Lecture 11: Algorithms for computing eigenvalues and eigenvectors and Principal Component Analysis

Francesco Preta
08/11/2020

Algorithms for computing eigenvalues and eigenvectors

We will discuss the following algorithms to compute eigenvalues and eigenvectors:

- **The power method**
- **The inverse power method**
- **The QR algorithm**

This is just a gentle introduction on the subject which requires much more depth. A more advance class on numerical analysis would treat these topics more thoroughly.

The power method

Purpose:

The power method algorithm finds the highest eigenvalue λ_1 of a square $n \times n$ matrix A, along with its eigenvector $\mathbf{v}_1$.

Assumptions:

- A has real eigenvalues
- A admits an eigenvalue λ_1 with algebraic and geometric multiplicity 1 which is, in absolute value, strictly greater than any other eigenvalue of A

Justification of the algorithm:

If A is diagonalizable, then for any $\mathbf{x} \in \mathbb{R}^n$ we can write

$$A^k\mathbf{x} = c_1\lambda_1^k\mathbf{v}_1 + c_2\lambda_2^k\mathbf{v}_2 + \ldots + c_n\lambda_n^k\mathbf{v}_n$$

where $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are the eigenvectors corresponding to the eigenvalues $\lambda_1, \ldots, \lambda_n$ and $c_1, \ldots, c_n$ are the coordinates of the original vector $\mathbf{x}$ in the basis of eigenvectors.

Justification of the algorithm (continued):

For large k, $A^k\mathbf{x} \sim c_1\lambda_1^k\mathbf{v}_1$, so that if we divide the expression by λ_1^k, we get

$$\frac{A^k\mathbf{x}}{\lambda_1^k} = c_1\mathbf{v}_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^k\mathbf{v}_2 + \ldots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\mathbf{v}_n$$

so that $\frac{A^k\mathbf{x}}{\lambda_1^k} \to c_1\mathbf{v}_1$

Power method algorithm:

1. Select an initial vector $\mathbf{x}_0$ whose largest coordinate is 1.
2. For $k = 0,1,...$:
   A. Compute $A\mathbf{x}_k$.
   B. Let μ_k be the entry of $A\mathbf{x}_k$ with largest absolute value.
   C. Let $\mathbf{x}_{k+1} = \frac{\mathbf{x}_k}{\mu_k}$.
3. If $\mathbf{x}_0$ is not an eigenvector for another eigenvalue, the sequence $\{\mu_k\}$approaches λ_1 and the sequence $\mathbf{x}_k$ approaches $\mathbf{v}_1$.

Example:

$$A = \begin{bmatrix} 6 & 5 \\ 1 & 2 \end{bmatrix}, \mathbf{x}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Applying the algorithm for the first three iterations gives:

Speed of convergence

The convergence to $\mathbf{v}_1$ depends on the ratio $\frac{\lambda_2}{\lambda_1}$ since that constitutes the main source of error between $\frac{A^k\mathbf{x}}{\lambda_1^k}$ and $c_1\mathbf{v}_1$.

This hints to the reason behind the requirement of λ_1 having multiplicity one, or otherwise the eigenvector produced by the algorithm would converge slowly and produce a choice strongly dependent on the algorithm itself.

Wrong initiation

If we choose an input vector $\mathbf{x}_0 = c_1\mathbf{v}_1 + \ldots + c_n\mathbf{v}_n$with $c_1 = 0$, then the algorithm would not converge to $\mathbf{v}_1$. However the probability of doing so by chance is very close to 0, so we can ignore such possibility.

The inverse power method

Purpose:

Finds the eigenvalues and the eigenvectors of a square matrix A, given an estimation for the eigenvalues.

Assumptions:

- A has all real eigenvalues with multiplicity one.
- All the eigenvalues of A are nonzero

Justification of the algorithm:

If A is an invertible matrix with eigenvalues $\lambda_1, \ldots, \lambda_n$, then A^{-1} has eigenvalues $\lambda_1^{-1}, \ldots, \lambda_n^{-1}$ with the same eigenvectors of A

Justification of the algorithm (continued):

If A has eigenvalues $\lambda_1, \ldots, \lambda_n$, for which we have an estimation $\alpha_1, \ldots, \alpha_n$ then the matrix $A - \alpha_i I$ will have an eigenvalue $\lambda_i - \alpha_i$ close to 0.

Justification of the algorithm (continued)

As long as α_i is not exactly equal to λ_i, it makes sense to consider the matrix $B = (A - \alpha_i I)^{-1}$, which will have a very large eigenvalue $(\lambda_i - \alpha_i)^{-1}$.

If the estimates $\alpha_1, \ldots, \alpha_n$ are close to the actual eigenvalues, so that $(\lambda_j - \alpha_i)^{-1}$ is significantly smaller than $(\lambda_i - \alpha_i)^{-1}$ for $j \neq i$, then the power method on B converges very fast.

The algorithm:

1. Given estimates $\alpha_1, \ldots, \alpha_n$ to eigenvalues $\lambda_1, \ldots, \lambda_n$, for $i = 1,...,n$:
   A. Choose $\mathbf{x}_0$ having coordinate with the largest absolute value equal to 1.
   B. For $k = 1,2,...$:
      - Solve for $\mathbf{y}_{k,i}$ the equation $\mathbf{x}_{k,i} = (A - \alpha_i)\mathbf{y}_{k,i}$.
      - Let $\mu_{k,i}$ be the entry of $\mathbf{y}_{k,i}$ with the largest possible absolute value.
      - Evaluate $\nu_{k,i} = \alpha_i + \frac{1}{\mu_{k,i}}$.
      - Compute $\mathbf{x}_{k+1,i} = \frac{\mathbf{y}_{k,i}}{\mu_{k,i}}$
   C. Then the sequence $\{\nu_{k,i}\}$ converges to λ_i and the sequence $\mathbf{x}_{k,i}$ converges to the eigenvector $\mathbf{v}_i$

Computational issues

$B_i = (A - \alpha_i I)^{-1}$ does not appear in the algorithm. This happens because finding the largest eigenvector of the inverse is more numerically unstable than solving the equation $\mathbf{x}_{k,i} = (A - \alpha_i I)\mathbf{y}_{k,i}$.

The QR algorithm

Basis for more complicated algorithms

The two previous algorithms, power method and inverse power method, are simple tools for practical applications. Built-in methods in numpy and MATLAB generally use more sophisticated algorithms. The basis of those is generally the QR algorithm.

Purpose:

Finding the eigenvalues of a square $n \times n$ matrix A. The eigenvectors have to be found with another method.

The algorithm:

1. Consider a square matrix A and let $A_0 = A$
2. For $k = 0,1,\ldots,$:
   A. Compute the QR decomposition $A_k = Q_k R_k$.
   B. Consider $A_{k+1} = R_k Q_k$.
3. The elements on the diagonal of A_k are the eigenvalues of A.

Similarity relations

Definition

Two matrices B and C are **similar** if there exists an invertible matrix Q such that $C = QBQ^{-1}$.

In general, every diagonalizable matrix is similar to the diagonal matrix Λ containing its eigenvalues. Viceversa, if two matrices are similar, they have the same eigenvalues.

Proof:

Justification of the algorithm

In this case, we have that $A_k = Q_k R_k$, so that $R_k = Q_k^{-1} A_k$. Then

$$A_{k+1} = R_k Q_k = Q_k^{-1} A_k Q_k$$

Therefore for each k, A_k and A_{k+1} are similar. That means that $A_0 = A$ and A_k are similar for each k and therefore they have the same eigenvalues.

Under some specific conditions, A_k converges to an upper triangular matrix, having eigenvalues on the principal diagonal.

Assumptions:

A is a matrix for which the QR algorithm converges (there is no easy closed condition for which that is guaranteed).

Example:

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Rate of convergence:

The rate of convergence to 0 of the element $A_{i,j}$ is given by $\frac{|\lambda_i|}{|\lambda_j|}$ so that if two eigenvalues are very close to each other, the corresponding element will converge very slowly.

Principal Component Analysis and Dimensionality Reduction

Matrix of observations

In what follows we will consider a $n \times m$ **matrix of observations** X. Each datapoint $\mathbf{x}_i$, for $i = 1,\ldots,n$ represents the i-th row vector of X having m features $\mathbf{x}_{i,j}$ for $j = 1,\ldots,m$.

Mean-deviation form

Definition

Let X be an observation matrix with rows $\{\mathbf{x}_i\}_{i=1}^n$. Then the **sample mean $\mathbf{m}$** is the vector

$$\mathbf{m} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$$

and the **mean-deviation form** of X is the matrix

$$\hat{X} = \begin{bmatrix} \hat{\mathbf{x}}_1 \\ \hat{\mathbf{x}}_2 \\ \cdots \\ \hat{\mathbf{x}}_n \end{bmatrix}$$

where $\hat{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{m}$ for $i = 1,\ldots,n$.

Example:

$$X = \begin{bmatrix} 1 & 2 & 1 \\ 4 & 2 & 13 \\ 7 & 8 & 1 \\ 8 & 4 & 5 \end{bmatrix}$$

Sample covariance matrix

Definition

Let X be an observation matrix and $\hat{X}$ its mean-deviation form. Then the **sample covariance matrix** S of X is

$$S = \frac{1}{n-1}\hat{X}^T\hat{X}$$

Meaning of covariance

The sample covariance matrix is a $m \times m$ matrix which measures the covariance between features. If two features are highly correlated, then removing one of them does not change significantly the information we have about our data. On the other hand, two features with 0 covariance are called **uncorrelated** and there is no way of knowing the value of one from the value of the other.

Example:

$$\hat{X} = \begin{bmatrix} -4 & -2 & -4 \\ -1 & -2 & 8 \\ 2 & 4 & -4 \\ 3 & 0 & 0 \end{bmatrix}$$

Variance and total variance

The coordinates on the diagonal represent the **variance** of each feature. The **total variance** is the sum of all such values, given by the trace $\mathrm{Tr}(S)$.

Principal Component Analysis

In principal component analysis, we plan to find a change of variable $\mathbf{y} = \mathbf{x}P$ for P a $m \times m$ matrix, with the property that the new features, obtained as linear combinations of the old ones, are uncorrelated and arranged in order of decreasing variance. In particular, we would like to find the matrix P to be orthogonal, so that $\mathbf{y}P^T = \mathbf{x}$. In this way, we obtain n observation vectors $\{\mathbf{y}_i\}_{i=1}^n$ with coordinates corresponding to the new features, and a new observation matrix Y.

Mean-deviation form for Y

If X is in mean-deviation form then Y is also in mean-deviation form:

Sample covariance for Y

Moreover, the corresponding sample covariance matrix S_y is

$$S_y = \frac{1}{n-1} Y^T Y = \frac{1}{n-1} P^T X^T X P = P^T S_x P$$

Principal components

Since we want to choose P such that S_y is a diagonal matrix with $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m \geq 0$, we have to diagonalize S_x, so that the i-th column of P correspond to the unit eigenvector $\mathbf{v}_i$ for the eigenvalue λ_i. Each $\mathbf{v}_i$ is called the i-th **principal component direction** of X and identifies a linear combination of the features which captures part of the variance, while being uncorrelated with the other principal components.

Example:

We can use a program to diagonalize the matrix S from the previous example. We obtain

$$\lambda_1 = 33.416 \qquad \mathbf{v}_1 = \begin{bmatrix} 0.059 \\ 0.229 \\ -0.972 \end{bmatrix}$$

$$\lambda_2 = 10 \qquad \mathbf{v}_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\lambda_3 = 6.584 \qquad \mathbf{v}_3 = \begin{bmatrix} 0.863 \\ -0.491 \\ -0.116 \end{bmatrix}$$

The coefficients of $\mathbf{v}_i$ express the linear combinations of the features that are uncorrelated to the other linear combinations. The eigenvalues represent their variance.

Dimensionality reduction

Principal component analysis is generally used to greatly reduce the dimensionality of data. In particular, by identifying a small number of principal components containing most of the variance (and therefore, most of the information about the data points), one can project the data points to the linear subspace spanned by those new features and reduce dimensions.

Example:

$$\lambda_1 = 33.416 \qquad \mathbf{v}_1 = \begin{bmatrix} 0.059 \\ 0.229 \\ -0.972 \end{bmatrix}$$

$$\lambda_2 = 10 \qquad \mathbf{v}_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\lambda_3 = 6.584 \qquad \mathbf{v}_3 = \begin{bmatrix} 0.863 \\ -0.491 \\ -0.116 \end{bmatrix}$$

Reducing the dimensions

We can project to a subspace spanned by $\mathbf{v}_1$ and $\mathbf{v}_2$ to retain most of the information about the original data.

In order to reduce the dimensionality and have a $n \times 2$ matrix (that is, n data points with two features), we can change basis $\hat{Y} = \hat{X}V$ and then consider only the first two coordinates. This can be done by multiplying $\hat{X}$ by the matrix $V_2 = [\mathbf{v}_1 \quad \mathbf{v}_2]$ and defining the new observation matrix as $\hat{Y}_2 = \hat{X}V_2$.

Dimensionality reduction and SVD

Since by the definition of S, $\mathbf{v}_1$ and $\mathbf{v}_2$ are the unitary eigenvectors associated with the largest singular values σ_1 and σ_2

$$\hat{Y}_2 = U\Sigma V^T V_2 = U\Sigma \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ \cdots & \\ 0 & 0 \end{bmatrix} = U \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \\ \cdots & \\ 0 & 0 \end{bmatrix}$$

In fact, this corresponds to first rewriting $\hat{X}$ in the coordinates given by the principal component directions and then truncating only to the first two coordinates.

Deciding the dimension

In the general case, in order to reduce the dimension from m to d, we have to implement a decision method for the value of d. This can be done either by deciding it a priori, or by choosing a threshold on the relative variance.

General dimensionality reduction

In general, in order to apply dimensionality reduction, we can consider the SVD of an observation matrix in mean-deviation form $\hat{X} = U\Sigma V^T$ with covariance matrix S. The matrix $\hat{Y} = \hat{X}V = U\Sigma$ corresponds to a change of basis for each single data point in the coordinates given by the principal component directions. Then we truncate up to the first d principal components.The resulting matrix $\hat{Y}_d$ is obtained by considering

$$\hat{Y}_d = \hat{X}V_d$$

where V_d is the $m \times d$ matrix having columns $\mathbf{v}_i$ for $i = 1,\ldots,d$.