Learning from Data with Linear Algebra

Gregory Aschenbrenner

University of Connecticut

December 9th, 2020
Is there a limit to how much we can learn from a data set? If so, how do we get there?
Outline

- 1 SVD
- 2 PCA
- 3 Norms
- 4 Numerical Linear Algebra
- 5 Randomized Linear Algebra
- 6 Changes in $A^{-1}$ from changes in $A$
- 7 Interlaced Eigenvalues
- 8 Compressed Sensing and Matrix Completion
- 9 Fourier Transforms
Important Ideas: The SVD

Definition:
The SVD is defined as follows $A = U \Sigma V^T$. Where $U$ is an orthogonal $n \times n$ matrix, $V$ is an orthogonal $m \times m$ matrix, and $\Sigma$ is an $n \times m$. $U$ holds our left singular vectors, $V$ holds our right singular vectors and $\Sigma$ holds our singular values. Note $\sigma_1^2$ to $\sigma_r^2$ are always nonzero eigenvalues of both $A^T A$ and $AA^T$. 
\[ A = U \Sigma V^T \]

\[
\begin{pmatrix}
  u_1 \\
  \vdots \\
  u_r \\
  u_{r+1} \\
  \vdots \\
  u_m \\
\end{pmatrix} \quad \begin{pmatrix}
  \sigma_1 \\
  \sigma_r \\
  0 \\
  \vdots \\
  \end{pmatrix} \quad \begin{pmatrix}
  v_1^T \\
  v_r^T \\
  v_{r+1}^T \\
  v_n^T \\
\end{pmatrix}
\]

\[ \text{col}(A) \quad \text{null}(A) \quad \text{row}(A) \]
Quick example of the SVD

\[
A = \begin{bmatrix}
6 & 2 \\
2 & 4 \\
7 & 8
\end{bmatrix}
\quad
AA^T = \begin{bmatrix}
40 & 20 & 58 \\
20 & 20 & 46 \\
58 & 46 & 113
\end{bmatrix}
\quad
A^T A = \begin{bmatrix}
89 & 76 \\
76 & 84
\end{bmatrix}
\]

\[
\Sigma = \begin{bmatrix}
12.74916 & 0 \\
0 & 3.23402113414883 \\
0 & 0
\end{bmatrix}
\]

\[
U = \begin{bmatrix}
-0.44729 & 0.84570 & -0.29104 \\
-0.33090 & -0.45880 & -0.82462 \\
-0.83092 & -0.27253 & 0.48507
\end{bmatrix}
\]

\[
V^T = \begin{bmatrix}
-0.71863 & -0.69538 \\
0.69538 & -0.71863
\end{bmatrix}
\]
Now that we have the SVD, what can it do?

The notion of PCA is to take the information available to us from the SVD and pick what information is important.

The real focus of utilizing PCA is to find the important pieces of the matrix and use these to make determinations about data sets. In this way PCA can be thought of as unguided learning in some sense, with only linear algebra guiding us.
What is the best rank $k$ matrix, $A_k$ that approximates $A$?

Using the idea of the SVD we define this matrix $A_k = \sigma_1 u_1 v_1^T + \ldots + \sigma_k u_k v_k^T$. Notice the matrix is composed of $k$ rank 1 matrices.

**Eckart-Young**

Thm: If $B$ has rank $k$ then $\|A - B\| \geq \|A - A_k\|$
Example of PCA

Using the $A$ in our SVD example, $A = \begin{bmatrix} 6 & 2 \\ 2 & 4 \\ 7 & 8 \end{bmatrix}$ What is the best $A_k$ to approximate $A$? It is $A_k = \sigma_1 u_1 v_1^T + \ldots + \sigma_k u_k v_k^T$! Thus the best rank 2 matrix that approximates would be $A_2 = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T = \begin{bmatrix} 12.7492 \\ 3.2340 \end{bmatrix}$
(a) PCA on wider spread of data  
(b) PCA on tightly grouped data

**Figure:** PCA in two very different data sets
So many norms so little time!

What is a norm?
A norm of a matrix, as we see below, is really a measure of the size of the elements of the matrix.

Types of norms useful here

Spectral Norm:
\[ \|A\|_2 = \max \frac{\|Ax\|}{\|x\|} = \sigma_1 \]

Forbenius Norm:
\[ \|A\|_F = \sqrt{\sigma_1^2 + \ldots + \sigma_r^2} \]

Nuclear Norm:
\[ \|A\|_N = \sigma_1 + \ldots + \sigma_r \]
What happens when our matrix is poorly behaved?

Three main cases: A is square and singular, ill-conditioned, or simply incredibly massive.

To solve these issues we use a variety of methods, including interesting concepts such as recursive and dynamic least squares, Numerical Linear Algebra, Randomized Linear Algebra, as well as manipulation of the column space’s basis to yield an easier problem.

Methods we are interested in

- Numerical Linear Algebra
- Randomized Linear Algebra
For our purposes we will utilize iterative techniques from Numerically Linear Algebra to orthogonalize a matrix of various sizes.

Methods Used
- Arnoldi Iterations
- Lanczos Iterations
- Conjugate Gradient

Definition:
Krylov Subspace: Given $A$ and $b$, we can compute $b$, $Ab$, ..., $A^{n-1}b$, the combination of these $n$ vectors produce a $n$th Krylov Subspace.
Arnoldi Iteration

(a) MATLAB Code

```
function [Q,H] = arnoldi(A,q1,m)

    n = length(A);
    if nargin < 3, m = n; end
    q1 = q1/norm(q1);
    Q = zeros(n,m); Q(:,1) = q1;
    H = zeros(min(m+1,m),n);

    for k=1:m
        z = A*Q(:,k);
        for i=1:k
            H(i,k) = Q(:,i)'*z;
            z = z - H(i,k)*Q(:,i);
        end
        if k < n
            H(k+1,k) = norm(z);
        end
        Q(:,k+1) = z/H(k+1,k);
    end
```

(b) Pseudocode

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1 = b/|b|$</td>
<td>$q_2, \ldots, q_k$ are known</td>
</tr>
<tr>
<td>Start with new $v$</td>
<td></td>
</tr>
<tr>
<td>For each known $q$</td>
<td>For each known $q$</td>
</tr>
<tr>
<td>Compute inner product</td>
<td>Compute inner product</td>
</tr>
<tr>
<td>Subtract projection</td>
<td>Subtract projection</td>
</tr>
<tr>
<td>Compute norm</td>
<td>Compute norm</td>
</tr>
<tr>
<td>New basis vector with norm 1</td>
<td></td>
</tr>
</tbody>
</table>

Figure: Implementation of Arnoldi Iterations
Given:

\[ A = \begin{bmatrix} 3 & 2 & 4 \\ 1 & 6 & 3 \\ 0 & 0 & 300 \end{bmatrix} \]

Let \( q_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \frac{b}{\|b\|} \) we receive what we expect, \( \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \)

Let \( q_1 = \begin{bmatrix} 3 \\ 6 \\ \pi \end{bmatrix} = \frac{b}{\|b\|} \) yields, \( \begin{bmatrix} 0.4050 & -0.1806 & -0.8963 \\ 0.8100 & -0.3838 & 0.4434 \\ 0.4241 & 0.9056 & 0.0091 \end{bmatrix} \)
Arnoldi Iterations on vector $\mathbf{b}$ and matrix $\mathbf{A}$ yields $\mathbf{Q}_k^T \mathbf{A} \mathbf{Q}_k = \mathbf{H}_k$ where $\mathbf{H}_k$ is Hessenberg Matrix

Hessenberg Matrix is a special square matrix, that is close to triangular. Upper Hessenberg has zero entries below subdiagonal; lower Hessenberg has zero entries above the first superdiagonal.

\[
\mathbf{A} = \begin{bmatrix}
1 & 4 & 2 & 3 \\
3 & 4 & 1 & 7 \\
0 & 2 & 3 & 4 \\
0 & 0 & 1 & 3
\end{bmatrix}
\]

\[
\mathbf{B} = \begin{bmatrix}
1 & 2 & 0 & 0 \\
5 & 2 & 3 & 0 \\
3 & 4 & 3 & 7 \\
5 & 6 & 1 & 1
\end{bmatrix}
\]
Lanczos Iteration

Really is just Arnoldi Iteration for a symmetric matrice. This simplifies the process.

(a) MATLAB Code

```matlab
% Initialize variables
Q = nan(n,k);
q = v / norm(v);
Q(:,1) = q;
d = nan(k,1);
od = nan(k-1,1);

% Perform Lanczos iterations
for i = 1:k
    z = mat * q;
d(i) = q' * z;
    z = z - Q(:,i:i) * Q(:,i:i)' * z;
    if (i ~= k)
        od(i) = norm(z);
        q = z / od(i);
        Q(:,i+1) = q;
    end
end

% Construct T
T = diag(d) + diag(od,-1) + diag(od,1);
% Return user-requested information
if (nargout == 2)
    varargout{1} = Q;
end
```

(b) Pseudocode

Lanczos iteration for $Sx = \lambda x$ (symmetric Arnoldi)

$q_0 = 0$. $q_1 = b/\|b\|$. Orthogonalize $b$, $Sb$, $Sb^2$.

For $k = 1, 2, 3, \ldots$

- $v = Sq_k$, Start with new $v$
- $a_k = q_k^T v$, Diagonal entry in $T$ is $a_k$
- $b_k = \|v\|$, Orthogonal to earlier $q$'s
- $q_{k-1} = v/b_k$, Off-diagonal entry in $T$ is $b_k$
- $q_k = \frac{v}{b_k}$, Next basis vector

**Figure:** Implementation of Lanczos Iterations
Conjugate Gradient Iteration

Solves $Sx = b$ when $S$ is positive definite.

(a) MATLAB Code

```
function x = conjgrad(A,b,tol)
    if norm(r) < tol
        return
    end
    y = -r; z = A*y; s = y'*z; t = (r'*y)/s; x = x + t*y;
    for k = 1:numel(b)
        r = r - t*z; if norm(r) < tol
            return;
        end
        B = (r'*z)/s; y = -r + B*y; z = A*y; s = y'*z; t = (r'*y)/s; x = x + t*y;
    end
end
```

(b) Pseudocode

```
Conjugate Gradient Iteration for Positive Definite $S$

$x_0 = 0, r_0 = b, d_0 = r_0$

for $k = 1$ to $N$
    $\alpha_k = (r_{k-1}^T r_{k-1})/(d_{k-1}^T S d_{k-1})$ step length $x_{k-1}$ to $x_k$
    $x_k = x_{k-1} + \alpha_k d_{k-1}$ approximate solution
    $r_k = r_{k-1} - \alpha_k S d_{k-1}$ new residual $b - S x_k$
    $\beta_k = (r_k^T r_k)/(r_{k-1}^T r_{k-1})$ improvement this step
    $d_k = r_k + \beta_k d_{k-1}$ next search direction

Notice: only 1 matrix-vector multiplication $Sd$ in each step
```
Randomized Linear Algebra

Idea:
Make random vector $x$ have one element $x_k$ s.t. $Ax$ is a random sample of the column space of $A$. Let $S$ be a random sample matrix with columns composed of $x$ vectors.

Let $A$ and $B$ be arbitrary matrices

$C = AS$ and $R = S^T B$ and $CR = ASS^T B \approx AB$
How to keep optimize further?

We can use probability! Specifically mean and variance.

**S** is n by s sampling matrix, with one nonzero value per column. Let \( A = [a_1, a_2, a_3] \).

1. See that \( AS = [a_1, a_2, a_3] \begin{bmatrix} s_{11} & 0 \\ 0 & 0 \\ 0 & s_{32} \end{bmatrix} = [s_{11}a_1 \quad s_{32}a_3]. \)

Now must choose values of \( s_{kj} \), but how?

2. Assign probability \( p_j \) to all n columns of A, with \( p_1 + p_2 + \ldots + p_n = 1 \)

3. Choose S columns with replacement (can choose more than once)
4 If column k of A (row k of B) is chosen multiply both by $\frac{1}{\sqrt{sp_k}}$
5 $(\text{col } k \text{ of } A)(\text{row } k \text{ of } B)/sp_k$ goes into random product $AB$

So far just the mean is correct, can we use norms to get a correct variance?
Yes, but they are technical to derive! (So we won’t)

Mean: $E[X] = \frac{1}{s}AB$
Variance: $E[\|AB - CR\|_F^2] = \frac{1}{s}(C^2 - \|AB\|_F^2)$
Changes in $A^{-1}$ from changes in $A$

**Easing into it**

$$M = I - uv^T \rightarrow M^{-1} = I + \frac{uv^T}{1 - v^Tu}$$

Correction term is rank 1. If $v^Tu = 1$ then $M^{-1}$ does not exist.

**V is $n \times k$, U is also $n \times k$**

$$M = I_n - UV^T \rightarrow M^{-1} = I_n + U(I_k - V^TU)^{-1}V^T$$

**Sherman-Morrison-Woodbury Formula**

$$M^{-1} = (A - UV^T)^{-1} = A^{-1} + A^{-1}U(I - V^TA^{-1}U)^{-1}V^TA^{-1}$$
Derivatives of $A^{-1}$

Let $B = A + \Delta A$

\[
B^{-1} - A^{-1} = B^{-1}(A - B)A^{-1} \rightarrow \frac{\Delta A^{-1}}{\Delta t} = -(A + \Delta A)^{-1} \frac{\Delta A}{\Delta t} A^{-1}
\]

Approaches \[\frac{dA^{-1}}{dt} = -A^{-1} \frac{dA}{dt} A^{-1}\]

Applications

- The Kalman Filter, is the updating of dynamic least squares. Thus even when there is no new data the state vector $x$ changes with time. Think GPS and the time between new data and what is happening on the ground.
- Quasi-Newton Update Methods, updates the Jacobian Matrix in the classical Newton Method instead of recalculating.
Interlacing Eigenvalues

Fundamental Question: How does each $\lambda$ change as $A$ changes?

Interesting answer, as it is matters what time period we are observing change over. When we take the derivative, $\frac{d\lambda}{dt}$ we have minimal problems are the derivative is a linear operator. But what about $\lambda(A + \Delta A)$? Turns out this is a very difficult question to answer.

Let $S$ be changed to $S + uu^T$

We say $uu^T$ is positive semidefinite ($x^T M x \geq 0 \in \mathbb{R}^n$). This addition only increases the value of the eigenvalues, thus the change in the eigenvalues is $\lambda_1 \geq \lambda_2 \geq \ldots$ to $z_1 \geq z_2 \geq \ldots$. 
Interlacing Eigenvalues

Each $z_i$ of $S + uu^T$ is not smaller than $\lambda_i$ or greater than $\lambda_{i-1}$. This idea forces the relation $z_1 \geq \lambda_1 \geq z_2 \geq \lambda_2 \geq \ldots \geq z_n \geq \lambda_n$. 
Given a time dependent $A$

We start with formula $A(t)x(t) = \lambda(t)x(t)$ simply multiply by $y^T$ and use $y^T x = 1$. Yields $\lambda = y^T Ax$, this the derivative and cancelling terms yields $\frac{d\lambda}{dt} = y^T \frac{dA}{dt} x$.

When the change to $S$ is $\theta uu^T$ what happens?

Actually yields the Secular Equation,

$$\frac{1}{\theta} = u^T (zl - S)^{-1} u = \sum_{k=1}^{n} \frac{c_k^2}{z - \lambda_k}.$$  
$z$ are the $n$ eigenvalues of $S + \theta uu^T$. 
$z_k$ doesn’t go past $\lambda_{k-1}$
Compressed Sensing and Matrix Completion

Idea behind Compressed Sensing is taking a sparse signal and completing it to its full state from incompletely data.

For Compressed Sensing to function properly the basis, $V$, must use as few $v_n$ as possible. Further the signal must have another basis, $W$ to represent it. For Compressed Sensing there must be incoherence of $V$ and $W$, or entries of $V^T W$ are small. Common choice for $V$ and $W$ is $F$ (Fourier Matrix) and $I$, as the entries of $F$ have equal size.

Matrix Completion seeks to take an incomplete matrix $A_0$ and complete it to $A$ while keeping rank low as possible.

More formally seeks to minimize $\|A\|_N$ subject to $A = A_0$ in the known entries.
Matrix Completion

Say we know K entries in $n \times n$ matrix of rank $r$, can we perfectly recover the rest of the data?

While it may seem odd, yes we can. Perfect recovery of $A$ is highly probable! Must be careful as we can force failures.
Fourier Transforms

What is a Fourier Transform?

<table>
<thead>
<tr>
<th>Type</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real series</td>
<td>$f(x) = a_0 + a_1 \cos x + b_1 \sin x + a_2 \cos 2x + b_2 \sin 2x + \cdots$</td>
</tr>
<tr>
<td>Complex series</td>
<td>$f(x) = c_0 + c_1 e^{ix} + c_{-1} e^{-ix} + c_2 e^{2ix} + c_{-2} e^{-2ix} + \cdots$</td>
</tr>
<tr>
<td>Fourier integrals</td>
<td>$f(x) = \int_{-\infty}^{\infty} \hat{f}(k) e^{ikx} , dk$</td>
</tr>
<tr>
<td>Discrete series</td>
<td>$f = c_0 b_0 + c_1 b_1 + \cdots + c_{N-1} b_{N-1} = \text{Fourier matrix } F \times c$</td>
</tr>
</tbody>
</table>

The reason all of the coefficients have nice formula is due to orthogonality!
Fourier Transforms

Fourier Transform Matrix $F$ and Discrete Fourier Transform Matrix $\Omega$

\[
F_4 = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & i & i^2 & i^3 \\
1 & i^2 & i^4 & i^6 \\
1 & i^3 & i^6 & i^9
\end{bmatrix}
\]

\[
\Omega_4 = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -i & (-i)^2 & (-i)^3 \\
1 & (-i)^2 & (-i)^4 & (-i)^6 \\
1 & (-i)^3 & (-i)^6 & (-i)^9
\end{bmatrix}
\]

\[
F\Omega = NI
\]