Atividade 2: Principal Component Analysis

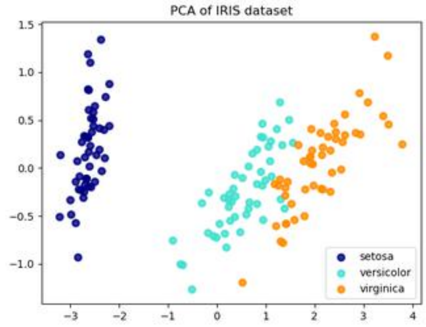
Análise de componentes principais, ou principal component analysis (PCA) é uma técnica utilizada para redução do espaço amostral, onde por meio de cáculos matriciais é possível reduzir o número de dimensões de um conjunto de dados, de forma a não perder suas características principais, representadas pelas componentes principais dado conjunto de entrada. Este trabalho apresenta um implementação do método em linguagem Java 1.8, para a disciplina de tópicos especiais em aprendizagem.

Introdução

Este artigo mostra como a aplicação que implementa o análise de componentes principais foi codificada. Para isto, uma breve explicação é feita sobre cada método desenvolvido, que compõe o exercício. O objetivo do trabalho é implementar um modelo simples que atenda os requisitos da disciplina.

Método

O método de análise dos componentes principais (do inglês Principal Component Analysis, ou PCA) busca transformar um conjunto de dados em um sistema com número menor dimensões, de forma a preservar as características dos dados originais. O método considera a variância das coordenadas de entrada, de forma a encontrar um plano de dimensão inferior em relação aos dados de entrada que represente a maior variância dos dados em cada uma das dimensões deste novo plano.



Após captar os dados, devemos transladar os dados subtraindo a média de cada coluna. Para isto, é calculado sobre as características, suas médias, subtraindo de cada amostra tal valor.
Após isto, calcula-se a matriz de covariância. Para obter a dimensão desta, é necessário saber o número de características do conjunto de dados. Então, a matriz de covariância representará as semelhanças entre as dimensões:

$$C = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix}$$

Com a covariância entre as dimensões dada por:

$$cov(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)}$$

Para encontrar as componentes principais é necessário encontrar os Autovalores da matriz de entrada, e para isso precisa-se encontrar os seus Autovetores. Para isto, a matriz de entrada deve ser quadrada, com isto, é necessário verificar se existe um vetor V, não nulo, que multiplicado pela matriz de entrada resulte em um vetor multiplo de V, da mesma forma que seria obtido pela multiplicação de um escalar λ real, diferente de zero pelo vetor V.

$$T\vec{v} = \lambda\vec{v}$$

Sabendo que existe tal vetor V quando a matriz de entrada é Possível e Indeterminada (SPI), que pode ser verificada no calculo de seu determinante, que resulta em zero, pode-se calcular os autovetores da seguinte forma:

$$\begin{bmatrix} a-\lambda & b \\ c & d-\lambda \end{bmatrix} \begin{bmatrix} v1 \\ v2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Além disso, sabe-se que o número máximo de autovalores de um sistema linear é igual a sua dimensão, pode-se calcular todos eles para cada conjunto de dados de entrada.

O autovetores correspondentes aos autovalores de uma matriz são suas componentes principais, para determinar qual deve ser utilizada, verificamos o correspondente maior autovalor, pois seu autovetor abrigará a maior variância do conjunto de dados da dimensão.

Por fim, deve-se fazer a multiplicação do vetor de características escolhidas transposto pela matriz original transposta, coforme visto a seguir:

$$dado\_final = caracteristicas\_escolhidas^T * dado\_original^T$$

Desenvolvimento

O método dos mínimos quadrados foi implementado na linguagem de programação Java em sua versão 8. Diferente métodos foram criados que em conjunto implementam o PCA.

Multiplicação de Matrizes

```
public static double[][] multiplicaMatriz(double[][] matriz_1,

double[][] matriz_2, int tamanho){

    double[][] r = new double[tamanho][tamanho];

    double v = 0;



    for (int i = 0; i < tamanho; i++) {
```

```
        // for (int j = 0; j < matriz_2[j].length; j++){
        for (int j = 0; j < tamanho; j++){
            for (int k = 0; k < tamanho; k++) {
                v = v + (matriz_1[i][k] * matriz_2[k][j]);
            }
            r[i][j] = v;
            v = 0;
        }
    }
    return r;
}
```

Calculo da Matriz Adjunta

```
public static  double[][] calculaAdjunta(double[][] matriz, int
tamanho) {
    double[][] adjunta = new double[tamanho][tamanho];
    if (tamanho == 2) {
        adjunta[0][0] = matriz[1][1];
        adjunta[0][1] = -matriz[1][0];
        adjunta[1][0] = -matriz[0][1];
        adjunta[1][1] = matriz[0][0];

    } else if (tamanho == 3) {
        double[][] temp = new double[2][2];

        for (int i = 0; i < tamanho; i++) {
            for (int j = 0; j < tamanho; j++) {
                temp[0][0] = matriz[(i + 1) % 3][(j + 1) % 3];
                temp[0][1] = matriz[(i + 1) % 3][(j + 2) % 3];
                temp[1][0] = matriz[(i + 2) % 3][(j + 1) % 3];
                temp[1][1] = matriz[(i + 2) % 3][(j + 2) % 3];

                adjunta[i][j] = calculaDeterminante(temp, 2);
            }
        }
    }
    return calculaTransposta(adjunta, tamanho);
}
```

Calculo da Matriz Inversa

```
public static double[][] calculaInversa(double[][] matriz, int
tamanho) {
    double[][] inversa = new double[tamanho][tamanho];
    double[][] adjunta = calculaAdjunta(matriz, tamanho);
```

```
    double determinante = calculaDeterminante(matriz,
tamanho);
    for (int i = 0; i < tamanho; i++){
        for (int j = 0; j < tamanho; j++){
            inversa[i][j] = (1/determinante) * adjunta[i][j];
        }
    }
    return inversa;
}
```

Calculo da Matriz Transposta

```
public static double[][] calculaTransposta(double[][] matriz, int
tamanho) {
    double[][] transp = new double[tamanho][tamanho];



    for (int i = 0; i < tamanho; i++) {
        for (int j = 0; j < tamanho; j++) {
            transp[j][i] = matriz[i][j];
        }
    }
```

```
    return transp;
}
```

Calculo do Determinante

```
public static double calculaDeterminante(double[][] matriz, int
tamanho) {
    double det = 0;

    if (tamanho == 2)
        det =   (matriz[0][0] * matriz[1][1]) - (matriz[0][1] *
matriz[1][0]);



    if (tamanho == 3) {
        for (int i = 0; i < tamanho; i++) {

            det = det + (matriz[0][i] * (matriz[1][(i + 1) % 3] *
matriz[2][(i + 2) % 3] - matriz[1][(i + 2) % 3] * matriz[2][(i + 1) %
3]));
        }
    }
```

```
    return det;

}
```

Resultados

Abaixo estão representados de forma matricial os resultados obtidos a partir dos dados de exemplo de entrada.

Height x Shoe size

- Matriz de Covariância

$$C = \begin{bmatrix} 10.3222 & 5.31111 \\ 5.31111 & 4.45556 \end{bmatrix}$$

- Autovetor

$$\lambda = \begin{bmatrix} 1 & 1.69468 \\ -1.69468 & 1 \end{bmatrix}$$

- Componetes Principais

$$\begin{bmatrix} 10.3222 & 5.31111 \\ 5.31111 & 4.45556 \end{bmatrix} \begin{bmatrix} 1 & 1.69468 \\ -1.69468 & 1 \end{bmatrix}$$

$$=$$

$$\begin{bmatrix} 1.32157 & 0 \\ 0 & 13.4562 \end{bmatrix} \begin{bmatrix} 1 & 1.69468 \\ -1.69468 & 1 \end{bmatrix}$$

Portanto,

$$\begin{bmatrix} 1.32157 & 22.804 \\ -2.23964 & 13.4562 \end{bmatrix}$$

Boiling point at the Alps

- Matriz de Covariância

$$C = \begin{bmatrix} 33.1739 & 17.3464 \\ 17.3464 & 9.12111 \end{bmatrix}$$

- Autovetor

$$\lambda = \begin{bmatrix} 1 & 1.91014 \\ -1.91014 & 1 \end{bmatrix}$$

- Componetes Principais

$$\begin{bmatrix} 33.1739 & 17.3464 \\ 17.3464 & 9.12111 \end{bmatrix} \begin{bmatrix} 1 & 1.91014 \\ -1.91014 & 1 \end{bmatrix}$$

$$=$$

$$\begin{bmatrix} 0.0398992 \\ 42.2551 \end{bmatrix} \begin{bmatrix} 1 & 1.91014 \\ -1.91014 & 1 \end{bmatrix}$$

Portanto,

$$\begin{bmatrix} 0.0398992 & 80.7131 \\ -0.076213 & 42.2551 \end{bmatrix}$$

Books x Grades

- Matriz de Covariância

$$C = \begin{bmatrix} 2.05128 & 2.71795 & 11.7692 \\ 2.71795 & 18.2974 & 34.4564 \\ 11.7692 & 34.4564 & 279.074 \end{bmatrix}$$

- Autovetor

$$\lambda = \begin{bmatrix} 0.0429869 & -33.3339 & -0.713884 \\ 0.130089 & 3.3279 & -7.45112 \\ 1 & 1 & 1 \end{bmatrix}$$

- Componetes Principais

$$\begin{bmatrix} 2.05128 & 2.71795 & 11.7692 \\ 2.71795 & 18.2974 & 34.4564 \\ 11.7692 & 34.4564 & 279.074 \end{bmatrix} \begin{bmatrix} 0.0429869 & -33.3339 & -0.713884 \\ 0.130089 & 3.3279 & -7.45112 \\ 1 & 1 & 1 \end{bmatrix}$$

$$=$$

$$\begin{bmatrix} 284.063 \\ 1.42686 \\ 13.9335 \end{bmatrix} \begin{bmatrix} 0.0429869 & -33.3339 & -0.713884 \\ 0.130089 & 3.3279 & -7.45112 \\ 1 & 1 & 1 \end{bmatrix}$$

Portanto,

$$\begin{bmatrix} 12.211 & -47.563 & -9.94692 \\ 36.9535 & 4.74846 & -103.82 \\ 284.063 & 1.42686 & 13.9335 \end{bmatrix}$$

US Census

- Matriz de Covariância

$$C = \begin{bmatrix} 1100 & 2227.83 \\ 2227.83 & 4599.6 \end{bmatrix}$$

- Autovetor

$$\lambda = \begin{bmatrix} 1 & 0.486145 \\ -0.486145 & 1 \end{bmatrix}$$

- Componetes Principais

$$\begin{bmatrix} 1100 & 2227.83 \\ 2227.83 & 4599.6 \end{bmatrix} \begin{bmatrix} 1 & 0.486145 \\ -0.486145 & 1 \end{bmatrix}$$

=

$$\begin{bmatrix} 16.9492 \\ 5682.65 \end{bmatrix} \begin{bmatrix} 1 & 0.486145 \\ -0.486145 & 1 \end{bmatrix}$$

Portanto,

$$\begin{bmatrix} 16.9492 & 2762.59 \\ -8.2398 & 5682.65 \end{bmatrix}$$

Conclusão

A partir do algoritmo implementado foi possível executar os exemplos indicados com sucesso, absorvendo os conceitos teóricos e as aplicações do PCA. Os dados utilizados na entrada de testes foram os seguintes:

- US Census
- Books x Grades
- Boiling point at the Alps
- Height x Shoe size

O algoritmo se mostrou eficiênte, onde sua propriedade de redução de dimensionalidade mostrou resultados satisfatórios principalmente no quesito exploração de dados.

Referências

1. AUTOVALORES E AUTOVETORES. Disponível em (https://www.respondeai.com.br/resumos/33/capitulos/1). Acesso em: 15 de out. de 2017

2. Implementing a Principal Component Analysis. Disponível em (http://sebastianraschka.com/Articles/2014_pca_step_by_step.html). Acesso em: 15 de out. de 2017

3. Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning:

4. Data Mining, Inference, and Prediction. Springer, 2001.