Data Mining Primer

Alex Sverdlov
alex@theparticle.com

Introduction

Data Mining is a short name of a conglomeration of methods that extract information out of data. Such methods include: managing data (database systems), looking for patterns (statistics), learning things (machine learning), and making intelligent decisions (artificial intelligence).

The field is somewhat limitless in a sense that anything can be described as "data mining" to some extent. For example, a scientist makes an observation, collects data, forms a hypothesis, tests hypothesis against collected data, makes predictions, decides whether hypothesis is good or bad on whether it makes good predictions. Every step in that process is some branch of data mining, or machine learning, or artificial intelligence, etc.

In other words, we are interested in extracting useful information out of data.

Stats Primer

Samples are subsets of a population. Stats on samples are often used to figure out things about the population. The same dataset may be a sample in some scenario and a population in the other. For example, a population of all the test scores in the class may be a sample of all the test scores in the school.

Location

Mean, or average, provides a typical (average) example of the data. We compute it via:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

We often use μ to refer to population mean, and $\bar{x}$ to sample mean. They are both computed in the same way.

The Law of Large Numbers tells us that sample mean approaches population mean. That means that the difference $\bar{x} - \mu$ approaches zero as we sample more values.

Mean is considered a measure of 'location' of the data: mean is often used as the center of the dataset.

Another measure of location is percentile; it is a value which bounds a certain percentage of observations. For example, for 95th percentile, 95% of values are below it and 5% are above. One straight forward way to calculate it is to sort the dataset, pick a value right below and right above the desired percentile, and then do a weighted average.

Median is just 50th percentile; it is the middle value (or weighted average of the two middle values). It is often used as a more stable version of the 'center of the dataset'.

Dispersion

While the average/median tells us *where* the data is (where the center is), it doesn't tell us how spread out it is. We get that from variance:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

or sample variance:

$$s^2 = \frac{1}{(n-1)}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Notice the difference between σ^2 and s^2. The basic idea is that since we are not dealing with the whole population, we know sample variance has to be higher than population variance; the extra factor adjusts for that uncertainty.

The standard deviation is just a square root of variance:

$$\sigma = \sqrt{\sigma^2}$$

Replace σ with s to get sample standard deviation.

Just as with mean vs median, there is also interquartile range: is it the range of the middle 50% of the values; in other words, 75th percentile minus 25th percentile.

There's also coefficient of variation, which is a percentage measure:

$$cv = \frac{s}{\bar{x}} \times 100\%$$

Standard error is:

$$se = \frac{s}{\sqrt{n}}$$

A normal distribution with mean 0 and standard deviation of 1 is known as: Z distribution. It is used to normalize the dataset, and allow for direct comparisons of different magnitude data:

$$z = \frac{x - \bar{x}}{s}$$

Practically, we often do not want to calculate $\bar{x}$ before calculating variance. We can fix that with a bit of algebra:

$$s^2 = \frac{1}{(n-1)}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2/n}{(n-1)}$$

The above formulation means that in order to calculate mean and standard deviation (or z-scores), we need to keep track of only: N, $\sum x$, and $\sum x^2$.

Another thing worth mentioning is *absolute deviation*: instead of summing over squares of differences, we can sum over absolute values of differences.

Covariance & Correlation

Variance can be extended to more than one variable via covariance:

$$cov(x, y) = (x - \bar{x})(y - \bar{y})$$

Covariance matrix is symmetric. Off-diagonal elements determine the amount of covariance between variables.

If we normalize covariance (divide it by standard deviations of x and y) we get correlation:

$$corr(x, y) = cov(x, y)/sd(x)sd(y)$$

To compute correlation in a single pass, we can write it as:

$$corr(x, y) = (E(x * y) - E(x) * E(y))/(\sqrt{E(x^2) - E(x)^2} * \sqrt{E(y^2) - E(y)^2})$$

where $E(x)$ is the expected value of x. Correlation matrix is symmetric, with every elements being in the -1 to 1 range.

Covariance is often used as a quick way to fit 2D lines to points. For example, a slope of a 2D line is:

$$slope(x, y) = cov(x, y)/var(y)$$

The y-intercept is:

$$intercept = avg(x) - slope(x, y) * avg(y)$$

Central Limit Theorem

The sample distribution of sample means will approach the normal distribution. Note that the theorem does not say which distributions we start out with—they will all approach the *normal distribution*.

Simple Novelty Detection

With a few assumptions (many of which are often ignored in practice), z-scores (e.g. standard deviation) can be used to detect unusual samples. For example, if a z-score is above 2, then that sample is "unusual", in a sense that (if our assumption of normal distribution pans out) it is unlike 95% of the other samples. If z-score is above 3, then it is about 99%. With such measurements, we can convince ourselves that the stock market crash of 2008 occurs once every few million years.

This kind of novelty detection can be applied online: maintain a window of count, sum, and sum of squares of (perhaps 20 minutes, or 20 days, etc. Then using these you can calculate average, and standard deviation (or z-scores), and then quickly apply the novelty check towards the previously unseen sample.

k-Means Clustering

Clustering is a process of grouping similar items together. k-Means is an iterative process of finding those groups. We start with k random points, call them "means", then proceed to assign each sample to exactly one (closest to that sample) mean. After all the samples have been assigned to a mean, we recalculate the mean using the samples assigned to it. Then repeat (reassign all samples to their closest mean).

The iteration stops when we no longer re-assign samples from one mean to the next. The k means represent k groups within the data. For any new (previously unseen) sample, we can find which group it belongs to by comparing it to k means, and picking the closest one.

The k-Means algorithm is an example of a unsupervised learning. We don't tell the algorithm what we consider important—it figures it out all by itself. For example, if we quantize a million customers and ask k-Means to come up with 7 clusters, it will do just that—bucket each customer into a cluster of similar customers.

There is no magic to the procedure. For it to work, the *distance* measure must be defined and be meaningful in some way.

Other Clustering Methods

k-Means is the simplest of all clustering methods. It requires knowledge of k, or how many clusters we want to end up with. There are two hierarchical clustering methods: agglomerative and divisive.

Agglomerative clustering methods start with each sample being in a cluster by itself—and then iteratively merging the two closest clusters, eventually ending up with 1 big cluster including everything (and a whole hierarchy of clusters).

Divisive clustering works in reverse. Everything starts out as part of one big cluster, and then iterations split the existing clusters creating new ones.

Most clustering schemes assume that the centers of clusters are 'points. Linear manifold clustering attempts to fit data to low dimensional linear manifolds. For example, a 0-manifold is a point, a 1-manifold is a line, a 2-manifold is a plane, etc.. The clustering works by sample N points (two for a line, 3 for a plane, etc.), there is a certain chance that all points belong to the same linear manifold cluster—so synthesized an orthogonal basis using the sampled points (perhaps using SVD).

Compare all points to the synthesized manifold—generate a histogram of distances to manifold. If the histogram has two humps, then we have a linear manifold cluster. The points in the cluster are removed, and iterations continue looking for other clusters.

Mean Classifier

Classification is essentially assigning a group label (usually 1 or -1) to new samples. Unlike clustering, we generally start with some training data (samples with labels). The mean classifier is perhaps the simplest: we find the mean of all positive (label +1), and negative (label -1) samples. We can then compare new samples to the two means, and pick closest one.

Some applications require a hyper plane "rule" (a linear discriminator). We can get that by taking a vector from negative to positive mean, and use ratio of variances (of negative and positive samples) to determine whether threshold should be.

Least Squares: Fitting lines, curves, hyper planes

Certain relatively simple problems come up very often. One such problem is solving linear equations. An example may be: find a line that passes through points $(2, 13)$ and $(3, 17)$. We need to solve for a and b in the below equations:

$$2a + b = 13$$

$$3a + b = 17$$

Such things are easier to write in matrix form:

$$\begin{bmatrix} 1 & 2 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} = \begin{bmatrix} 13 \\ 17 \end{bmatrix}$$

What we have is a classic equation: $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$, where we need to solve for $\boldsymbol{w}$. Rearranging things a bit, we end up with two solutions for $\boldsymbol{w}$[1]:

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

$$\boldsymbol{w} = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{y}$$

With these, we learn that $\boldsymbol{w}$ is:

$$\begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

In other words, the line is: $4x + 5 = y$.

We can use this method to solve any such linear system! Often, the matrices $\boldsymbol{X}^T\boldsymbol{X}$ or $\boldsymbol{X}\boldsymbol{X}^T$ will not be invertible—so we modify the solutions to always create invertible matrices:

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

[1]The least squares loss function is derived from an assumption that the sample data set S is generated by a smooth function with Gaussian noise. The probability of the sample data S, is a product of probabilities of individual points, which is proportional to

$$P \propto \prod_{i=1}^{L} e^{-\frac{1}{2}\left(\frac{y_i - f_{w_1,\ldots,w_M}(\boldsymbol{x}_i)}{\sigma}\right)^2}$$

where σ is the sample data standard deviation, which we assume to be constant. Maximizing this probability is equivalent to minimizing the negative of its logarithm, which is equivalent to the sum of squares loss function.

$$\boldsymbol{w} = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$$

Here, $\boldsymbol{I}$ is an appropriately sized identity matrix, with λ being a small constant, such as 0.001.

Why are there two solutions, and which one would we use? That depends on the shape of our problem: Let us do a slightly more complicated example: find a line that passes through points $(2, 13)$, $(3, 17)$, $(5, 23)$, $(7, 29)$, $(11, 31)$, $(13, 37)$. We need to solve for a and b in the below equations:

$$2a + b = 13$$

$$3a + b = 17$$

$$5a + b = 23$$

$$7a + b = 29$$

$$11a + b = 31$$

$$13a + b = 37$$

Rewriting in matrix form:

$$\begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 5 \\ 1 & 7 \\ 1 & 11 \\ 1 & 13 \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} = \begin{bmatrix} 13 \\ 17 \\ 23 \\ 29 \\ 31 \\ 37 \end{bmatrix}$$

Plugging in solutions for $\boldsymbol{w}$, we get:

$$\begin{bmatrix} 11.4437 \\ 1.9836 \end{bmatrix}$$

In other words, the line is approximately: $1.9836x + 11.4437 = y$. Notice that this line does not fit any of the points perfectly, yet it approximately fits all of them! So why the two solutions? The $\boldsymbol{X}^T\boldsymbol{X}$, or primal solution, needs to invert a 2×2 matrix, while the $\boldsymbol{X}\boldsymbol{X}^T$, or dual solution, needs to invert a 6×6 matrix—in this case, doing the primal solution is much faster. For situations when we have few points in many dimensions, solving the dual solution is often faster. For example:

$$\begin{bmatrix} 1 & 2 & 3 & 5 & 7 \\ 1 & 3 & 5 & 7 & 11 \\ 1 & 5 & 7 & 11 & 13 \end{bmatrix} \begin{bmatrix} c \\ b \\ a \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

Solving for $\boldsymbol{w}$ is much simpler via the dual method ($\boldsymbol{X}\boldsymbol{X}^T$) as it only requires inverting a 3×3 matrix, and not the 5×5 matrix required by the primal ($\boldsymbol{X}^T\boldsymbol{X}$) solution. The solution by either method is

$$\begin{bmatrix} 1.627295 \\ -0.075596 \\ -1.320899 \\ 1.476102 \\ -0.556916 \end{bmatrix}$$

Notice that the above isn't a line; but a hyperplane!

Non-Linear Embedding

Now for a bit of magic: this linear method can fit non-linear functions, via the process of embedding. For example, if you want to fit $y = Be^{Ax}$, we can take log of both sides to get $\ln(y) = Ax + \ln(B)$, which is linear. Now you simply use that form in $\boldsymbol{X}$ and $\boldsymbol{y}$ and what you're fitting will be the non-linear $y = Be^{Ax}$.

Similarly, to fit power function $y = B * x^a$ you can take log of both sides to get $\ln(y) = \ln(B) + a * \ln(x)$, which is now linear.

This embedding is very powerful. The idea is to embed your non-linear data into some higher dimensional space that perhaps has linear structures, and then use a linear solver.

To fit polynomials we "embed" higher dimensions that are powers of x. For example, instead of

$$\begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 5 \\ 1 & 7 \\ 1 & 11 \\ 1 & 13 \end{bmatrix}$$

from example above, which would fit a line, you can fit a 3rd degree polynomial just by tweaking that matrix to be:

$$\begin{bmatrix} 1 & 2 & 2^2 & 2^3 \\ 1 & 3 & 3^2 & 3^3 \\ 1 & 5 & 5^2 & 5^3 \\ 1 & 7 & 7^2 & 7^3 \\ 1 & 11 & 11^2 & 11^3 \\ 1 & 13 & 13^2 & 13^3 \end{bmatrix}$$

The resulting solutions will have the form $y = D + Cx + Bx^2 + Ax^3$. This can be extended to any degree polynomial you care to fit.

Kernel Trick

The embedding procedure can be avoided by noticing $\boldsymbol{X}\boldsymbol{X}^T$ in the primal solution for w. The $\boldsymbol{X}\boldsymbol{X}^T$ can represent the *kernel* function, such as inner join of ith row with jth row, or such inner join squared, etc.

The kernel 'trick is that treating $\boldsymbol{X}\boldsymbol{X}^T$ as a kernel often allows for very complicated non-linear embeddings—even into infinite dimensions—without us ever actually calculating the embedding itself.

Interpolation & Extrapolation

Once we get the 'line' (or polynomial, or hyperplane, etc.), what can we do with it? Well, we can fill in missing values—for sample, lets say we have values from 1 to 100, but we have some gaps in the middle of the dataset. We can fill those in simply by plugging the values into the 'line' (or whatever we've fit). That is called interpolation.

The other thing we can do is project our queries outside the sample used to fit the line. For example, we've fit an exponential curve to earnings data for the last 2 years, and we would like to guess what the earnings will be next quarter. This is called extrapolation—and is often much less precise than interpolation.

Least Squares Discriminator

While the 'least squares' method described above is used primarily for interpolation and extrapolation, a similar technique can be used for classification. Given a training set:

$$S = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_L, y_L)\}$$

where $y_i \in \{-1, +1\}$ indicates the class. Our model is a hyperplane, with weights $\boldsymbol{w}$, and distance D, such that:

$$w_1 x_1 + \cdots + w_N x_N = D$$

With such a hyperplane, we get a notion of things being in 'front' of the plane and in the 'back' of the plane. If we plug $\boldsymbol{x}$ into the plane equation (represented by $\boldsymbol{w}$ and D), and get a positive value, then $\boldsymbol{x}$ is in front of the plane, etc.

To turn this problem into the 'least squares' problem described above consider the dual solution. We only used the inner products to find the interpolating line. Now we need to incorporate the y_i values into that Gram matrix. What we end up with is known as a Hessian matrix:

$$H_{ij} = y_i y_j \boldsymbol{x}_i \boldsymbol{x}_j$$

Notice that the H matrix is essentially the kernel multiplied by $y_i y_j$, e.g.

$$H_{ij} = y_i y_j \boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

This allows for non-linear classifiers.

We can obtain a KKT (Karush-Kuhn-Tucker) system:

$$\left[\begin{array}{c|c} 0 & \boldsymbol{y}^T \\ \hline \boldsymbol{y} & H \end{array}\right] \left[\begin{array}{c} -D \\ \hline \boldsymbol{\alpha} \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline \boldsymbol{1} \end{array}\right]$$

where $\boldsymbol{y} = (y_1, \ldots, y_L)$, $\mathbf{1} = (1_1, \ldots, 1_L)$, H is the Hessian matrix, and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_L)$ are Lagrange multipliers. We can then solve for $\boldsymbol{w}$ via:

$$\boldsymbol{w} = \boldsymbol{X}^T \left[\boldsymbol{\alpha} \times \boldsymbol{y}\right]$$

where $\boldsymbol{\alpha} \times \boldsymbol{y}$ is an element-wise multiplication.

Maximum Margin Classifier

In general, we want as much separation between classes as possible—we want the classifier to find the maximum buffer. More on this in class.

Support Vector Machines

SVMs are not really machines. They are maximum margin classifiers, with other nice features.

One of the major problems with the least squares method is that it becomes impractical with a relatively low number of samples. For N input samples, we would need an N by N Gram matrix, with most matrix multiplications taking $O(N^3)$ operations—consider a modest problem with 10000 samples to get an idea of how quickly this becomes impractical.

This is where *Support Vector Machines* come in. SVMs are binary classifiers, identical to least squares discriminator in every way, except they don't use all the input samples for training. The important points, as far as classification is concerned, are the ones on the boundaries. If we use just the boundary points, the classifier will be just as good as if we used all the points. The big question now is how to find the boundary points.

Most SVM algorithms have a notion of 'working set'; where, in every iteration, the algorithm picks a 'working set' of input points to use for training. The working set is generally relatively small. Some techniques pick points for the working set which have the maximum influence on the resulting classifier (essentially picking inputs with the corresponding largest Lagrange multipliers). Google for SMO algorithm.

SVMs are often used in conjunction with kernels.