Assignment 3

Artificial Intelligence

November 14, 2020

Written Component Due Date: November 22, 2020 at 11:59pm
Programming Component Due Date: November 29, 2020 at 11:59pm

Written Component: Please submit a PDF containing your solutions called LastName-FirstName-A3.pdf, where you replace LastName and FirstName with your last and first names. You may write out your answers on paper, take pictures of your work, compile them into a PDF and submit in that manner if you prefer not to type out the answers.

Programming Component: Please submit your source code and a README outlining how to run your program.

Suppose that you have a problem where the predictive attributes $A_1, ..., A_k$ are numeric and the classification attribute CE is discrete, with 2 or more values. Let dom(C) be the set of possible values of the classification attribute.

For any instance X, let $\vec{u}$(X) be the vector of predictive attributes. For instance if instance X has predictive attributes $X.A_1$ = 2, $X.A_2$ = 6, $X.A_3$ = 1, and classification attribute $X.C$ = a, then $\vec{u}$(X) = (2, 6, 1).

Consider the following classification method.

Learning: From training set T. For each value v ∈ dom(C), let T_v be the set of instances X ∈ T, where $X.C$ = v. For each value v of the classification attribute, compute the centroid:

$$\vec{g}_v = \frac{1}{|T_v|} \sum_{Y \in T_v} \vec{u}(Y) \qquad (1)$$

The vectors $\vec{g}_v$ are the output of the learning stage.

Classification: To classify a new instance Z, find the closest of the exemplar vectors in terms of the Euclidean distance. Note that it is actually easier to compute the distance squared, and minimize that. (Assume that no two exemplar vectors are equal, and that you never get a test item where two exemplars are tied for closeness.)

$$v = argmin_{w \in dom(C)} d^2(\vec{g}_w, \vec{u}(Z)) \qquad (2)$$

, where

$$d^2(\vec{x}, \vec{y}) = \sum_{i=1}^{k} (x_i - y_i)^2 \qquad (3)$$

Predict that Z has classification attribute v.

For example, suppose the following table is the training set T, with dom(C) = a,b,c.

Then, the following exemplar vectors are computed:
$\vec{g}_a = \frac{1}{3}$((1,1,2) + (2,1,1) + (2,0,1)) = (1.67, 0.67, 1.33)
$\vec{g}_b = \frac{1}{2}$((0,1,2) + (3,2,0)) = (1.50, 1.50, 1.00)
$\vec{g}_c = \frac{1}{4}$((3,3,0) + (0,3,0) + (3,2,1) + (0,3,3)) = (1.50, 2.75, 1.00)

Now, if a new instance $\vec{z}$ = (2,2,2) arrives, we compute $d^2(\vec{z}, \vec{g}_a)$ = 2.33, $d^2(\vec{z}, \vec{g}_b)$ = 3.50, $d^2(\vec{z}, \vec{g}_c)$ = 1.81; so $\vec{z}$ would be classified as an instance of category c.

This choice of exemplar points minimizes the objective function which is the sum of the distance squared from each data point in the training set to the exemplar of its category.

A_1	A_2	A_3	C
1	1	2	a
2	1	1	a
2	0	1	a
0	1	2	b
3	2	0	b
3	3	0	c
0	3	0	c
3	2	1	c
0	3	3	c

Table 1: Training Set

Objective Function

$$O_T(\vec{g}_1, ..., \vec{g}_c) = \sum_{v \in dom(T)} \sum_{\vec{z} \in T_v} d^2(\vec{g}_v, \vec{z}) \tag{4}$$

1. Let N be the number of instances in the training set; k, the number of predictive attributes; and c, the number of different values of the classification attribute. What (order of magnitude) is the running time for training? For classification?

2. This method can easily be made robust relative to null values, both in the training data and in the test instance; that is, it is possible to use training instances where some of the predictive attributes are null and it is possible to classify test instances where some of the predictive attributes are null. Explain how.

3. Compute the value of the objective function for the training set provided above.

4. Given an example in which there is some category a such that no data points in the training set are classified as being of category a, even though there are data points labelled a in the training set. (Hint: there are examples with 1 predictive attribute and 3 categories.)

5. Suppose that you have one predictive attribute A, and the classification attribute depends as follows on A as follows:

   - If $X.A < 0$ then X.C = a;
   - If $0 < X.A < 1$ then X.C = b;
   - If $1 < X.A < 5$ then X.C = c;
   - If $5 < X.A < 6$ then X.C = d;
   - If $6 < X.A$ then X.C = e;

   Show that there is no set of exemplars that will always correctly classify test items.

6. A variant of this method would be to allow q exemplar points for each category. A new instance is classified using the closest of the cq exemplar points. This allows a more expressive notation; for instance, the class of points that are categorized as a given category can be non-convex or even disconnected.

   Propose a method for choosing q exemplar points. (Assume that each category includes much more than q instances in the training set.) This is an open-ended question; there's no definitive right answer. Say something intelligent about it.

7. For the categories in Question 5, find a set of at most 2 exemplars per category that will always correctly classify test items.

Programming Component In this programming assignment, you will implement a gradient-based classifier for a classification problem with numerical predictive attributes and a discrete classification attribute.

Classifier The classifier is the same as the one in the written portion, and we will use the same notations. The learning algorithm produces a set of c exemplar vectors to be used during the classification phase . The classifier classifies a test point according to the closest exemplar vector.

However, we will modify how the classification algorithm works. First, we will modify the objective function to charge a "cost" only to misclassified points. Let G be a tuple of exemplar vectors G = $\{\vec{g}_1$, ..., $\vec{g}_c\}$. For a given training example Y, where Y.C = v, let $\vec{r}_G$(Y) be the closest exemplar vector to Y (assume there are no ties). As in the written portion, let $\vec{u}$(Y) be the vector of predictive attributes of Y and let v = Y.c be the classification attribute. Thus if $\vec{r}_G = \vec{g}_v$, then G is correctly classifying Y. Therefore, the quantity:

$$d^2(\vec{g}_v, \vec{u}(Y)) - d^2(\vec{r}_G(Y), \vec{u}(Y)) \tag{5}$$

is zero if G correctly classifies Y and positive if it does not.

We therefore can associate an error cost with Y for classifies G:

$$Cost_G(Y) = min(M, d^2(\vec{g}_v, \vec{u}(Y)) - d^2(\vec{r}_G(Y), \vec{u}(Y))) \tag{6}$$

The quantity M is externally provided. The point of putting a maximum on the cost is to make sure that distant outliers do not have too much of an influence.

The objective function for G with respect to a given training set T is the sum of the costs of the instances in Y: $O_T(G) = \sum_{Y \in T} Cost_G(Y)$.

The classification algorithm is to use gradient descent over this objective function to find a local minimum.

It is easily seen that the objective function is zero if and only if all points in the training set are correctly classified; and that it is continuous and piecewise differentiable.

The negative gradient of O with respect to an exemplar vector $\vec{g}_v$ is computed as follows:

Let $P_{G,v}(T)$ be the set of all instances Y in T such that Y.C = v but $\vec{r}_G \neq \vec{g}_v$ and $Cost_G(Y) < M$; that is the points that are labeled v but misclassified, and have a cost less than M.

Let $Q_{G,v}(T)$ be the set of all instances Y in T such that Y.C $\neq$ v, but $\vec{r}_G = \vec{g}_v$ and $Cost_G(Y) < M$; that is the points that are classified as v but whose label is actually something else and have a cost less than M.

Then the negative gradient of O with respect to $-\vec{g}_v$ is given by:

$$-\nabla g_v(O) = 2 * \sum_{Y \in P_{G,v}(T)} (\vec{u}(Y) - \vec{g}_v) + 2 * \sum_{Y \in Q_{G,v}(T} (\vec{u}(Y) - \vec{g}_v) \tag{7}$$

Intuitively, you want to move $\vec{g}_v$ toward data points that should be labelled v but are not, and away from data points that are labelled v but should be labelled something else.

However, if a data point is too far from its proper exemplar, then we give it up (for the time being) as hopeless, and don't consider it in the calculations.

Therefore the gradient descent classification algorithm works like this:

```
function gradDescent(training set T; double stepSize; double epsilon; double M) {
    initialize vectors: g_1,...,g_c arbitrarily; //exemplar points
    previousCost = infinity;
    PrevCost = infinity;
    PrevAccuracy = computeAccuracy(g_1, ..., g_c, T);
    loop {
        TotalCost = 0.0;
        for (v = 1 to c) {
            n_v = 0; // (n is a vector)
        }
        for (each datapoint Y in T) {
            v = Y.c;
            find g_w cloest to u(Y);
            if (w != v) {
                Cost = distSquared(g_v, u(Y)) - distSquared(u(Y), g_w);
                if (Cost < M) {
                    n_v += u(Y) - g_v;
                    n_w += g_w - u(Y);
                    TotalCost += Cost;
                } else {
                    Cost += M;
                }
            }
        }
        if (TotalCost < epsilon) {
            return g_1, ..., g_c; // Training complete
        }
        if (TotalCost > (1-epsilon)*PrevCost) {
            return g_1,...,g_c; // improvement too small
        }
        for (v = 1 to c) {
            h_v = g_v + step_size * n_v;
        }
        NewAccuracy = computeAccuracy(h_1, ..., h_c, T);
        if (NewAccuracy < PrevAccuracy) {
            return g_1, ..., g_c;
        }
        for (v = 1 to c) {
            g_v = h_v;
        }
        PrevCost = TotalCost;
        PrevAccuracy = NewAccuracy;
    }
}
```

Program

The values of parameters stepSize, epsilon and M will be given in command line arguments, as specified below. Another command line argument will be the number of iterations of random restart to run.

The driver program will call gradDescent a number of times, with different initializations of the vectors $\vec{g}_1, ..., \vec{g}_c$. On the first pass, they will be initialized to be the centroid of their respective categories, as in the problem in the written component. On the remaining passes, they will be initialized randomly. For each attribute, find the minimum and maximum value in the training set; then sample uniformly between the minimum and maximum to randomly initialize the vectors.

After the classifier has returned, the fraction of points in the training set classified correctly is computed. (This class of classifiers is too inexpressive for overfitting to be a concern; unless the training set is extremely small, the percentage correct on the training set and on the test set will be pretty much the same.)

The training set should be in a CSV (comma separated values) file. You may assume that there are no null values. The predictive attributes are floating point numbers, and the values of the classification attributes are single lower-case letters 'a' through 'z'. (There are not more than 26 values.) For example, the sample data from the written component would like the following as an input file:

```
1.0,1.0,2.0,a
2.0,1.0,1.0,a
2.0,0.0,1.0,a
0.0,2.0,1.0,b
3.0,2.0,0.0,b
3.0,3.0,0.0,c
0.0,3.0,0.0,c
3.0,2.0,1.0,c
0.0,3.0,3.0,c
```

Command Line Arguments Your program classify should take five or six command line arguments, in sequence: the name of the file for the training set; the values of stepSize, epsilon and M; the number of random restarts, and, optionally "-v" to indicate verbose output. (The -v is optional when running your program, but it is mandatory to add support for it as a part of the assignment).

For example in Java, the command to execute your program might be: java classify -v training.csv 0.1 0.01 0.2 100

Output In non-verbose output, just output the fraction of the training set correctly classified in each run, and, at the end, the best accuracy obtained in any run.

In the verbose output, output a trace of the sequence of values of the exemplar vectors, and of the successive values of the accuracy.