CISC 452 Assignment #3

Alex Globus

November 18, 2016

1 Principal Component Analysis Algorithm

An indispensable feature of data analysis is to pick out key features that are more consequential for classification than others. One technique is to transform the input space such that the important information is retained but with reduced dimensionality.

Principal Component Analysis (PCA) is such a technique. It attempts to find a lower dimensional subspace on to which it can project the data such that the sum of the squared projection errors is minimized. In other words, we wish to reduce the dimensionality of the data with as little error as possible. The algorithm finds an orthogonal set of transformation vectors in the input space that allows the data to be projected along these vectors. These vectors are the eigenvectors of the correlation matrix of the input vector. The projection corresponds to the eigenvalues.

For this assignment, we have a set of data with two dimensions, $x_i \in \mathbb{R}^2$. We would like to reduce this data to one dimension. To reduce the data from n dimensions to k dimensions, we must first compute the covariance matrix.

$$\sigma = \frac{1}{m} \sum_{i=1}^{n} (x_i)(x_i)^T$$

Then, we calculate the eigenvectors of covariance matrix. Observe that the result is an $n \times n$ matrix since our input matrix x_i is an $n \times 1$ matrix; its transpose is a matrix of size $1 \times n$. The columns of the covariance matrix are eigenvectors. The eigenvector with the largest eigenvalue is the direction along which the data set has the maximum variance. Taking n of the eigenvectors with the largest eigenvalues and combining them in a matrix gives us our transformation matrix W. To transform the input vector, we simply multiply it with the transformation matrix.

2 Neural Network Component

For n input dimensions that we wish to reduce to m dimensions, our network has n input nodes and m output nodes. There is no hidden layer/nodes. The network is a simple feed-forward network with one simple weight change rule.

Each output node y calculates its output according to equation 1. The weights are initialized randomly and are changed according to equation 2. The ℓ in the equation refers to the iteration. The assignment mandates that only one iteration be performed, so it is not necessary to keep track of it in our implementation.

$$y_j = \sum_{i=1}^{n} w_{ji} x_i \tag{1}$$

$$\Delta W_\ell = \eta_\ell y_\ell x_\ell^T - K_\ell W_\ell \tag{2}$$

$$K_\ell = y_\ell y_\ell^T$$

3 Code

We set our learning rate η to 0.1 and calculate the weight change delta using the following code.

3.1 Weight update algorithm

```
for (size_t j = 0; j < m; j++) { // m = 1 in this case
  for (size_t i = 0; i < set.rows(); i++ ) {
    Eigen::RowVector2d x = set.row(i);
    double y = weight.col(0).dot(x);
    Eigen::RowVector2d delta = LEARN_RATE * (y * x.transpose()) - (y*y*weight);
    weight += delta.transpose();
  }
}
```

3.2 Program Output

```
Learning Rate = 0.1
Statistically calculated PCA transform for 1 dimension:
-0.603864
-0.797088
```

```
NN Calculated PCA:
-0.213587
 0.233197
```

4 Results

We multiply the eigenvector to the mean normalized input to get the final output. After doing this, we can see that even with only one iteration our network manages to calculate an appropriate weight vector. I set the learning rate to one and ran the program again, this time with values that are much closer to the statistically calculated PCA transform weight.

```
Learning Rate = 1
Statistically calculated PCA transform for 1 dimension:
-0.603864
-0.797088
NN Calculated PCA:
-0.673328
 0.739347
```

I generated a wav file using the weight vector where the learning rate is one. You can hear the voice filtered out, which makes sense. The music has more variability in its frequency than the voice data, which is why its preserved.