CSCE 633: Machine Learning

Assignment #3

Submitted by,

Raaja Prabhu Uma Jaganathan

824006931

Introduction

Instance-based classifiers are classifiers that classify a test-data to be of same class as the most similar training sample. The similarity is measured considering the samples as points in an n-dimensional (n represents the number of input attributes) space and distance is computed between the test-data and all the training samples, the sample with the least distance is considered to be the most similar sample. This, in a way, is a good measure because the variance in all the attributes is considered while computing the distance; on the other hand, if an attribute is irrelevant, considering the difference in values of this attribute could essentially result in noise. Hence using feature selection could help us by removing the attributes that do not significantly contribute in classification. Similarly, feature weighting could be used to weight the features based on their relevance in classification and the proportion of the weights could be used in computing the distance, leading to a more accurate classification. The disadvantage of this method is that there isn't a hypothesis that is learnt and to classify a sample, distances to all the training samples are to be computed, which could be computationally intensive if the number of samples is huge.

Implementation details

Data representation

I represented data in a 3-dimensional array of strings; the first dimension is of size 10, used to represent data for each fold of cross-validation. The training-data for each fold are represented in the 2-dimensional arrays of strings, each row representing an input sample, each column corresponds to the sample's value for each attribute.

Normalization

I normalized all the continuous attributes between 0 & 1 for a fair impact of the all features on distance or similarity.

Distance computation

I considered sum of squares of the differences in each attribute to represent the distance. The difference for continuous attribute is just the difference in their normalized values, while the difference for discrete attributes is 0.5 if they are not equal and 0 otherwise. I used 0.5 for discrete attributes because, in comparison with continuous attributes, a value of 1 would mean that the samples represent the extremes of the continuous attribute, which could be a drastic measure. Hence I felt a value between 0.4 and 0.7 would be a good estimate of the difference. This showed improvement in datasets that had both continuous and discrete attributes (Thoracic Surgery dataset), while datasets having only discrete or continuous attributes won't be affected by this weighting.

Classification

For K-NN classification, I compute the distances from the test-sample to all the training samples and consider only the K least of them and weight the corresponding classes proportional to the inverse of their distances, then classify the test-sample as the class having the highest weight (indicating the most likely class of the test-sample). Note that the distance I used here was the square of the distance from Cartesian geometry and in most cases, this performed better than the using the actual distance itself.
I also tried classifying the test-sample to the class of the majority of the K most similar (or closest) samples, but the performance wasn't as good as the previous approach.
I experimented with different K values in {1, 3, 5, 7, 9, 11, 13, 15}.

Feature Selection

I tried Stepwise Forward Selection (SFS) and Stepwise Backward Elimination (SBE).

Stepwise Forward Selection

I had set aside one-third of the training-samples for validation of SFS. Below is the pseudo code for my implementation:

```
SFS(traing-samples):
    1. CurrentAccuracy = 0
    2. while(true)
        1. TempAccuracy = 0
        2. For each unselected feature 'f'
            1. Consider 'f' to be selected
            2. Classify the validation set with the set of selected features
            3. If the classification-accuracy > TempAccuracy
                1. TempAccuracy = classification-accuracy
                2. BestFeature = f
            4. Unselect 'f', so that it isn't considered for classification
        3. If TempAccuracy > CurrentAccuracy
            1. CurrentAccuracy = TempAccuracy
            2. Mark BestFeature to be selected for further classification
        4. Else
            1. Break the while loop
    3. Return the set of features selected for classification
```

Stepwise Backward Elimination

Here again, I had set aside one-third of the training-samples for validation of SBE. Below is the pseudo code for my implementation:

```
SBE(traing-samples):
    1. Select all the features for classification
    2. CurrentAccuracy = classification accuracy on the validation-set
    3. while(true)
        1. TempAccuracy = 0
        2. For each selected feature 'f'
            1. Unselect 'f', so that it isn't considered for classification
            2. Classify the validation set with the set of selected features
            3. If the classification-accuracy > TempAccuracy
                1. TempAccuracy = classification-accuracy
                2. FeatureToDrop = f
            4. Add 'f' back, so that it is considered for further classification
        3. If TempAccuracy > CurrentAccuracy
            1. CurrentAccuracy = TempAccuracy
            2. Drop FeatureToDrop so that it isn't considered for further classification
        4. Else
            1. Break the while loop
    4. Return the set of features selected for classification
```

NTGrowth

I experimented with NTGrowth, as suggested in the publication. Below is the detailed description of my implementation. The algorithm resulted in significant reduction of the training-samples' size (to about 10% of the actual size), however the classification accuracy declined; more on this in results section.

Pseudo code

```
NTGrowth(traing-samples):
    1. For each sample 't' in the set of training samples
        1. If the set of acceptable samples from the so far visited training samples is empty
            1. Update the classification records of a randomly chosen number of closest samples to 't'
        2. Else
            1. Select the closest acceptable sample 'c' from the so far visited training samples
            2. If the classes of 'c' & 't' are the same, then discard 't'
            3. Update the classification records of all the non-discarded training-samples visited so far
        3. For all the non-discarded training-samples visited so far
            1. If the classification accuracy is significantly greater than the sample's class' frequency
                1. Accept the sample
            2. Elseif the classification accuracy is significantly lesser than the sample's class' frequency
                1. Discard the sample
            3. Else
                1. Make the sample not-acceptable, without discarding it
    2. Return all the acceptable samples to be used as training-samples for further classification
```

Classification Accuracy

I used 10-fold cross-validation to report the classification accuracies; I split the data into 10 subsets, I then used 1 subset (in turns) to test the Instance-based Learner, tuned (with NTGrowth and feature selection) based on rest of the data.

Results

Experiment with 'k'

Accuracies	**K = 1**	**K = 3**	**K = 5**	**K = 7**	**K = 9**	**K = 11**	**K = 13**	**K = 15**
Car Evaluation	78.68	84.89	87.03	89.01	90.56	92.55	<u>93.90</u>	93.68
Iris	<u>95.73</u>	<u>95.73</u>	95.22	95.22	94.6	94.6	95.22	95.22
Chess (King-Rook vs. King-Pawn)	90.57	96.28	97.14	97.42	96.85	<u>97.71</u>	97.14	97.14
Tic-Tac-Toe Endgame	80.55	89.12	92.36	94.53	95.97	96.66	97.00	<u>98.69</u>
Wine	94.39	96.03	96.99	<u>97.58</u>	97.51	97.51	97.51	96.96

Different datasets performed better for different values of 'k', this could indicate that: for datasets performing better with higher 'k', there isn't a clear boundary (or hyperplane) that separates the samples from different classes, hence more number of neighboring samples are needed to classify a test-sample.

Experiment with feature selection

I experimented feature selection with a fixed K value (=5). I also considered using the best K value (in terms of accuracy from the previous experiment), but then after feature selection, some-other K could perform better than the earlier K, hence to analyze the performance of Feature selection alone, I used a single value for K = 5.

Stepwise Forward Selection

SFS	Total number of attributes	Average number of attributes selected	Classification accuracy with SFS	Classification accuracy without SFS
Car Evaluation	6	5	94.77	87.03
Iris	4	2.3	93.33	95.22
Chess (King-Rook vs. King-Pawn)	36	10.2	96.42	97.14
Thoracic Surgery	16	2	83.95	80.66
Tic-Tac-Toe Endgame	9	1.3	62.86	92.36
Wine	13	3.4	91.97	96.99

With SFS, the accuracy of 'Car Evaluation' and 'Thoracic Surgery' has increased while it has decreased a little for 'Iris' and 'Wine', which can because of using a validation set to select the features. We could also see that the number of chosen attributes for 'Thoracic Surgery' was just 2 while it had 16, showing significant drop in number of attributes considered for classification. The accuracy has decreased significantly for 'Tic-Tac-Toe' dataset, which could because SFS considers features one-by-one, hence the accuracy might not improve significantly with a single feature alone but with a set of features, we could also notice that the number of selected features was around 1, adding to our reasoning. We could also observe that for 'Wine' dataset, with just 3.4 attributes (on average) out of the 13, the classification accuracy hasn't dropped significantly. Similarly, for the 'Chess' dataset, with just 10.2 features out of 36, the accuracy didn't reduce drastically; the non-significant drop in the accuracy could suggest over-fitting for the validation set used.

Stepwise Backward Elimination

SBE	Total number of attributes	Average number of attributes selected	Classification accuracy with SBE	Classification accuracy without SBE
Car Evaluation	6	5	94.77	87.03
Iris	4	3.3	94.04	95.22
Chess (King-Rook vs. King-Pawn)	36	28.9	97.71	97.14
Thoracic Surgery	16	13.6	81.15	80.66
Tic-Tac-Toe Endgame	9	8.8	92.08	92.36
Wine	13	12	95.98	96.99

We could see that, on average, more number of attributes being selected, when compared with SFS. The reasoning could be that: here we are trying to remove features one-by-one to obtain a good accuracy, hence we could get stuck in a local maxima (same argument would hold for SFS). For 'Tic-Tac-Toe' dataset, the reduction in the number of attributes wasn't significant, suggesting that all the attributes were significant in classification.

Experiment with NTGrowth

NTGrowth	Average number of training samples	Average number of samples after NTGrowth	Classification accuracy with NTGrowth	Classification accuracy without NTGrowth
Car Evaluation	1555.2	228.2	80.8	87.03
Iris	135	19.5	95.44	95.22
Chess (King-Rook vs. King-Pawn)	2876.4	383.5	90.14	97.14
Thoracic Surgery	423	11.4	84.4	80.66
Tic-Tac-Toe Endgame	862.2	130.9	81.51	92.36
Wine	160.2	23.6	93.0	96.99

NTGrowth helped in increasing the accuracies with 'Iris' and 'Thoracic surgery' datasets, but accuracy decreased with other datasets. One important thing to note here is that the training-samples size reduced to about 15% of the actual size, which is huge improvement. For 'Thoracic surgery' dataset, with just 2.5% of the data, the classification accuracy improved with NTGrowth. Hence NTGrowth is extremely efficient in removing the redundant or noisy samples while not worsening the classification accuracy badly.

Comparison with other classification algorithms

	IBL accuracy (best version)	95% confidence interval	Neural Network accuracy	95% confidence interval	ID3 accuracy	95% confidence interval	p value from T-test (IBL vs NN)	p value from T-test (IBL vs ID3)
Car Evaluation	94.77	0.9423 to 0.9530	92.89	0.9093 to 0.9484	90.9063	0.8618 to 0.9208	0.332587	0.085207
Iris	95.73	0.9407 to 0.9738	95.83	0.9419 to 0.9746	84.2302	0.7566 to 0.9762	0.488592	0.104834
Chess (King-Rook vs. King-Pawn)	97.71	0.9744 to 0.9797	54.85	0.5206 to 0.5763	99.1217	0.9877 to 0.9989	0.00001	0.095233
Tic-Tac-Toe Endgame	98.69	0.9832 to 0.9905	99.51	0.9928 to 0.9973	83.9206	0.8125 to 0.902	0.221075	0.000052
Wine	97.58	0.9642 to 0.9873	100	1 to 1	59.4167	0.5012 to 0.7861	0.197744	0.000377

95% confidence interval (in the context of our problem) is a range of probability values for an arbitrary sample to get correctly classified; the probability (for classification) will be in this range with a probability of 95%.

The p values from t-tests are computed with significance level 0.01, hence if the value of p is less than 0.01, then the difference is significant (highlighted in blue, in the table above), else it isn't significant.

Analysis

We could observer that IBL performed very good on average, with accuracy > 94% for all datasets. From the confidence intervals, we could see that IBL performed significantly better than ID3 for 'Car Evaluation', 'Tic-Tac-Toe' and 'Iris' datasets, while Neural Networks performed similar to IBL for the same datasets. Another observation was that: using distance-weighting of the neighbors for classification performed better than just taking a vote on the nearest neighbors and this seems logical, since the closer samples have more impact on classification than the farther samples.

We could also notice that both Neural Networks and ID3 performed poorly on some datasets, while the other was performing better (for chess and wine datasets); whereas IBL consistently performed good on all the datasets with a very good accuracy. However, on the flip side, Neural Networks and Decision Trees are relatively much quicker in classifying a test-samples while IBL is a bit slow (since it has to compute the similarity with each training-sample). Hence it would be a wise decision to analyze the number of samples, performance of various algorithms (in terms of both time and accuracy) while choosing a classification algorithm to better suit the requirements.