GEAR: Generic, Efficient, Accurate kNN-based Regression

by

Aditya Desai, Himanshu Singh, Vikram Pudi

in

Intl Conf on Knowledge Discovery and Information Retrieval
(KDIR)

Valencia, Spain

Report No: IIIT/TR/2010/40

Centre for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
October 2010
GEAR

Generic, Efficient, Accurate kNN-based Regression

Submitted for blind Review

Keywords: Regression, prediction, k-nearest neighbours, Generic, Accurate

Abstract: Regression algorithms are used for prediction (including forecasting of time-series data), inference, hypothesis testing, and modeling of causal relationships. Statistical approaches although popular, are not generic in that they require the user to make an intelligent guess about the form of the regression equation. In this paper we present a new regression algorithm GEAR – Generic, Efficient, Accurate kNN-based Regression. In addition to this, GEAR is simple and outlier-resilient. These desirable features make GEAR a very attractive alternative to existing approaches. Our experimental study compares GEAR with fourteen other algorithms on five standard real datasets, and shows that GEAR is more accurate than all its competitors.

1 INTRODUCTION

Regression analysis has been studied extensively in statistics (Humberto Barreto; Y. Wang and I. H. Witten, 2002), there have been only a few studies from the data mining perspective. The algorithms studied from a data mining perspective mainly fall under the following broad categories - Decision Trees (L. Breiman, J. Friedman, R. Olshen, and C. Stone, 1984), Support Vector Machines (W. Chu and S.S. Keerthi, 2005), Neural Networks (M. Ware, 2005), Nearest Neighbour Algorithms (E. Fix and J. L. H. Jr., 1951)(P. J. Rousseveeu and A. M. Leroy, 1987), Ensemble Algorithms(L. Breiman, 1996)(R.E. Schapire, 1999) among others. It may be noted that most of these studies were originally for classification, but have later been modified for regression (I. H. Witten and E. Frank).

In this paper we present a new regression algorithm GEAR (Generic, Efficient, Accurate and Robust kNN-based Regression). GEAR has the following desirable features:

- **Generic**: Our approach assumes that dependent variable changes smoothly with change in independent variable, an assumption which is valid for most real-life datasets. Hence, it will work “out-of-the-box” and doesn’t require to be tinkered with for every application domain and dataset.

- **Efficient**: GEAR is based on the nearest neighbour (k-NN) approach and is modified so that the resulting time complexity is $O((\log n)^4)$ and space complexity is $O(n)$ for every test instance (where $n$ is the number of tuples in the training data).

- **Accurate**: Our experimental study in Section 5.1 shows that GEAR provides more accurate estimates than its competitors on several datasets. Among the algorithms we included for comparison are the best available algorithms from the Weka toolkit (I. H. Witten and E. Frank).

- **Robust**: We include techniques such as “noisy neighbour elimination” and “gravity based weighting” to make GEAR extremely robust.

- **Parameterless**: Even though the algorithm uses four parameters, our experimental results demonstrate that it is relatively insensitive to variations in the parameter values. Moreover, the algorithm automatically sets the optimal values of these parameters.

- **Simple**: The design of GEAR is simple, as it is based on the k-NN approach, making it easy to implement, maintain, embed and modify as the situation demands.
• **Outlier Resilience:** The output of GEAR for a particular input record \( T \) is dependent only on the nearest neighbours of \( T \) and is therefore insensitive to far-away outliers.

The remainder of the paper is organized as follows: In Section 2 we discuss related work and in Section 3 we describe the concept the optimal fitting line. In Section 4 we present the GEAR algorithm. Then, in Section 5 we experimentally evaluate our algorithm and show the results. Finally, in Section 6, we conclude our study and identify future work.

## 2 RELATED WORK

In this section, we describe work related to the new regression algorithm proposed in this paper.

### Traditional statistical approaches

Most existing approaches (Y. Wang and I. H. Witten, 2002) (Humberto Barreto) follow a “curve fitting” methodology that requires the form of the curve to be specified in advance. This requires that regression problems in each special application domain are separately studied and solved optimally for that domain. Another problem with the curve fitting approaches is outlier (extreme cases) sensitivity. Outliers can seriously bias the results by pulling or pushing the regression curve in a particular direction, leading to biased regression coefficients. In addition to this they fit the regression equation to the entire plane and hence are unlikely to capture inherent relationships.

### Global fitting methods

Approaches like Generalized Projection Pursuit regression (Lingjaerde, Ole C. and Liestøl, Knut, 1999) is a new statistical method which constructs a regression surface \( y = f(X) \) by estimating the form of the function \( f \) in such a manner that it best fits the dataset \( X \) without using any parameter. Lagrange by Dzeroski et al. (Dzeroski S., Todorovski L., 1995) generates the best fit equation over the observational data by constructing a large number of equation alternatives. Due to huge search space the approaches are computationally expensive. Lagrange (Todorovski L., 1998) is a modification of Lagrange in which grammars of equations are generated from domain knowledge and the best-fit equations are determined by filtering the forms using the generated grammar. However, the algorithm requires prior domain knowledge. Support vector machine (W. Chu and S.S. Keerthi, 2005) (SVM) is a new data mining paradigm applied for regression. However these techniques involve complex abstract mathematics thus resulting in techniques that are more difficult to implement, maintain, embed and modify as the situation demands. Neural networks are another class of data mining approaches that have been used for regression (M. Ware, 2005) and dimensionality reduction as in Self Organizing Maps (Haykin, Simon). However, neural networks are complex “black box” models and hence an in depth analysis of the results obtained is not possible. Data mining applications typically demand an “white box” model where the prediction can be explained to the user, since it is to be used for decision support. Ensemble based learning (L. Breiman, 1996) (R.E. Schapire, 1999) is a new approach to regression where a number of machine learning algorithms are combined to build a learner having an accuracy better than the individual learners. A major problem associated with ensemble based learning is to determine the relative importance of each individual learner. This is usually done by assigning weights to individual learners where a high weight implies a higher relative importance of the learner. The problem with ensemble learning is that the resultant ensembles are extremely large and are prone to large train and test times in addition to overfitting and for this reason various ensemble pruning and weighting algorithms have been proposed in the past. These approaches being highly complex act like black box models and hence prove to be futile when it comes to decision support. However, all these algorithms suffer from the problem that they try to fit the entire data to a particular structure and hence cannot capture inherent relationships in different localities of the dataset.

### Decision Trees

One of the first data mining approaches to regression were *regression trees* (L. Breiman, J. Friedman, R. Olshen, and C. Stone, 1984), a variation of decision trees where the predicted output values are stored at the leaf node. These nodes are finite and hence the predicted output is limited to a finite set of values which is in contrast with the problem of predicting a continuous variable as required in regression.

### k-Nearest Neighbour

Finally, another class of data mining approaches that have been used for regression include *nearest neighbour techniques* (E. Fix and J. L. H. Jr., 1951)(P. J. Rousseweu and A. M. Leroy, 1987). These methods estimate the response value as a weighted sum of the responses of all neighbours, where the weight is inversely proportional to the distance (generally normalized euclidean) from the input tuple. These algorithms are known to be simple and reasonably outlier resistant. Although these approaches have the desirable property of simplicity, they have relatively low accuracy because of the problem of determining the correct
number of neighbours and the fact that they assume that all dimensions contribute equally. This is often not the case as some features which have a higher degree of correlation with the dependant variable have a higher influence on the dependant variable value than other variables.

In this paper, we enhance the power of nearest neighbour predictors which intrinsically handle local variations. We eliminate the problems associated with nearest neighbour methods like choice of number of neighbours and difference in importance of dimensions. We use a novel weighting criterion which determines the relative importance of dimensions. This is used in combination with a unique stability criterion which determines the appropriate number of neighbours. Our algorithm uses a gravity based criterion to set the importance of neighbours. A similar technique was used in another context for clustering in (Yong Shi, Yuqing Song, Aidong Zhang, 2003).

3 OPTIMAL FITTING LINE

The key idea in this paper is derived from the fact that typically in regression, dependant variable changes smoothly when each independent variables value is increased (or decreased). Specifically, this translates to an approximately linear relationship between the dependent and independent variables within very small neighbourhoods. Thus, a central component of our approach is to fit a line onto the points in a small neighbourhood.

In this section, we propose a method which gives the parameters of the line which minimizes local error and is referred to as optimal fitting line. Local error for the line is defined as the mean of squared error if the line was used for prediction of points of the neighbourhood. A low value of error indicates that the line so obtained fits the neighbourhood well and has captured the linearity of the locality and hence can be used for high quality prediction in this locality. Let the $k$ points have values $\{(x_1,y_1),\ldots,(x_k,y_k)\}$ in dimension $x$ and $y$ and let the variable to be predicted be $y$. Let the line have the form $y = \alpha x + \beta$. Thus, the value of the dependant variable that is predicted for tuple $i$ is $\alpha x_i + \beta$. Let the error in prediction for tuple $i$ be denoted as $e_i$ and is equal to $|y - \alpha x_i - \beta|$. Thus, the local mean error (LME) is denoted as

$$LME(\alpha, \beta) = \frac{\sum_{i=1}^{k} e_i}{k} = \frac{\sum_{i=1}^{k} (y_i - \alpha x_i - \beta)^2}{k}$$

By minimizing LME where $\alpha$ and $\beta$ are the parameters.

$$\alpha = \frac{\sum_{j=1}^{k} y_j \sum_{i=1}^{k} x_i - k \sum_{i=1}^{k} y_i x_i}{\sum_{j=1}^{k} x_j \sum_{i=1}^{k} x_i - k \sum_{i=1}^{k} x_i^2}, \quad \beta = \frac{\sum_{j=1}^{k} y_j - \alpha \sum_{i=1}^{k} x_i}{k}$$

(2)

Thus, we get the equation of the optimal fitting line.

However, not all points in the neighbourhood of the input tuple are equally important for line construction. Thus, we weight points to denote their importance. Also, the constructed line should be more accurate in predicting the values of more important points. The intuition requires us to construct a line that minimizes the weighted mean local error (WLME) given as:

$$WLME = \frac{\sum_{i=1}^{k} w_i e_i}{k}$$

(3)

By minimizing WLME where $\alpha$ and $\beta$ are the parameters.

$$\alpha = \frac{\sum_{j=1}^{k} w_j y_j \sum_{i=1}^{k} w_i x_i - k \sum_{i=1}^{k} w_i y_i x_i}{\sum_{j=1}^{k} w_j x_j \sum_{i=1}^{k} w_i x_i - k \sum_{i=1}^{k} w_i x_i^2}, \quad \beta = \frac{\sum_{j=1}^{k} w_j y_j - \alpha \sum_{i=1}^{k} w_i x_i}{k}$$

(4)

This, leads us to the following theorem:

**Theorem:** Given $k$ points with weights $w_1, \ldots, w_k$ the regression line that minimizes the WLME is $y = \alpha x + \beta$ where

$$\alpha = \frac{\sum_{j=1}^{k} w_j y_j \sum_{i=1}^{k} w_i x_i - k \sum_{i=1}^{k} w_i y_i x_i}{\sum_{j=1}^{k} w_j x_j \sum_{i=1}^{k} w_i x_i - k \sum_{i=1}^{k} w_i x_i^2}, \quad \beta = \frac{\sum_{j=1}^{k} w_j y_j - \alpha \sum_{i=1}^{k} w_i x_i}{k}$$

(5)

The theorem can be proved by retracing the steps as above. As is the intuition behind nearest neighbour predictors, the importance of a neighbour for prediction is inversely proportional to its distance from the input tuple. Thus we assign, $w_i = \frac{1}{\text{dist}_i}$ where $\text{dist}_i$ is the distance of the $i^{th}$ closest neighbour from the input tuple.

4 THE GEAR ALGORITHM AND DESIGN

In this section we present the GEAR (Generic, Efficient, Accurate & Robust $k$NN-based Regression) algorithm. GEAR is also simple and outlier-resilient. These desirable features make GEAR a very attractive alternative to existing approaches. In section 4.1, we present the design of our algorithm, followed by a pseudo code of the same in section 4.2. Finally in section 4.3, we present the computational complexity of the algorithm.

4.1 Design of the Algorithm

The design of GEAR is based on the assumption that value of the dependant variable varies smoothly with
the variation in values of dependant variable. Now, every smooth curve can be modelled to be a combination of piecewise linear curves. This intuition becomes the basis of our paper, where the task at hand is determining the nature of linearity in the locality of the test tuple. The simplest approach to solving this problem is to construct a line using the two closest neighbours of the input tuple and thus approximating the linearity of the regression curve in this region to the linearity given by the line so constructed.

It may be noted that a standard kNN algorithm bounds values i.e. the values output by a kNN-algorithm are always between the minimum and maximum in the dataset. Our algorithm alleviates this problem by constructing a 1-dimensional predictor and hence approximates the linearity in the given region thus giving more accurate estimates. However, the above mentioned basic solution is highly susceptible to bias in case one of the two closest neighbours turns out to be noisy or are not a correct reflection of the neighbourhood. A solution to this problem is to construct a line minimizing the distance from k-neighbours instead of 2, with higher weights associated to closer neighbours and lower weights associated with far away ones. However, setting the value of k to an optimal value is not a trivial task as a small value may bias the line so constructed while a large value of k may yield a curve with loss of local knowledge and is thus the crux of our paper. In short, we chose the best line as the line which minimizes the mean weighted distance of the k-neighbours from the line where weights for a point are inversely proportional to the square of distance from the input test tuple.

It may be noted that some dimensions model variations of dependant variable w.r.t independent variable well while some dimensions model such variations poorly. One of the reasons for this may be that this dimension may be noisy or may be simply because of the fact that the two dimensions are uncorrelated. Also, it may be noted that two dimensions may be influencing each other highly in some regions and may be completely uncorrelated in some other regions. We present a solution to this problem which is described here in brief.

The line that we constructed using the neighbours is used to predict the values of dependant variables of these very neighbours and the mean error is computed. A small value of error indicates that the line fits the neighbourhood of the input tuple in a given direction, while a large value of error indicates that the prediction done using the line is a poor estimate of the dependant variable values. Thus, the predictors so constructed are weighed in such a manner that the weight and thus importance of the predictor is inversely proportional to the mean error of prediction in this dimension. The final result is thus a weighted sum of values output by individual predictors.

### 4.2 The GEAR Algorithm

In this section we present the GEAR algorithm. The notation used in this section is shown in Table 1. Additional notation is as follows: The training data has d-dimensions with feature variables \((A_1, \ldots, A_d)\) and the value of the feature variable \(A_i\) corresponding to the \(i^{th}\) tuple can be accessed as \(Data[i][j]\). The value of the dependant variable of the training tuple corresponding to id \(i\), can be accessed as \(y[i]\). The test tuple \(T\) consists of values of d-dependant variables where the value of feature variable \(A_i\) can be accessed as \(T_i\).

We now describe the GEAR algorithm by using the pseudo code in Algorithm 1-4. As mentioned earlier GEAR is based on \(k=NV\) approach.

In order to determine the correct value of number of neighbours \((k)\) for a given test tuple \(T\), we use an iterative procedure which takes four optional parameters as input. These parameters correspond to error threshold \((\text{Error}_\text{Threshold})\), lower \((\text{min})\), upper bound \((\text{max})\) of \(k\) and Stopping Factor \((\eta)\). Note that these parameters are optional and can also be set to their optimal values automatically. The correct choice of \(k\) is a result of an exhaustive search in between \(\text{min}\) and \(\text{max}\). If the parameters are not set the algorithm automatically sets \(\text{min}\) to a sufficiently low value (about five) and \(\text{max}\) to a sufficiently high value (tuples in training data/2). The default value of \(\text{Error}_\text{Threshold}\) is \(\infty\) and of \(\eta\) is 1.

For each value of \(k\), we determine the \(k\)-nearest neighbours of \(T\) (line 4 of Algorithm 1) and the id’s of these neighbours are stored in the list Closest Neighbours. In line 1 (of Algorithm 2), we iterate over all dimensions and in the subsequent line (line 2 of Algo-

| \(k\) | The number of closest neighbours used for prediction. |
| \(\text{Error}_\text{Threshold}\) | The error threshold |
| \(y\) | The response variable. |
| \(D\) | The Training Data |
| \(d\) | Number of dimensions in \(D\) |
| \(n\) | Number of training tuples in \(D\) |
| \(A_i\) | Denotes a feature |
| \(X\) | The feature vectors space. \(X = (A_1, \ldots, A_d)\) |
| \(T\) | A tuple in \(X\)-space. \(T = (t_1, \ldots, t_d)\) |
| \(\text{RID}\) | Tuple in \(D\) with record id RID |
| \(v_i, \text{RID}\) | Value of attribute \(A_i\) of \(T_{\text{RID}}\) |
| \(N_{L,y}\) | Value of dependent variable for the \(L^{th}\) closest neighbour of \(T\) |

### Table 1: Notation
Algorithm 1 Pseudo Code

1: MinimumError ← ∞; OutputValue ← 0;
2: for k = min to max do
3: ErrorUsingK ← 0; Count ← 0;
4: ClosestNeighbours ← GetNeighbours(Data, k, T)
5: // Returns id of k - closest neighbours of T.
6: // In line 1-Dimensional Predictor (Algorithm 2)
7: Weighting dimensions (Algorithm 3)
8: ErrorUsingK ← ErrorUsingK / Count;
9: // Mean error in prediction of k-neighbours when k-neighbours used
10: if MinimumError > ErrorUsingK then
11: MinimumError ← ErrorUsingK;
12: LIndex ← k // Value of k at which minimum is found
13: end if
14: if (k - LIndex) > η * log(n)² then break
16: end if
17: end for
18: return OutputValue;

Algorithm 2 Inline function 1 - Dimensional predictor

1: for i = 1 to d do
2: Li ← ConstructLine(ClosestNeighbours, i); // Construct line in plane of (A_i, y) using Section 3 (Referred to as a one - dimensional predictor). Equation of line - y = αA_i + β, where L_i[1] = α and L_i[2] = β
3: e_i ← 0
4: for j = 1 to k do
5: PredictVal_j ← Data[ClosestNeighbours[j]][i] * L_i[1] + L_i[2]; // Predicting value of j_th neighbour using line L_i
6: ActualVal_j ← y[ClosestNeighbours[j]]; // Dependant variable value of the j_th closest neighbour.
7: e_j ← e_j + (PredictVal_j - ActualVal_j)²; // Calculating error in prediction using one-dimensional predictor. Distance is distance of the j_th closest neighbour from T
8: end for
9: PredictedTest ← T_i * L_i[1] + L_i[2]; // Predicting the value of the test tuple using one-dimensional predictor constructed above
10: end for

Algorithm 3 Weighting Dimensions

1: for i = 1 to d do
2: if e_i < (min(e_i) * Error_Threshold) then
3: w_i ← max(e_i); // Assigning weights for each dimension, as inversely proportional to the error in prediction using only that dimension.
4: else
5: w_i ← 0 // Setting weight as 0 for high error dimensions
6: end if
7: end for

Weights are assigned weights (Algorithm 3) by assigning weights for dimensions (w_i) to be inversely proportional to the error in prediction for this dimension(e_i). The dimensions corresponding to high error are assigned weight 0 (in line 5 of Algorithm 3). Using these weights, the value of dependant variable for test tuple is predicted (line 2-6 of Algorithm 4) as a weighted sum of values predicted by one dimensional predictors earlier (line 10 of Algorithm 2). Similarly, dependant variable values of neighbours are predicted (line 8-13 of Algorithm 4) as a weighted sum of values output by one-dimensional predictors for neighbour tuples (line 6 of Algorithm 2). The mean error in prediction of neighbours (line 7-16 of Algorithm 4) using k-neighbours is computed and is stored in ErrorUsingK. A lower value of this error indicates that the weight values chosen using k-neighbours were appropriate for the neighbourhood and hence PredictVal corresponding to the min-
Algorithm 4 Inline function d-dimensional predictor

1: PredictVal ← 0; Weights ← 0;
2: for i = 1 to d do
3: PredictVal ← PredictVal + PredictedTesti * wi;
4: Weights ← Weights + wi;
5: end for
6: PredictVal ← PredictVal / Weights; // Predict the value of dependant variable of the test sample as the weighted some of outputs of one-dimensional predictors.
7: for j = 3 to k do
8: FinalPredictValj ← 0; FinalWeightsj ← 0;
9: for i = 1 to d do
10: FinalPredictValj ← FinalPredictValj + wi * PredictValj;
11: FinalWeightsj ← FinalWeightsj + wi;
12: end for
13: FinalPredictValj ← FinalPredictValj / FinalWeightsj;
14: ActualValj ← yjClosestNeighbours[j]; //Dependant variable value of the j-th closest neighbour.
15: Factor ← (1 / Parameter)2
16: ErrorUsingK ← ErrorUsingK + Factor * (FinalPredictValj - ActualValj)2; // Sum of square of deviation from actual value
17: Count ← Count + Factor
18: end for

Algorithm 4 Inline function d-dimensional predictor

1: PredictVal ← 0; Weights ← 0;
2: for i = 1 to d do
3: PredictVal ← PredictVal + PredictedTesti * wi;
4: Weights ← Weights + wi;
5: end for
6: PredictVal ← PredictVal / Weights; // Predict the value of dependant variable of the test sample as the weighted some of outputs of one-dimensional predictors.
7: for j = 3 to k do
8: FinalPredictValj ← 0; FinalWeightsj ← 0;
9: for i = 1 to d do
10: FinalPredictValj ← FinalPredictValj + wi * PredictValj;
11: FinalWeightsj ← FinalWeightsj + wi;
12: end for
13: FinalPredictValj ← FinalPredictValj / FinalWeightsj;
14: ActualValj ← yjClosestNeighbours[j]; //Dependant variable value of the j-th closest neighbour.
15: Factor ← (1 / Parameter)2
16: ErrorUsingK ← ErrorUsingK + Factor * (FinalPredictValj - ActualValj)2; // Sum of square of deviation from actual value
17: Count ← Count + Factor
18: end for

The k-nearest neighbour computation is generally a bottleneck in terms of its complexity. We rectify this problem of high-dimensionality and high response time by modifying the routine to search for nearest neighbours by incorporating a simple but effective index structure.

Intuitively, the algorithm spreads out along each dimension starting from the tuple closest to the input tuple (in that dimension) in order to find its k nearest neighbors. In each dimension it selects the nearest k tuples as “candidates” using Binary Search algorithm (Donald Knuth, 1997) and later removes those which are found to be far away along other dimensions. The resulting tuples may not be the true k nearest neighbours, but the approximation has borne out to be quite good in practice, as demonstrated in our experiments. The algorithm uses the following five data structures:

- **Attribute List**: The attribute list $\overline{A}_i$ is defined for all independent feature variables $A_i$ and each tuple in $\overline{A}_i$ is of the form $\langle v_i, \text{RID} \rangle$ where $v_i, \text{RID}$ is the value of feature variable $A_i$ for record-id RID in $D$. The attribute list is maintained with attribute values in sorted order. RID of the $j^{th}$ tuple of the attribute list $A_i$ is accessed as $\overline{A}_i[j][2]$ and the corresponding value is accessed as $\overline{A}_i[j][1]$. Similar notation is used for the lists below.

- **Predicted List**: The predicted attribute list $\overline{P}_L$ is maintained as $\langle \text{RID}, v_{d, \text{RID}} \rangle$ where $v_{d, \text{RID}}$ is the value of dependant variable for record-id RID in $D$. The attribute list is maintained with record-id’s in sorted order.

- **Data Info List**: The DataInfo list $\overline{DI}_L$ is maintained of the form $\langle \text{RID}, s_{\text{RID}} \rangle$ where $s_{\text{RID}}$ is the distance between the input tuple and the tuple with record-id RID.

- **Distance List**: The Distance list $\overline{D}_L$ is maintained of the form $\langle \text{RID}, s_{\text{RID}} \rangle$ where $s_{\text{RID}}$ is the distance between the input tuple and the tuple with record-id RID.

Initially the first three $\langle \overline{A}_L, \overline{P}_L, \overline{DI}_L \rangle$ lists are written to disk with nothing loaded in the main memory. The rest are empty and loaded in the main memory. The nearest neighbour computation for finding k-neighbours of an input tuple $T = (t_1, \ldots, t_n)$ proceeds as follows:

Upon simple analysis, it is clear from the mentioned algorithm that the space complexity is $O(n + kd)$ and the time complexity is $O(\log n + (kd)\log(kd))$, where $n$ is the number of tuples in $D$ and $d$ is the number of feature variables. Since $\overline{D}_L$ can contain a maximum of $kd$ points ($k$-neighbours are found for $d$ dimensions) and as $k \ll n, kd$ can be ignored with respect to $n$ thus, $O(n + kd) \approx O(n)$.

### 4.2.1 Noisy Neighbor Elimination

It can be clearly seen from Algorithm 1 that the stability of prediction is inversely proportional to the error...
Algorithm 5 Modified Nearest Neighbour

1: for \( i = 1 \) to \( d \) do
2: list \( \leftarrow \text{load}(A_i) \)
3: closest \( \leftarrow \text{BinarySearch}(A_i, T) \)
4: \( \text{LeftDistance} \leftarrow \text{closest}.\text{Right} \leftarrow \text{closest} \)
5: \( \text{LeftDistance} \leftarrow \infty \), \( \text{RightDistance} \leftarrow \infty \)
6: for counter = 1 to \( k \) do
7: if \( \text{Left} > 1 \) then
8: \( \text{LoadedTuple} \leftarrow \text{DI}_L[A_i][\text{Left} - 1][2] \)
9: \( \text{Distance} \leftarrow \text{Distance}((\text{LoadedTuple}, T)) \)
10: end if
11: if \( \text{Right} < n \) then
12: \( \text{LoadedTuple} \leftarrow \text{DI}_L[A_i][\text{Right} + 1][2] \)
13: \( \text{Distance} \leftarrow \text{Distance}((\text{LoadedTuple}, T)) \)
14: end if
15: if \( \text{LeftDistance} < \text{RightDistance} \) then
16: \( S_L[A_i][\text{Left} - 1][2] \leftarrow \text{LeftDistance} \) // Updating list with new distances
17: \( \text{LeftDistance} \leftarrow \text{LeftDistance} \) // Updating list with new distances
18: else
19: \( S_L[A_i][\text{Right} - 1][2] \leftarrow \text{RightDistance} \)
20: end if
21: \( S_L \leftarrow \text{Min}(S_L, k) \) // Updating list retaining \( k \) closest points
22: \( \text{TOP}_k \leftarrow \text{Load}(\text{TOP}_k, \text{DI}_L) \) // Loading tuples with record id in \( S_L \)
23: end for

in prediction in each dimension. Due to this relation, noisy neighbors may have an adverse effect on the relations between feature variables. Hence it becomes necessary to eliminate these noisy neighbors. The algorithm to do the same is given in Algorithm 6 and uses the \( k \) nearest neighbours of input tuple.

The reason for using both mean (\( \mu \)) and median (\( \text{med} \)) is that the noisy neighbors may seriously affect the value of the mean and thus may give a large \( \text{std} \) value which may not be of any use to eliminate the noisy point. The above algorithm is applied before line 4 (Algorithm 1) and the noisy neighbours are eliminated from future computation. For our experiments, we take the value of \( \gamma \) as 3.

Algorithm 6 Noisy Neighbour Elimination

1: \( \mu \leftarrow \text{Mean}(N_{1,y}, \ldots, N_{k,y}) \)
2: \( \text{std}_1 \leftarrow \frac{\sum_{i=1}^{N_{1,y}} \gamma_i}{\text{med}} \)
3: \( \text{med} \leftarrow \text{Median}(N_{1,y}, \ldots, N_{k,y}) \)
4: \( \text{std}_2 \leftarrow \frac{\sum_{i=1}^{N_{1,y}} \gamma_i}{\text{med}} \)
5: if \( \text{std}_1 < \text{std}_2 \) then
6: \( \text{dev} \leftarrow \text{std}_1 \), \( \text{center} \leftarrow \mu \)
7: else
8: \( \text{dev} \leftarrow \text{std}_2 \), \( \text{center} \leftarrow \text{med} \)
end if
10: if \( N_{1,y} \not\in [\text{center} - \gamma \ast \text{dev}, \text{center} + \gamma \ast \text{dev}] \) then
11: Eliminate \( i^{th} \) closest neighbour
end if

4.3 Computational Complexity

An analysis of the pseudo-code reveals the following information about the computational cost associated in executing GEAR for a single test tuple. The nearest-neighbour search algorithm, as can be seen in Algorithm 5 has a time complexity of \( O(\log n + kd \log kd) \). This is followed by the line construction algorithm which is \( O(1) \) and prediction of values of neighbours which is \( O(k) \). As this procedure is repeated \( d \) times corresponding to \( d \) dimensions, the computational efficiency for this procedure is \( O(kd) \).

This is followed by the procedure “Weighting dimensions”, which is a simple iteration over \( d \) dimensions and hence is \( O(d) \). The final procedure computes values of the input tuple and all its neighbours using one dimension predictors and hence is \( O(kd) \). It may be noted that noisy neighbour elimination is \( O(k) \). Thus the overall computational complexity for given \( k \) is \( O(\log n + kd + d + k + kd(\log kd)) \). As we vary \( k \) from \( \min \) to \( \max \) the resulting computational cost is

\[
\sum_{k=\min}^{\max} O(\log n + kd + d + k + kd(\log kd)) \tag{6}
\]

Experimental results have revealed that the maximum number of neighbours to be considered is \( O((\log n)^2) \) and hence the resulting time complexity is approximately equal to \( O((\log n)^3) \).

5 Experimental Study

In this section we describe the experimental settings and results in Section 5.1. In Section 5.2 we discuss the scalability aspect of the algorithm followed by Section 5.3 which contains a discussion of the results.
5.1 Experimental Settings and Results

In this section we compare our regression algorithm against fourteen other algorithms on 5 datasets including a mix of synthetic and real life datasets. The algorithms used are available in the Weka toolkit (I. H. Witten and E. Frank). We experimented with different parameter values in these algorithms and selected those values that worked best for most datasets.

We used five datasets in our experimental study. Their details are shown in table 2. It may be noted that in the Table 2, Cat indicates number of categorical feature variables, Num denotes number of numeric feature variable, Tup indicates number of tuples, S denotes synthetic dataset and R denotes real-life datasets. The experimental results for these datasets are shown in Table 6.

All the results and comparison have been done using the leave one out comparison technique which is a specific case of n-folds cross validation, where the n is set to the number of instances of the dataset. The metrics used for comparison are Root Mean Square Error (RM) and Absolute Mean Error (AB). Only a maximum of top three performing algorithms are indicated (in boldface) where the ordering is in accordance with the principle of “Pareto Dominance” (Voorneveld, Mark, 2002).

5.2 Scalability Analysis

One of the most important requirement for real-world application of any algorithm is its scalability and hence GEAR was executed on synthetically generated training datasets ranging from 50 to 2000000 tuples. The size of the test set was 5000 tuples. Due to lack of space the results have been presented only for the Friedman-2 dataset. As can be seen from Figure 1, the algorithm shows a $O((\log n)^4)$ thus validating the evaluation done in Section 4.3 and hence demonstrating the Scalable nature of the algorithm. It may be noted that the value of $\eta$ used for experiments is 0.5. The computational environment is Ubuntu - Linux with Intel Core 2 Duo 2.80 GHz CPU and 2 GB RAM. The algorithm is implemented in python.

5.3 Discussion of Results

In this section we discuss the results in experimental results in Section 5 with respect to properties of GEAR described in Section 1

- **Generic**: As is evident from the experimental results we have compared GEAR with 14 other other algorithms on 5 numeric datasets. Exceptional relative performance on all kinds of datasets and domains validates the generic nature of GEAR.

- **Efficient**: The computational efficiency described in Section 4.3 demonstrates its efficient nature.

- **Accurate**: As is evident from the experimental results we have compared GEAR with 14 other algorithms on 3 standard synthetic and 2 real-life datasets. It was found that GEAR demonstrated the best accuracy and was more accurate or equivalent to all its competitors on almost all datasets.

- **Parameterless**: We note that although GEAR uses four parameters - min, max, Error Threshold and $\eta$ it is reasonably robust to variations in their values. For eg., it can be seen from Table 5 that there is a small variation in root mean square error for large variations in Error Threshold values. Similarly robustness to variations in value of $\eta$ can be seen in Table 4, thus validating the parameterless nature of GEAR.

- **Robust and Outlier Resistant**: The experiments on synthetic datasets were conducted by artificially adding a Gaussian noise. In addition to this, we used datasets like concrete and servo dataset which are reported to be highly non-linear functions. Excellent results on these datasets demonstrate the robustness and outlier resistant nature of GEAR.

As is evident from the experimental results we have compared GEAR with 14 other algorithms on 3 standard synthetic and 5 real-life datasets. It was found...
that GEAR demonstrated the best accuracy and was more accurate or equivalent to all its competitors on all datasets.

6 Conclusion

In this paper we have presented and evaluated GEAR, an algorithm for regression based on nearest neighbor methods. Evaluation was done against 14 competing algorithms on 5 standard synthetic and real-life datasets. Although simple, it outperformed all competing algorithms on most datasets. Unlike most other algorithms, GEAR can be used “out-of-the-box” without having to extensively tune or tweak it for each application domain and dataset.

Future work in this field includes application of GEAR to industrial and real world applications like stock market prediction among others. We also intend to remove the bottleneck of one dimensional prediction with the help of association rules. In addition GEAR can be applied on data-streams and can be used for active learning. It is also of great interest to experiment with different curve fitting measures instead of the linear curve fitting measure currently used.

REFERENCES


Y. Wang and I. H. Witten, Modeling for optimal probability prediction, 2002.

Humberto Barreto, An Introduction to Least Median of Squares, Chapter contribution to Barreto and Howland, Econometrics via Monte Carlo Simulation


Bart Hamers, Johan A. K. Suyken, Bart De Moor, Compactly Supported RBF Kernels for Sparsifying the Gram Matrix in LS-SVM Regression Models, Artificial Neural Networks, ICANN 2002.


Yong Shi, Yuqing Song and Aidong Zhang “A Shrinking-Based Approach for Multi-Dimensional Data Analysis” Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003


<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameter settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian Process</td>
<td>GaussianProcesses -L 1.0 -N 0 -K &quot;supportVector.RBFKernel -C 250007 -G 0.5&quot;</td>
</tr>
<tr>
<td>Isotonic Regression</td>
<td>IsotonicRegression</td>
</tr>
<tr>
<td>Least Median Square Regression (LMS)</td>
<td>LeastMedSq -S 4 -G 0</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>LinearRegression -S 0 -R 1.0E-8</td>
</tr>
<tr>
<td>Multi Layer Perceptron (MLP)</td>
<td>MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a</td>
</tr>
<tr>
<td>Pace Regression</td>
<td>PaceRegression -E eb</td>
</tr>
<tr>
<td>RBF Network</td>
<td>RBFNetwork -B 2 -S 1 -R 1.0E-8 -M -1 -W 0.1</td>
</tr>
<tr>
<td>Simple Linear Regression (SL)</td>
<td>SimpleLinearRegression</td>
</tr>
<tr>
<td>SMOreg Regression</td>
<td>SMOreg -S 0.001 -C 1.0 -T 0.001 -P 1.0E-12 -N 0 -K</td>
</tr>
<tr>
<td>SVMReg Regression</td>
<td>SVMReg -C 1.0 -N 0 -I waka.classifiers.functions. supportVector.PolyKernel -C 250007 -E 1.0</td>
</tr>
<tr>
<td>IBk (k-NN)</td>
<td>IBk.lazy.IBk -K 5 -W 0 -A &quot;weka.core.neighboursearch. LinearNNSearch -A wka.core.EuclideanDistance -R first-last&quot;</td>
</tr>
<tr>
<td>KStar</td>
<td>KStar -waka.classifiers.lazy.KStar -B 20 -M a</td>
</tr>
<tr>
<td>Additive Regression</td>
<td>AdditiveRegression -S 1.0 -I 10 -W</td>
</tr>
</tbody>
</table>

Table 3: Experimental Settings

<table>
<thead>
<tr>
<th>η-Values</th>
<th>Multi</th>
<th>Friedman-1</th>
<th>Friedman-2</th>
<th>Slump</th>
<th>Concrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.67</td>
<td>2.23</td>
<td>43.01</td>
<td>7.64</td>
<td>7.37</td>
</tr>
<tr>
<td>0.6</td>
<td>0.67</td>
<td>2.19</td>
<td>41.56</td>
<td>7.66</td>
<td>7.31</td>
</tr>
<tr>
<td>0.75</td>
<td>0.67</td>
<td>2.19</td>
<td>41.81</td>
<td>7.66</td>
<td>7.33</td>
</tr>
<tr>
<td>1.0</td>
<td>0.67</td>
<td>2.19</td>
<td>42.43</td>
<td>7.65</td>
<td>7.34</td>
</tr>
<tr>
<td>1.25</td>
<td>0.67</td>
<td>2.19</td>
<td>42.22</td>
<td>7.64</td>
<td>7.37</td>
</tr>
<tr>
<td>1.5</td>
<td>0.66</td>
<td>2.19</td>
<td>42.28</td>
<td>7.65</td>
<td>7.40</td>
</tr>
<tr>
<td>2.0</td>
<td>0.66</td>
<td>2.17</td>
<td>41.59</td>
<td>7.65</td>
<td>7.39</td>
</tr>
</tbody>
</table>

Table 4: Variation in RMSE with variation in η values for Multi, Friedman-1, Friedman-2, Slump and Concrete Datasets

<table>
<thead>
<tr>
<th>Error Threshold</th>
<th>Multi</th>
<th>Friedman-1</th>
<th>Friedman-2</th>
<th>Slump</th>
<th>Concrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0150</td>
<td>0.695</td>
<td>2.260</td>
<td>42.592</td>
<td>7.682</td>
<td>7.364</td>
</tr>
<tr>
<td>0.0200</td>
<td>0.678</td>
<td>2.235</td>
<td>42.012</td>
<td>7.678</td>
<td>7.330</td>
</tr>
<tr>
<td>0.0400</td>
<td>0.672</td>
<td>2.189</td>
<td>42.243</td>
<td>7.657</td>
<td>7.342</td>
</tr>
<tr>
<td>0.0800</td>
<td>0.670</td>
<td>2.188</td>
<td>42.384</td>
<td>7.656</td>
<td>7.344</td>
</tr>
<tr>
<td>0.1500</td>
<td>0.670</td>
<td>2.186</td>
<td>42.437</td>
<td>7.653</td>
<td>7.344</td>
</tr>
<tr>
<td>0.2000</td>
<td>0.670</td>
<td>2.186</td>
<td>42.427</td>
<td>7.653</td>
<td>7.344</td>
</tr>
<tr>
<td>0.3000</td>
<td>0.670</td>
<td>2.186</td>
<td>42.427</td>
<td>7.653</td>
<td>7.344</td>
</tr>
<tr>
<td>0.5000</td>
<td>0.670</td>
<td>2.186</td>
<td>42.427</td>
<td>7.653</td>
<td>7.344</td>
</tr>
<tr>
<td>1.0000</td>
<td>0.670</td>
<td>2.186</td>
<td>42.427</td>
<td>7.653</td>
<td>7.344</td>
</tr>
<tr>
<td>∞</td>
<td>0.670</td>
<td>2.186</td>
<td>42.427</td>
<td>7.653</td>
<td>7.344</td>
</tr>
</tbody>
</table>

Table 5: Variation in RMSE with Error Threshold variation on Multi, Friedman-1, Friedman-2, Slump and Concrete Datasets.
### Table 6: Experimental Results on Multi, Friedman-1, Friedman-2, Slump and Concrete Dataset

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Multi</th>
<th>Friedman-1</th>
<th>Friedman-2</th>
<th>Slump</th>
<th>Concrete</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEAR</td>
<td>AB 0.53, RM 0.67</td>
<td>AB 1.69, RM 2.19</td>
<td>AB 29.09, RM 42.43</td>
<td>AB 5.43, RM 7.65</td>
<td>AB 5.17, RM 7.34</td>
</tr>
<tr>
<td>Gaussian</td>
<td>AB 0.48, RM 0.61</td>
<td>AB 1.74, RM 2.21</td>
<td>AB 35.02, RM 53.21</td>
<td>AB 5.81, RM 7.26</td>
<td>AB 6.58, RM 8.38</td>
</tr>
<tr>
<td>Isotonic</td>
<td>AB 0.90, RM 1.13</td>
<td>AB 3.59, RM 4.32</td>
<td>AB 219.69, RM 285.11</td>
<td>AB 5.89, RM 7.52</td>
<td>AB 10.84, RM 13.52</td>
</tr>
<tr>
<td>LMS</td>
<td>AB 0.56, RM 0.71</td>
<td>AB 2.28, RM 2.90</td>
<td>AB 114.37, RM 161.02</td>
<td>AB 6.71, RM 10.58</td>
<td>AB 9.28, RM 16.55</td>
</tr>
<tr>
<td>LinearReg</td>
<td>AB 0.56, RM 0.71</td>
<td>AB 2.28, RM 2.88</td>
<td>AB 107.94, RM 144.30</td>
<td>AB 6.55, RM 7.79</td>
<td>AB 8.29, RM 10.46</td>
</tr>
<tr>
<td>Pace</td>
<td>AB 0.56, RM 0.71</td>
<td>AB 2.28, RM 2.89</td>
<td>AB 107.75, RM 144.47</td>
<td>AB 6.73, RM 7.9</td>
<td>AB 8.32, RM 10.52</td>
</tr>
<tr>
<td>RBF</td>
<td>AB 1.01, RM 1.26</td>
<td>AB 3.77, RM 4.71</td>
<td>AB 244.93, RM 315.83</td>
<td>AB 6.84, RM 8.56</td>
<td>AB 13.43, RM 16.67</td>
</tr>
<tr>
<td>SL</td>
<td>AB 0.90, RM 1.10</td>
<td>AB 3.56, RM 4.27</td>
<td>AB 220.86, RM 284.07</td>
<td>AB 6.53, RM 7.85</td>
<td>AB 11.87, RM 14.52</td>
</tr>
<tr>
<td>MLP</td>
<td>AB 0.63, RM 0.77</td>
<td>AB 2.09, RM 2.61</td>
<td>AB 30.08, RM 38.62</td>
<td>AB 6.03, RM 8.85</td>
<td>AB 6.58, RM 8.61</td>
</tr>
<tr>
<td>SMOReg</td>
<td>AB 0.56, RM 0.72</td>
<td>AB 2.27, RM 2.89</td>
<td>AB 107.27, RM 144.31</td>
<td>AB 5.79, RM 7.95</td>
<td>AB 8.23, RM 10.97</td>
</tr>
<tr>
<td>SVMReg</td>
<td>AB 0.56, RM 0.72</td>
<td>AB 2.27, RM 2.89</td>
<td>AB 107.41, RM 144.45</td>
<td>AB 5.79, RM 7.95</td>
<td>AB 8.23, RM 10.97</td>
</tr>
<tr>
<td>KStar</td>
<td>AB 0.62, RM 0.78</td>
<td>AB 1.96, RM 2.55</td>
<td>AB 56.06, RM 79.14</td>
<td>AB 6.45, RM 9.75</td>
<td>AB 6.08, RM 8.36</td>
</tr>
<tr>
<td>LWL</td>
<td>AB 0.83, RM 1.05</td>
<td>AB 3.47, RM 4.16</td>
<td>AB 168.19, RM 228.41</td>
<td>AB 5.83, RM 7.74</td>
<td>AB 10.55, RM 13.23</td>
</tr>
<tr>
<td>Additive</td>
<td>AB 0.68, RM 0.85</td>
<td>AB 2.30, RM 2.83</td>
<td>AB 142.93, RM 173.57</td>
<td>AB 6.05, RM 8.36</td>
<td>AB 6.67, RM 8.45</td>
</tr>
<tr>
<td>IBK</td>
<td>AB 0.57, RM 0.71</td>
<td>AB 1.74, RM 2.21</td>
<td>AB 50.57, RM 68.70</td>
<td>AB 5.83, RM 8.03</td>
<td>AB 6.10, RM 8.55</td>
</tr>
</tbody>
</table>