Statistical Methods in AI

Mini Project I

Pratyush Nigam | 200901063

Contents

Introduction

The aim of this project is to design classifiers and subsequent selection of classification algorithms. The design/selection of features and classifiers depend on the nature of the data-set. The purpose of the mini-project is to fulfill the challenges presented above and thereby implement and find a successful way of doing that.

Data-Sets

We use the following five data-sets for completing the project, four of which have extracted features and one contains raw data -

1) **Iris Dataset**: This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.
Predicted attribute: class of iris plant.
Attribute Information:
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

[1]

2) **Wine**: These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivates. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The attributes are -
1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash

5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10)Color intensity
11)Hue
12)OD280/OD315 of diluted wines
13)Proline

[2]

3) **Yeast**: Predicted Attribute: Localization site of protein. (non-numeric).

The references below describe a predecessor to this data set and its development. They also give results (not cross-validated) for classification by a rule-based expert system with that version of the data-set.

Reference: "Expert Sytem for Predicting Protein Localization Sites in Gram-Negative Bacteria", Kenta Nakai & Minoru Kanehisa, PROTEINS: Structure, Function, and Genetics 11:95-110, 1991.

Reference: "A Knowledge Base for Predicting Protein Localization Sites in Eukaryotic Cells", Kenta Nakai & Minoru Kanehisa, Genomics 14:897-911, 1992.

Attribute Information:
1. Sequence Name: Accession number for the SWISS-PROT database
2. mcg: McGeoch's method for signal sequence recognition.
3. gvh: von Heijne's method for signal sequence recognition.
4. alm: Score of the ALOM membrane spanning region prediction program.
5. mit: Score of discriminant analysis of the amino acid content of the N-terminal region (20 residues long) of mitochondrial and non-mitochondrial proteins.
6. erl: Presence of "HDEL" substring (thought to act as a signal for retention in the endoplasmic reticulum lumen). Binary attribute.
7. pox: Peroxisomal targeting signal in the C-terminus.
8. vac: Score of discriminant analysis of the amino acid content of vacuolar and extracellular proteins.
9. nuc: Score of discriminant analysis of nuclear localization signals of nuclear and non-nuclear proteins.
[3]

4) **P-R**: The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical

moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. We typically train on the first 16000 items and then use the resulting model to predict the letter category for the remaining 4000. See the article cited above for more details.

Attribute Information:

1. lettr capital letter (26 values from A to Z)
2. x-box horizontal position of box (integer)
3. y-box vertical position of box (integer)
4. width width of box (integer)
5. high height of box (integer)
6. onpix total # on pixels (integer)
7. x-bar mean x of on pixels in box (integer)
8. y-bar mean y of on pixels in box (integer)
9. x2bar mean x variance (integer)
10. y2bar mean y variance (integer)
11. xybar mean x y correlation (integer)
12. x2ybr mean of x * x * y (integer)
13. xy2br mean of x * y * y (integer)
14. x-ege mean edge count left to right (integer)
15. xegvy correlation of x-ege with y (integer)
16. y-ege mean edge count bottom to top (integer)
17. yegvx correlation of y-ege with x (integer)

[4]

5) **Digit Dataset**: Handwritten digits from 0 through 9.

Classifiers

A. Distance based classifiers

1) k-Nearest Neighbor -

In pattern recognition, the *k*-nearest neighbor algorithm (*k*-NN) is a method for classifying objects based on closest training examples in the feature space. *k*-NN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. The *k*-nearest neighbor algorithm is amongst the simplest of all learning algorithms: an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its *k* nearest neighbors (*k* is a positive integer, typically small). The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. [5]

2) Single Nearest Neighbor -

In a k-NN classifier, if the value of *k* = 1, then the object is simply assigned to the class of its

nearest neighbor.

3) Weighted k-NN -
Using the algorithm of k-NN classifier, when while determining the class of a sample, if the weight i.e. reciprocal of Euclidean/Manhattan/Minnowski is also added, then it becomes a weighted k-NN.

4) Nearest Mean –
This algorithm differs from k-NN in terms that the mean of all the distances is found out and then when the class is determined, then the mean gets used instead of the training data.

B. Linear Classifiers

1) Perceptron Learning:
a) Batch learning –
The batch perceptron algorithm for finding a solution vector can be stated very simply : the next weight vector is obtained by adding some multiple of the sum Batch of the misclassified samples to the present weight vector. We use the term "batch" training to refer to the fact that (in general) a large group of samples is used when computing each weight update.[6]

b) Single sample Fixed increment –
The fixed-increment Perceptron rule is the simplest of many algorithms that have been proposed for solving systems of linear inequalities. Geometrically, its interpretation in weight space is particularly clear. Since $\mathbf{a}(k)$ misclassifies $\mathbf{y}k$, $\mathbf{a}(k)$ is not on the positive side of the $\mathbf{y}k$ hyper plane $\mathbf{a}t\mathbf{y}k = 0$. The addition of $\mathbf{y}k$ to $\mathbf{a}(k)$ moves the weight vector directly to ward and perhaps across this hyperplane. Whether the hyperplane is crossed or not, the new inner product $\mathbf{a}t(k+1)\mathbf{y}k$ is larger than the old inner product $\mathbf{a}t(k)\mathbf{y}k$ by the amount _$\mathbf{y}k$_2, and the correction is clearly moving the weight vector in a good direction.[6]

c) Single sample Variable increment with margin –
The fixed increment rule can be generalized to provide a variety of related algorithms. The variant introduces a *variable increment* $\eta(k)$ and a margin b, and calls for a correction whenever variable
$\mathbf{a}t(k)\mathbf{y}k$ fails to exceed the margin. The update is given by increment
$\mathbf{a}(1)$ arbitrary
$\mathbf{a}(k + 1) = \mathbf{a}(k) + \eta(k)\mathbf{y}k \; k \geq 1,$
where now $\mathbf{a}t(k)\mathbf{y}k \leq b$ for all k. [6]

2) Minimum Squared error classifier using Pseudo inverse method –
Criterion function considered so far involve only misclassified samples. Now consider a criterion that involves *all* of the samples,not just misclassified ones. Previously we were interested in making all of the inner products $\mathbf{a}t\mathbf{y}i$ positive. Now try to make $\mathbf{a}t\mathbf{y}i = bi$ where bi are some arbitrarily specified positive constants. Thus replace the problem of solving a set of linear inequalities with more stringent but *better understood problem* of finding a solution to a set of linear equations. [6]

3) LMS procedure for minimum squared error learning -

For the case where atyi = b is a considered as a base case, this particular algorithm is called LMS for minimum squared error learning.

Implementations Details

Pseudo Inverse Code:

```
nc2 = zeros(c, c, n_sample(2));
for i = 1:c
    for j = i+1:c
        subtrg = zeros(1, n_sample(2));
        minimum = 0;

        % Finding the subset which belong to 'i'th and 'j'th class
        for k = 1:tr
            if (trg(k,1)==i || trg(k,1)==j)
                x=[1,trg(k,2:end)];
                subtrg = [subtrg; x];
                minimum = minimum + 1;
                if(trg(k,1) == j)
                    x = -x;
                end
            end
        end

        mod_min = minimum;
        YY = zeros(mod_min, n_sample(2));
        % Optimization of 'a' vector
        for k = 1:mod_min
            mod_i = mod(k, mod_min)+1;
            Y=[1,subtrg(mod_i,2:end)];
            if(subtrg(mod_i, 1) == j)
                Y = -Y;
            end
            YY(mod_i, :) = Y(1,:);
            %size(YY)
            e = ones(mod_min, 1);
            %size(e)
            e = e*b;

        end

        a = pinv(YY)*e;
        nc2(i,j,:) = a(1,:);
    end
end
```

k-NN Code:

```
for i = 1:b
    d = zeros(a, 2);
```


```
    for j = 1:a
        sum = 0;
        d(j,1)=trg(j,1);
        for h=2:n_sample(2)
            sum = sum + (abs(tst(i,h) - trg(j,h)))^p;
        end
        d(j,2)=sum^1/p;
    end
    class=zeros(1,c);
    d = sortrows(d,2);
    for j=1:k
        class(d(j,1)) = class(d(j,1)) + 1;
    end
    [max_val index]=max(class);
    if(tst(i,1) == index)
        accuracy = accuracy + 1;
    end
end
accuracy = accuracy/b;
```

Single Sample Variable Increment Code:

```
nc2 = zeros(c, c, n_sample(2));
for i = 1:c
    for j = i+1:c
        subtrg = zeros(1, n_sample(2));
        a = zeros(1, n_sample(2));
        constraint = 12000;
        n_misclassified = 0;
        minimum = 0;

        % Finding the subset which belong to 'i'th and 'j'th class
        for k = 1:tr
            if (trg(k,1)==i || trg(k,1)==j)
                subtrg = [subtrg; trg(k,:)];
                minimum = minimum + 1;
            end
        end

        mod_min = minimum;

        % Optimization of 'a' vector
        for k = 1:constraint
            mod_i = mod(k, mod_min)+1;
            Y=[1,subtrg(mod_i,2:end)];
            if(subtrg(mod_i, 1) == j)
                Y = -Y;
            end
            if(a*Y'<= b)
                n_misclassified = n_misclassified + 1;
                a = a + Y*(1/k); % 1/k is the decreasing function
            end
            if(mod_i == mod_min)
                if(n_misclassified < minimum)
                    minimum = n_misclassified;
                    best_a = a;
                end
```


```
                if n_misclassified==0
                    break;
                end
                n_misclassified=0;
            end
        end

        nc2(i,j,:) = best_a(1,:);
    end
end
```

Single Sample Variable Increment Code:

Same as the above, but for the multiplicand will be 1 instead of 1/k.

LMS Code:

```
nc2 = zeros(c, c, n_sample(2));
for i = 1:c
    for j = i+1:c
        subtrg = zeros(1, n_sample(2));
        a = zeros(1, n_sample(2));
        constraint = 12000;
        n_misclassified = 0;
        minimum = 0;

        % Finding the subset which belong to 'i'th and 'j'th class
        for k = 1:tr
            if (trg(k,1)==i || trg(k,1)==j)
                subtrg = [subtrg; trg(k,:)];
                minimum = minimum + 1;
            end
        end

        mod_min = minimum;

        % Optimization of 'a' vector
        for k = 1:constraint
            mod_i = mod(k, mod_min)+1;
            Y=[1,subtrg(mod_i,2:end)];
            if(subtrg(mod_i, 1) == j)
                Y = -Y;
            end
            if(a*Y'<= b)
                n_misclassified = n_misclassified + 1;
                a = a + Y*(1/k)*(b - a*Y'); % 1/k is the decreasing
function
            end
            if(mod_i == mod_min)
                if(n_misclassified < minimum)
                    minimum = n_misclassified;
                    best_a = a;
                end
                if n_misclassified==0
                    break;
```


```
                end
                n_misclassified=0;
            end
        end

        nc2(i,j,:) = best_a(1,:);
    end
end
```

1-NN:

Specific case of k-NN where k=1.

Weighted k-NN:

```
for i = 1:b
    d = zeros(a, 2);
    for j = 1:a
        sum = 0;
        d(j,1)=trg(j,1);
        for h=2:n_sample(2)
            sum = sum + (abs(tst(i,h) - trg(j,h)))^p;
        end
        d(j,2)=sum^1/p;
    end
    class=zeros(1,c);
    d = sortrows(d,2);
    for j=1:k
        class(d(j,1)) = class(d(j,1)) + 1/d(j,2);
    end
    [max_val index]=max(class);
    if(tst(i,1) == index)
        accuracy = accuracy + 1;
    end
end
```

Nearest Mean Code:

```
mean = zeros(c,n_sample(2));
count = zeros(c,1);
for i=1:a
    count(trg(i))=count(trg(i))+1;
    for j=2:n_sample(2)
        mean(trg(i,1),j) = trg(i,j) + mean(trg(i,1),j);
    end
end

for i = 1:b
    d = zeros(a, 2);
    for j = 1:a
        sum = 0;
        d(j,1)=trg(j,1);
        for h=2:n_sample(2)
            sum = sum + (abs(tst(i,h) - mean(j,h)))^p;
        end
        d(j,2)=sum^1/p;
```


```
    end
    class=zeros(1,c);
    d = sortrows(d,2);
    for j=1:k
        class(d(j,1)) = class(d(j,1)) + 1;
    end
    [max_val index]=max(class);
    if(tst(i,1) == index)
        accuracy = accuracy + 1;
    end
end
```

Experimental Results and Analysis

A. Distance based Classifier:

1) Single Nearest Neighbour:

Training Data	Distance Type	Iris	Letter	Wine	Yeast
60.00%	Euclidean	92.67%	67.46%	77.23%	41.66%
	Manhatten	93.33%	61.14%	71.94%	51.69%
	Minkowski p = 3	98.22%	69.30%	78.83%	57.19%
	Minkowski p = 5	94.11%	67.14%	64.38%	42.18%
80.00%	Euclidean	85.67%	63.35%	85.56%	53.20%
	Manhatten	90.33%	68.69%	80.56%	46.80%
	Minkowski p = 3	93.67%	62.30%	66.67%	60.17%
	Minkowski p=5	95.55%	71.20%	76.42%	38.82%

2) K-Nearest Neighbor:

Training Data	Distance Type	K	Iris	Letter	Wine	Yeast
60.00%	Euclidean	3	96.67%	67.04	75.22	54.37
		5	92.33%	66.72	63.44	52.22
	Manhatten	3	96.67%	65.28	75.22	51.88
		5	96.12%	65.92	77.22	57.23
	Minkowski	3	93.33%	66.72	68.28	52.20

	p = 3	5	95.00%	68.89	68.06	55.22
	Minkowski p=5	3	96.67%	66.88	59.72	54.55
		5	93.33%	67.68	66.67	55.22
80.00%	Euclidean	3	93.33%	67.09	72.22	55.89
		5	93.33%	70.61	66.67	52.86
	Manhatten	3	93.33%	59.11	69.44	49.83
		5	96.67%	71.25	72.22	59.26
	Minkowski p = 3	3	96.67%	65.81	61.11	50.51
		5	90.00%	72.84	69.44	53.20
	Minkowski p=5	3	93.33%	64.86	77.78	50.51
		5	95.50%	65.50	73.89	60.27

3) Weighted K-NN:

Training Data	Distance Type	K	Iris	Letter	Wine	Yeast
60.00%	Euclidean	3	98.33	67.04	79.17	38.05
		5	95	64.48	59.72	44.11
	Manhatten	3	95	65.28	73.61	39.90
		5	93.33	62.08	79.17	45.79
	Minkowski p = 3	3	95	64.96	68.06	38.05
		5	93.33	60.32	59.72	48.48
	Minkowski p=5	3	95	63.68	76.39	41.41
		5	93.33	61.44	70.83	44.61
80.00%	Euclidean	3	96.67	64.86	75	38.05
		5	96.67	65.81	66.67	42.42
	Manhatten	3	96.67	68.05	75	37.37
		5	96.67	61.02	83.33	48.15
	Minkowski p = 3	3	100	68.05	75	42.09
		5	90	62.94	61.11	44.44
	Minkowski p=5	3	96.67	63.58	72.22	37.04
		5	100	64.86	61.11	46.13

4) Nearest Mean:

Training Data	Distance Type	Iris	Letter	Wine	Yeast
	Euclidean	98.33%	41.6	68.06%	52.19

60.00%	Manhatten	91.67%	44	75	51.01
	Minkowski p = 3	93.33%	42.4	70.83	48.99
	Minkowski p=5	90.00%	37.28	75	49.49
80.00%	Euclidean	83.33%	51.12	75.00%	50.51
	Manhatten	86.67%	42.49	77.78	50.17
	Minkowski p = 3	96.67%	42.17	69.44	49.16
	Minkowski p=5	96.67%	40.89	80.56	48.15

B.Linear Classifier:

1)Perceptron Learning:

a.Batch Learning:

Training Data	Iris	Letter	Wine	Yeast
60.00%	61.25	98.71	82.51	35.48
80.00%	65.33	99.15	86.56	33.33

b.Single Sample Fixed increment:

Training Data	Iris	Letter	Wine	Yeast
60.00%	63.33	89.67	48.39	29.82
80.00%	33.33	95.51	57.11	34.29

c.Single Sample Variable increment with Margin:

Training Data	Iris	Letter	Wine	Yeast
60.00%	31.67	51	61.11	29.63
80.00%	70	41	72.22	30.98

2) Minimum Square Error:

Training Data	Iris	Letter	Wine	Yeast
60.00%	66.67	50.45	66.89	54.21
80.00%	50.33	65.66	51.22	64.33

3) LMS:

Training Data	Iris	Letter	Wine	Yeast
60.00%	34.37	0.11	31.29	36.20
80.00%	31.33	0	35.56	22.21

Analysis –

As from the above tables, we are able to see that for –

a) Iris Dataset – We have the best accuracy when we use weighted k-NN
b) Letter Dataset – The best accuracy comes when Batch learning is employed
c) Wine Dataset - The best accuracy comes when Batch learning is employed
d) Yeast Dataset – The best accuracy comes when k-NN is employed.

This experimental results show that different kinds of classifiers work best for different kinds of dataset. However, when the number of classes is increased, there is an increase in the misclassification of samples as well. Therefore, the selection of a suitable classifier is of eminent importance.

Conclusion

The conclusion can be derived that the learning is heavily based upon the kind of classifier and hence algorithm used as well as the nature of the training data. It is to be noted that all the algorithms try to approach the separating boundary, however to achieve an ideal situation, some or the other stakeholders involved get sacrificed in each one of the algorithms since each one of them is based on a particular assumption.

References

[1] - http://archive.ics.uci.edu/ml/datasets/Iris
[2] - http://archive.ics.uci.edu/ml/datasets/Wine
[3] - http://archive.ics.uci.edu/ml/datasets/Yeast
[4] - http://archive.ics.uci.edu/ml/datasets/Letter+Recognition
[5] - http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm
[6] - Pattern Classification, 2nd Edition - Duda and Hart