Project 5: Analytical Methods

Owen O'Reilly | 11/17/2024 | AMS-595

Summary

This project was divided into four sections: a pagerank algorithm, principal component analysis, linear regression of four-dimensional data, and gradient descent of a given loss function.

Script 1: PageRank Algorithm

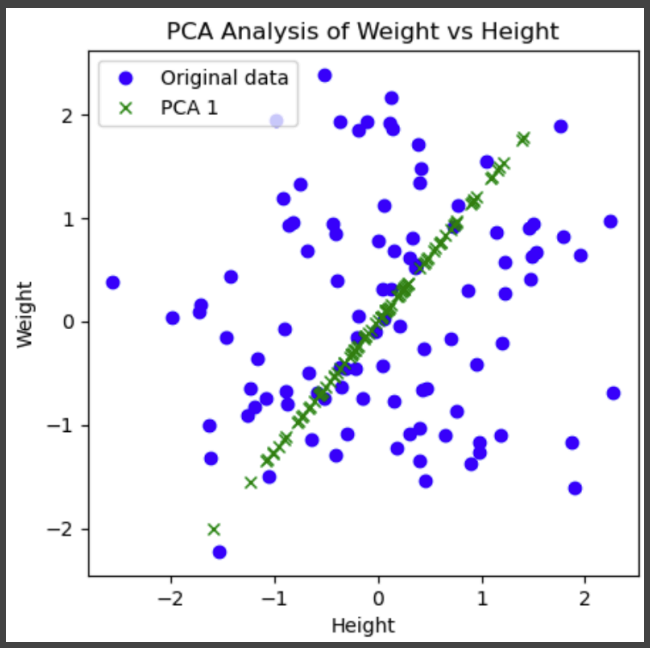
The initial matrix provided is constructed by calculating the portion of links on a given page p_i that link to another page p_j and assigning $M_{i,j}$ that value. Once this is done, the entries of the normalized major eigenvector of the matrix(corresponding to an eigenvalue of ~1) represent the normalized PageRank scores of each page p_i.

From the simple matrix provided, the best-scoring pages appear to be p_2 and p_3, with normalized PageRanks of 0.333 apiece. p_1 comes second with 0.2083, and p_0 last with 0.125.

Script 2: Principal Component Analysis for Dimension Reduction

PCA is an essential tool in dealing with higher-dimensional data, as it allows for elucidation of major trends that can be plotted in human-friendly dimensional spaces. Here, we are converting 2D data to 1D data, so the process is very lossy with regards to the reconstructed data, but at much higher dimensions this loss becomes more tolerable.

A standardized height/weight 100x2 dataset was provided. The script generates a 2x2 covariance matrix from the data corresponding to the two degrees of freedom in the dataset, and calculates the eigenvalues/vectors of this matrix. The principal components are the eigenvectors produced from this operation, and the magnitude of their effect is indicated by the corresponding eigenvalue magnitudes. A representation of the dataset from the perspective of one component was generated by element-wise multiplication of each dataset entry by the component's eigenvector. The original dataset (blue) was plotted alongside this dimension-reduced PCA representation (green, see below). The linear nature of the reconstructed data provides an excellent visual of the dimension-reducing action of PCA.



Script 3: Linear Regression via Least Squares

The system was set up as a matrix-based least squares problem, where a set of coefficients was to be generated based on some "training" housing data and corresponding prices.

The Scipy *linalg.lstsq* method was used to generate a least-squares solution for the weights, and this weight matrix was then used to attempt to predict the value of another house. From an intuition standpoint, this predicted value seems wildly off-base compared to the training set, which may indicate some non-linearities in the least-squares solution. However, weights returned by the *linalg.solve* provide a similar solution, so it is equally possible this is the best model that can be generated from the limited data. It is also worth noting that while the *solve* method performed equally well in this instance, it relies on the dimensionality of the training data being the same as the number of predictive variables, which is not tractable in many situations. In contrast, *lstsq* does not seem to require this, making it more effective in many applications.

Script 4: Gradient Descent for Minimizing Loss Function

This script uses the Scipy *optimize.minimize()* function and a defined mean squared error loss function to iteratively "correct" a random array towards a target value. Of note - the *minimize* function only accepts a 1D array, so the 100x50 array is passed in as a 5000-value array. Loss values are logged at each step and plotted once

the minimization completes (see below). Minimization tends to complete within 5 steps, and the loss improves logarithmically.