Contents

1 Abstract 2

2 AI decleration 2

3 Relevant theory 2
- 3.1 Symmetric matrices . . . 2
- 3.2 Similar matrices . . . 2
- 3.3 Orthogonal matrices . . . 2
- 3.4 Eigenvalues and Eigenvectors . . . 2
- 3.5 The Spectral Theorem . . . 3
- 3.6 Singular Value Decomposition (SVD) . . . 3
- 3.7 Principle Component Analysis . . . 4
- 3.8 Phylogenetic PCA . . . 5
  - 3.8.1 Evolutionary covariance matrix . . . 5

4 Demonstrative Work 5

1 Abstract

2 AI decleration

3 Relevant theory

3.1 Symmetric matrices

A symmetric matrix is one whose transpose is equal to itself

$$S = S^T$$

Recall rules for matrix transpose:

$$(A^T)^T = A$$
$$(AB)^T = B^T A^T$$

Using these rules, we can prove that the product of any matrix with its transpose is symmetric:

$$(AA^T)^T = (A^T)^T A^T = AA^T$$
$$(A^T A)^T = A^T (A^T)^T = A^T A$$

3.2 Similar matrices

Matrices A and B are called "similar", if there exists an invertible matrix M, s.t.
$B = M^{-1}AM$. Similar matrices also share the same eigenvalues .

If at least one of A and B are invertible, then AB and BA are similar. When the invertible (full-rank) matrix is not square, we can choose a proof that only uses the left or right inverse product as the case may be.

$$B(AB)B^{-1} = (BA)(BB^{-1}) = BA$$
$$A^{-1}(AB)(A^{-1})^{-1} = A^{-1}(AB)A = (A^{-1}A)(BA) = BA$$

The remaining two cases are identical to the above with A and B swapped.

3.3 Orthogonal matrices

This term is slightly confusingly used to describe matrices with ortho*normal* columns. I will use the letter Q to symbolize such orthogonal matrices.

3.4 Eigenvalues and Eigenvectors

Given any square matrix: $A_{n\times n}$, any vector $x_{n\times 1}$, and any real number $\lambda \in \mathbb{R}$.
When the following equation is true, x is an eigenvector of A, and λ is its matching eigenvalue:

$$Ax = \lambda x$$

Solving this expression leads to a system of equations with an infinite solution space. However, it is convenient to select *normalized* eigenvectors:

$$\forall x_i \left(\|x_i\|^2 = 1\right)$$

3.5 The Spectral Theorem

The *Spectral Theorem* provides a useful decomposition for symmetric matrices:

$$S = Q\Lambda Q^T \tag{1}$$

$$S_{n\times n} = S^T$$

$$\Lambda_{n\times n} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

Q consists of S's normalized eigenvectors:

$$Q = \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}$$

$$Q^T = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix}$$

$$\forall q_i \, (Sq_i = \lambda_i)$$

$$\forall q_i \left(\|q_i\|^2 = 1\right)$$

3.6 Singular Value Decomposition (SVD)

Next, I will be discussing *Singular Value Decomposition.* It is similar to the Spectral decomposition discussed in the last section, but is applicable to *all* matrices, not just symmetrical ones.

The result we want is a decomposition of A into left/right orthonormal "singular vectors" U and V, connected by a diagonal matrix of "singular values" Σ:

$$A = U\Sigma V^T \tag{2}$$

$$\begin{aligned} A^T &= (U\Sigma V^T)^T \\ &= (\Sigma V^T)^T U^T \\ &= V\Sigma^T U^T \end{aligned}$$

We cannot perform a spectral decomposition directly on the non-symmetric matrix A, so instead we look at the spectral decompositions of A^TA and AA^T .

$$(A^TA) = V\Lambda_1 V^T$$

$$(AA^T) = U\Lambda_2 U^T$$

If A is full-rank and invertible, $A^T A$ and AA^T are similar and thus their nonzero eigenvalues are equal.

Note that there are faster ways of computing SVD than working with these products directly, this is just useful for understanding and proving SVD.

(proof not finished yet)

$$\begin{aligned}
A^T A &= U\Sigma V^T V \Sigma^T U^T \\
&= U\Sigma\Sigma^T U^T && V \text{ is orthonormal, so } V^T V = I \\
&= U\Sigma^2 U^T && \Sigma \text{ diagonal and therefore symmetric, so } \Sigma^T = \Sigma
\end{aligned}$$

.. similarly

$$AA^T = V\Sigma^T U^T U \Sigma V^T = V\Sigma\Sigma^T V^T = V\Sigma^2 V^T$$

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V_{n\times n}$$

We can ignore what is in the null space of $A^T A$ to reduce dimensions without losing any data.

$$A_{m\times n} = U_{m\times r}\Sigma_{r\times r}V_{r\times n}$$

$$\Sigma_{n\times n} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \end{bmatrix}$$

3.7 Principle Component Analysis

From the previous section on SVD, we know that any matrix can be expressed as a product of an orthogonal, a diagonal, and another orthogonal matrix:

$$A = U\Sigma V^T$$

maybe explain how rest of sigmas/singvals are zero leading to only R pieces
also not entirely clear myself on why $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_R$

This can be further expressed as a sum of R rank-1 pieces, R being the original matrix A's rank.

$$= \sigma_1 u_1 v_1^T + \ldots + \sigma_R u_R v_R^T$$

These pieces are individually know as *principal components*, and are, as the name suggests, key to PCA.

Picking any $k \leq R$, A_k is defined as the sum of the first k principal components:

$$A_k = \sigma_1 u_1 v_1^T + \ldots + \sigma_k u_k v_k^T$$

We claim that this matrix A_k is the closest possible approximation of A with rank k. *...maybe explain norms and def. of "approxomation", add proofs for Eckart-Young*

This claim is neatly defined in the *Eckart-Young Theorem*: *italics?*

$$\text{rank}(B) = k \implies \|A - B\| \geq \|A - A_k\| \tag{3}$$

3.8 Phylogenetic PCA

3.8.1 Evolutionary covariance matrix

$$\hat{r} = \frac{1}{1^T C^{-1} 1} \left(1^T C^{-1} X \right)$$

$$\hat{R} = \frac{1}{n-1} \left(X - \hat{r}^T \right)^T C^{-1} \left(X - \hat{r}^T \right)$$

4 Demonstrative Work

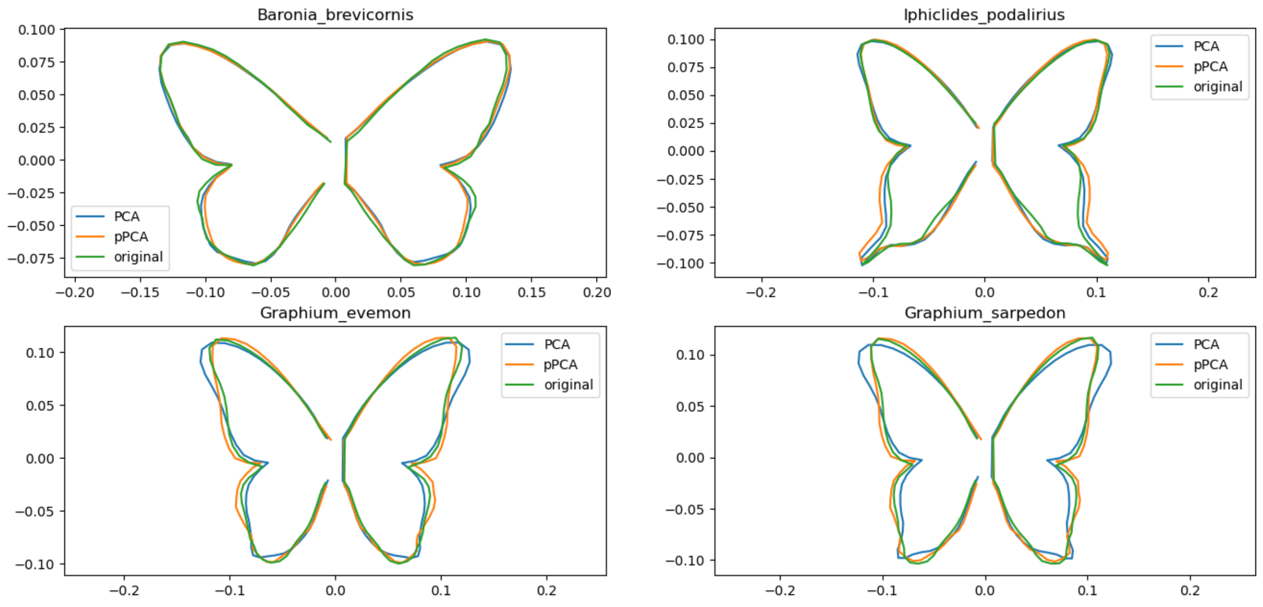

Figure 1: Sample of reconstructions with k=2

```
def design_matrix_slow(n, m):
    des = np.zeros((n * m, m))

    for i in range(n * m):
        for j in range(m):
            if (j - 1) * n < i >= j * n:
                des[i, j] = 1.

    return des
```

```
def design_matrix(n,m):
    des = np.zeros((n*m, m))
    i_indices = np.arange(n*m)
    j_indices = np.arange(m)
    # Use broadcasting to create a mask
    mask = (j_indices[:, None] * n <= i_indices) & (i_indices < (j_indice
    des[mask.T] = 1.0
    return des
```