Christopher Stoll and Michael Crouse
635 Advanced Algorithms
March 27th, 2014

This project had two goals: understand and use single value decomposition to compress an image, and then use SVD to implement principal component analysis to reduce the dimensionality of a large dataset. To achieve this goal, we used the open-source Eigen library for C++, an open-source library for half-precision floating point numbers called Half, and the WEKA library from The University of Waikato. Our image compression program was implemented in C++, and our PCA experimentation was done in a mixture of C++ (using the Eigen library for SVD) and WEKA.

Single value decomposition (hereafter abbreviated SVD) is a powerful linear algebra equation that can be derived for any m*n matrix A. It is written as A = USV*, where U is an m*m matrix comprised of the eigenvectors of AA*, V is the transpose of an n*n matrix comprised of the eigenvectors of A*A and S is a m*n diagonal matrix comprised of the singular values of AA* (and A*A), which are the square root of the eigenvalues. The eigenvectors are usually shortened to unit vectors, meaning the length of the vector is reduced to 1 while maintaining the original direction. The eigenvectors and singular values for U, V and S are sorted by decreasing order of the singular values. The resulting decomposition's rank r is at most the smallest of m or n, but may be smaller if some of the singular values are zero. Any columns of U or rows of V* with position greater than r are unimportant to the calculation of A as they will be multiplied by zero vectors and can be discarded.

Similarly, we can choose to reduce the rank of the SVD to some integer value k, meaning we retain only the largest k singular values and their corresponding eigenvectors in U and V. The resulting matrix is still m*n, as multiplying an m*k, k*k, and k* n matrix still results in an m*n matrix. We can determine what percentage of the information of A is stored in our reduced form by summing up the squares of the singular values and then determining what proportion of the total the eigenvectors we kept represents. Similarly, the error remaining in the shortened dataset can be derived from the next singular value. In this way we could dynamically determine the best k value for the matrix. To store the resulting decomposition, we will need to store k * (n+m+1) values rather than m*n values. Depending on our k value this can lead to a significantly smaller data footprint.

When using SVD to compress a grayscale pgm file, there are some additional issues with storing the eigenvectors and singular values. The individual grayscale values can be stored in 1 byte, plus a 5 byte header to store the height, width and depth of the image. Our U, S and V are all floating point values which require significantly more storage – the traditional double-precision float requires 8 bytes. In order to bring down the storage requirement for our compressed binary we attempted to use half-precision (16 bit) floats, but ran into overflow trouble with images with many white pixels or large images. The overflow was relieved by storing the singular values (which can get quite large) in doubles but using half precision for the eigenvectors. Some precision is lost using this storage method, but using SVD to compress data is lossy to begin with and the effect is difficult to detect with high k values. In the end, our compressed data required 2k* (m+n) + 8k + 7 bytes, while simply storing the pgm in binary required m*n + 5 bytes.

Our results on greyscale images were mixed. For images like the 400x300 image of a field with trees in the background seen in Figure I, results were impressive. At k=16, the image is still quite blurry, but at k=32 the stakes in the field are clearly visible, as are some details of the trees in the background. At this level of compression, the image only requires ~45Kb, a 62% reduction in storage compared to the 120kb original. An increase to k=48 or even k=64 fails to noticeably improve the image.

Figure I – From left to right, then top to bottom – original, k=16, k=32, k=48

Other images were less amenable to compression using this method. For example, Fig.II showing three people playing in the snow causes significant trouble. The coats the subjects are wearing are dark and thus are stored in lower eigenvectors since white is stored as 255 and would be considered more "important" to the decomposition. Because of this, chunks of their coats are missing until k approaches r. At k=32, most of the image looks remarkably similar to the original and its storage requirements of 20.4kb is slightly lower than the 23kb required for the original, but the missing parts of the coats ruin the effect. At k=64, most of the artifacts are gone, but at that point the compressed image is larger than the original! Some initial testing suggests there may be a heuristic that could improve these images but we had not found one at the time of writing.

Figure II – From left to right – original, k=16, k=32

With images that had sharp contrast we noticed a lot of artifacts in the white area of the image. Interestingly, if we scaled the eigenvalues by some factor of k (a preliminary heuristic of 1+1/k produced good results) it would remove the artifacts and also improve the definition of the borders of objects in the image. However, if the image had a light background (such as the snow in Fig.II) all details of the background would be obliterated. As seen in Fig.III, this can be very useful at low rank approximations. This well defined shape with low storage requirements could be very useful for applications such as image recognition by expert systems.

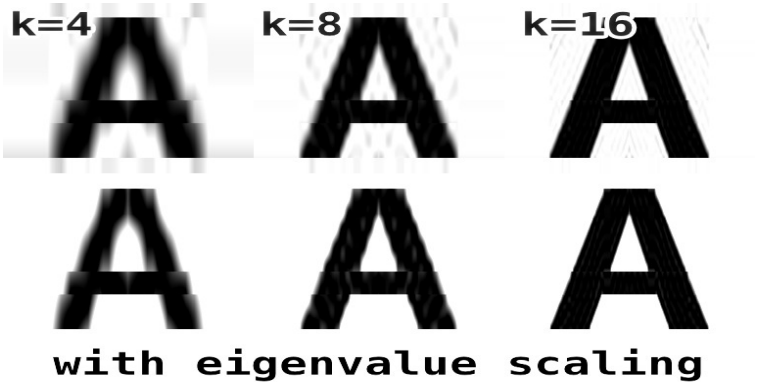

Figure III – scaling eigenvalues can sharpen images noticeably

Principal component analysis is a statistical method that attempts to interpret data in n dimensions by rotating the mean-centered data around a new set of axes called principal components. In order to get these components we calculate the eigenvectors and eigenvalues of the covariance matrix, which is a square matrix comprised of the covariance between the individual variables with the variances within the variables in the diagonal. For example, a covariance for a three variable dataset would look like this:

$$\begin{bmatrix} \mathrm{cov}(x,x) & \mathrm{cov}(x,y) & \mathrm{cov}(x,z) \\ \mathrm{cov}(y,x) & \mathrm{cov}(y,y) & \mathrm{cov}(y,z) \\ \mathrm{cov}(z,x) & \mathrm{cov}(z,y) & \mathrm{cov}(z,z) \end{bmatrix}$$

Figure IV – covariance matrix

Note that cov(x,y) is equal to the summation of all the x and y variables in the dataset multiplied with each other and then divided by n-1, where n is the number of examples in the dataset[1]. Normally, before the covariance matrix is calculated the mean of each variable will be subtracted to the dataset which has the effect of centering the dataset about the mean. It is also possible to divide the dataset by each variable's standard deviation, which results in standardized data instead of centered data. The resulting matrix in this case is called the correlation matrix rather than the covariance matrix. This can be useful if the dataset includes data with wildly different bases of measurement. For example, while testing a PCA implementation we used a made-up sample of students including height and GPA as potential variables. Using centered data, the resulting PCA continually attempted to correlate the height of the student with other factors while ignoring the GPA, due to the much higher variance between heights. Standardization allowed us to compare these variables and correctly deduce that the height was not very important along the first dimension of the PCA. However, if all items in the dataset are of comparable scale, the covariance matrix will produce superior results.

After either centering or standardizing the matrix, it is interesting to note that the covariance matrix for A turns out to be equal to A*A. While we could calculate the eigenvectors of this matrix directly using either of these methods, the same information can be derived from SVD. Specifically, the columns of V store the eigenvectors of A*A and the eigenvalues are the singular values of S squared. Once we have our eigenvectors and eigenvalues, interpreting them in terms of the original variables is surprisingly simple. Each eigenvector's components can be mapped to the original variables in the order they were in the original matrix – if we decomposed a matrix of age, height and GPA, the columns of V will contain 3 eigenvectors, each of which will have a correlation value for age, height and GPA. A negative value will mean that the variable will decrease as variables with positive values increase. We simply need to choose a cutoff for whether an individual variable is important to us for each eigenvector. In our example matrix for student performance using the correlation matrix, the first eigenvector notes a strong contribution from age and final grade (> .5) with a slightly weaker contribution from midterm grade (.46). Taking a close look at the example data, it is clear that the older half of the students had noticeably higher average grades than the younger half.

1	**age**	**height**	**GPA**	**midterm**	**final**
2	21	170	3.6	65	90
3	28	155	3.2	80	80
4	19	183	2.9	75	70
5	32	185	3.8	92	96
6	25	172	2.6	82	86
7	22	144	3.1	75	50

Figure V – example matrix using imaginary data

The main dataset we chose to study was a set of public record data from the Barberton police department. Due to the nature of the data, we were unable to determine any useful information about this dataset. When using the covariance matrix, the results skew heavily towards the involveCountComplainant field. Using the correlation matrix, our first principal component instead latches onto the fields race=0.0 and gender=0.0, which are placeholder values for when the race or gender of the person interacting with the police department is not available to record. The third and fourth eigenvectors do appear to be related to male and female offenders specifically (as they correlate between gender=F or M, WarrantCount and LastWarrantBail) but even then, other than the gender the other fields are relatively weak (~.3). We also had a look at some publicly available data about passengers aboard the Titanic, but found no strong correlations there either.

0.8785 1 -0.379race=0.0-0.369gender=0.0+0.342dobYear+0.309race=W+0.307height...
0.7842 2 0.434involveTotal+0.338gender=0.0+0.322involveCountVictim+0.308involveCountC...
0.7008 3 -0.476gender=F+0.424gender=M+0.313warrantCount-0.266offenseCount+0.258lastWarrantBail...
0.6385 4 0.529gender=F-
0.515gender=M+0.352warrantCount+0.333lastWarrantBail+0.206involveCountArrestee...
0.5851 5 -0.568race=B+0.432race=W-0.322medicalCount-0.285offenseCount-0.266height...

Figure VI – first five eigenvectors for the correlation PCA on our Barberton police data, with what percentage of the data variance is left unresolved on the left

Finally, we built a number of experimental programs regarding PCA. This included several Python scripts and a c++ implementation that used the Eigen library used for our image compression program to perform PCA. These turned out to be very helpful in understanding the inner workings of PCA, such as the difference between the correlation and covariance matrices and how the eigenvectors connect to the original data. In the end we performed most of the calculations on WEKA as it handled complex data (including non-numeric data) better than our experimental code could and allowed us to start interpreting our dataset more quickly.

Resources:

1. Lindsay I Smith. February 2002. A Tutorial on Principal Component Analysis.
Retrieved from
http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf
2 Kirk Baker. March 2005. Singular Value Decomposition Tutorial.
Retrieved from
http://www.ling.ohio-state.edu/~kbaker/pubs/Singular_Value_Decomposition_Tutorial.pdf
3 Half – IEEE 754 based half-precision floating point library
Retrieved from http://half.sourceforge.net/
4 Eigen – C++ template library for linear algebra
Retrieved from http://eigen.tuxfamily.org/index.php?title=Main_Page
5 Weka 3 – Data Mining Software in Java
Retrieved from http://www.cs.waikato.ac.nz/ml/weka/