

CMPE 621
PATTERN RECOGNITION
PROJECT REPORT

Instance Based Learning Algorithms
Implementation and Analysis

by

M. Toygar KARADENIZ

Bogaziçi University

1996

Introduction

Supervised learning algorithms can perform better, if some of the specific instances are stored to do the direct matching next time or if some scheme of partial matching is supported. Learning which is strengthened by a set of domain specific instances, is an interesting approach and more interestingly such kind of algorithms are said to perform acceptably well. Instance based learning algorithms simply use some instances to ease the classification task, so they both need a similarity function and describe probabilistic concepts. The probabilistic concept description ability of these algorithms constitute the real power of them.

In this project report, we will be presenting the implementation of three such kind of instance based learning algorithms. The implementation has been mainly done in MATLAB interpreter environment and the system has been trained and performance measured by using a set of 8-by-8 pixels digit images. More information about these will be given in a dedicated section.

The paper organization is as follows : After this introductory part, we will be giving the outline of each algorithm in turn. Then the training and test domains are going to be introduced. The results obtained by running these algorithms on the generated noisy mixture set of digit images, are presented next. And finally, after the conclusion part, the MATLAB source codes of the algorithms are given as an appendix.

Instance Based Learning Algorithms

Concepts In Common

Each algorithm, as it is an instance based learning algorithm, uses some components in common with the other instance based learning algorithms. These components can be summarized as follows.

- **Concept Description** : This is the actual output of the algorithms. It consists of the set of specific instances which are going to be used for the matching purposes. In our implementation, the concept description is a n-by-8 matrix where 'n' is eight times the number of saved instances. So, each pixel is saved. Moreover, there must be some way to represent which class a particular instance in the concept description belongs to. For this purpose, in this implementation an array is used which can be addressed with the index of an image in the concept description.
- **Similarity Function** : This function returns a value between 0 and 1, and shows the amount of similarity between two instances. In this project, the similarity function is implemented in such a way that it calculates the number of different pixels and return a ratio of this number to the total number of pixels examined (i.e. 64). So, a value of one means that two instances are twins.
- **Classification Function** : This function considers the result of similarity function and the current class assignment in the concept description and does the classification job. It simply assigns the instances to the class of best fit concept description member.
- **Concept Description Adder** : The main purpose of the instance based algorithm, is to reach a final concept description. So, some mechanism of addition to the temporary description is needed. This report's implementation dedicates a full MATLAB function for this purpose.
- **Concept Description Remover** : The first and second algorithms works with just adding new instances to the concept description, but never removing any. The third algorithm removes some instance when necessary in addition to adding them. So, we need to implement a mechanism to do this removal. This report's implementation does this, too.

IBL1 Algorithm

This is the simplest one among the three instance based learning algorithms which are proposed to be implemented in this project. It tries to assign the class of

concept description instance which has the highest similarity to the newly presented instance, just like the other algorithms. But the decision making about the concept description improvement is not done. Every presented instance is added to the concept description without doing any checks.

The pseudo-code of the IBL1 algorithm can be given as follows :

```

CD  $\leftarrow$  0
for each x  $\in$  training set do
  for each y  $\in$  CD
    sim[y]  $\leftarrow$  similarity(x,y)
  ymax  $\leftarrow$  some y  $\in$  CD with maximal sim[y]
  if class(x) = class(ymax)
    then classification  $\leftarrow$  correct
  else classification  $\leftarrow$  incorrect
  CD  $\leftarrow$  CD  $\cup$  {x}
    
```

IBL Algorithm 1

As it saves all of the seen instances, IBL1 algorithm has a high space complexity, but it has somewhat larger chance to find the best similar instance to a presented one. More information about these will be given in results section, but here, we can say simply that IBL1 is memorizing not learning.

IBL2 Algorithm

IBL2 algorithm is very much like the IBL1 algorithm. The only difference is that IBL2 saves only the misclassified instances. The idea behind is the usual places of mostly wrong classified instances. These are nearly always lies in the boundary between two classes. So, only if these are fully saved, the rest which are far from boundaries, can be easily deduced by using the similarity function.

The pseudo-code of the IBL2 algorithm can be given as follows :

```

CD  $\leftarrow$  0
for each x  $\in$  training set do
  for each y  $\in$  CD
    sim[y]  $\leftarrow$  similarity(x,y)
  ymax  $\leftarrow$  some y  $\in$  CD with maximal sim[y]
  if class(x) = class(ymax)
    then classification  $\leftarrow$  correct
  else
    classification  $\leftarrow$  incorrect
  CD  $\leftarrow$  CD  $\cup$  {x}
    
```

IBL Algorithm 2

IBL2 has obviously less storage requirement than the IBL1 algorithm. That is simply because it does not save right classified instances. But, it must have worse performance than IBL1, usually with the instances lying in the boundaries.

IBL3 Algorithm

The last of the three instance based learning algorithms, is IBL3. This is the most powerful one among the others. Its power comes from the point that it can do the classification process on not only the maximum similar concept description instance but also on a set of near-similar instances if the maximum of the similarity set can not exceed the predefined threshold of acceptability. If the best similar concept description instance is enough good to be acceptable, then the algorithm reduced to somewhat very like IBL2. In that condition, still IBL3 does not reduce fully to IBL2. Because, it has a different management scheme of concept description updates. When any instance's similarity performance degrades, it is just removed from the concept description, as low level of similarity performance means that this instance is not heavily used. So, it can be safely removed.

The pseudo-code of the IBL3 algorithm can be given as follows :

```

CD ← 0
for each x ∈ training set do
  for each y ∈ CD do
    sim[y] ← similarity(x,y)
    if ∃{y ∈ CD | acceptable(y)}
    then ymax ← some acceptable y ∈ CD with maximal sim[y]
    else
      i ← a randomly selected value in {1,|CD|}
      ymax ← some y ∈ CD that is the i-th most similar instance to x
    if class(x) = class(ymax)
    then classification ← correct
    else
      classification ← incorrect
      CD ← CD ∪ {x}
  for each y ∈ CD do
    if sim[y] >= sim(ymax)
    then
      update y's classification record
      if y's record is significantly poor
      then CD ← C - {y}
  
```

IBL Algorithm 3

The storage requirements of IBL3 is even lesser than IBL2, because of its removal mechanism. So, the most concentrated concept description is the concept description of IBL3. Moreover, since it tries intensively (by using acceptability) to right classify the instance, IBL3 performs better than both IBL1 and IBL2.

Domain Of Instance Sets

In this particular implementation of IBL algorithms which is presented throughout this report, the instance sets are generated by using 8-by-8 pixel digit images. Each of these is given to MATLAB interpreter as an 8-by-8 matrix of 1's and 0's. So, where there is a one that pixel is full and where there is a zero that one is empty. Some example of digit representations used in this project, are shown in **Figure 1** below :

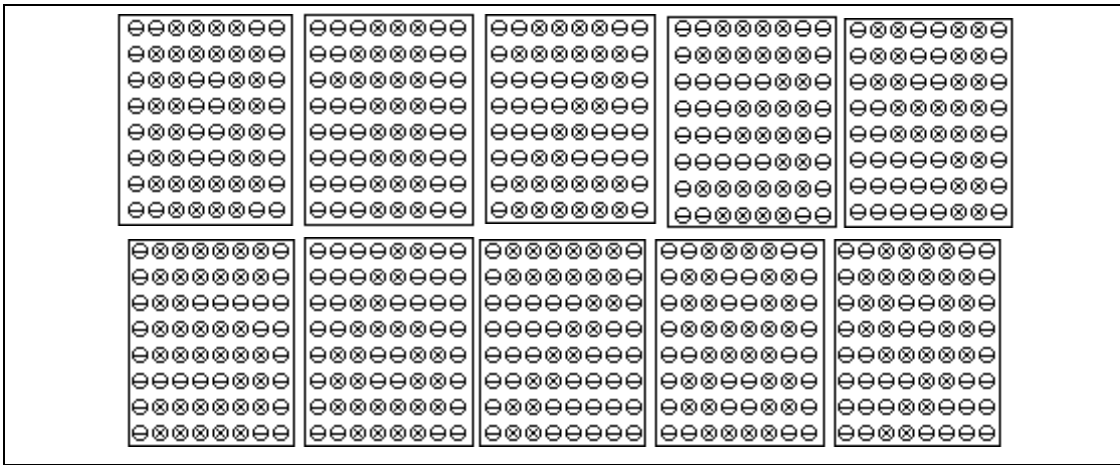


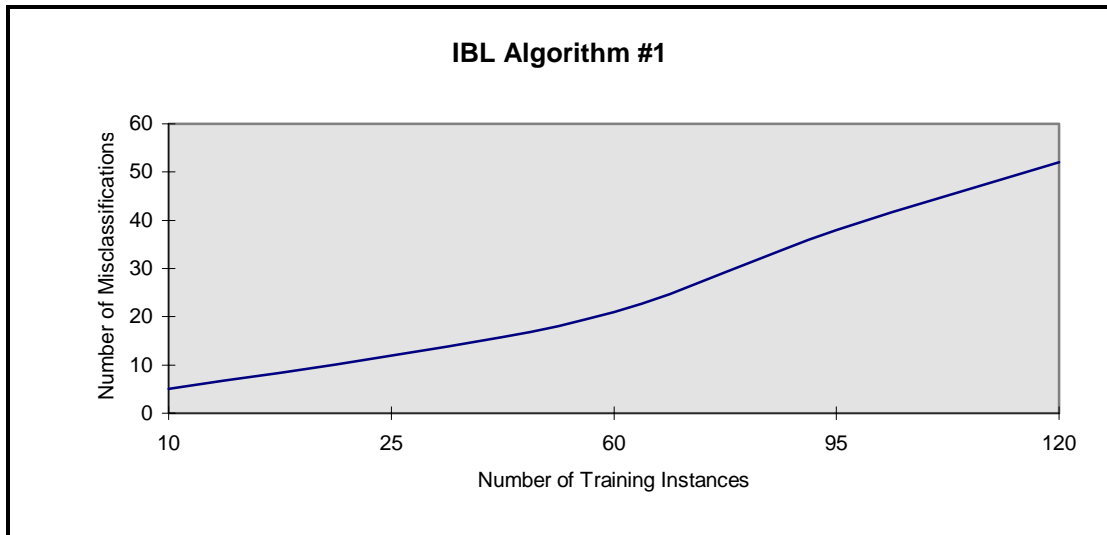
Figure 1

Figure 1 shows the perfect digit images which included no noise at all. But, in addition to these, to do the normal training we need some noisy ones, also. So, these noisy patterns are generated randomly reversing a particular amount of pixels of perfect digit images. And the training is done by this mixture set.

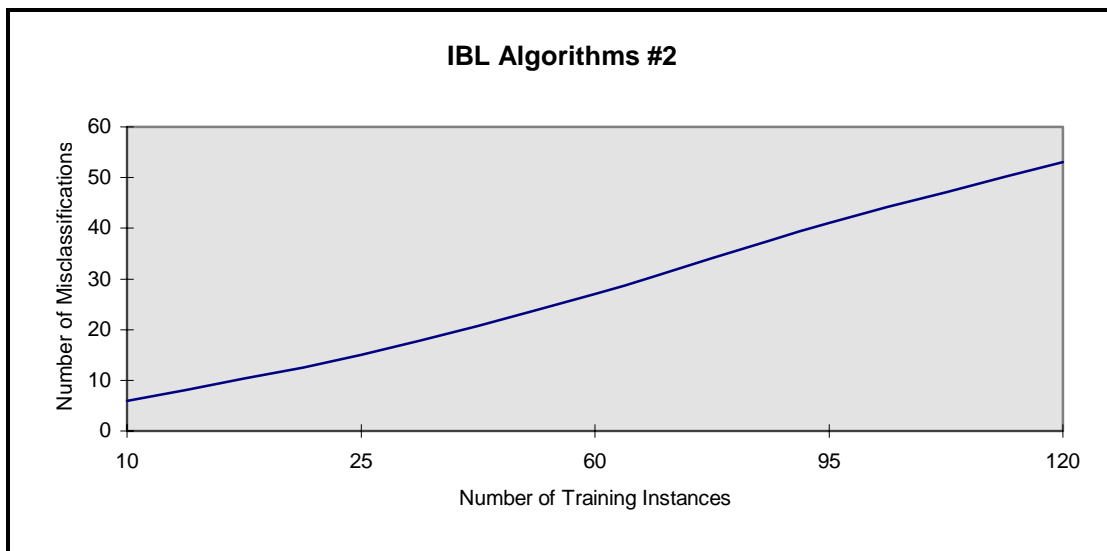
Deviations Of This Implementation From The Original Algorithms

- The original algorithms are designed for numerical values, but we used digit images instead.
- In the original algorithms, concept description is set to NULL in the beginning, but we give an initial concept description consisting of the perfect images in **Figure 1** and their shifted ones by one pixel column to the left and to the right, to ease the start of learning.
- In IBL3, when a new instance is going to be saved into the concept description, we give it a beginning classification value. That value shows how many times it has been used as the most similar instance. When that instance is entered into the concept description, it has a beginning value which is the half of maximum classification value in concept description. This pre-value is given just to prevent the system to remove that new instance from the concept description immediately. We are just giving it a free chance to stay in that concept description for a while. If it is a poor one than after sometime, it will be automatically removed in any circumstance.

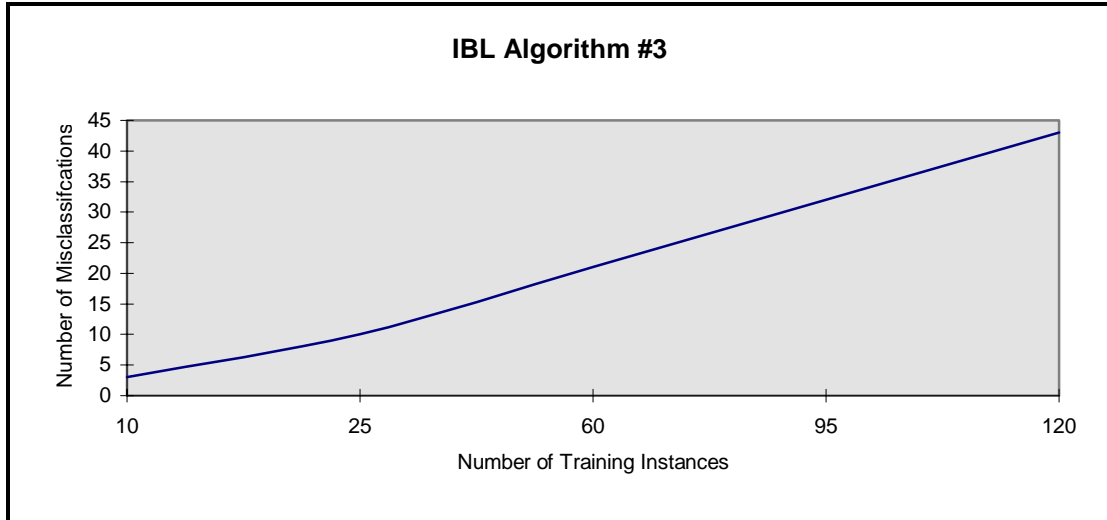
Graphical Results



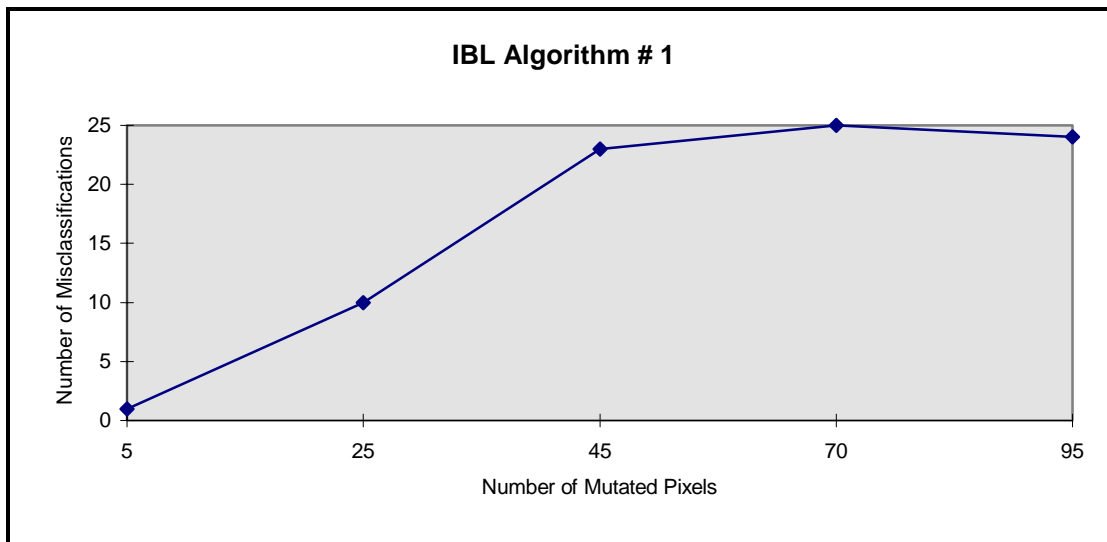
Graph 1



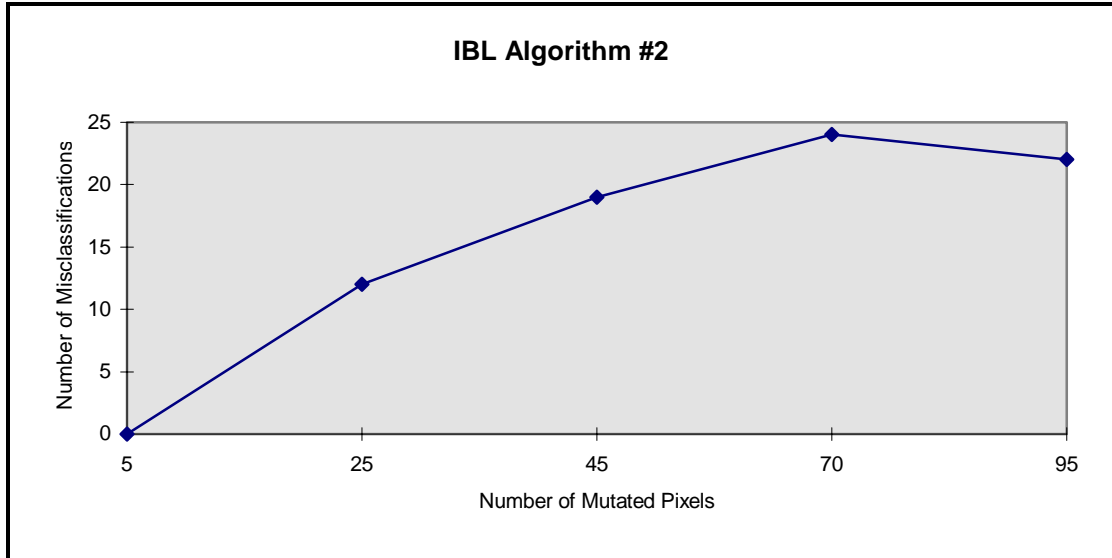
Graph 2



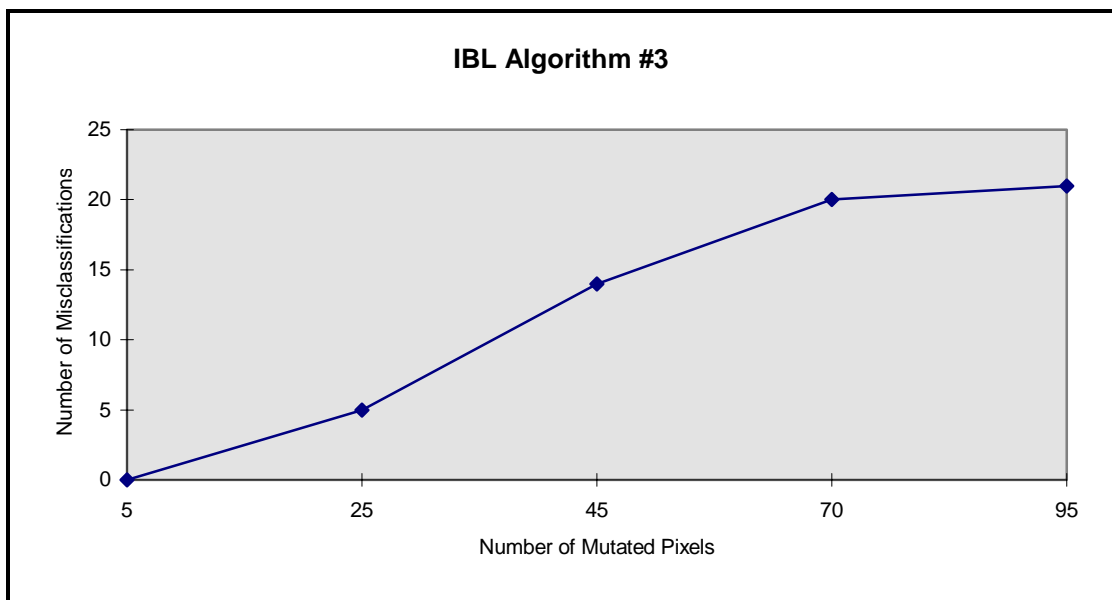
Graph 3



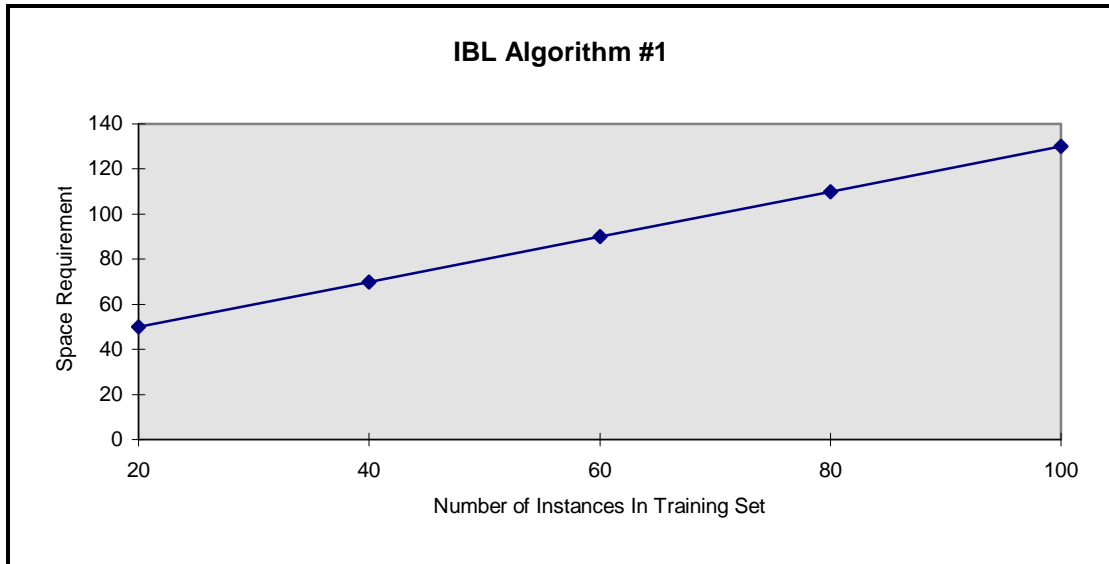
Graph 4



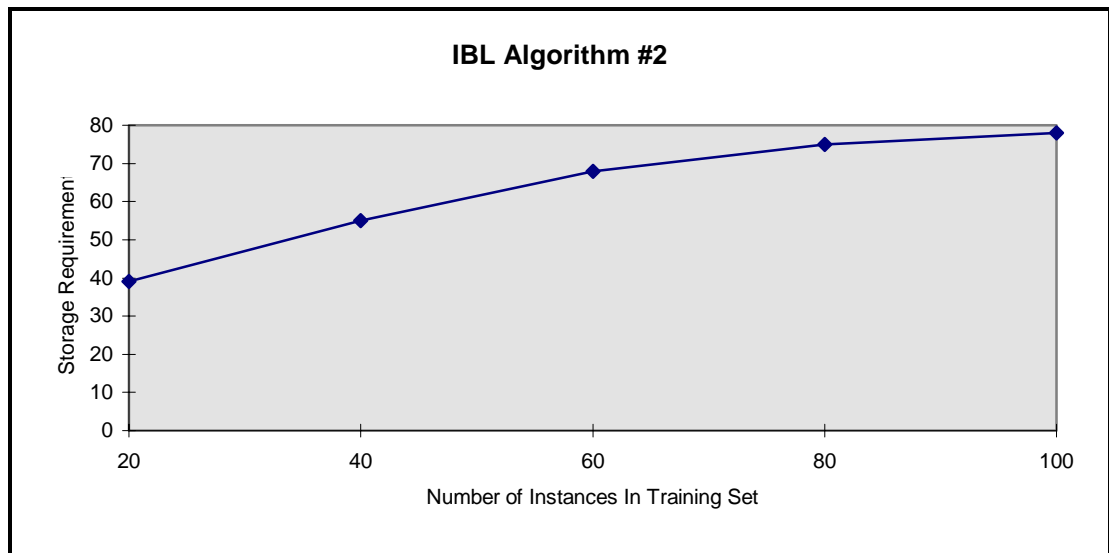
Graph 5



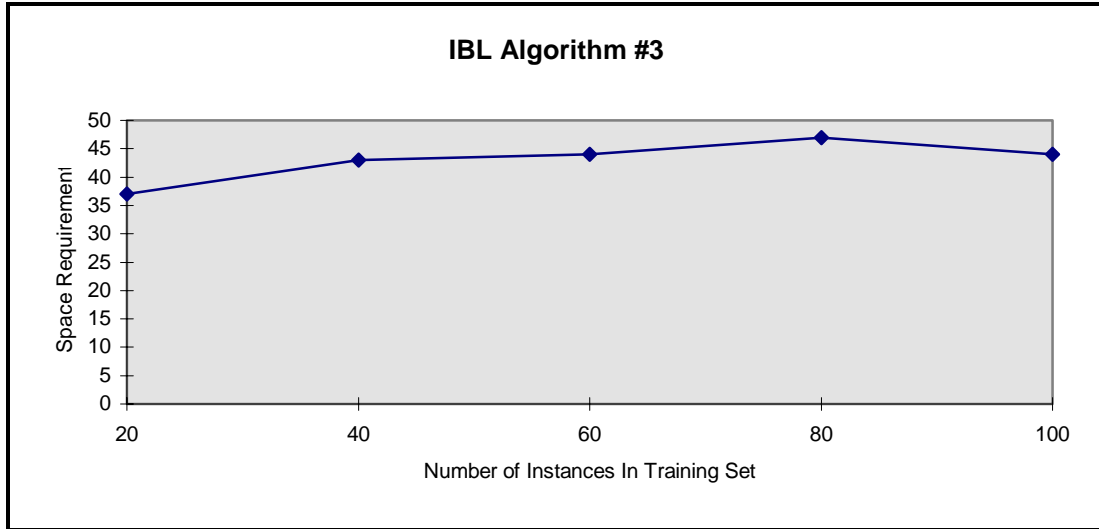
Graph 6



Graph 7



Graph 8



Graph 9

Conclusion

Throughout this project, three different instance based algorithms have been implemented and analysed according to some criteria. As the training domain, a set of perfect and noisy 8-by-8 pixel digit images have been used. Although that domain is not much exhaustive, we get some results with significant importance. To summarize shortly, if we imagine a line connecting the concepts of memorizing something and generalization of it, these algorithms are far from the generalization end and near to the memorization end. It means that they can not generalize well in very noisy domains. Moreover, their speed is not so good either. They seem to perform even worse than neural networks in some particular domains. But, the idea lying behind them is interesting enough to examine. As a final word, we can say that these instance based algorithms may be used as the last chance, whenever any other approach comes to an end. They may work then, especially if the domain of expertise is suitable to memorize.

Appendix - MATLAB Source Codes

```

add_conc.m-----
% CmpE 621 Term Project
% M.Toygar KARADENIZ

% Add To Concept Description

function y = add_conc(item,concept,conc_count);

for i = 1 : 8
    concept((conc_count-1)*8+i,:) = item(i,:);
end;

y = concept;

get_exam.m-----
% CmpE 621 Term Project
% M.Toygar KARADENIZ

% Get One Instance Function

function y = get_exam(concept,which);

which = which - 1;
result = [];
for i = 1 : 8
    result(i,:) = concept(which*8+i,:);
end;

y = result;

ibl1.m-----
% CmpE 621 Term Project
% M.Toygar KARADENIZ

% Main Body
mutate_cnt = 100;
train_cnt = 60;

% Create Training Set
conc_count = 1;
concept = [];
concept_class = [];
load_ini;

disp('Preparing Test Set ...');
train;
test_set = train_set;
test_class = train_class;
disp('Preparing Train Set ...');
train_set = [];
train_class = [];
train;

% IBL Algorithm #1

for toy = 1 : 2

```

```

IBL1_wrong_class = 0;
if toy == 2
    train_set = test_set;
    train_class = test_class;
end;
for loop = 1 : train_cnt
    mutated = get_exam(train_set,loop);
    instance_class = train_class(loop);
    similarity = [];
    for j = 1 : conc_count - 1
        similarity(j) = similar(get_exam(concept,j),mutated);
    end;
    [max_i max_j] = max(similarity);
    mutated_class = concept_class(max_j);
    if mutated_class ~= instance_class
        IBL1_wrong_class = IBL1_wrong_class + 1;
    end;
    if toy == 1
        concept = add_conc(mutated,concept,conc_count);
        concept_class(conc_count) = instance_class;
        conc_count = conc_count + 1;
    end;
end;
IBL1_wrong_class
end;

ibl2.m-----

% CmpE 621 Term Project
% M.Toygar KARADENIZ

% Main Body
mutate_cnt = 100;
train_cnt = 60;

% Create Training Set
conc_count = 1;
concept = [];
concept_class = [];
load_ini;

disp('Preparing Test Set ...');
train;
test_set = train_set;
test_class = train_class;
disp('Preparing Train Set ...');
train_set = [];
train_class = [];
train;

% IBL Algorithm #2
conc_count = 1;
concept = [];
concept_class = [];
load_ini;

for toy = 1 : 2
    IBL2_wrong_class = 0;
    if toy == 2
        train_set = test_set;
        train_class = test_class;
    end;
    for loop = 1 : train_cnt
        mutated = get_exam(train_set,loop);

```

```

instance_class = train_class(loop);
similarity = [];
for j = 1 : conc_count - 1
    similarity(j) = similar(get_exam(concept,j),mutated);
end;
[max_i max_j] = max(similarity);
mutated_class = concept_class(max_j);
if mutated_class ~= instance_class
    IBL2_wrong_class = IBL2_wrong_class + 1;
    if toy == 1
        concept = add_conc(mutated,concept,conc_count);
        concept_class(conc_count) = instance_class;
        conc_count = conc_count + 1;
    end;
end;
end;
IBL2_wrong_class
end;

ibl3.m-----

% CmpE 621 Term Project
% M.Toygar KARADENIZ

% Main Body
mutate_cnt = 100;
train_cnt = 30;
poor_limit = 0.1;
acceptable = 0.90;

% Create Training Set
conc_count = 1;
concept = [];
concept_class = [];
load_ini;

disp('Preparing Test Set ...');
train;
test_set = train_set;
test_class = train_class;
disp('Preparing Train Set ...');
train_set = [];
train_class = [];
train;

% IBL Algorithm #3
conc_count = 1;
concept = [];
concept_class = [];
load_ini;
for i = 1 : conc_count-1
    class_rec(i) = 0;
end;

for toy = 1 : 2
    IBL3_wrong_class = 0;
    if toy == 2
        train_set = test_set;
        train_class = test_class;
    end;
    for loop = 1 : train_cnt
        loop
            mutated = get_exam(train_set,loop);
            instance_class = train_class(loop);
            similarity = [];
            for j = 1 : conc_count - 1

```



```
conc_count = conc_count + 1;

load digit0_3.toy;
concept = add_conc(digit0_3,concept,conc_count);
conc_count = conc_count + 1;

load digit1_1.toy;
concept = add_conc(digit1_1,concept,conc_count);
conc_count = conc_count + 1;

load digit1_2.toy;
concept = add_conc(digit1_2,concept,conc_count);
conc_count = conc_count + 1;

load digit1_3.toy;
concept = add_conc(digit1_3,concept,conc_count);
conc_count = conc_count + 1;

load digit2_1.toy;
concept = add_conc(digit2_1,concept,conc_count);
conc_count = conc_count + 1;

load digit2_2.toy;
concept = add_conc(digit2_2,concept,conc_count);
conc_count = conc_count + 1;

load digit2_3.toy;
concept = add_conc(digit2_3,concept,conc_count);
conc_count = conc_count + 1;

load digit3_1.toy;
concept = add_conc(digit3_1,concept,conc_count);
conc_count = conc_count + 1;

load digit3_2.toy;
concept = add_conc(digit3_2,concept,conc_count);
conc_count = conc_count + 1;

load digit3_3.toy;
concept = add_conc(digit3_3,concept,conc_count);
conc_count = conc_count + 1;

load digit4_1.toy;
concept = add_conc(digit4_1,concept,conc_count);
conc_count = conc_count + 1;

load digit4_2.toy;
concept = add_conc(digit4_2,concept,conc_count);
conc_count = conc_count + 1;

load digit4_3.toy;
concept = add_conc(digit4_3,concept,conc_count);
conc_count = conc_count + 1;

load digit5_1.toy;
concept = add_conc(digit5_1,concept,conc_count);
conc_count = conc_count + 1;

load digit5_2.toy;
concept = add_conc(digit5_2,concept,conc_count);
conc_count = conc_count + 1;

load digit5_3.toy;
concept = add_conc(digit5_3,concept,conc_count);
conc_count = conc_count + 1;

load digit6_1.toy;
concept = add_conc(digit6_1,concept,conc_count);
conc_count = conc_count + 1;
```

```

load digit6_2.toy;
concept = add_conc(digit6_2,concept,conc_count);
conc_count = conc_count + 1;

load digit6_3.toy;
concept = add_conc(digit6_3,concept,conc_count);
conc_count = conc_count + 1;

load digit7_1.toy;
concept = add_conc(digit7_1,concept,conc_count);
conc_count = conc_count + 1;

load digit7_2.toy;
concept = add_conc(digit7_2,concept,conc_count);
conc_count = conc_count + 1;

load digit7_3.toy;
concept = add_conc(digit7_3,concept,conc_count);
conc_count = conc_count + 1;

load digit8_1.toy;
concept = add_conc(digit8_1,concept,conc_count);
conc_count = conc_count + 1;

load digit8_2.toy;
concept = add_conc(digit8_2,concept,conc_count);
conc_count = conc_count + 1;

load digit8_3.toy;
concept = add_conc(digit8_3,concept,conc_count);
conc_count = conc_count + 1;

load digit9_1.toy;
concept = add_conc(digit9_1,concept,conc_count);
conc_count = conc_count + 1;

load digit9_2.toy;
concept = add_conc(digit9_2,concept,conc_count);
conc_count = conc_count + 1;

load digit9_3.toy;
concept = add_conc(digit9_3,concept,conc_count);
conc_count = conc_count + 1;

for i = 1 : 10
    concept_class((i-1)*3+1) = [i-1];
    concept_class((i-1)*3+2) = [i-1];
    concept_class((i-1)*3+3) = [i-1];
end;

mutate.m-----

% CmpE 621 Term Project
% M.Toygar KARADENIZ

% Mutation Function

function y = mutate(to_mutate,how_many);

[fst_i fst_j] = size(to_mutate);
for cnt = 1 : how_many
    i = round(fst_i*rand(1));
    j = round(fst_j*rand(1));
    if i == 0
        i = 1;
    end;
    if j == 0
        j = 1;
    end;
end;

```

```

end;
if to_mutate(i,j) == 0
    to_mutate(i,j) = 1;
elseif to_mutate(i,j) == 1
    to_mutate(i,j) = 0;
end;
end;

y = to_mutate;

rem_conc.m-----

% CmpE 621 Term Project
% M.Toygar KARADENIZ

% Remove From Concept Description

function y = rem_conc(concept,item_no,conc_count);

newconcept = [];

for i = 1 : item_no - 1
    for j = 1 : 8
        newconcept((i-1)*8+j,:) = concept((i-1)*8+j,:);
    end;
end;

for i = item_no + 1 : conc_count - 1
    for j = 1 : 8
        newconcept((i-1)*8+j,:) = concept(i*8+j,:);
    end;
end;

y = newconcept;

similar.m-----

% CmpE 621 Term Project
% M.Toygar KARADENIZ

% Similarity Function

function y = similar(first,second);

[fst_i fst_j] = size(first);
[sec_i sec_j] = size(second);

if ((fst_i ~= sec_i)|(fst_j ~= sec_j))
    disp('Sizes are different ...');
    pause;
    quit;
end;

same = 0;
for i = 1 : fst_i
    for j = 1 : fst_j
        if first(i,j) == second(i,j)
            same = same + 1;
        end;
    end;
end;

all = fst_i * fst_j;
y = same / all;

train.m-----

```

```
% CmpE 621 Term Project
% M.Toygar KARADENIZ

% Create Sample Sets

for i = 1 : train_cnt
    which = round(rand(1)*(conc_count-1));
    if which == 0
        which = 1;
    end;
    for j = 1 : 8
        temp_obj(j,:) = concept((which-1)*8+j,:);
    end;
    mutated = mutate(temp_obj,mutate_cnt);
    for j = 1 : 8
        train_set((i-1)*8+j,:) = mutated(j,:);
    end;
    train_class(i) = concept_class(which);
end;
```

an example of digit representation-----

```
0 0 1 1 1 1 0 0
0 1 1 1 1 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 0 0 1 1 0
0 1 1 1 1 1 1 0
0 0 1 1 1 1 0 0
```