
    if $counter == L$ then
        Set $counter = 0$.
        $cond = cond \cup$ mcnn($typical$).
        $typical = \phi$.
    end if
end for
6. Stop and $cond$ is final condensed set.
```

The main difference between Method 1 and Method 2 occurs in step 5. In this algorithm, the data coming in one by one is classified using *cond* and if it is misclassified,it is stored in *typical*. When L data points have been analyzed, the misclassified data points *typical* are used and MCNN algorithm is run on these data points. The condensed set which results is added on to *cond*. It classifies each incoming data sample using the current condensed set *cond* as in Method 1. For each data sample, if it is misclassified, it is added to *typical* set and the *counter* is incremented by one. When *counter* is equal to L, it applies MCNN rule on set *typical* and sets *counter* value to zero. It adds the condensed set computed from *typical* to the current condensed set and sets $typical = \phi$. As in Method 1, it computes the final condensed set after a full scan of the data set.

The approach used in Method 2 computes a smaller condensed set than Method 1 but it takes more time to converge. The streaming data sets have strict time constraints. Due to fast execution, Method 1 gives better results for data streams to achieve time constraints. If we want to get a smaller size condensed set, Method 2 will give better results by computing a smaller condensed set *cond*. We can use one of the methods according to our requirements.

In streaming data sets, whenever we want to have a condensed set for classification, we use the current condensed set as the final condensed set. During classification of the testing data set, the algorithm will use this condensed set instead of the complete training data set. It takes some time for condensation but after condensation, classification requires less time using the condensed set.

In this algorithm, as the streaming data keeps coming, the number of misclassified samples comes down and the number of prototypes being added to the condensed set will decrease and finally will be very minimal.

3 Distributed approach for prototype selection

The sequential algorithm gives efficient and reduced condensed set but requirements increase when dataset size increases. In this part, we introduce a distributed algorithm for prototype selection to handle huge data sets and to achieve better time and memory bounds.

The training set T is divided into equally sized disjoint training set blocks on each node. Here node is a processor of distributed architecture. Each node has its own training set and computes the condensed set of that training set in parallel.

Figure 1 shows the distributed architecture for prototype selection. The training dataset is divided on p nodes. Here c is the total number of classes in the dataset. One node is the root node which handles task division and gathers results from all other nodes to get the final result. Here node id ‘0’ is the root node.

Initially our algorithm finds the centroid of each class using all nodes in parallel. As we explained in our proposed sequential algorithm, initially the condensed set starts by adding the centroid of each class from the initial training set $T^{'}$ into this condensed set.

The algorithm 4 is used to compute the condensed set from the initial training set with the distributed approach. In step 1, $S^i[j]$ indicates sum of all elements of class label j on node id i. $T^i[j]$ indicates number of elements of class label j on node id i. After calculating $S^i[j]$ and $T^i[j]$, all nodes send both variables to root node.

In step 2, the root node gathers all data from different nodes and computes the center point of each class using both these variables. The root node computes $S[j]$ and $T[j]$ for each class label j. Here $S[j]$ is the total sum of all elements of class j in the initial training set. $T[j]$ represents the total number of elements of class j. It then finds $c[j]$ which is the center point for every class j. The root node does all this computation and broadcasts the result to all nodes.

In step 3, all nodes have the center point of each class. In this step, all nodes compute the local centroid $C^i[j]$ in their own training set block. Here local centroid point $C^i[j]$ is the nearest neighbor of center point $c[j]$ in class j in the local training set block T.

In step 4, the root node computes the global centroid point of each class. In this step, it assigns the nearest neighbor of center point $c[j]$ among all local centroid points $C^i[j]$ as the global centroid point for class label j. The algorithm adds all global centroid points of each class in the condensed set ΔS and broadcasts ΔS to all nodes. All nodes will start with this condensed set ΔS.

In step 5, ΔS is added to the set ΔS^i at all nodes (for node with id i, its local condensed set is denoted by ΔS^i).

In step 6, for each node, initial training set block is classified with the condensed set ΔS^i. All misclassified data points are added to set *typical* . If set *typical* is not empty then apply step 1-5 again on the *typical* set to compute prototypes from this set.

In step 7, the root node collects ΔS^i of every node i and merges them into the condensed set S. The condensed set S is the condensed set computed from the initial training set. The root node broadcasts the condensed set S to all nodes. The condensed set computed by distributed approach for initial training set is the same as the condensed set computed by our sequential Method 1.

In algorithm 5, for the remaining incoming data points, on each node, the data points are classified with condensed set S and the misclassified data points are added to the condensed set S^i. In the streaming data set, whenever we want to have a condensed set for classification, it will merge all the condensed sets S^i into the final condensed set S on the root node. During classification of test data, the algorithm will use this condensed set instead of the complete training data set.

Table 2. Datasets used in experiments

Dataset	No. of Samples	No. of features	No. of classes
Optical dig. rec.	5,620	64	10
Pen dig. rec.	10,992	16	10
Letter image rec.	20,000	16	26
Gisette	15,300	5000	2
Forrest cover type	581,012	54	7

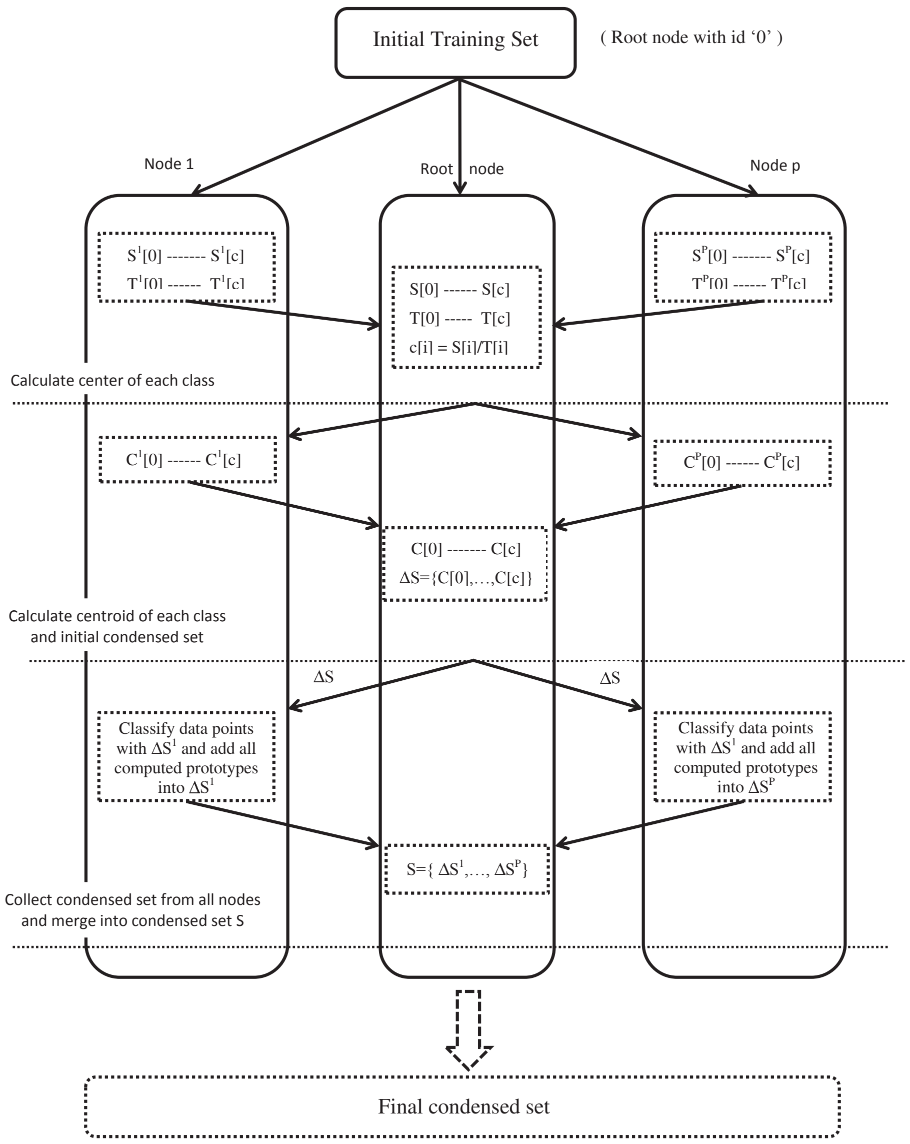
Initial Training Set
(Root node with id '0')
Node 1
Root node
Node p
S¹[0] ------- S¹[c]
T¹[0] ------ T¹[c]
S[0] ------ S[c]
T[0] ----- T[c]
c[i] = S[i]/T[i]
Sᴾ[0] ------- Sᴾ[c]
Tᴾ[0] ------ Tᴾ[c]
Calculate center of each class
C¹[0] ------ C¹[c]
Cᴾ[0] ------ Cᴾ[c]
C[0] ------- C[c]
ΔS={C[0],…,C[c]}
Calculate centroid of each class and initial condensed set
ΔS
ΔS
Classify data points with ΔS¹ and add all computed prototypes into ΔS¹
Classify data points with ΔS¹ and add all computed prototypes into ΔSᴾ
S={ ΔS¹,…, ΔSᴾ}
Collect condensed set from all nodes and merge into condensed set S
Final condensed set

Figure 1. Distributed architecture for prototype selection.

Table 1. Accuracy (%) obtained on streaming data

Dataset	kNN	CNN	MCNN	WECU	Method 1	Method 2
Optical recog	97.99	93.84	92.65	91.60	93.71	91.04
Pen digit recog	97.89	90.94	89.37	89.37	95.43	91.57
Letter recog	95.29	91.29	91.46	88.97	90.06	89.02
Gisette	96.20	88.91	87.20	89.90	91.70	90.30
Forest cover	84.07	73.55	71.59	72.78	75.04	71.52

Algorithm 4 Distributed approach to find condensed set from initial training set

```
Input: $T^{`}$ = initial training set block, condensed set $\Delta S^i = \phi$, node id $i$.
1.
for all $x_i$ in $T^{`}$ do
   if ( $label(x_i) == j$ ) then
      Calculate sum $S^i[j]$ of all elements of this class $j$.
      Calculate number of elements $T^i[j]$ of this class $j$.
      Send $S^i[j]$ and $T^i[j]$ to root node.
   end if
end for
2.
if ( node id $i == 0$ ) then
   for all class label $j$ do
      S[j] = calculate total sum of ($S^1[j], S^2[j], \ldots, S^p[j]$).
      T[j] = calculate total sum of ($T^1[j], T^2[j], \ldots, T^p[j]$).
      c[j] = S[j]/T[j]. Broadcast c[j] to all nodes.
   end for
end if
3.
for all class label $j$ in $T^{`}$ do
   Compute local centroid point $C^i[j]$, which is nearest neighbor of c[j] in this class.
   Send $C^i[j]$ to root node.
end for
4.
if ( node id $i == 0$ ) then
   for all class label $j$ do
      C[j] = nearest neighbor in ($C^1[j], C^2[j], \ldots, C^p[j]$).
      Add C[j] to initial condensed set $\Delta S$.
   end for
   Broadcast $\Delta S$ to all nodes.
end if
5. Set $\Delta S^i = \Delta S^i \cup \Delta S$ and $typical = \phi$.
6. Classify all data points of $T^{`}$ with condensed set $\Delta S^i$ ( for node $i$, its condensed set is denoted by $\Delta S^i$).
for all $x$ in $T^{`}$ do
   Find neigh = nearest-neighbor($x$ , $\Delta S^i$) .
   if $label(x) \neq label(neigh)$ then
      Update $typical$ by adding $x$.
   end if
end for
if $typical \neq \phi$ then
   Set $T^{`} = typical$.
   Go to step 1.
end if
7.
if ( node id $i == 0$ ) then
   Collect $\Delta S^i$ of every node $i$ and merge them into condensed set $S$.
   Broadcast $S$ to all nodes.
end if
```

Algorithm 5 Distributed approach to find condensed set from remaining incoming data points

```
Input: $R^{`}$= Incoming training set block, condensed set $S^i = \phi$, node id $i$.
1. For remaining incoming data points from $R^{`}$:
Classify data points with condensed set $S^i$ ( for node with id $i$, its condensed set is denoted by $S^i$).
for all $x$ in $R^{`}$ do
  Find neigh = nearest-neighbor($x$ , $S^i$) .
  if $label(x) \neq label(neigh)$ then
    update $S^i$ by adding $x$ into it.
  end if
end for
2.
if ( node id $i == 0$ ) then
  Collect $S^i$ of every node $i$ and merge them into final condensed set $S$.
end if
```

Table 3. Time taken for condensation process in seconds

Dataset	CNN	MCNN	Method 1	Method 2
Optical recog	74.53	107.56	31.35	38.41
Pen digit recog	78.88	121.39	47.37	58.98
Letter recog	781.82	1142.12	410.57	574.35
Gisette	2065.74	3036.46	1426.95	1823.38
Forest cover	23849.17	32513.98	12943.99	17963.45

Table 4. Total execution time in seconds

Dataset	KNN	CNN	MCNN	WECU	Method 1	Method 2
Optical recog	219.37	108.54	123.48	103.16	51.40	57.02
Pen digit recog	367.25	113.99	151.39	109.37	84.04	96.46
Letter recog	1881.05	1088.75	1258.75	835.05	587.91	746.05
Gisette	8592.91	3261.94	5136.13	2409.37	1732.53	2318.86
Forest cover	67485.97	39481.85	45541.17	23109.37	17489.35	20942.11

Table 5. Size of dataset for initial phase

Dataset	Method 1	Method2
Optical recog	260	260
Pen digit recog	230	230
Letter recog	702	702
Gisette	978	978
Forest cover	12922	12922

4 Results

The data sets used for our experiments are given in Table 2. The last data set, Forest cover type data set is a large data set having 581,012 data points. It is about predicting the forest cover type from given observations. This set has been used to show results on a large dataset. The letter recognition data set is used to classify black-and-white rectangular pixel displays in one of the 26 capital letters of the English alphabet. The other two datasets are used for handwritten digits recognition. These are the Optical recognition of handwritten digits data set and the Pen-based recognition of handwritten digits data set. GISETTE dataset is related to a handwritten digit recognition problem. It is used to separate the digits '4' and '9' which are generally confused for each other. This dataset is one of five datasets of the NIPS 2003 feature selection challenge. The results of using Method 1 and Method 2 is compared with results using WECU, MCNN and CNN. It is also compared with results using kNN where no condensation is carried out.

Table 1 shows the classification accuracy obtained using the prototype set obtained by all the methods. Figure 2 gives the comparison of classification accuracy on different datasets for all methods. Table 3 shows the time taken for the condensation method using Method 1 and Method 2 and also using CNN and MCNN. Table 4 gives details of the total execution time for the various algorithms.

From Table 1, the classification accuracy obtained by using k-nn rule on the complete dataset is the highest accuracy. There is no condensation of the dataset and the entire training set is used for classification. The execution time given in Table 4 for this approach, is the overall time it takes to classify the test data set using the entire training data set. As expected this takes the most time.

In Table 1, the classification accuracy obtained using CNN and MCNN is then reported. Both these approaches require the complete data set simultaneously to compute the condensed set from the training dataset. The given total execution time from Table 4 for both approaches is the total time taken for computation of condensed set and classification of test data using that condensed set. The condensation time given in Table 3 is the time taken to carry out the condensation process with a particular approach.

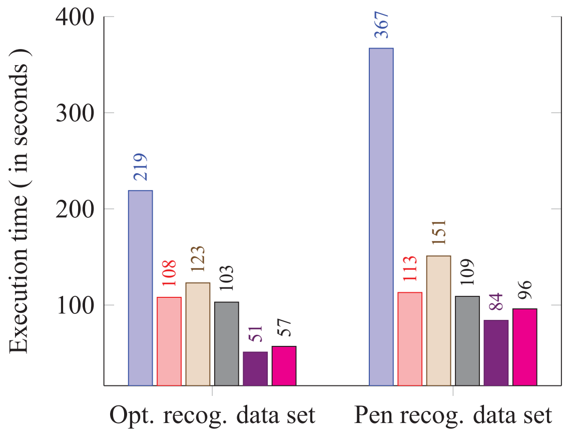

Figure 3. Execution time comparison of all six algorithms for Optical recog. data set and Pen recog. data set

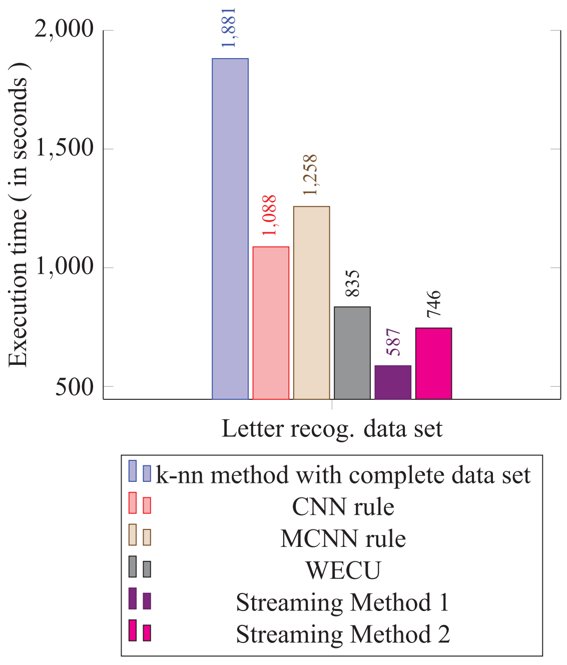

Figure 4. Execution time comparison of all six algorithms for Letter recog. data set

Table 6. Size of condensed set

Dataset	CNN	MCNN	Method1	Method 2
Optimal recog	306	68	217	143
Pen digit recog	313	45	256	189
Letter recog	1532	189	708	504
Gisette	195	71	152	108
Forest cover	31204	18809	24763	19045

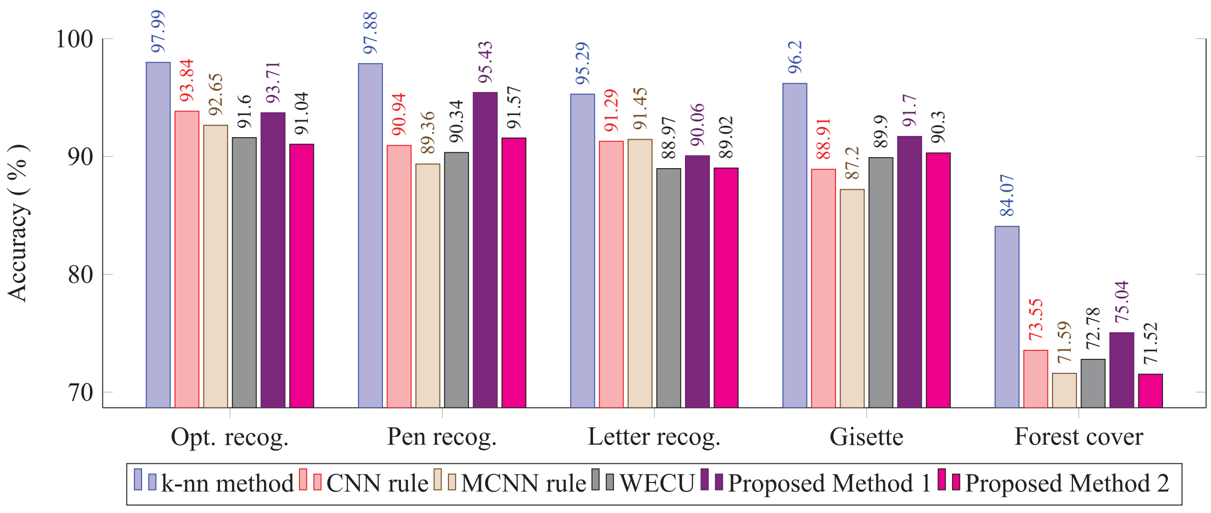

Figure 2. Accuracy comparison of both proposed methods and existing methods with different data sets

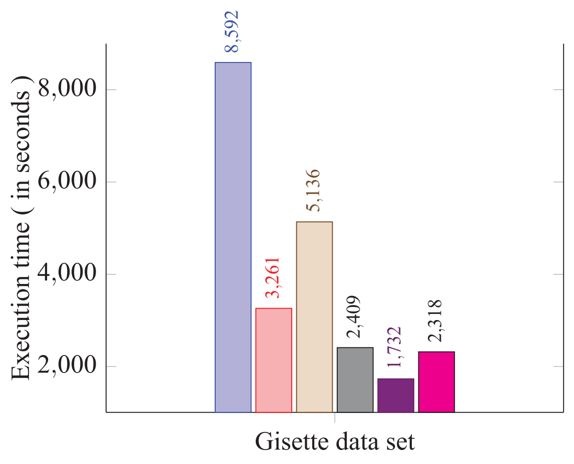

Figure 5. Execution time comparison of all six algorithms for Gisette data set

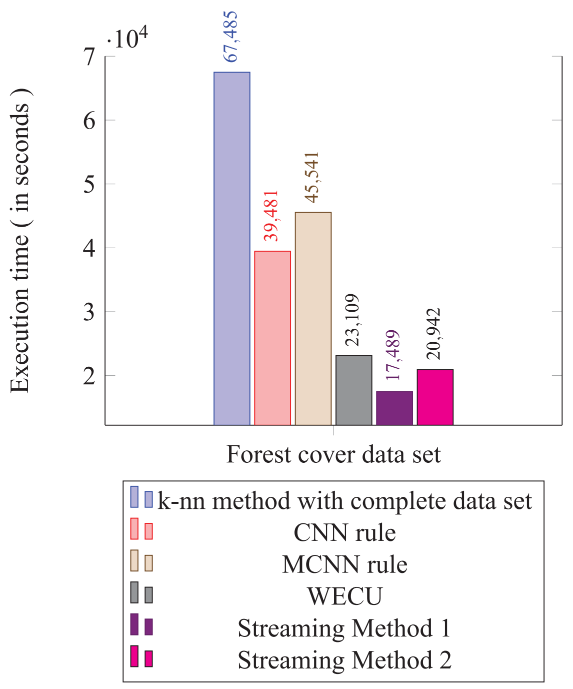

Figure 6. Execution time comparison of all six algorithms for Forest cover type data set

Table 1 shows performance results of WECU (Weighted Ensemble with one-class Classification based on Updating of data chunk) approach used by Czarnowski and Jedrzejowicz [14], where an ensemble classifier is used for classification of data streams. It uses 10% of original dataset as threshold to use data chunks at a particular time.

Table 1 also shows the classification accuracy obtained using Method 1 and Method 2 which are proposed in this paper. The total execution time in both rows is the sum of time taken by the algorithm to compute the condensed set and in classification using this computed condensed set. Both methods have some initial data points as the dataset for initial phase. Table 5 gives the number of initial data points used in Method 1 and Method 2. Both proposed methods start with MCNN rule on the initial data points and then proceed with remaining data points. The condensed set produced by the proposed methods give better classification accuracy on most datasets than other approaches since it includes all important required data points in condensed set with respect to their distribution.

From Table 5, it can be seen that for the optical recognition of handwritten digits dataset, the proposed methods uses 260 data samples from the training data set as the dataset for the initial phase. Method 1 gives better results than Method 2 but has more data points in the final condensed set. We can see that in these two methods, the difference in accuracy is 1-2%. The size of the condensed set by Method 1 is 217 but by Method 2, it is 143. In Method 2, step 6 shows that it does not add misclassified data points in the condensed set directly. It again applies MCNN rule on the computed set of misclassified data points. It reduces the size of the overall condensed set. On the other hand, Method 2 takes more time for the condensation process than Method 1. It happens again due to applying MCNN rule iteratively. Method 1 gives better accuracy than WECU approach, CNN rule and MCNN rule on the complete data set andMethod 2. WECU approach requires more time than both proposed methods. On this data set, MCNN rule gives 92.65% and Method 1 gives 93.71% accuracy.

Experimental results show that Method 1 with Pen-based recognition of handwritten digits dataset gives an accuracy of 95.43% which is very close to the accuracy given when the complete training dataset is used for classification as shown in Table 1. WECU approach gives 90.94% accuracy which is less than both the proposed methods and also requires more execution time. MCNN rule also gives less accuracy than both proposed methods. With this data set, the algorithm first uses 230 data samples as the initial training dataset to compute initial condensed set. Both generate condensed sets which are very small in size. The condensed set computed by Method 1 contains only 256 data samples from the training dataset. Method 2 gives a condensed set which have 189 data points from the training dataset. As shown in the results, both methods have a small execution time for condensation process.

After condensation, it used this smaller size condensed set for classification of test data which also takes very less time. So both methods takes very less overall execution time compared to WECU, CNN rule and MCNN rule. We can see from Table 3 that condensation process of Method 1 is three times faster than condensation process of MCNN rule. The proposed methods need a one-pass scan of the whole training set, which reduces execution time and gives faster algorithms.

From Table 5, it can be seen that the proposed methods use 702 data samples of training dataset as the initial dataset for letter recognition data set. This data set has 26 class labels. With this data set, the proposed Method 1 produces condensed set of size 708 from the total training set. It gives accuracy around 90% with this dataset. Method 1 gives classification accuracy of 90.06% which is more than Method 2 which gives 89.02% accuracy. WECU approach gives 88.97 % accuracy. Experimental results show that MCNN rule gives better accuracy of 91.456% on the letter recognition dataset which requires more execution time and entire training dataset on the disk space at the same time.

Gisette dataset has 5000 attributes. It gives 91.70% accuracy with the proposed Method 1, which used a condensed set for classification. The condensed set computed by Method 1 has only 152 data points. Method 1 also requires very less execution time compared to all other methods.

To investigate advantages of the proposed methods, we conducted experiments on a large data set. We used Forest Cover type dataset which has 581,012 data samples. From Table 4, classification of test data with the whole training set takes a very long time to execute. CNN and MCNN rule on this big dataset also takes too much execution time. During condensation process, it iteratively works on the misclassified data samples from the whole training dataset which takes a long time for large datasets. The proposed methods used 12922 data samples as the initial training dataset to compute initial condensed set. After computing the condensed set, both methods require a one-pass scan of the remaining data samples which requires less execution time. The condensed set computed by Method 1 have more data samples than Method 2 but gives more accuracy than Method 2. Method 1 gives 75.04% classification accuracy with the computed condensed set. Method 2 gives an accuracy which is very close to the accuracy given by MCNN rule which is around 71%.

From the experimental results, we can see that the proposed methods give better results than the results obtained by ANNCAD approach[10] for both letter recognition and forest cover datasets.

Experimental results in Table 1 show that the proposed methods gives good accuracy with less execution time. On every data set, the proposed methods compute very small condensed sets as can be seen in Table 6. The results show that after condensation, the algorithms use the condensed set and also give good accuracy. We can observe from the results that both the proposed methods give better results than the existing WECU approach for data streams and also with less execution time and memory space. It is evident in all five datasets, that there is a huge time saving . Also, the entire dataset do not need to be stored. Only the initial training set and the condensed set formed needs to be stored.

To give an idea of the huge savings in time, Figure 3, Figure 4, Figure 5 and Figure 6 give the time taken by the various algorithms for all the datasets for sequential algorithms.

Table 7 shows the comparison between distributed approach and sequential approach of our algorithm. The distributed approach uses the same approach as our sequential approach for MCNN. The calculation of the centroid and the closest point to the centroid is done using a number of processors leading to saving time. It can be seen from Table 7 that the size of the condensed set and the classification accuracy in the distributed method is almost the same as for the sequential method. We can observe that the total execution time is reduced with respect to higher number of parallel nodes. It gives better results when we have large datasets. From Table 7, for forest cover dataset, sequential streaming Method 1 takes around 120 minutes but distributed algorithm takes around 17 minutes only. The distributed approach reduces the time complexity with almost the same classification accuracy.

Table 7. Experimental results for distributed approach on different datasets

Datasets				
	Algorithms	Letter recog.	Gisette	Forest cover
Total Execution Time (in seconds)	Method 1	2.70	260.82	7346.74
	Distributed Method (p=2)	1.58	140.19	3848.48
	Distributed Method (p=4)	1.06	77.86	2135.54
	Distributed Method (p=8)	.32	34.61	997.83
Size of Condensed Set	Method 1	708	152	24763
	Distributed Method (p=2)	861	225	25300
	Distributed Method (p=4)	803	247	26162
	Distributed Method (p=8)	846	239	26034
Classification Accuracy (%)	Method 1	90.06	91.70	75.04
	Distributed Method (p=2)	88.53	90.50	72.16
	Distributed Method (p=4)	89.31	90.47	73.28
	Distributed Method (p=8)	88.79	90.68	72.37

5 Conclusion

In this paper, we propose two prototype selection methods for data compression on data streams. The algorithms give good results for both streaming datasets and large datasets. We do not need to store the complete training dataset in memory at the same time. Streaming data comes as a sequence of data points with infinite length. The proposed methods do not require the complete dataset at a particular time, therefore they handle memory requirements and strict time constraints for streams. It has also been shown how for large data by dividing it into chunks or using one pattern at a time, this algorithm can be used leading to considerable saving in time and space complexity.

We have discussed both proposed methods with regard to accuracy and execution time and also compared with existing approaches. Method 1 requires less execution time and Method 2 computes smaller condensed set. Both methods have their own advantages.

We also propose a distributed approach for our sequential streaming method. The distributed approach for prototype selection reduces time complexity and gives good classification accuracy. The benefits of distributed approach over sequential method has been shown by comparing results of both approaches.

Future work includes doing more experimentation on larger datasets and streaming data both for sequential algorithms as well as the distributed framework.

References

[1] Lakhpat Meena and V. Susheela Devi, Prototype Selection on Large and Streaming Data, International Conference on Neural Information Processing(ICONIP 2015), 2015.

[2] M. Narasimha Murty and V. Susheela Devi, Pattern Recognition: An Algorithmic Approach, Springer and Universities Press, 2012.

[3] T.M. Cover, P.E. Hart, Nearest neighbor pattern classification, IEEE Trans. on Information Theory, IT-13: 21-27, 1967.

[4] P.E. Hart, The condensed nearest neighbor rule. IEEE Trans. on Information Theory, IT-14(3): 515-516, 1968.

[5] G.W. Gates, The reduced nearest neighbour rule, IEEE Trans. on Information Theory, IT-18 (3): 431-433, 1972

[6] V. Susheela Devi, M. Narasimha Murty. An incremental prototype set building technique, Pattern Recognition, 35: 505-513, 2002.

[7] F. Angiulli, Fast Condensed Nearest Neighbor Rule, Proc. 22nd International Conf. Machine Learning (ICML '05), 2005

[8] Angiulli, Fabrizio, and Gianluigi Folino, Distributed nearest neighbor-based condensation of very large data sets, Knowledge and Data Engineering, IEEE Transactions on 19.12, 2007, 1593-1606, 2007.

[9] B. Karacali and H. Krim, Fast Minimization of Structural Risk by Nearest Neighbor Rule, IEEE Trans. Neural Networks, vol. 14, no. 1, pp. 127-134, 2003.

[10] Law, Yan-Nei and Zaniolo, Carlo, An adaptive nearest neighbor classification algorithm for data streams, In Knowledge Discovery in Databases: PKDD 2005, pp. 108120, Springer, 2005.

[11] J. Beringer, E. Hüllermeier, Efficient instance-based learning on data streams, Intelligent Data Analysis, 11 (6) 627-650, 2007

[12] K. Tabata, Maiko Sato, Mineichi Kudo, Data compression by volume prototypes for streaming data, Pattern Recognition, 43: 3162-3176, 2010

[13] Salvador Garcia, Joaquin Derrac, Prototype selection for nearest neighbor classification: Taxonomy and Empirical study, IEEE Trans. on PAMI, 34: 417-435, 2012.

[14] Ireneusz Czarnowski, Piotr Jedrzejowicz, Ensemble classifier for mining data streams, 18th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems(KES 2014), Procedia Computer Science, 35: 397-406, 2014.

[15] Jacob Bien, Robert Tibshirani, Prototype selection for interpretable classification, Annals of Applied Statistics, Vol. 5, No. 4, 2403-2424, 2011.

[16] Shikha V. Gadodiya, Manoj B. Chandak, Prototype selection algorithms for kNN Classifier: A Survey, International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), Vol. 2, Issue 12, pp. 4829-4832, 2013.

[17] Nele Verbiest, Chris Cornelis, Francisco Herrera, FRPS: A fuzzy rough prototype selection method, Vol. 46, Issue 10, 2770-2782, 2013.

[18] Juan Li, Yuping Wang, A nearest prototype selection algorithm using multi-objective optimization and partition, 9th International Conference on Computational Intelligence and Security, 264-268, 2013.

V. Susheela Devi is a lecturer at the Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India. She obtained her Ph.D. degree in 2000 from Indian Institute of Science, Bangalore. She works in the fields of pattern recognition, machine learning and soft computing.

Lakhpat Meena completed his Master of Engineering degree from the Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India. He worked on algorithms for large and streaming data and parallel computation.