Atividade 3 **Yan A. S. Duarte**
PEL 208 Tópicos Especiais em Aprendizagem
Prof. Reinaldo A. C. Bianchi Entrega: 25/10/2017

Introdução

Esse relatório tem como objetivo detalhar a teoria, a implementação, os resultados e a conclusão da terceira atividade do curso. A propósta do exercício é implementar os métodos Linear Discriminant Analysis (LDA) e Most Discriminant Features (MDF).

Teoria

Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) é um método que procura a combinação linear de características que conseguem separar as amostras em grupos distintos. Para que isso aconteça, a técnica tenta maximizar a distância entre a classes e minimizar a distância intra-classes.

Para calcular a LDA de um conjunto de dados, utilizamos a equação:

$$(S_w^{-1} S_b)P = P\Lambda$$

onde:

- P são os autovetores;
- Λ são os autovalores;
- $(S_w^{-1} S_b)$ é a multiplicação da matriz de dispersão intra-classes invertida pela matriz de dispersão entre classes.

O calculo utilizado para encontrar as matrizes de dispersão são:

- Entre classes

$$S_b = \sum_{i=1}^{g} N_i(x_i - \bar{x})(x_i - \bar{x})^T$$

- Intra-classes

$$S_w = \sum_{i=1}^{g}(N_i - 1)S_i = \sum_{i=1}^{g}\sum_{j=1}^{N_i}(x_{i,j} - \bar{x_i})(x_{i,j} - \bar{x_i})^T$$

Onde:

- $\bar{x}$ é a média geral

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{g}\sum_{j=1}^{N_i} x_{i,j}$$

- $\bar{x}$ é a média da classe

$$\bar{x}_i = \frac{1}{N_i} \sum_{j=1}^{N_i} x_{i,j}$$

- $x_{i,j}$ é uma amostra da classe π_i
- N_i é o número de amostras da classe π_i
- g é o total de classes.

Most Discriminant Features

É a aplicação da técnica LDA no subconjunto de dados que mais discriminam o conjunto de dados obtido através da técnica PCA. Assim, chegamos a equação:

$$(P_{pca}^T S_w P_{pca})^{-1}(P_{pca}^T S_b P_{pca})P = P\Lambda$$

Implementação

Para a elaboração do exercício foi utilizada a linguagem de programação C++. Foram criadas funções separadas para calcular cada etapa do método que serão descritas a seguir.

scatter_between_class

Função para calcular a matriz de dispersão entre classes.

```
vector< vector<double> > Matrix::scatter_between_class(
    vector< vector< vector<double> > > vector_of_class,
    vector< vector<double> > class_mean,
    vector<double> sample_mean
){
    vector< vector <double> > Sb(
        vector_of_class[0][0].size(),
        vector<double>(vector_of_class[0][0].size()));

    for (int c=0; c<vector_of_class.size(); c++){
        int N = vector_of_class[c].size();

        Sb = this->matrices_sum(
            Sb,
            this->matrices_multiplication(
                N,
                this->matrices_multiplication(
                    this->transpose(
                        this->matrices_subtraction(
                            sample_mean, class_mean[c]
                        )
                    ),
                    this->transpose(
                        this->transpose(
                            this->matrices_subtraction(
                                sample_mean,
                                class_mean[c]
                            )
                        )
                    )
```

```
                )
            )
        );
    }

    return Sb;
}
```

scatter_within_class

Função para calcular a matriz de dispersão intra-classes.

```
vector< vector <double> > Matrix::scatter_within_class(
    vector< vector< vector<double> > > vector_of_class,
    vector< vector<double> > class_mean
){
    vector< vector <double> > Sw(
        vector_of_class[0][0].size(),
        vector<double>(vector_of_class[0][0].size())
    );

    for (int c=0; c<vector_of_class.size(); c++){
    for (int i=0; i<vector_of_class[0].size(); i++){
      Sw = this->matrices_sum(
          Sw,
          this->matrices_multiplication(
              this->transpose(
                  this->matrices_subtraction(
                      vector_of_class[c][i], class_mean[c]
                  )
              ),
              this->transpose(
                  this->transpose(
                      this->matrices_subtraction(
                          vector_of_class[c][i], class_mean[c]
                      )
                  )
              )
          )
      );
    }
}

return Sw;
};
```

mdf

Função para executar e exibir o método Most Discriminant Features.

```
void Matrix::mdf(
    vector< vector<double> > X,
    vector< vector<double> > y,
    bool exclude_first_col,
    int num_classes
){

    if (exclude_first_col==true)
    X = this->exclude_first_col(X);
```

```
vector< vector < vector <double > > > classes_x =
    this->get_separated_classes(X, y, num_classes);

vector <double> sample_mean = this->data_mean(X);

vector< vector <double> >
    class_mean = this-> get_mean_of_classes(classes_x);

vector< vector <double> >
    Sb=this->scatter_between_class(classes_x, class_mean, sample_mean);

vector< vector <double> >
    Sw=this->scatter_within_class(classes_x, class_mean);



int lin = X.size(),
col = X[0].size();

vector< vector<double> > new_X = X;

// Subtract the mean.
for (int j=0; j<col; j++)
for (int i=0; i<lin; i++)
new_X[i][j] -= sample_mean[j];

// Calculate the covariance matrix
vector< vector<double> > covariance =
    this->covariance(new_X, false, new_X[0]);

// Calculate the eigenvectors and eigenvalues of the covariance matrix.
vector <double> autovalores_pca = this->autovalores(covariance);
vector< vector <double> > autovetores_pca =
    this->autovetores(covariance, autovalores_pca);

cout<<"\nMedias\n";
for (int i = 0; i<sample_mean.size();i++){
cout<<sample_mean[i]<<"\t";
}
cout<<"\n\n\n";

this->show_matrix(class_mean, "Medias das classes");
cout<<"\n\n\n";

this->show_matrix(covariance, "Matrix de covariancia");

cout<<"\n\n\nAutovalores PCA\n";
for (int i = 0; i<autovalores_pca.size();i++){
cout<<autovalores_pca[i]<<"\t";
}
cout<<"\n\n\n";

this->show_matrix(autovetores_pca, "Autovetores PCA");
cout<<"\n\n\n";

this->show_matrix(this->matrices_multiplication(covariance,autovetores_pca),
      "Matrix de covariancia * Autovetores PCA");
cout<<"\n\n\n";
```

```
    vector< vector<double> > temp = autovetores_pca;

    for (int i = 0; i<temp.size(); i++){
    for (int j = 0; j<temp[0].size(); j++){
    temp[i][j] *= autovalores_pca[j];
    }
    }

    this->show_matrix(temp, "Autovalores * Autovetores PCA");
    cout<<"\n\n\n";

    vector <double> autovalores_lda =
        this->autovalores(this->matrices_multiplication(this->inversa(Sw),Sb));

    vector< vector <double> > autovetores_lda =
        this->autovetores(this->matrices_multiplication(this->inversa(Sw),Sb),
                          autovalores_lda);

    this->show_matrix(Sw, "Sw");
    cout<<"\n\n\n";

    this->show_matrix(this->inversa(Sw), "Sw^-1");
    cout<<"\n\n\n";

    this->show_matrix(Sb, "Sb");
    cout<<"\n\n\n";

    cout<<"Autovalores LDA\n";
    for (int i = 0; i<autovalores_lda.size();i++){
    cout<<autovalores_lda[i]<<"\t";
    }
    cout<<"\n\n\n";

    this->show_matrix(autovetores_lda, "Autovetores LDA");
    cout<<"\n\n\n";

    vector< vector <double> > temp_mdf;

    temp_mdf = this->matrices_multiplication(
    this->inversa(
        this->matrices_multiplication(
            this->matrices_multiplication(
                this->transpose(autovetores_pca),
                Sw),
            autovetores_pca)),
        this->matrices_multiplication(
            this->matrices_multiplication(
                this->transpose(autovetores_pca),Sb),
            autovetores_pca)
    );

    this->show_matrix(
    temp_mdf,
    "(P^T * Sw * P)^-1 * (P^T * Sb * P)");
    cout<<"\n\n\n";
};
```

Testes

Para testar o funcionamento das funções foi utilizado o dataset Iris somente as características *Sepal Width*, *Petal Length*, *Petal Width*. Foram utilizadas apenas essas três características devida a limitação que o sistema possui de realizar algumas funções apenas com matrizes quadradas de dimensão 3x3 e devida a necessidade de saber quantas classes a amostra tem.

Resultados

Iris DataSet (Sepal Width | Petal Length | Petal Width)

Médias

$$\mu = \begin{bmatrix} 3.054 \\ 3.75867 \\ 1.198867 \end{bmatrix}$$

Medias das classes

$$\begin{bmatrix} 3.418 & 1.464 & 0.244 \\ 2.77 & 4.26 & 1.326 \\ 2.974 & 5.552 & 2.026 \end{bmatrix}$$

Matriz de covariância PCA.

$$C = \begin{bmatrix} 0.188004 & -0.321713 & -0.117981 \\ -0.321713 & 3.11318 & 1.29639 \\ -0.117981 & 1.29639 & 0.582414 \end{bmatrix}$$

Autovalores PCA

$$x = \begin{bmatrix} 3.6928 \\ 0.0340654 \\ 0.15673 \end{bmatrix}$$

Autovetores PCA

$$\lambda = \begin{bmatrix} -0.251794 & -0.145175 & 6.18389 \\ 2.37636 & -0.436194 & 0.234419 \\ 1 & 1 & 1 \end{bmatrix}$$

Conferência

$$\begin{bmatrix} 0.188004 & -0.321713 & -0.117981 \\ -0.321713 & 3.11318 & 1.29639 \\ -0.117981 & 1.29639 & 0.582414 \end{bmatrix} \begin{bmatrix} -0.251794 & -0.145175 & 6.18389 \\ 2.37636 & -0.436194 & 0.234419 \\ 1 & 1 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 3.6928 \\ 0.0340654 \\ 0.15673 \end{bmatrix} \begin{bmatrix} -0.251794 & -0.145175 & 6.18389 \\ 2.37636 & -0.436194 & 0.234419 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\therefore \begin{bmatrix} -0.929824 & -0.00494545 & 0.969199 \\ 8.77542 & -0.0148591 & 0.0367405 \\ 3.6928 & 0.0340654 & 0.15673 \end{bmatrix} = \begin{bmatrix} -0.929824 & -0.00494545 & 0.969199 \\ 8.77542 & -0.0148591 & 0.0367405 \\ 3.6928 & 0.0340654 & 0.15673 \end{bmatrix}$$

Matriz de dispersão intra-classes Sw.

$$C = \begin{bmatrix} 17.035 & 8.12 & 4.9132 \\ 8.12 & 27.22 & 6.2536 \\ 4.9132 & 6.2536 & 6.1756 \end{bmatrix}$$

Matriz de dispersão intra-classes Sw invertida.

$$x = \begin{bmatrix} 0.0790879 & -0.0119072 & -0.0508633 \\ -0.0119072 & 0.0497007 & -0.0408552 \\ -0.0508633 & -0.0408552 & 0.243874 \end{bmatrix}$$

Matriz de dispersão entre classes Sb.

$$x = \begin{bmatrix} 10.9776 & -56.0552 & -22.4924 \\ -56.0552 & 436.644 & 186.908 \\ -22.4924 & 186.908 & 80.6041 \end{bmatrix}$$

Autovalores LDA

$$x = \begin{bmatrix} 30.29999 \\ -4.44089e-014 \\ 0.277724 \end{bmatrix}$$

Autovetores LDA

$$\lambda = \begin{bmatrix} -0.628115 & -0.397312 & 0.780876 \\ 0.483007 & -0.479062 & -0.325435 \\ 1 & 1 & 1 \end{bmatrix}$$

$(P_{pca}^T S_w P_{pca})^{-1}(P_{pca}^T S_b P_{pca})$.

$$C = \begin{bmatrix} 29.7474 & 0.105047 & -1.51019 \\ 63.0833 & 0.272534 & -1.8994 \\ -6.07643 & -0.0127263 & 0.537109 \end{bmatrix}$$

Conclusão

Nesse relatório foi apresentado como se faz o método Most Discriminant Features em um conjunto de dados.

Foi implementado, em linguagem c++, um algorítmo para executar esse método e, para testar o algorítimo, foi utilizado o dataset Iris com 3 características de cada amostra..

Referências

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2001.*

[2] Linear discriminant analysis - Wikipedia Disponível em (https://goo.gl/9RoQ85). Acesso em: 27 de out. de 2017

[3] *Linear discriminant analysis - University of Toronto, Faculty of Applied Science & Engineering Disponível em (https://goo.gl/UZUE4P). Acesso em: 27 de out. de 2017*