Prototype Selection for Nearest Neighbor Classification: Survey of Methods

Salvador García, Joaquín Derrac, José Ramón Cano, and Francisco Herrera

Abstract

Prototype selection is a research field which has been active for more than four decades. As a result, a great number of methods tackling the prototype selection problem have been proposed in the literature.

This technical report provides a survey of the most representative algorithms developed so far. A widely used categorization (edition, condensation and hybrid methods) has been employed to present them and describe their main characteristics, thus providing a first insight into the prototype selection field which may be useful for every practitioner who needs a quick reference about the existing techniques and their particularities.

Index Terms

Prototype selection, nearest neighbor, taxonomy, condensation, edition, classification.

I. Introduction

The nearest neighbors classifier is one of the most used and known techniques for performing recognition tasks. It has also demonstrated to be one of the most interesting algorithms in the data mining field in spite of its simplicity. However, the nearest neighbors classifier suffers from several drawbacks; such as high storage requirements, low efficiency in classification responde and low noise tolerance. These weaknesses have been the subject of study of many researchers and many solutions have been proposed. This technical report provides a survey of the prototype selection methods proposed in the literature.

II. Survey of Instance Selection Algorithms

In the instance selection problem, let's assume that there is a set T, composed by p instances. This set is divided in training (TR, composed by n instances) and test set (TS, composed by t instances). Each instance x_i from T consists of pairs $(x_{i1}, x_{i2}, ..., x_{id})$ and w which defines the corresponding class label. T contains p instances, which have d input attributes each one and they should belong to one of the Ω classes. Let $S \subseteq TR$ be the subset of selected instances resulted for the execution of the instance selection algorithm.

Algorithms for instance selection may be classified in three type groups: edition, condensations algorithms and hybrids.

J. Derrac and F. Herrera are with the Department of Computer Science and Artificial Intelligence, CITIC-UGR (Research Center on Information and Communications Technology), University of Granada, 18071 Granada, Spain.
E-mails: jderrac@decsai.ugr.es, herrera@decsai.ugr.es

S. García and J.R. Cano are with the Department of Computer Science, University of Jaén, 23071, Jaén, Spain.
E-mails: sglopez@ujaen.es, jrcano@ujaen.es

III. Condensation Algorithms

This set includes the techniques which aim to retain the points which are closer to the decision boundaries, also called border points.

Considering their search direction they can be classified as:

A. Incremental

- **Condensed Nearest Neighbor Rule (CNN)** [1] - This algorithm finds a subset S of the training set TR such that every member of TR is closer to a member of S of the same class than to a member of S of a different class. It begins by randomly selecting one instance belonging to each output class from TR and putting them in S. Then each instance in TR is classified using only the instances in S. If an instance is misclassified, it is added to S, thus ensuring that it will be classified correctly. This process is repeated until there are no instances in TR that are misclassified. This algorithm ensures that all instances in TR are classified correctly, though it does not guarantee a minimal set.
- **Ullman algorithm (Ullmann)** [2] - This method tries to improve the CNN and RNN algorithms by introducing two new concepts: Firstly, a new parameter is introduced to the computation of the similarity measure, γ. It helps to obtain smoother boundaries between the prototypes selected in S, because instances in TR are only considered misclassified if their distance to a wrong class prototype is lesser than the distance to a correct class prototype plus γ.

  The second improvement consist on a new iterative method to find the best subset S. It employs a binary nxn matrix (n is the number of prototypes in TR), to represent which instances are currently the nearest neighbors of the same class of each instance. Then, the method starts with S = empty and adds to S only those instances which leads to correct classification of its nearest neighbors and lies further than it from any member of any other class, working through all the positions of the matrix.
- **Tomek Condensed Nearest Neighbor (TCNN)** [3] - Tomek presents two modifications based on the CNN algorithm. The method 1 is similar to CNN, but, when an instance x_i is misclassified (because its nearest neighbor in S, s is from the opposite class), instead of adding it to S the method finds the nearest neighbor of s which is a member of the same class of x_i, and adds it to S.

  The method 2 is also a modification of CNN, where instead of use instances in TR to build S, only a subset of TR, C is employed. C is composed by the instances of TR which have the same class as its nearest neighbors.
- **Mutual Neighborhood value Algorithm (MNV)** [4] - This algorithm is a modification of CNN, where the MNV measure is employed. The MNV between two instances, x_i and $x_i^{'}$ is based on how many times x_i is present on the neighborhood of the k-nearest neighbors of $x_i^{'}$, and, respectively, how many times x_i is present on the neighborhood of the k-nearest neighbors of $x_i^{'}$. MNV algorithm perform these steps:

  Stage 1:

  1) For each sample x_i of the training set TR, find the nearest neighbor x_j from the opposite class. Subsequently, with respect to x_j, considering only samples belonging to the opposite class, find the

nearest neighbor rank R of x_i. Now the MNV of x_i with respect to x_j is MNV(x_i,x_j) = 1+R. This value of MNV will be associated with the sample x_i alone and not with x_j. Also record the Euclidean distance *d* between x_i and x_j, and associate it with x_i. Samples that are near the decision boundary will have low values of MNV and *d*.

2) Using the results from 1), order the n samples according to MNV in ascending order. If the MNVs of some of the samples are identical, order such samples according to distances *d*, in ascending order. Store this ordered set in ORDER.
3) The first sample of ORDER is placed in STORE.
4) The next sample in ORDER is classified by the NN rule, using the samples that are present in STORE. If the classification is wrong, add that sample to STORE.
5) Step 4) is repeated till all the samples in ORDER are tested.
6) After one pass through ORDER, apply steps 4) and 5) to the samples retained in ORDER. Repeat this procedure until there are no transfers of samples from ORDER to STORE in a pass. The present contents of STORE constitute the first modified condensed training set.

Stage 2:

When the deletion of a sample in the condensed subset produces no change in the classification of any member of the complete training set, the deleted sample may be excluded from the condensed set. This idea is used to make a further reduction in the number of samples constituting the modified condensed set.

7) A sample x_k of STORE (on completion of step 6)) is placed in SIFT.
8) All the samples in ORDER are classified by the NN rule using the samples that are now present in STORE. If there is any misclassification, transfer Z back to STORE, else retain it in SIFT.
9) Steps 7) and 8) are repeated for all the samples in STORE. The final contents of STORE constitute the second modified condensed training set.

- **Modified Condensed Nearest Neighbor (MCNN)** [5] - This algorithm is similar to CNN but, instead of adding a instance to the set S when it is misclassified, it flags all the instances misclassified and, when all the instances in TR have been tested, a representative example of each class is added to S, generating it as the centroid of the misclassified examples in each class. The process is conducted iteratively until no instance in TR is misclassified.
- **Generalized Condensed Nearest Neighbor (GCNN)** [6] - The GCNN algorithm tries to improve the CNN algorithm. Firstly, the initial prototypes are selected as the most voted from each class (considering a vote as to be the nearest instance to other of the same class). Then, the CNN rule is applied, but a new instance x is considered classified correctly only if its nearest neighbor x_i in S is from its same class, and the distance between x and x_i is lower than $dist$, where $dist$ is the distance between x and its nearest enemy in S.

- **Fast Condensed Neighbor algorithms family (FCNN)** [7] - The **FCNN1** algorithm starts by introducing in S the centroids of each class. Then, for each prototype p in S, its nearest enemy inside its Voronoi region is found, and add to S. This process is performed iteratively until no enemies are found on a single iteration.
  Fast Condensed Nearest Neighbor 2 (**FCNN2**): The FCNN2 algorithm is similar to FCNN1 but, instead of adding the nearest enemy on each Voronoi region, is added the centroid of the enemies found in the region.
  Fast Condensed Nearest Neighbor 3 (**FCNN3**): The FCNN3 algorithm is similar to FCNN1 but, instead of adding one prototype per region in each iteration, only one prototype is added (the one which belongs to the Voronoi region with most enemies). In FCNN3, S is initialized only with the centroid of the most populated class.
  Fast Condensed Nearest Neighbor 4 (**FCNN4**): The FCNN3 algorithm is similar to FCNN2 but, instead of adding one prototype per region in each iteration, only one centroid is added (the one which belongs to the Voronoi region with most enemies). In FCNN4, S is initialized only with the centroid of the most populated class.
- **Prototype Selection based on Clustering (PSC)** [8] - To build the S set, the PSC first employs the C-Means algorithm to extract clusters from the set TR of training prototypes. Then, for each cluster c, if it is homogeneous (all prototypes belongs to the same class), then its centroid is added to S. If it is not homogenous, then their majority class c_m is computed, and every instance which do not belongs to c_m in the cluster is add to S, along with its nearest neighbor in class c_m.

B. Decremental

- **Reduced Nearest Neighbor (RNN)** [9] - RNN starts with S = TR and removes each instance from S if such a removal does not cause any *other* instances in TR to be misclassified by the instances remaining in S. It will always generate a subset of the results of CNN algorithm.
- **Selective Nearest Neighbor (SNN)** [10] - SNN extends CNN such that every member of TR must be closer to a member of S of the same class than any member of TR (instead of S) of a different class. Further, the method ensures a minimal subset satisfying theses conditions.
  It starts by constructing a binary nxn matrix A (where n) is the number of instances in TR), where A_{ij} is set to 1 when instance *j* is of the same class as instance i, and it is closer to instance i than i's nearest *enemy*, i.e., the nearest neighbor of i in TR that is of a different class than i. A_{ii} is always set to 1.
  Once this array is set up, the following 5 steps are taken until no columns remain in the array:
  1) For all columns i that have exactly one bit on, let j be the row with the bit on in column i. All columns with a bit on in row j are removed, row j is removed, and instance j is added to S.
  2) For all rows j, delete row j if for all (remaining) columns i and for some (remaining) row k, $A_{ji} \leq A_{ki}$. In other words, row j is deleted if for some other row k, whenever row j contains a 1, row k also contains a 1. In this case instance j is *not* added to S.
  3) Delete any column i if for all (remaining) rows j and some (remaining) column k, $A_{ji} \geq A_{jk}$. In other

words, columns i is deleted if there is some other columns k that has zeroes in every row that column i does. Again instance i is *not* added to S.

4) Continue to repeat steps 1-3 until no further progress can be made. If no columns remain in the array, the S is complete and the algorithm is finished. Otherwise, go on to step 5.
5) Find the row j that when included in S requires the fewest other rows to also be included in S. This is done as follows:
   a) For each remaining row j, assume that instance j will be added to S, and that row j and any (remaining) columns with a bit on in row j will be deleted (but do not actually remove row j or the column yet). Subject to this assumption, find the fewest number of additional rows it would take to get at least as many 1's as there are remaining columns. From the minimum found for each row j, keep track of the absolute minimum found by any row j.
   b) For each row j in (a) that resulted in the absolute minimum number of additional rows that *might* be needed, actually remove j and columns with bits on in row j and call the algorithm recursively beginning with step 1. If the minimum number of rows was really used, the add j to S and stop: S is complete. Otherwise, restore row j and the removed columns, and try the next possible row j.
   c) If no row j is successful in achieving the minimum number, increment the absolute minimum and try (b) again until successful.

- **Shrink (Shrink)** [11] - This algorithm starts with S = TR, and then removes any instances that would still be classified correctly by the remaining subset. This is similar to RNN, except that it only considers whether the *removed* instance would be classified correctly, whereas RNN considers whether the classification of *other* instances would be affected by the instance's removal.
- **Minimal Consistent Set (MCS)** [12] - The purpose of this algorithm is to find a Minimal consistent set of instances which will be able to classify all the training instances in TR. It performs the following steps:
  1) Define an initial consistent set to be the given training data set, since the given set is by definition consistent with itself.
  2) For a specific sample in the given training data set, determine the nearest sample distance among all the samples from all classes other than its own in the consistent set, Le., identify and store the Nearest Unlike Neighbor (NUN) distance of the sample from the consistent set.
  3) For this same sample, identify all the neighboring samples from its own class in the given data set which are closer than this NUN distance and cast an approval vote to each of these samples in the given set by incrementing the corresponding vote registers, while noting this voter's (sample) identity by updating the corresponding voter lists.
  4) Repeat Step 2 and 3 for all samples in the given training set, which results in a list of the number of votes received by each sample in the given set along with the records of identity of its voters.
  5) Create a potential candidate consistent set consisting of all samples in the given set which are either (a)

already present in the current consistent set or (b) whose inclusion will not create an inconsistency; i.e., the sample should not be nearer to any member of any other class than that member's current NUN distance. In the first iteration, the entire consistent set (i.e., the given set) remains as the candidate consistent set as all samples satisfy condition (a)

6) Identify the most voted sample in this candidate consistent list and designate it as a member of a newly selected consistent set and identify all of its contributing voters.
7) Delete these voters from all the voter lists wherein they currently appear and correspondingly decrement the appropriate vote counts.
8) Repeat Step 6 and Step 7 till all the voters have been accounted for by the selected consistent set.
9) Now with this selected consistent set, the NUN distances of the input samples are likely to be greater than before as some of the original NUN samples may no longer be in the selected consistent set. Accordingly, repeat Step 2 using this selected consistent set to determine the NUN distance thresholds for each sample in the given set.
10) Repeat Step 3 through 8 using all the samples in the given set to identify a new consistent set. This process of recursive application of step 2 through 8 is continued till the selected set is no longer getting smaller. It is easy to see that under this procedure this final subset remains consistent, i.e., is able to classify all samples in the original set correctly.

- **Modified Selective Algorithm (MSS)** [13] - Let R_i be the set of all x_i in TR such that x_j is of the same class of x_i and is closer to x_i than the nearest neighbor of x_i in TR of a different class than x_i. Then, MSS is defined as that subset of the TR containing, for every x_i in TR, that element of its R_i that is the nearest to a different class than that of x_i.

  An efficient algorithmic representation of the *MSS* method is depicted as:

```
$Q = TR$
Sort the instances $\{x_j\}_{j=1}^{n}$ according to increasing values
of enemy distance $(D_j)$.
For each instance $x_i$ do
    $add \leftarrow FALSE$
    For each instance $x_j$ do
        If $x_j \in Q \wedge d(x_i, x_j) < D_j$ then
            $Q \leftarrow Q - \{x_j\}$
            $add \leftarrow TRUE$
    If $add$ then $S \leftarrow S \cup \{x_i\}$
    If $Q = \emptyset$ then return $S$
```

- **PEBS Algorithm (PEBS)** [14] - This algorithm is based in the PENN algorithm (see section III-B). The method proceeds as follows:

```
TR = PENN(TR), $S = \emptyset$
$C$ = getClassSet(TR), $C$= $C_i$ | i= 0,..., $\Omega$
For all i,j, where i<j, $C_i \neq \emptyset$, and $C_j \neq \emptyset$
    $S_{ij} = \emptyset$, $C'_i = C_i$, $C'_j = C_j$, $C_{ij} = C_i \bigcup C_j$
    Acc[l]=0, l=0, 1, ..., L, L=100
    While(true)
        $S'_{ij}$ = $SI_2(C'_i, C'_j, S_{ij})$
        $S_{ij}$ = $S_{ij} \bigcup S'_{ij}$
        $C'_i$=$C'_i$ - $S'_{ij}$, $C'_j$ = $C'_j$ - $S'_{ij}$
        Acc[l]=ValidateNBModel($S_{ij}$,$C_{ij}$)
        if (Acc[l] $\leq$ Acc[l-1])
            $S_{ij}$ = old; break;
            continue
        old = $S_{ij}$, l++
    $S$ = $S \bigcup S_{ij}$
Return S
```

C. Batch

- **Improved KNN (IKNN)** [15] - The IKNN defines weights and attractive capacities of every pattern in TR to perform a condensing process. It can be briefly described as follows:
  1) Calculate the weight of each pattern and keep only one out of all identical patterns. Set l=1.
  2) Calculate the attractive capacity for each pattern in the updated set R.
  3) Check if all attractive capacities are no greater than Γ. If true, set S=R and stop.
  4) Eliminate the patterns which exhibit capacities greater than Γ and are among the σ(l) portion of patterns corresponding to the highest capacities.
  5) Set l=l + 1 Go to Step 2.

  In addition to the condensing procedure, the IKNN algorithm takes into account the difference of the norms between the prototypes and the test instances, to improve further the results of the classification process.
- **Patterns by Ordered Projections (POP)** [16] - This algorithm consists on eliminating the examples that are not in the limits of the regions which they belong. For it, each attribute is studied separately, making a sorting and increasing a value, called *weakness*, associated to each one of the instances, if it is not in a limit. The instances with a value of *weakness* equal to the number of attributes are eliminated.

- **Max Nearest Centroid Neighbor (Max-NCN)** [17] This algorithm is based on the Nearest Centroid Neighborhood (NCN) concept, defined by:

  1) The first NCN of x_i is also its NN, y_1.
  2) The i-th NCN, y_i, $i \geq 2$, is such that the centroid of this and previously selected NCN, $y_1, ..., y_i$ is the closest to x_i.

  MaxNCN algorithm can be written as follows:

```
For each instance $x_i$ do
    $neighbors\_number[x_i] = 0$
    $neighbor = next\_neighbor(x_i)$
    While $neighbor.class == x_i.class$ do
        $neighbors\_vector[x_i] = Id(neighbor)$
        $neighbors\_number[x_i] ++$
        $neighbor = next\_neighbor(x_i)$
    End while
End for
While $Max\_neighbors() > 0$ do
    $EliminateNeighbors(id\_Max\_neighbors)$
End while
```

- **Reconsistent (Reconsistent)** [17] - The Reconsistent algorithm is an enhanced version of the Iterative MaxNCN. When it has been applied to the set TR the subset resulting is processed by a condensing method (CNN), employing as reference set the original training set TR.
- **Template Reduction KNN (TRKNN)** [18] - The TRKNN method introduces the concept of nearest neighbors chains. Every chain is assigned to one instance, and it is built by finding the nearest neighbors of each element of the chain, which belongs alternatively to the class of the starting instance, or to a different class. The chain is stopped when an element is selected twice to belong to the chain.

  By construction the distances between the patterns in the chain form a non-increasing sequence, thus the last elements of the chain will be near from the decision boundaries. The TRKNN method will employ this property to drop all instances which are far away from the decision boundaries.

IV. Edition Algorithms

These algorithms edit out noises instances as well as close border class, leaving smoother decision boundaries. They also retain all internal points, because an internal instance may be labeled as the same class of its neighbors.

Considering their search direction they can be classified as:

A. Decremental

- **Edited Nearest Neighbor (ENN)** [19] - Wilson developed this algorithm which starts with S = TR and then each instance in S is removed if it does not agree with the majority of its k nearest neighbors.
- **Repeated-ENN (RENN)** [19] - It applies the ENN algorithm repeatedly until all instances remaining have a majority of their neighbors with the same class.
- **Multiedit (Multiedit)** [20] - This method proceeds as follows.

```
Let $S = TR$.
Do
    Let $R = S$.
    Let $S = \emptyset$ and randomly split $R$ into $b$ blocks: $R_1, ..., R_b$,
    where $b > 2$
    For each $b_i$ block
        Add to $S$ the prototypes from $R_{b_i}$ that are
        misclassified using the $k$-NN rule with $R_{(b_i+1) mod\, b}$.
While $S \neq R$.
```

- **Relative Neighborhood Graph Edition (RNGE)** [21] - A Proximity Graph (PG), $G = (V, E)$, is an undirected graph with a set of vertices $V = TR$, and a set of edges, E, such that $(t_i, t_j) \in E$ if ad only if t_i and t_j satisfy some neighborhood relation. In this case, we say that t_i and t_j are *graph neighbors*. The graph neighbors of a given point constitute its *graph neighborhood*. The graph neighborhood of a subset, S $\subseteq V$, consists of the union of all the graph neighbors of every node in S. The scheme of editing can be expressed in the following way.

  1) Construct the corresponding PG.
  2) Discard those instances that are misclassified by their graph neighbors (by the usual voting criterion).

  The PGs used are the **Gabriel Graph Edition (GGE)** and the **Relative Neighborhood Graph Edition (RNGE)**.

  Gabriel Graph Editing (GGE): - The *GGE* is defined as follows:

$$(t_i, t_j) \in E \Leftrightarrow d^2(t_i, t_j) \leq d^2(t_i, t_k) + d^2(t_j, t_k), \forall t_k \in T, k \neq i, j. \quad (1)$$

  where $d(\cdot, \cdot)$ be the Euclidean distance.

Relative Nearest Graph Editing (RNGE): - Analogously, the set of edges in the *RNGE* is defined as follows:

$$(t_i, t_j) \in E \Leftrightarrow d(t_i, t_j) \leq max(d(t_i, t_k), d(t_j, t_k)), \forall t_k \in T, k \neq i, j. \tag{2}$$

- **Modified Edited Nearest Neighbor (MENN)** [22] - This algorithm, similar to ENN, starts with S = TR and then each instance x_i in S is removed if it does not agree with all of its k+l nearest neighbors, where *l* are all the instances in S which are at the same distance that the last neighbor of x_i.
  In addition, MENN works with a prefixed number of pairs (k,$k^{'}$). k is employed as the number of neighbors employed to perform the editing process, and $k^{'}$ is employed to validate the edited set S obtained. The best pair found is employed as the final reference set (if two or more sets are found as optimal, then both are employed in the classification of the test instances. A majority rule is used to decide the output of the classifier in this case).
- **Nearest Centroid Neighbor Edition (NCNEdit)** [23] - The NCN Editing algorithm applies the NCN classification rule to perform an edition process over the training set TR. The NCN classification rule can be defined as:
  1) The first NCN of x_i is also its Nearest Neighbor, y_1.
  2) The i-th NCN, y_i, i $>$ 1, is such that the centroid of this and previously selected NCN, $y_1 ... y_i$ is the closest to x_i.

  Having defined the NCN scheme, the editing process consists in set S=TR an discard from S every prototype misclassified by the NCN rule.
- **Edited Normalized Radial Basis Function (ENRBF)** [24] This algorithm is an *Edited* version of *NRBF* [25]. *NRBF* estimates probability of k-th class given vector v and training set TR:

$$P(k|v, TR) = \sum_{i \in I^k} \bar{G}_i(v; v_i), \tag{3}$$

  where $I^k = \{i : (v_i, y_i) \in TR \wedge y_i = k\}$, and $\bar{G}_i(v; v_i)$ is defined by

$$\bar{G}_i(v; v_i) = \frac{G(v; v_i, \sigma)}{\sum_{j=1}^{n} G(v; v_j, \sigma)}, \tag{4}$$

  and $G(v; v_i, \sigma)$ (σ is fixed) is defined by $G(v; v_i, \sigma) = e^{-\frac{||v - v_i||^2}{\sigma}}$.
  The *ENRBF* eliminates all vectors if only:

$$\exists_{k \neq y_i} \; P(y_i|v, TR^i) < \alpha P(k|v, TR^i), \tag{5}$$

  where $TR^i = TR - \{v_i, y_i\}$, and $\alpha \in (0, 1]$.

- **Edited Normalized Radial Basis Function 2 (ENRBF2)** [24] - This algorithm removes given instance x_i from the training set TR if the criterion $P(y_i|x_i;TR) \cdot \beta < P(y_i|x_i;TR^i)$ where $\beta \epsilon (0,1]$ and $TR^i = TR - (x_i, y_i)$. This means that if removing instance x_i probability that this instance belongs to the class y_i is not significantly reduced then such instance can be removed.
- **Edited Nearest Neighbor Estimating Class Probabilistic (ENNProb)** [26] - This method employs a probabilistic k-NN rule, where the class of a instance is decided as a weighted probability of its nearest neighbors class (each neighbor has the same apriori probability, and the weight associated to it is the inverse of its distance). The editing process is performed starting with S=TR an deleting from S every prototype misclassified by this probabilistic rule.
- **Edited Nearest Neighbor Estimating Class Probabilistic and Threshold (ENNTh)** [26]
  This modification of ENNProb defines a threshold μ in the k-NN rule, which does not take into account instances which is assigned probability is lower than the threshold.

B. Batch

- **All k-NN (AllKNN**) [27] - All k-NN is an extension of ENN. The algorithm, for i = 0 to *k* flags as bad any instance not classify correctly by its i nearest neighbors. When the loop is completed *k* times, it removes the instances flagged as bad.
- **Model Class Selection (MoCS)** [28] - Brodley's algorithm for reducing the size of the training set TR is to keep track of how many times each instance was one of the k nearest neighbors of another instance, and whether its class matched that of the instance being classified. If the number of times was wrong is greater than the number of times it was correct then it is thrown out.

V. Hybrids Algorithms

Hybrids methods try to find the smallest subset S which lets keep or even increase the generalization accuracy in test data. For doing it, it allows the removal of internal and border points.

A. *Incremental*

- **Instance-Based Learning Algorithms Family (IB3)** [29][30] - A series of *instance-based* learning algorithms are presented. IB1 was simply the 1-NN algorithm, used as a baseline.

  IB2: - It starts with S initially empty, and each instance in TR is added to S if it is not classified correctly by the instances already in S (with the first instance always added). IB2 is similar to CNN, except that IB2 not seed S with one instance of each class and does not repeat the process after the first pass through the training set.

  IB3: - The IB3 algorithm proceeds as follows:

```
For each instance $x_i$ in $TR$
    Let $a$ be the nearest acceptable instance in $S$ to $x_i$.
    (if there are no acceptable instances in $S$, let $a$ be a
    random instance in $S$)
    If $class(a) \neq class(x_i)$ then add $x_i$ to $S$.
    For each instance $s$ in $S$
        If $s$ is at least as close to $x_i$ as $a$ is
            Then update the classification record of $s$
            and remove $s$ from $S$ its classification
            record is significantly poor.
Remove all non-acceptable instance from $S$.
```

  An instance is *acceptable* if the lower bound of its accuracy is statistically significantly higher (at a 90% confidence level) than the upper bound on the frequency of its class. Similarly, an instance is dropped from S if the upper bound on its accuracy is statistically significantly lower (at a 70% confidence level) than the lower bound on the frequency of its class. Other instances are kept in S during training, and then dropped at the end if they do not prove to be acceptable.

  The expression for the upper and lower bounds of the confidence level interval is:

$$\frac{p + z^2/2g \pm z\sqrt{\frac{p(1-p)}{g} + \frac{z^2}{4g^2}}}{1 + z^2/g} \tag{6}$$

  where for the *accuracy* of an instance in S, g is the number of classification attempts since introduction of the instance to S (i.e., the number of times it was at least as close to x_i as a was), p is the accuracy of such attempts (i.e., the number of times the instance's class matched x_i's class, divided by g), and z is the confidence (0.9 for acceptance, 0.7 for dropping). For the frequency of a class, p is the frequency (i.e. proportion of instances so far that are of this class), g is the number of previously processed instances, and z is the confidence.

B. Decremental

1) Filter:

- **Variable Similarity Metric (VSM)** [31] - In order to reduce storage and remove noisy instances, an instance x_i is removed if all k of its neighbors are of the same class, even if they are of a different class than x_i. The instance is only removed if its neighbors are at least 60% sure of their classification. The VSM method typically uses a fairly large k (i.e., $k = 10$).
- **Polyline Functions (PF)** [32]
  This method is based in the definition of Polyline functions, where the instances finally selected in S are represented as vertex in the Polyline functions.
- **Decremental Reduction Optimization Procedure Algorithms Family (DROP3)** [33] In order to present these reduction techniques, we need to define some concepts. Each instance x_i has k nearest neighbors where k is typically a small odd integer. x_i also has a nearest *enemy*, which is the nearest instance with a different output class. Those instances that have x_i as one of their k nearest neighbors are called $associates$ of x_i.
  DROP1: - It uses the following basic rule to decide if it safe to remove an instance from the instance set S (where S = TR originally):
  Remove x_i if at least as many of its associates in S would be classified correctly without x_i.
  The algorithm *DROP1* proceeds as follows.

```
DROP1(Training set $TR$): Selection set $S$.
Let $S = TR$.
For each instance $x_i$ in $S$:
    Find the $k+1$ nearest neighbors of $x_i$ in $S$.
    Add $x_i$ to each of its neighbors' lists of associates.
For each instance $x_i$ in $S$:
    Let $with = \#$ of associates of $x_i$ classified correctly with
    $x_i$ as a neighbor.
    Let $without = \#$ of associates of $x_i$ classified correctly
    without $x_i$.
    If $without \geq with$
        Remove $x_i$ from $S$.
        For each associate $a$ of $x_i$
            Remove $x_i$ from $a$'s list of nearest neighbors.
            Find a new nearest neighbor for $a$.
            Add $a$ to its new neighbor's list of associates.
        For each neighbor $b$ of $x_i$
            Remove $x_i$ from $b$'s lists of associates.
    Endif
Return $S$.
```

DROP2: - In this method, the removal criterion can be restated as:

Remove x_i if at least as many of its associates in TR would be classified correctly without x_i.

Using this modification, each instance x_i in the original training set TR continues to maintain a list of its $k+1$ nearest neighbors in S, even after x_i is removed from S. This means that instances in S have associates that are both in and out of S, while instances that have been removed from S have no associates.

DROP2 also changes the order of removal of instances. It initially sorts the instances in S by the distance to their nearest enemy. Instances are then checked for removal beginning at the instance furthest from its nearest enemy.

DROP3: - It is a combination of *DROP2* and *ENN* algorithms. *DROP3* uses a noise-filtering pass *before* sorting the instances in S (Wilson *ENN* editing). After this, it works identically to *DROP2*.

DROP4: - It is identical to DROP3 except that instead of blindly applying ENN, the noise-filtering pass removes each instance only if it is (1) misclassified by its *k* nearest neighbors, and (2) it does not hurt the classification of other instances.

DROP5: - It modifies DROP2 so that the instances are considered for removal beginning with instances that are nearest to their nearest enemy, and proceeding outward.

- **Decremental Encoding Length (DEL)** [34] - It is similar than DROP3, except that it uses the encoding length heuristic to decide in each case whether the instance can be removed. This serves as a noise-reduction pass, but will also cause most internal points to be removed as well. By removing points near the decision boundary first, the decision boundary is smoothed. After this pass, the furthest-to-nearest pass as done by DROP2 is done repeatedly until no further improvement can be made.
- **Prototype Selection by Relative Certainty Gain (PSRCG)** [35] - In order to describe this algorithm, we before should clarify some previous concepts:

  Definition 1: The Quadratic Entropy is a function QE from $[0,1]^{\Omega}$ in $[0,1]$.

$$QE : [0,1]^{\Omega} \rightarrow [0,1],\, (\gamma_1, ..., \gamma_{\Omega}) \rightarrow QE((\gamma, ..., \gamma_{\Omega})) = \sum_{j=1}^{\Omega} \gamma_j (1-\gamma_j) \tag{7}$$

  where $\sum_{j=1}^{\Omega} \gamma_j = 1$.

  Definition 2: Let $N(x_i)$ be the neighborhood of an instance x_i belonging to the learning set S:

$$N(x_i) = \{x_j \in S / x_i \, is\, linked\, by\, an\, edge\, to\, x_j\, in\, the\, KNN\, graph\} \cup \{x_i\} \tag{8}$$

  $\{x_i\}$ is inserted in $N(x_i)$ to allow the elimination of mislabeled instances.

  Definition 3: Let $U_{loc}(x_i)$ be the local uncertainty around x_i:

$$U_{loc}(x_i) = \sum_{j=1}^{\Omega} \frac{n_{ij}}{n_{i.}} (1 - \frac{n_{ij}}{n_{i.}}) \tag{9}$$

where $n_{i.} = card\{N(x_i)\}$ and $n_{ij} = card\{x_l \in N(x_i) | Y(x_l) = y_j\}$ where $Y(x_l)$ describes the class of x_l among Ω classes.

Definition 4: Let U_{tot} be the total uncertainty in S with s instances:

$$U_{tot} = \sum_{i=1}^{s} \frac{n_{ij}}{n_{..}} \sum_{j=1} \Omega \frac{n_{ij}}{n_{i.}}(1 - \frac{n_{ij}}{n_{i.}}) \tag{10}$$

where $n_{..} = \sum_{i=1}^{s} n_{i.} = s + 2card\{E\}$ and E is the set of all the edges in the KNN graph.

Definition 5: Let RCG be the Relative Certainty Gain:

$$RCG = \frac{U_0 - U_{tot}}{U_0} \tag{11}$$

where U_0 is the uncertainty computed directly from the *a priori* distribution.

$$U_0 = \sum_{j=1}^{\Omega} \frac{n_j}{s}(1 - \frac{n_j}{s}) \tag{12}$$

where $n_j = card\{\omega_i / Y(\omega_i) = y_j\}$.

The ***PSRCG*** algorithm consist in starting with the whole set of instances, and in a first step removing irrelevant examples containing noise. The principle is based on the searching for the worst instance which allows, once deleted, to improve the information in the data set. An instance is eliminated at the step i if and only if two conditions are filled:

1) the RCG after the deletion is better than before.
2) $RCG_i > 0$.

Once this procedure is completed, then it executes a post-process which removes the center of the remaining clusters. The pseudocode of this algorithm is following presented:

```
$i \leftarrow 0$; $s = |S|$
Build the kNN graph on the learning set
Compute $RCG_1$
Repeat
    $i \leftarrow i + 1$
    Select the instance $\omega = max(U_{loc}(\omega_j))$
    If 2 instances have the same $U_{loc}$
        select the example having the smallest number of
        neighbors
    Local modifications of the kNN graph
    Compute $RCG_{i+1}$ after removing $\omega$
    $s \leftarrow s - 1$
Until $(RCG_{i+1} < RCG_i)$ or not $(RCG_{i+1} > 0)$
Remove instances having a null uncertainty with their
    $(k + 1)$-NN
```

- **C-Pruner (CPruner)** [36] - First it is necessary to introduce some concepts underlying this algorithm.

  Definition 6: For an instance x_i in TR, the k nearest neighbors of x_i make up its *k-reachability* set, denoted as *k-reachability(x_i)*.

  Definition 7: For an instance x_i in TR, those instances with similar class label to that of x_i, and have x_i as one of their nearest neighbors are called the *k-coverage* set of x_i, denoted as *k-coverage(x_i)*.

  Definition 8: For instance x_i in TR, if x_i can be classified correctly by *k-reachability(x_i)*, then we say x_i is *implied* by *k-reachability(x_i)*, and x_i is a *superfluous* instance in TR.

  Definition 9: For instance x_i in TR, x_i is a *critical* instance, if the following conditions holds:

  At least one instance x_j in *k-coverage(x_i)* is not implied by *k-reachability(x_j)*, or

  After x_i is deleted, at least one instance x_j in *k-coverage(x_i)* is not implied by *k-reachability(x_j)*.

  Definition 10: For instance x_i in TR, if x_i is not a superfluous instance and —*k-reachability(x_i)*— ¿ —*k-coverage(x_i)*—, then x_i is a *noisy* instance.

  Rule 1: Instance pruning rule

  For an instance x_i in TR, if it can be pruned, it must satisfy one of the following two conditions:

  It is a noisy instance;

  It is a superfluous instance, but not a critical one.

  Rule 2: Rule for deciding the order of instances removal

  Let H-kNN(x_i) be the number of the instances of its class in kNN(x_i), and D-NE(x_i) be the distance of x_i to its nearest enemy.

  For two prunable instances x_i and x_j in TR,

  If H-kNN(x_i) $>$ H-kNN(x_j), x_i should be removed before x_j;

  If H-kNN(x_i) $=$ H-kNN(x_j) and D-NE(x_i) $>$ D-NE(x_j), x_j should be removed before x_i;

  If H-kNN(x_i) $=$ H-kNN(x_j) and D-NE(x_i) $=$ D-NE(x_j), the order of removal is random decided.

  Following, we present the *C-Pruner* algorithm.

```
$S = TR$
For all $x_i \in S$ do
    Compute k-reachability($x_i$) and k-coverage($x_i$)
For all $x_i \in S$ do
    If $x_i$ is a noisy instance
        Remove $x_i$ from $S$
        For all $x_j \in$ k-coverage($x_i$)
            Remove $x_i$ from k-reachability($x_j$)
            Update k-reachability($x_j$)
        For all $x_j \in$ k-reachability($x_i$)
```

```
                    Remove $x_i$ from *k-coverage*($x_j$)
        Sort the removal order of instances in $S$ according to rule 2
        For all $x_i \in S$
            If $x_i$ satisfies rule 1
                Remove $x_i$ from $S$
                For all $x_j \in$ *k-coverage*($x_i$)
                    Remove $x_i$ from *k-reachability*($x_j$)
                    Update *k-reachability*($x_j$)
        Return $S$
```

- **Support Vector Based Prototype Selection (SVBPS)** [37] - The SVBPS method firstly learns a SVM employing a proper kernel function. The Support Vectors found in the procedure are post processed with the DROP2 algorithm, adopting its result as the final S set.
- **Noise Removing based on Minimal Consistent Set (NRMCS)** [38] - The NRMCS algorithm is as follows:
  1) For each sample, identify all the neighboring samples from its own class in the given data set which are closer than its NUN distance and cast an approval vote to each of these samples in the given set.
  2) Create a potential candidate consistent set consisting of all samples in the given set which are either (a) already present in the current consistent set or (b) whose inclusion will not create an inconsistency. In the first iteration, the entire consistent set remains as the candidate consistent set as all samples satisfy condition(a).
  3) Delete the samples which neither vote nor are voted except itself from potential candidate consistent set.
  4) Identify the most voted sample in this candidate , and designate it as a member of a newly selected consistent set and identify all of its contributing voters.
  5) Delete these voters wherein they currently appear and correspondingly decrement the appropriate vote counts.
  6) Repeat Step 2 through 4 till all the voters have been accounted for by the selected consistent set.
  7) Repeat Step 1 through 5 using all the samples in the given data set to identify a new consistent set.
- **Class Conditional Instance Selection (CCIS)** [39]
  - This algorithm consists of the following two phases:
  – Class Conditional selection phase (CC). It removes from the training set outliers, isolated points and points close to the 1NN decision boundary. This phase aims at enlarging the hypothesis margin and reducing the empirical error.

```
$(x_1, ..., x_n)$ = TR sorted in decreasing order of Score
$S = (x_1, ..., x_{k_0})$
i = $k_0$ + 1
```

```
$go\_on$ = 1
ub = n - |$\{a$ s.t. Score($a$)$\leq$ 0 $\}$|
While i < ub and $go\_on$ do
     Temp = S $\bigcup$ $\{x_i\}$
     If $\epsilon^S \leq \epsilon^A$ then
          $go\_on$ = 0
     If $\epsilon^{Temp} < \epsilon^S$ and $go\_on$ then
          S = Temp
          i = i + 1
     else
          $go\_on$ = 0
```

– Thin-out selection phase (THIN). It thins out points that are not important to the decision boundary of the resulting 1NN rule. This phase aims at selecting a small number of instances without negatively affecting the 1NN empirical error.

```
$S_f$ = $\{$ $x\epsilon S$ with in-degree $G_{bc}^{S} > 0$ $\}$
$S_{prev} = S$
$S_1$ = $S \backslash S_f$
$go\_on$ = 1
While $go\_on$ do
     $S_t$ = $\{$ $a\epsilon S_1$ with in-degree $G_{bc}^{S_t}$ > 0
          and with in-degree $G_{bc}^{S_{prev}}$ or in $G_{wc}^{S_{prev}} > 0\}$
     $go\_on$ = $\epsilon^{S_f \bigcup S_t} < \epsilon^{S_f}$
     If $go\_on$ then
          $S_f$ = $S_f \bigcup S_t$
          $S_prev$ = $S_1$
          $S_1$ = $S \backslash S_f$
```

2) Wrapper:

- **Backward Sequential Edition (BSE)** [40] - The BSE algorithm starts with S=TR. Each instance x_i is tested to find how the performance of the k-NN is increased when x_i is removed from S. The instance which removal causes the best increase in the performance is finally deleted from S, and the process is repeated until no further increases in the performance of the classifier are found. To increase the efficiency of the method, the authors suggested the use of ENN or DROP procedures as a first stage of the BSE algorithm.

C. Batch

- **Iterative Case Filtering (ICF)** [41] - ICF defines *local set* $L(x)$ which contain all cases inside largest hypersphere centered in x_i such that the hypersphere contains only cases of the same class as instance x_i. Authors define two properties, *reachability* and *coverage*:

$$Coverage(x_i) = \{x'_i \in TR : x_i \in L(x'_i)\}, \tag{13}$$

$$Reachability(x_i) = \{x'_i \in TR : x'_i \in L(x_i)\}, \tag{14}$$

  In the first phase ICF uses ENN algorithm to remove the noise from the training set. In the second phase ICF algorithm removes each instance x for which the $Reachability(x_i)$ is bigger than the $Coverage(x_i)$. This procedure is repeated for each instance in TR. After that ICF recalculates *reachability* and *coverage* properties and restarts the second phase (as long as any progress is observed).

- **Hit-Miss Network Algorithms (HMN)** [42] - Hit-Miss Networks are directed graphs where the points are the instances in TR and the edges are connections between an instance and its nearest neighbor from each class. The edge connecting a instance with a neighbor of its same class is called ”Hit”, and the rest of its edges are called ”Miss”.

  The **HMNC** method builds the network and removes all nodes not connected. The rest of the nodes are employed to build the final S set.

  Hit Miss Network Edition (**HMNE**): Starting from the output of HMNC, HMNE applies these four rules to prune the network:

  1) Every point with more ”Miss” edges than ”Hit” edges is flagged for removal.
  2) If the size of non flagged points of a class is too low, edges with at least one ”Hit” from those classes are unflagged.
  3) If there are more than three classes, some points of each class with low number of ”Miss” are unflagged.
  4) Points which are the ”Hit” of a 25% or more instances of a class are unflagged.

  Finally, the instances which remain flagged for removal are deleted from the network, in order to build the final S set.

  Hit Miss Network Edition Iterative (**HMNEI**): The HMNE method can be employed iteratively until the generalization accuracy of 1-NN on the original training set with the reduced set decreases.

D. Mixed+Wrapper

- **Explore (Explore)** [43] Cameron-Jones used an *encoding length heuristic* to determine how good the subset S is in describing TR. His algorithms use cost function defined by:

$$COST(s, n, x) = F(s, n) + s\ \log_2(\Omega) + F(x, n - s) + x\ \log_2(\Omega - 1) \tag{15}$$

where n is the number of instances in TR, s is the number of instances in S, and x defines the number of badly classified instances (basing on S). Ω is the number of classes in the classification task. $F(s,n)$ is the cost of encoding which s instances if the n available are retained, and is defined as:

$$F(s,n) = \log^* \left(\sum_{j=0}^{s} \frac{n!}{j!(n-j)!} \right) \tag{16}$$

$\log^* n = arg\ min_k F(k) \geq n$, k is integer, and $F(0) = 1, F(i) = 2^{F(i-1)}$.

ELH algorithm starts from the empty set and adds instances if only the minimize the cost function.

ELGrow additionally tries to remove instances if it helps to minimize the cost function. **Explore** extends ELGrow by 1000 mutations to try to improve the classifier. Each mutation tries adding an instance to S, removing one from S, or swapping one in S with one in TR - S, and keeps the change if it does not increase the cost of the classifier.

- **Generational Genetic Algorithm (GGA)** [44] - The basic idea in GGA is to maintain a population of chromosomes, which represent plausible solutions to the particular problem that evolves over successive iterations (generations) through a process of competition and controlled variation. Each chromosome in the population has an associated fitness to determine which chromosomes are to be used to form new ones in the competition process. This is called selection. The new ones are created using genetic operators such as crossover and mutation. The classical model of genetic algorithms is the GGA, which consists of three operations:

  1) Evaluation of individual fitness.
  2) Formation of a gene pool (intermediate population) through selection mechanism.
  3) Recombination through crossover and mutation operators.

  The selection mechanism produces a new population, P(t), with copies of chromosomes in P(t-1). The number of copies received for each chromosome depends on its fitness; chromosomes with higher fitness usually have a greater chance of contributing copies to P(t). Then, the crossover and mutation operators are applied to P(t). Crossover takes two individuals called parents and produces two new individuals called the offspring by swapping parts of the parents. In its simplest form the operator works by exchanging sub-strings after a randomly selected crossover point. The crossover operator is not usually applied to all pairs of chromosomes in the new population. A random choice is made, where the likelihood of crossover being applied depends on probability defined by a crossover rate.

  Mutation serves to prevent premature loss of population diversity by randomly sampling new points in the search space. Mutation rates are kept small however, otherwise the process degenerates into a random search. In the case of bit strings, mutation is applied by flipping one or more random bits in a string with a probability equal to the mutation rate.

  Termination may be triggered by reaching a maximum number of generations or by finding an acceptable solution by some criterion.

- **Steady-State Genetic Algorithm (SSGA)** [44]

  In these algorithms usually only one or two offspring are produced in each generation. Parents are selected to produce offspring and then a replacement/deletion strategy defines which member of the population will be replaced by the new offspring. The basic algorithm steps of SSGA are the following:

  1) Select two parents from the population P.
  2) Create an offspring using crossover and mutation.
  3) Evaluate the offspring with the fitness function.
  4) Select an individual in P, which may be replaced by the offspring.
  5) Decide if this individual will be replaced.

  In step 4, the replacement strategy can be chosen (e.g., replacement of the worst, the oldest, or a randomly chosen individual). In step 5, the replacement condition can be chosen (e.g., replacement if the new individual is better, or unconditional replacement). A widely used combination is to replace the worst individual only if the new individual is better.

- **Population Based Incremental Learning (PBIL)** [44] - PBIL is a specific evolutionary algorithm designed for binary search spaces. The PBIL algorithm attempts to explicitly maintain statistics about the search space to decide where to sample next.

  The object of the algorithm is to create a real valued probability vector, V_p, which, when sampled, reveals high quality solution vectors with high probability. For example, if a good solution can be encoded as a string of alternating 0's and 1's, a possible final V_p would be 0.01, 0.99, 0.01, 0.99, etc. Initially, the values of V_p are set at 0.5. Sampling from this vector yields random solution vectors because the probability of generating a 1 or 0 is equal. As the search progresses, the values of V_p gradually shift to represent high evaluation solution vectors through the following process:

  1) A number of solution vectors ($N_{samples}$) are generated based upon the probabilities specified in V_p.
  2) V_p is pushed towards the generated solution vector with the highest evaluation, S_{best}: $V_p[i] = V_p[i] \cdot (1 - LR) + S_{best}[i] \cdot LR$, where LR is the learning rate, which specifies how close the steps are to the best solution.
  3) V_p is pushed far away from the worst evaluation, S_{worse}, where S_{best} and S_{worse} differ. This is accomplished as follows: $If S_{best}[i] <> S_{worse} then V_p[i] = V_p[i] \cdot (1 - Negat_L R) + S_{best}[i] \cdot Negat\_LR$, where Negat_LR is the negative learning rate, which specifies how far the steps are from the worst solution.
  4) After the probability vector is updated, sampling the updated probability vector produces a new set of solution vectors, and the cycle is continued.

  Furthermore, PBIL applies mutations to V_p, with an analogous aim as mutation in genetic algorithms: to inhibit premature convergence. Mutations affect V_p with low probability, P_m, in a random direction, Mut_Shif.

  It must be pointed out that there are problems setting the LR and Negat_LR in PBIL, which affects the convergence and the quality of solutions.

- **Cerveron's Tabu Search (CerveronTS)** [45] - To apply Tabu Search to instance selection problem, the selected subset is represented by a 0/1 bit string. The k-th bit, when valued as 0 or 1 denotes the absence or presence of the k-th instance in the selected subset, respectively.

  The approach proposed employs the generic Tabu Search metaheuristic. Two changes are suggested to improve the performance of the method in the task of perform a PS:
  - Generate the initial solution with the application of CNN.
  - Employ as initial solution a set with a randomly picked prototype from each class, disabling sample deletion in the search process of the TS.
- **Estimation of Distribution Algorithm (EDA)** [46] - The EDAs can be employed to perform PS process. In EDA there are neither crossover nor mutation operators, the new population is sample from a probability distribution which is estimated from the selected individuals.

  This approach employs a binary representation to perform PS. A brief pseudocode is shown as follows:

```
$D_0 \longleftarrow$ Generate $N$ individuals (initial population) randomly
Repeat for $l$=1,2,... until a stop criterion is met
    $D_{l-1}^U \longleftarrow$ Select $U \leq N$ individuals from $D_{l-1}$
          according to a selection method
    $p_l$(x)=p(x|$D_{l-1}^s$) $\longleftarrow$ Estimate the joint probability
          distribution of an individual being among selected individuals
    $D_l \longleftarrow$ Sample $N$ individuals (new population) from $p_l$(x)
End Repeat
```

- **Intelligent Genetic Algorithm (IGA)** [47] - IGA is a generational genetic algorithm that incorporates an Intelligent Crossover (IC) operator. IC builds an Orthogonal Array (OA) from two parents of chromosomes and searches within the OA for the two best individuals according to the fitness function. It takes about $2^{log_2(\gamma+1)}$ fitness evaluations to perform an IC operation, where γ is the number of bits that differ between both parents. Note that the application of IC on large-size chromosomes (resulting chromosomes from large size data sets) could consume a high number of evaluations.
- **Zhang's Tabu Search (ZhangTS)** [48] - To apply Tabu Search to instance selection problem, authors describe the method as follows.

  The selected subset is represented by a 0/1 bit string, the k-th bit 0 or 1 denotes the absence or presence of the k-th instance in the selected subset. Let S_{curr}, S_{next} and S_{best} be the current, next and the best reference subsets, respectively. TL is a first in first out tabu list. It has a predefined length l (tabu tenure). The neighborhood of each solution S_{curr} is defined as $N(S_{curr})$ consisting of all subsets that differ from S_{curr} in only one sample addition, denoted by $N^+(S_{curr})$, or deletion, denoted by $N^-(S_{curr})$.

  In adding or deleting an instance to or from s_{curr}, it considers the change of the classification error rate ($e(\cdot)$) and the distance between the original data set and the current selected subset, which is the sum of the distances between each sample in the original data set and its nearest neighbors of the same class in the selected subset. Two heuristic criteria are used in searching the best solution S_{next} among all candidate solutions in $N^{+}(S_{curr})$:

1) Search the S_{next} of the minimal error rate in the candidate subsets $N^{+}(S_{curr})$. If $e(S_{next}) < e(S_{curr})$, then S_{next} is the best solution in the candidate subsets. If there is more than one solution having the minimal error rate, then select the one that has the minimal distance from the original data set.
2) For the minimal error rate S_{next} in the candidate solutions $N^{+}(S_{curr})$, if $e(S_{next}) \geq e(S_{curr})$, then consider selecting such solutions in $N^{+}(S_{curr})$ which could correctly classify at least one of the samples that are wrongly classified by S_{curr}.

A complete description of the subset selection algorithm based on tabu search is as follows:

1) Input the original training data set TR, specify the tabu list length l and the error rate threshold et.
2) Generate an initial solution S_{init}, set $S_{curr} = S_{init}$, $S_{best} = TR$. Let $TL = \emptyset$, $k = 0$.
3) a) Find the best solution S_{next} in the neighborhood of S_{curr}. There are two cases:
   If $e(S_{curr}) > et$, then search the best solution S_{next} in $N^{+}(S_{curr})$ according to criterion 1 and 2.
   If $e(S_{curr}) \leq et$, then search the best solution S_{next} among all the solutions in $N^{-}(S_{curr})$ according to the minimal error rate and minimal distance criterions.
   b) If S_{next} is in TL and does not satisfy the aspiration criterion, then let $N(S_{curr}) = N(S_{curr}) - \{S_{next}\}$, goto 3(a); Otherwise, let $S_{curr} = S_{next}$.
   If $e(S_{curr}) \leq et$ and $|S_{curr}| < |S_{best}|$, or $|S_{curr}| = |S_{best}|$ and $e(S_{curr}) < e(S_{best})$, then let $S_{best} = S_{curr}$.
   c) If termination condition is satisfied, stop and output the S_{best}, otherwise insert the S_{curr} into TL, $k = k + 1$, Goto (3).

- **CHC (CHC)** [44] - During each generation the CHC develops the following steps.
  1) It uses a parent population of size N to generate an intermediate population of N individuals, which are randomly paired and used to generate N potential offsprings.
  2) Then, a survival competition is held where the best N chromosomes from the parent and offspring populations are selected to form the next generation.

  CHC also implements a form of heterogeneous recombination using HUX, a special recombination operator. HUX exchanges half of the bits that differ between parents, where the bit position to be exchanged is randomly determined. CHC also employs a method of incest prevention. Before applying HUX to two parents, the Hamming distance between them is measured. Only those parents who differ from each other by some number of bits (mating threshold) are mated. The initial threshold is set at $L/4$, where L is the length od the chromosomes. If no offspring are inserted into the new population then the threshold is reduced by one.
  No mutation is applied during the recombination phase. Instead, when the population converges or the search stops making progress, the population is reinitialized to introduce new diversity to the search. The chromosome representing the best solution found over the course of the search is used as a template to reseed the population.

Reseeding of the population is accomplished by randomly changing 35% of the bits in the template chromosome to form each of the other $N-1$ new chromosomes in the population. The search is the resumed.

- **Genetic Algorithm based on Mean Squared Error, Clustered Crossover and Fast Smart Mutation (GA-MSE-CC-PSM)** [49] - This algorithm included the definition of a novel mean square error based fitness function, a novel clustered crossover technique, and a fast smart mutation scheme.

  The fitness function employed was based on a mean square error measure. As their authors stated, the error surface defined by this function is smoother than those obtained using counting estimator based functions, making easier the obtaining of its local minimum.

  The clustering crossover operator firstly performs a k-means clustering process over the chromosomes, extracting the centroids of all the clusters found (the number of clusters is established randomly). Then, the centroids are employed with a classical random point cross operator to generate the individuals of the new generation.

  The Fast Smart Mutation procedure computes the effect of the change of one bit of the chromosome over its fitness value, testing all the possibilities. The change which produces the better fitness value is accepted as the result of the mutation operator. It is applied to every individual of the population.
- **Steady-state memetic algorithm (SSMA)** [50] - The SSMA was proposed to cover a drawback of the conventional evolutionary PS methods that had appeared before: their lack of convergence when facing large problems. SSMA makes use of a local search or meme specifically developed for this prototype selection problem. This interweaving of the global and local search phases allows the two to influence each other; i.e. SSGA chooses good starting points, and local search provides an accurate representation of that region of the domain. A brief pseudocode of the SSMA is shown as follows:

```
Initialize population
While (not termination-condition) do
    Use binary tournament to select two parents
    Apply crossover operator to create offspring ($Off_1$,$Off_2$)
    Apply mutation to $Off_1$ and $Off_2$
        Evaluate $Off_1$ and $Off_2$
        For each $Off_i$
            Invoke Adaptive-$P_{LS}$-mechanism to obtain $P_{LS_i}$ for $Off_i$
            If $\upsilon(0,1)$ < $P_{LS_i}$ then
                Perform meme optimization for $Off_i$
            End if
        End for
    Employ standard replacement for $Off_1$ and $Off_2$
End while
Return the best chromosome
```

- **COoperative COevolutionary Instance Selection (CoCoIS)** [51] - The cooperative algorithm presents the following steps:

```
Initialize population of combinations
Initialize subpopulations of selectors
While Stopping criterion not met do
     For N iterations do
          Evaluate population of combinations
          Select two individuals by roulette selection and
               perform two point crossover
          Offspring substitutes two worst individuals
          Perform mutation with probability $P_{mutation}$
     For M iterations do
          Foreach subpopulation i do
               Evaluate selectors of subpopulation i
               Copy Elitism% to new subpopulation i
               Fill (1-Elitism)% of subpopulation i by HUX crossover
               Apply random mutation with probability $P_{random}$
               Apply RNN mutation with probability $P_{rnn}$
          Evaluate population of combinations
```

Where N and M are the number of iterations.

E. *Fixed+Wrapper*

- **Monte Carlo 1 (MC1)** [52]
  - This method obtains a high number of *f* random samples (typically *f*=100) of *m* instances (typically *m*=3) of TR. The classification accuracy of each sample over TR is measured with a 1-NN classifier. Then, the best sample is selected as the selected subset S.
- **Random Mutation Hill Climbing (RMHC)** [52] - It randomly selects a subset S from TR which contains a fixed number of instances s ($s = \%|TR|$). In each iteration, the algorithm interchanges an instance from S with another from TR - S. The change is maintained if it offers better accuracy.

References

[1] P. E. Hart, "The condensed nearest neighbour rule," *IEEE Transactions on Information Theory*, vol. 18, no. 5, pp. 515–516, 1968.

[2] J. Ullmann, "Automatic selection of reference data for use in a nearest-neighbor method of pattern classification," *IEEE Transactions on Information Theory*, vol. 24, pp. 541–543, 1974.

[3] I. Tomek, "Two modifications of cnn," *IEEE Transactions on systems, Man, and Cybernetics*, vol. 6, no. 6, pp. 769–772, 1976.

[4] K. Gowda and G. Krishna, "The condensed nearest neighbor rule using the concept of mutual nearest neighborhood," *IEEE Transactions on Information Theory*, vol. 29, pp. 488–490, 1979.

[5] V. Devi and M. Murty, "An incremental prototype set building technique," *Pattern Recognition*, vol. 35, no. 2, pp. 505–513, 2002.

[6] C.-C. L. F. Chang and C.-J. Lu, "Adaptive prototype learning algorithms: Theoretical and experimental studies," *Journal of Machine Learning Research*, vol. 7, pp. 2125–2148, 2006.

[7] F. Angiulli, "Fast nearest neighbor condensation for large data sets classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 11, pp. 1450–1464, 2007.

[8] J. C.-O. J.A. Olvera-López and J. Martínez-Trinidad, "A new fast prototype selection method based on clustering," *Pattern Analysis and Applications*, vol. In press, 2010.

[9] G. W. Gates, "The reduced nearest neighbour rule," *IEEE Transactions on Information Theory*, vol. 18, no. 3, pp. 431–433, 1972.

[10] G. L. Ritter, H. B. Woodruff, S. R. Lowry, and T. L. Isenhour, "An algorithm for a selective nearest neighbor decision rule," *IEEE Transactions on Information Theory*, vol. 21, no. 6, pp. 665–669, 1975.

[11] D. Kibler and D. W. Aha, "Learning representative exemplars of concepts: An initial case study," Proceedings of the Fourth International Workshop on Machine Learning. Morgan Kaufmann, 1987, pp. 24–30.

[12] B. Dasarathy, "Minimal consistent set (mcs) identification for optimal nearest neighbor decision system design," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, pp. 511–517, 1994.

[13] R. Barandela, N. Cortés, and A. Palacios, "The nearest neighbor rule and the reduction of the training sample size." Proceedings of the IX Symposium of the Spanish Society for Pattern Recognition, 2001.

[14] G. Li, N. Japkowicz, T. Stocki, and R. Ungar, "Instance selection by border sampling in multi-class domains," *Lecture Notes in Artificial Intelligence*, no. 5678, pp. 209–221, 2009.

[15] Y. Wu, K. Ianakiev, and V. Govindaraju, "Improved k-nearest neighbor classification," *Pattern Recognition*, vol. 35, no. 10, pp. 2311–2318, 2002.

[16] J. C. Riquelme, J. S. Aguilar-Ruiz, and M. Toto, "Finding representative patterns with ordered projections," *Pattern Recognition*, vol. 36, pp. 1009–1018, 2003.

[17] M. Lozano, J. S. Sánchez, and F. Pla, "Reducing training sets by ncn-based exploratory procedures," I Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA'03). Lecture Notes in Computer Science 2652, 2003, pp. 453–461.

[18] H. Fayed and A. Atiya, "A novel template reduction approach for the k-nearest neighbor method," *IEEE Transactions on Neural Networks*, vol. 20, no. 5, pp. 890–896, 2009.

[19] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 2, no. 3, pp. 408–421, 1972.

[20] F. J. Ferri, J. V. Albert, and E. Vidal, "Consideration about sample-size sensitivity of a family of edited nearest-neighbor rules," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 29, no. 4, pp. 667–672, 1999.

[21] J. S. Sánchez, F. Pla, and F. J. Ferri, "Prototype selection for the nearest neighbour rule through proximity graphs," *Pattern Recognition Letters*, vol. 18, pp. 507–513, 1997.

[22] K. Hattori and M. Takahashi, "A new edited k-nearest neighbor rule in the pattern classification problem," *Pattern Recognition*, vol. 33, no. 3, pp. 521–528, 2000.

[23] J. Sánchez, R. Barandela, A. Marques, R. Alejo, and J. Badenas, "Analysis of new techniques to obtain quality training sets," *Pattern Recognition Letters*, vol. 24, no. 7, pp. 1015–1022, 2003.

[24] N. Jankowski and M. Grochowski, "Comparison of instances selection algorithms i. algorithms survey," *Lecture Notes in Computer Science*, vol. 3070, pp. 598–603, 2004.

[25] N. Jankowski, *Data regularization*, 2000, pp. 209–214.

[26] F. Vázquez, J. Sánchez, and F. Pla, "A stochastic approach to wilson's editing algorithm." Proceedings of the Second Iberian Conference on Pattern Recognition and Image Analysis, 2005, pp. 35–42.

[27] I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Transactions on systems, Man, and Cybernetics*, vol. 6, no. 6, pp. 448–452, 1976.

[28] C. E. Brodley, "Adressing the selective superiority problem: Automatic algorithm/model class selection," Proceedings of the Tenth International Machine Learning Conference. MA, 1993, pp. 17–24.

[29] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, pp. 37–66, 1991.

[30] D. W. Aha, "Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms," *International Journal of Man-Machine Studies*, vol. 36, pp. 267–287, 1992.

[31] D. G. Lowe, "Similarity metric learning for a variable-kernel classifier," *Neural Computation*, vol. 7, no. 1, pp. 72–85, 1995.

[32] U. Lipowezky, "Selection of the optimal prototype subset for 1- nn classification," *Pattern Recognition Letters*, vol. 19, no. 10, pp. 907–918, 1998.

[33] D. R. Wilson and T. R. Martínez, "Instance pruning techniques," Proceedings of the Fourteenth International Conference (ICML'97). Morgan Kaufmann, 1997, pp. 403–411.

[34] ——, "Reduction techniques for instance-based learning algorithms," *Machine Learning*, vol. 38, pp. 257–286, 2000.

[35] M. Sebban and R. Nock, "Instance pruning as an information preserving problem," Seventeenth International Conference on Machine Learning (ICML-00). Morgan Kaufmann, 2000, pp. 855–862.

[36] K. P. Zhao, S. G. Zhou, J. H. Guan, and A. Y. Zhou, "C-pruner: An improved instance prunning algorithm." Second International Conference on Machine Learning and Cybernetics (ICMLC'03), 2003, pp. 94–99.

[37] Y. Li, Z. Hu, Y. Cai, and W. Zhang, "Support vector based prototype selection method for nearest neighbor rules," *Lecture Notes in Computer Science*, vol. 3610, pp. 528–535, 2005.

[38] X. Wang, B. Wu, Y. He, and X. Pei, "Nrmcs : Noise removing based on the mcs." Proceedings of the Seventh International Conference on Machine Learning and Cybernetics, 2008, pp. 24–30.

[39] E. Marchiori, "Class conditional nearest neighbor for large margin instance selection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 2, pp. 364–370, 2010.

[40] J. Olvera-López, J. Martínez, and J. Carrasco, "Edition schemes based on bse," *Lecture Notes in Computer Science*, vol. 3773, pp. 360–367, 2005.

[41] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learing algorithms," *Data Mining and Knowledge Discovery*, vol. 6, pp. 153–172, 2002.

[42] E. Marchiori, "Hit miss networks with applications to instance selection," *Journal of Machine Learning Research*, vol. 9, pp. 997–1017, 2008.

[43] R. M. Cameron-Jones, "Instance selection by encoding length heuristic with random mutation hill climbing." Prceedings of the Eighth Australian Joint Conference on Artificial Intelligence, 1995, pp. 99–106.

[44] J. R. Cano, F. Herrera, and M. Lozano, "Using evolutionary algorithms as instance selection for data reduction in kdd: An experimental study," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 6, pp. 561–575, 2003.

[45] V. Cerverón and F. Ferri, "Another move toward the minimum consistent subset: a tabu search approach to the condensed nearest neighbor rule," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 31, no. 3, pp. 408–413, 2001.

[46] B. Sierra, E. Lazkano, I. Inza, M. Merino, P. Larraaga, and J. Quiroga, "Prototype selection and feature subset selection by estimation of distribution algorithms. a case study in the survival of cirrhotic patients treated with tips," *Lecture Notes in Computer Science*, vol. 2101, pp. 20–29, 2001.

[47] S.-Y. Ho, C.-C. Liu, and S. Liu, "Design of an optimal nearest neighbor classifier using an intelligent genetic algorithm," *Pattern Recognition Letters*, vol. 23, no. 13, pp. 1495–1503, 2002.

[48] H. Zhang and G. Sun, "Optimal reference subset selection for nearest neighbor classification by tabu search," *Pattern Recognition*, vol. 35, pp. 1481–1490, 2002.

[49] R. Gil and X. Yao, "Evolving edited k-nearest neighbor classifiers," *International Journal of Neural Systems*, vol. 18, no. 6, pp. 459–467, 2008.

[50] S. García, J. Cano, and F. Herrera, “A memetic algorithm for evolutionary prototype selection: A scaling up approach,” *Pattern Recognition*, vol. 41, no. 8, pp. 2693–2709, 2008.

[51] N. García, J. Romero, and D. Ortiz, “A cooperative coevolutionary algorithm for instance selection for instance-based learning,” *Machine Learning*, vol. 78, no. 3, pp. 381–420, 2010.

[52] D. B. Skalak, “Prototype and feature selection by sampling and random mutation hill climbing algorithms,” Proceedings of the Eleventh International Conference on Machine Learning (ML94). Morgan Kaufmann, 1994, pp. 293–301.