CS 493: Algorithms for Massive Data Sets	**Classification and clustering**
March 14, 2002	**Scribe: Edith Elkind**

1 Naive Bayes classifier

Consider the task of picking a hypothesis that gives the best explanation for the observed data. More formally, suppose that we are given a version space $V = \{v_j\}$ with known apriori probabilities $P(v_j)$. Denote the observed data by $a_1, \ldots, a_n$. We are interested in $V_{MAP} = \text{argmax}_{v_j \in V} P(v_j \mid a_1, \ldots, a_n) = \text{argmax}_{v_j \in V} P(a_1, \ldots, a_n \mid v_j)P(v_j)$.

The naive Bayes classifier replaces $P(a_1, \ldots, a_n \mid v_j)$ by $\prod_{i=1,\ldots,n} P(a_i \mid v_j)$. Although usually the observed events are not independent, this approximation works well in practice and reduces storage requirements from exponential (the probability for each subset of possible events and each version) to polynomial (number of possible events $\times$ number of versions).

For example, suppose that we are trying to classify a collection of newspaper articles based on words that occur in them. The possible classes include "sports", "entertainment", etc. We can approximate probabilities by frequencies:

$$P(a_i \mid v_j) = \frac{\#\text{docs in class } j \text{ that contain } i}{\#\text{docs in class } j} = \frac{n_j^i}{n_j}.$$

However, the numerator of this expression can be equal to zero, so using this formula is problematic. The technique used to overcome this difficulty is called *smoothing*. Namely, n_j^i/n_j is replaced with $(n_j^i + pm)/(n_j + m)$, where p is some apriori probability and m is sufficiently big. This can be interpreted as adding to our database m documents that contain this word with probability p. Usually, p is set to $1/|\text{vocabulary}|$, and m is set to $|\text{vocabulary}|$, so the latter expression becomes $(n_j^i + 1)/(n_j + |\text{vocabulary}|)$.

Also, in general, the probability of seeing a specific word may depend on the word position (that is, $P(w_1 = \text{"goal"} \mid \text{class} = \text{"sports"}) \neq P(w_2 = \text{"goal"} \mid \text{class} = \text{"sports"})$), but usually it is assumed to be the same.

Other tricks used here are to exclude very common words (articles, prepositions), or very unusual words.

2 Expectation Maximization (EM)

This is an approach to learning in the presence of hidden variables. It was introduced by Dempster et al. in 1977. We explain it by example.

Suppose that we are trying to learn a mixture of two Gaussians. That is, our data is generated by picking one of the two Gaussians at random and then choosing an instance according to the selected distribution.

Recall that a normal distribution is given by the formula

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}.$$

In our case, $\sigma_1 = \sigma_2 = \sigma$ is given, and our goal is to output a maximum likelihood hypothesis for μ_1, μ_2.

Note that if there is only one distribution, its mean can be estimated by the sample mean. This follows from the fact that the maximum likelihood hypothesis is the one that minimizes the square error, and obviously, $\min_\mu \sum_{i=1,\ldots,n} (\mu - x_i)^2$ is achieved at $\mu = (x_1 + \ldots + x_n)/n$. Hence, if we knew which instances are generated by which distribution, we could easily solve the problem. Following this intuition, we can think of each instance as a triple of the form $\langle x_i, z_{i1}, z_{i2} \rangle$, where z_{ij} is 1 if x_i is generated by jth distribution and 0 otherwise; the variables z_{ij} are *hidden*.

The EM algorithm starts with an arbitrary hypothesis $h = \langle \mu_1, \mu_2 \rangle$ and then works iteratively by estimating the expected values of hidden variables based on the current hypothesis for $\langle \mu_1, \mu_2 \rangle$, and then adjusting the current hypothesis. That is, each iteration consists of two steps:

- Compute $E[z_{ij}]$ assuming $h = \langle \mu_1, \mu_2 \rangle$:

$$E[z_{ij}] = \frac{P(x \text{ is generated by } j)}{P(x \text{ is generated by } 1) + P(x \text{ is generated by } 2)} = \frac{e^{-\frac{1}{2}(\frac{x-\mu_j}{\sigma})^2}}{e^{-\frac{1}{2}(\frac{x-\mu_1}{\sigma})^2} + e^{-\frac{1}{2}(\frac{x-\mu_2}{\sigma})^2}}.$$

- Calculate $h' = \langle \mu'_1, \mu'_2 \rangle$ assuming $z_{ij} = E[z_{ij}]$:

$$\mu'_j = \sum_{i=1}^{n} z_{ij} x_i / \sum_{i=1}^{n} z_{ij}.$$

It can be proved that each iteration increases the likelihood of the data given the current hypothesis, unless a local maximum is achieved. However, this algorithm does not always find the global maximum.

This approach generalizes to an arbitrary number of Gaussians (but does not perform well if the distributions are close to each other) and also to higher dimensions (in this case, μ is a vector and $P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{\|x-\mu\|}{\sigma})^2}$). Even more generally, it can be used in other learning situations involving hidden variables.

The general setting for EM algorithm involves a set of m independently drawn instances and two sets of variables $X = \{x_1, \ldots, x_m\}$ and $Z = \{z_1, \ldots, z_m\}$ that correspond to the observed data and hidden variables, respectively. Set $Y = X \cup Z$.

We assume that the set of all hypotheses is parametrized by θ and denote the current hypothesis by h. Then, the goal is to find θ that maximizes $E[\ln P[Y \mid h] \mid X]$.

The algorithm consists of two steps:

Step 1 (Expectation) Calculate $Q(h' \mid h) = E[\ln P[Y \mid h'] \mid h, X]$ (this is a score for each possible hypothesis h' based on h, X).

Step 2 (Maximization) Set $h' = \text{argmax}\ Q(h' \mid h)$.

It is not hard to see that the algorithm we presented earlier is a particular case of this approach.

3 Clustering

The general idea of clustering can be described as follows: given a set of points, divide them into clusters so that similar points belong to the same cluster, while dissimilar points belong to different clusters.

The example of two Gaussians is an instance of this paradigm: for each point, we are trying to guess the distribution that generated it, thus dividing the points into two groups.

In general, the pairwise distances between points are given by a matrix, although in some cases there might be a closed expression for the distance as well (e.g., $l_p = (\sum_i |x_i - y_i|^p)^{1/p}$).

The methods used in practice can be divided into two big groups: hierarchical (bottom-up) and partitional (top-down).

The hierarchical methods start with clusters that consist of a single element. They then merge "closest" clusters and update the distances between clusters. This step is repeated until only one cluster remains. The entire process can be described by a tree-like diagram.

We consider three flavors of this approach, which differ by how they recompute the distances.

Single link: Suppose that we merge A and B to obtain C. Then for any cluster D, set $d(D, C) = \min\{d(D, A), d(D, B)\}$.

Complete link: In the same setting, set $d(D, C) = \max\{d(D, A), d(D, B)\}$.

Average: In the same setting, set $d(D, C) = (d(D, A) + d(D, B))/2$.

Not all of these methods perform well on all examples: bad inputs for single link and complete link are given by Fig. 1 and Fig. 2, respectively. In practice, complete link is the one that is used more often.

Among partitional methods, we identify mixture resolving (e.g., EM algorithm), square error (k means), and graph-theoretical methods.

MST-based method belongs to the class of graph-theoretical methods. It finds the minimum spanning tree in the graph whose vertices are the given points and the edge weights are the corresponding distances, and consecutively breaks down the most expensive edges. It is equivalent to single-link.

In k means algorithm, the goal is to form k clusters so that the expression

$$\sum_{j=1}^{k} \sum_{i=1}^{n_j} \|x_i^j - c_j\|$$

Figure 1: Hard example for single link

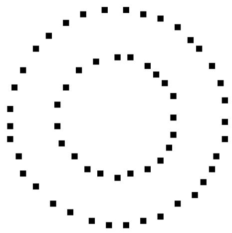

Figure 2: Hard example for complete link

is minimized (here x_i^j is the ith point in the jth cluster, c_j is the center of the jth cluster, and n_j is the number of points in the jth cluster).

The algorithm starts by guessing k cluster centers. It then forms k clusters by assigning each point to the closest center, and recomputes the centers of these clusters. This step is repeated until some convergence criterion is met.

This method is similar to expectation maximization, but differs from it in that in the end each point is assigned to a specific cluster, while in EM each point is assigned a probability of belonging to each cluster. Generally, methods of the first type are described as *hard clustering*, while methods of the second type are described as *fuzzy clustering*.

3.1 Provable heuristics

Although finding exact solutions for many natural clustering problems is NP-hard, in many cases, it is possible to come up with provable heuristics.

For example, assume that the distance function satisfies the triangle inequality, and consider two possible goals:

k-center Minimize $\max_{i,j} \|x_i^j - c_j\|$.

k-median Minimize $\sum_{j=1}^{k} \sum_{i=1}^{n_j} \|x_i^j - c_j\|$.

Assume that the number of clusters is fixed and all points and all possible centers are vertices of a weighted graph.

Both of these problems are NP-hard. For k centers, it is easy to come up with a 2-approximation algorithm and NP-hard to approximate it within a factor of $2 - \epsilon$. For k medians, it is NP-hard to approximate it within some constant factor c, where $c > 1.3$; the best known algorithm gives a $3 + \epsilon$-approximation for any fixed $\epsilon > 0$.

3.2 2-approximation algorithm for k centers

Suppose that the optimal solution for k centers has radius R. We construct a solution that has radius $2R$ and uses at most k centers. To achieve this, we pick an arbitrary point x, build a cluster that consists of all points whose distance from x is at most $2R$, and then repeat this process with remaining points until all points are distributed between clusters. To see that the number of clusters is at most k, consider the sequence of points $x_1 = x, x_2, \ldots$ that we picked during this process. Note that the pairwise distances between these points are at least $2R$, so they cannot belong to the same cluster in the optimal solution. To turn this idea into an actual algorithm, we have to show that it is possible to guess the radius of the optimal solution. However, it is easy to see that in graph-theoretic setting this is indeed the case.

Another algorithm that achieves the same approximation factor works as follows: pick an arbitrary point x_1, set $x_2 = \operatorname{argmax} d(x, x_1)$, $x_3 = \operatorname{argmax} \min\{d(x, x_1), d(x, x_2)\}$, etc. The proof of correctness is left as exercise.

4 References

- Machine Learning, Tom Mitchell, McGraw Hill, 1997 (chapter 6).
- Data clustering: a review, A. K. Jain, M. N. Murty, P. J. Flynn, ACM Computing Surveys, 31(3), 1999.