Implementing Bayes Classifiers

Gregory Bint
Evan Huus
Simon Pratt *
Carleton University

March 25, 2014

1 Background

Recall that if we have a sufficiently large collection of sample data drawn from an inherently random set of variables then the *Central Limit Theorem* [1] states that this data will approximate a *Gaussian* or *Normal* distribution [5].

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

Here μ is the mean and σ^2 is the variance. This follows from the idea that we *expect* many of the samples to be close to the expected value, or mean, of the underlying random variable.

When working with data that has multiple attributes or *dimensions*, such as the iris data, there is no such thing as *the* expected value, or *the* sample value. Instead, we use vectors and matrices to record each dimension of our data.

More specifically, if our data has d dimensions (the iris data has 4), then:

$$x \text{ becomes } \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \text{ and } \mu \text{ becomes } M = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_d \end{bmatrix}$$

and σ^2 becomes the $d \times d$ matrix Σ, whose contents depend on the type of classifier you are making, discussed below.

If we modify Equation 1, above, to use these multi-dimensional variables, we get the Gaussian multivariate formula[4], which looks as follows:

$$\frac{1}{\sqrt{(2\pi)^d|\Sigma|}}e^{-\frac{(\vec{x}-M)^T\Sigma^{-1}(\vec{x}-M)}{2}} \tag{2}$$

*Simon can be reached at spratt@scs.carleton.ca for questions.

We can then simplify Equation 2 to the following:

$$\frac{1}{\sqrt{(2\pi)^d|\Sigma|e^{D'(\vec{x},M,\Sigma)}}} \tag{3}$$

Where D' is the square of the *Mahalanobis* distance function [2]. The square of the *Mahalanobis* distance function is defined as:

$$D'(\vec{x}, M, \Sigma) = (\vec{x} - M)^T \Sigma^{-1} (\vec{x} - M)$$

2 Naïve Bayes Classifier

The first thing to note is that the following formulas only work to classify between two different classes. If your training data contains more than two classes, your first step is to choose two classes of interest and disregard all other data.

The steps we follow to build the classifier are as follows:

1. Partition the training data into two sets, those belonging to class A and those belonging to class B.
2. Calculate the Normal Distribution for each set.
3. Form the *classification equation*.

2.1 Partition the data

This step is easy, simply look over the training data and place each instance into the appropriate set according to its class. Recall that we know the class of each sample of our training data.

2.2 Calculate the Normal Distribution Equations

Calculating the normal distribution of the training data is just a matter of calculating M and Σ. In our case, we are calculating the Sample Mean and the Sample Covariance [7] since we are working with sample data.

Sample mean is nothing more than a simple average. For each dimension i of the sample data, calculate the sum total of the values over all instances of the class partition, and then divide by the number of instances to get μ_i.

In Naïve Bayes, our covariance matrix is a diagonal matrix of the following form:

$$\begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_d^2 \end{bmatrix}$$

Each σ_i represents the variance of the particular dimension of the sample data. Variance is calculated with the following:

$$\sigma_i^2 = E[(x_i - \mu_i)^2]$$

Calculating this is just a matter of looping over every sample, and for each one, we calculate the value $x_i - \mu_i$ (recall that we calculated all the μ's earlier), add that value to a running total (i.e. the summation step), and when finished, divide by the number of samples.

Now that we have M and Σ for *each* class, we have done all the hard work of finding the Normal Distribution for this class. We now use these two variables to create the classification equation.

2.3 Build the Classification Equation

Let M_A and Σ_A be the mean vector and covariance matrix for class A, and let M_B and Σ_B be the mean vector and covariance matrix for class B.

In simple terms, classification is performed by plugging an instance of input data, $\vec{x}$, from our testing set, and checking whether class A or class B is more likely (remember: this is all just probabilities). This is equivalent to deciding if the output value of the Normal Distribution equation is higher for class A's equation, or class B's.

$$\frac{1}{\sqrt{(2\pi)^d|\Sigma_A|e^{D'(\vec{x},M_A,\Sigma_A)}}} \gtrless_{\omega_B}^{\omega_A} \frac{1}{\sqrt{(2\pi)^d|\Sigma_B|e^{D'(\vec{x},M_B,\Sigma_B)}}} \tag{4}$$

This formula gives the probability that a given data point is in the distribution. Given the covariance matrix and mean vector for both class A and class B, after many steps of simplification (which can be seen in Appendix A), we get the following inequality:

$$ln|\Sigma_B| - ln|\Sigma_A| + D'(\vec{x}, M_B, \Sigma_B) - D'(\vec{x}, M_A, \Sigma_A) \gtrless_{\omega_B}^{\omega_A} 0 \tag{5}$$

For performing the matrix calculations necessary to create this expression, we recommend the Jama library, available at http://math.nist.gov/javanumerics/jama/, if you are programming in Java, or the NumPy library, available at http://numpy.scipy.org/, if you are programming in Python.

2.4 Classify!

Equation 5 is the final product we are looking for. With this equation in hand, we can classify any instance of our data, $\vec{x}$, by simply plugging it in. The output from the equation will be a single, scalar number. If that number is greater than 0, then $\vec{x}$ *most likely* belongs to class A, otherwise $\vec{x}$ belongs to class B.

3 Optimal Bayes Classifier

The only difference between Naïve Bayes and Optimal Bayes classifiers is the covariance matrix used. In a Naïve Bayes classifier, the covariance matrix is just a diagonal matrix (Section 2.2), but in an Optimal Bayes classifier, we use a full matrix instead, which has the following structure:

$$\begin{bmatrix} \sigma_1\sigma_1 & \sigma_1\sigma_2 & \cdots & \sigma_1\sigma_d \\ \sigma_2\sigma_1 & \sigma_2\sigma_2 & \cdots & \sigma_2\sigma_d \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_d\sigma_1 & \sigma_d\sigma_2 & \cdots & \sigma_d\sigma_d \end{bmatrix}$$

Where each entry is calculated by:

$$\sigma_i \sigma_j = E[(x_i - \mu_i)(x_j - \mu_j)]$$

All of the other steps and procedures for training and testing are the same, the only difference is the complexity of the covariance matrix.

4 Programming a Classifier

Now that we understand the theory, we can implement a working classifier. We are given n data points, each consisting of k numerical observations and a classification.

This data is often stored as a file wherein each line contains observations, an indication of the classification, and possibly extraneous data. This must be parsed and stored in some convenient data structure of your own design.

4.1 Partition the Data into Training and Testing Sets

We begin by partitioning the dataset into a *training set* and a *testing set*. The purpose of the *training set* is to build the covariance matrix and mean vector for each class. The purpose of the *testing set* is to provide independent data with which to test the accuracy of the classifier.

The simplest way to partition the dataset is to choose the first $\lceil n/2 \rceil$ tuples for the *training set* and the remaining $\lfloor n/2 \rfloor$ tuples for the *testing set*. We will discuss other partitioning schemes in Section 4.6.

4.2 Partition the Training Set by Class

Next, we further partition the *training set* by class. If there are two classes, A and B, let the set of data points belonging to class A be T_A and similarly let the set of data points belonging to B be T_B.

It is important to note that if either T_A or T_B is empty, classification will almost certainly be hopelessly inaccurate. It is important to have some strategy to handle this case, and a reasonable strategy would be to throw an error.

4.3 Calculate the Mean Vector

A mean vector is a k-dimensional vector M (recall that k is the number of numerical observations in each data point), in which the i^{th} element is the mean of all i^{th} numerical observations. It is constructed by first building a sum vector whose i^{th} element is the sum of all i^{th} numerical observations, and then dividing each element by the number of data points.

We will need to do this twice, once to build the mean vector M_A for training data T_A and again, separately, to build M_B for the training set T_B.

4.4 Calculate the Covariance Matrix

Whether we are building a Naïve Bayes or Optimal Bayes classifier, we must calculate the variances for each observation, similar to how we calculated the mean for each observation in the previous step.

We construct the covariance matrix by instantiating a $k \times k$ matrix, Σ, with zeroes, and iterating over the data points. For each data point (itself a k-dimensional vector), we subtract our mean vector M. We take the outer product of the result (itself times its transpose) which results in a new $k \times k$ matrix. This new matrix gets added to Σ. Once we've iterated over every data point, we divide each element of Σ by *one less than* the number of data points.

As with the mean vector, we must repeat this procedure twice, as we must independently build Σ_A (with respect to M_A and T_A) and Σ_B (with respect to M_B and T_B).

The formula for Σ_A is as follows:

$$\Sigma_A = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - M_A)(x_i - M_A)^T$$

If we are building a Naïve Bayes classifier, we can take the "full" covariance matrices we just created for the Optimal Bayes classifier and simply set every element to 0, except for those found along the main diagonal (that is, the diagonal starting at the top left and ending at the bottom right).

4.5 Classify!

We now have everything we need in order to use Equation 5, repeated here for reference:

$$ln|\Sigma_B| - ln|\Sigma_A| + D'(\vec{x}, M_B, \Sigma_B) - D'(\vec{x}, M_A, \Sigma_A) \gtrless_{\omega_B}^{\omega_A} 0$$

Recall that $\vec{x}$ is a vector of numerical observations from our *testing set* that we would like to classify. We calculate the classification value, c, as follows:

$$\begin{aligned} c &= ln|\Sigma_B| - ln|\Sigma_A| + D'(\vec{x}, M_B, \Sigma_B) - D'(\vec{x}, M_A, \Sigma_A) \\ &= ln|\Sigma_B| - ln|\Sigma_A| + (\vec{x} - M_B)^T \Sigma_B^{-1} (\vec{x} - M_B) - (\vec{x} - M_A)^T \Sigma_A^{-1} (\vec{x} - M_A) \end{aligned}$$

If c is greater than zero, the observations most likely belong to class A, otherwise class B, and our program should report as much.

Note: Be careful when calculating the inverse of the covariance matrix, because it may be singular, i.e. have no inverse. In this case, your library may provide a *pseudo-inverse*[3] which is a safe approximation of the true inverse. You will also need to calculate the *pseudo-determinant*[6], since the determinant of a singular matrix is 0, and the logarithm of 0 is undefined.

4.6 Testing

In the first step, we partitioned the sample into a *Training Set* and a *Testing Set*. To test, we simply run the classifier against all the observations in the *Testing Set* and keep a score of how many are correctly classified.

To more accurately test the classifier, you could randomize which observations are used to train the classifier.

You could also use K-fold cross-validation in which the observations are partitioned into K groups and $K-1$ of the groups are used to train and the remaining group is used to test. Using the same partitioning, a new classifier is trained on $K-1$ of the groups and a different group is used to test. This is repeated K times such that every group is used to test once.

Finally, both randomization and K-fold cross-validation can be used together and the results averaged.

A Derivation

The full derivation from Equation 4 to Equation 5 is as follows:

$$
\begin{aligned}
\frac{1}{\sqrt{(2\pi)^d|\Sigma_A|e^{D'(\vec{x},M_A,\Sigma_A)}}} &\gtrless_{\omega_B}^{\omega_A} \frac{1}{\sqrt{(2\pi)^d|\Sigma_B|e^{D'(\vec{x},M_B,\Sigma_B)}}} \\
\left(\frac{1}{\sqrt{(2\pi)^d|\Sigma_A|e^{D'(\vec{x},M_A,\Sigma_A)}}}\right)^2 &\gtrless_{\omega_B}^{\omega_A} \left(\frac{1}{\sqrt{(2\pi)^d|\Sigma_B|e^{D'(\vec{x},M_B,\Sigma_B)}}}\right)^2 \\
\frac{1}{(2\pi)^d|\Sigma_A|e^{D'(\vec{x},M_A,\Sigma_A)}} &\gtrless_{\omega_B}^{\omega_A} \frac{1}{(2\pi)^d|\Sigma_B|e^{D'(\vec{x},M_B,\Sigma_B)}} \\
\frac{1}{|\Sigma_A|e^{D'(\vec{x},M_A,\Sigma_A)}} &\gtrless_{\omega_B}^{\omega_A} \frac{1}{|\Sigma_B|e^{D'(\vec{x},M_B,\Sigma_B)}} \\
\ln\left(\frac{1}{|\Sigma_A|e^{D'(\vec{x},M_A,\Sigma_A)}}\right) &\gtrless_{\omega_B}^{\omega_A} \ln\left(\frac{1}{|\Sigma_B|e^{D'(\vec{x},M_B,\Sigma_B)}}\right) \\
-\ln|\Sigma_A| - \ln(e^{D'(\vec{x},M_A,\Sigma_A)}) &\gtrless_{\omega_B}^{\omega_A} -\ln|\Sigma_B| - \ln(e^{D'(\vec{x},M_B,\Sigma_B)}) \\
\ln|\Sigma_B| - \ln|\Sigma_A| + D'(\vec{x},M_B,\Sigma_B) - D'(\vec{x},M_A,\Sigma_A) &\gtrless_{\omega_B}^{\omega_A} 0
\end{aligned}
$$

References

[1] Wikipedia. Central limit theorem. `http://en.wikipedia.org/wiki/Central_limit_theorem`.

[2] Wikipedia. Mahalanobis distance. `https://en.wikipedia.org/wiki/Mahalanobis_distance`.

[3] Wikipedia. Moore-penrose pseudoinverse. `https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_pseudoinverse`.

[4] Wikipedia. Multivariate normal distribution. `https://en.wikipedia.org/wiki/Multivariate_normal_distribution`.

[5] Wikipedia. Normal distribution. `http://en.wikipedia.org/wiki/Normal_distribution`.

[6] Wikipedia. Pseudo-determinant. `https://en.wikipedia.org/wiki/Pseudo-determinant`.

[7] Wikipedia. Sample mean and sample covariance. `http://en.wikipedia.org/wiki/Sample_mean_and_sample_covariance`.