Artificial Intelligence - Problem Set 5

Mark Xavier (xaviem01)

May 12, 2019

1. Problem 1

   (a) For training, we walk through each example once, determine the classification to determine which centroid to compute, then convert the attributes to vectors and sum them. Each of these summed vectors is then multiplied by one over the number of items in that classification. Then based on input, training is simply $\approx O(N)$ running time, not taking into account the time for division.

   For testing, we take the new instance and find the distance between this instance and that of the centroids, then determine which centroid is closes. There are c centroids, and there are k predictive attributes, each an element in the vectors. Then, for c centroids we have to determine the distance between two k element arrays, leading to a running time of $\approx O(ck)$.

   (b) The approach in cases of missing values is to simply use the values that have been obtained and compare them to corresponding values in other vectors. If a new instance comes in $z =< a_1, \emptyset, a_3 >$, then in all comparisons we simply grab the first and third element of each vector when we test distance. During the training phase, we can simply use the attributes we have. We could even remove attributes from the sets if they are missing from certain instances as long as we can avoid removing a large set of instances (for instance, if many instances do not include a_2 then we can simply remove a_2 from all instances) - this would lower some of the predictive power of our algorithm since we're removing information, but this would help in the case of missing attributes.

   (c) Assuming that we generate vectors from a Gaussian distribution from a given mean and standard deviation, then we can cluster groups of vectors using that mean and deviation. Then the generative model is that we have some categories that each have a vector that represents the mean and a vector representing the deviation. Then through a stochastic process, we generate some number of vectors using that mean and deviation vector through each classification, and those generated vectors serve as our labeled data.

   (d) We can get unpredictable results from the following set-up:

Predictive Attribute	Classification
1	b
1	a
2	a
2	c

   In the case above, we see that category a actually sits in-between categories b and c, and that the distance from a to b and from a to c is 1. Then when the centroid for a is calculated, it is calculated as 1.5, which is 0.5 away from centroid a. However, this is directly on the classification line for b and c, each of which are 0 away from either point in a (we can even make them 0.1 or such away). This means that the centroid distances for b and c will classify both points in a as being in either b or c. Therefore, though category a exists in training and is linearly separable, without enough points to define better centroids for b and c, we classify everything as b or c.

   (e) The problem here is with the width of the windows given for $X.A$. An example where this fails is simple, assume that the centroid calculated for $X.C = a$ is very close to 0. The centroid for $X.C = b$ is also close to 0, at most a euclidean distance of 1 away from 0. Now if a data point close to 0 appears, we run into an issue, does it classify as a or b? If on the other hand the centroid for $C = a$ is extremely far from 0, we may categorize more values as b than a by mistake, since values closer to 0 are closer to category b than a.

   This situation extends through the example, categories a and e have very large category windows, b and d have small category windows, and c has a medium sized window. The differing windows means that, at least distance-wise, points will appear closer to categories that they are not actually a part of, since any $X.A_i$ can take on any value within the given windows.

   (f) One method for doing this is to compute exemplars by computing some combinations of the training data. For example, every 2 or 3 vectors encountered in a given category c can be averaged to find a mean vector, but this may cause issues for vectors very far apart. Another method is to simply determine a halfway point between two vectors, and keep that as a centroid. The latter method holds information about both vectors, and as many as $N - 1$ of these can be created in linear time (we can obtain many more by finding distances between all vectors, but this running time is $\approx O(N^2)$. Either of these sets of centroids stores some kind of information about the vectors used to compute them, and therefore can be used in place of the centroid(s) calculated by the prompt.

(g) The exemplars here a simple to define, they are all as close as possible to their respective boundaries (2 exemplars where boundaries are two-sided, only when if they are open). An example set is included in the table below:

X.A	X.C
-0.01	a
0.01	b
0.99	b
1.01	c
4.99	c
5.01	d
5.99	d
6.01	e

2. Problem 2

   (a) Naive-Bayes will classify vector $< 2, 2, 2 >$ as being in category b.

   (b) 3-Nearest-Neighbors identifies the vectors at indices 0, 1, and 7 as being the nearest to 2, 2, 2 (categories a, a, and c). This identifies the vector as being in category a.

3. Problem 3

   (a) The error function simply sums up the weight times the vector $< x, y, 1 >$ for all misclassified points, which turns out to be $(1 - 1 - 3) + (2 - 2 - 3) + (3 - 4 - 3) = -10$, so the cost function is 10.

   (b) Since correctly labeled points are multipled by 0, we can ignore them, Looking at the 3 mislabeled points above, the first was labeled red but was blue, therefore we get $< 1, 1, 1 >$. For the other two, they were red but labeled blue, so they get multiplied by -1, given $< -2, -2, -1 >$ and $< -3, -4, -1 >$. Summing these gives: $< -4, -5, -1 >$.

   (c) The new weight vector = $< 1.4, 1.5, -2.9 >$.

   (d) The change is that now the updated weights get closer to categorizing the first point correctly to blue, however it doesn't work yet (the result is 0, which is just under the threshold). So in actual terms, classifications don't change at all.

   (e) The error function computes the same value as in the first part (10) as no vectors have been re-categorized after the weights were updated.