Aprendizado de Máquina - Relatório

Prática 2 - Classificadores

1 - Questão 01

Nessa questão foram criados 3 (Três) arquivos distintos, sendo dois deles os classificadores requeridos e o terceiro arquivo é o programa principal que faz uso dos classificadores e organiza os resultados obtidos de acordo com o que foi requerido. Abaixo descrevemos como se deu a construção de cada um dos arquivos.

Classificador de Distância Euclidiana Mínima (DEM)

O primeiro arquivo contém o classificador de distância Euclidiana Mínima, durante a construção do classificador foram separados 80% do conjunto de dados para treinamento e com os 20% restante foram efetuados testes. No processo de desenvolvimento foi necessário normalizar o conjunto de dados. Para a normalização dos dados é necessário calcular a média geral de cada atributo e seu desvio padrão, após fazer isso para cada atributo do conjunto de dados, este será o resultado da subtração dele com sua média, dividido pelo seu respectivo desvio padrão.

Depois dos dados normalizados foi efetuado um embaralhamento aleatório do conjunto de dados, deste modo é possível trabalhar com todas as classes disponíveis. Visto que a não realização desse embaralhamento pode ocasionar pegar dados de somente uma parte das classes disponíveis. A partir deste novo conjunto de dados, foi feito a separação das classes e o cálculo dos centróides para as mesmas.

Sendo assim, após o cálculo dos centróides foi iniciado a classificação dos atributos do conjunto de dados de teste. Esta classificação foi feita de acordo com a distância entre o atributo e o seu possível centróide, para calcular essa distância como esperado foi utilizada a distância Euclidiana mínima. Assim, um atributo qualquer do conjunto de teste pertence aquela uma determinada classe, se e somente se a distância entre eles for menor do que a distância entre ele e as demais classes.

Após classificar os dados de teste, de acordo com o conjunto de testes original foi feita a separação da suas classes, com intuito de sabermos exatamente a que classe cada atributo pertence e a partir disso poder criar a matriz de confusão do classificador. Para a criação dessa matriz foram feitas duas funções em especial, a primeira retorna a quantidade de atributos que foram classificados corretamente e a outra retorna a quantidade de atributos que pertencem a uma outra classe passada como parâmetro.

Uma matriz de confusão é uma tabela que mostra o resultado da classificação automática, comparando-o com o resultado real. Onde as colunas correspondem às classes reais e as linhas correspondem às classes preditas. Pois como existem mais duas classes, o formato correto de representação da matriz deve ser seguir esse padrão e não aquele utilizado para duas classes(Com a utilização de Falso Positivo e Falso Negativo).

Classificador de Distância de Mahalanobis (DMM)

Para a construção deste classificador foi utilizado o mesmo conceito do anterior, porém neste não foi utilizado a normalização dos dados, mas foi acrescentado o cálculo da matriz de covariância (Explica na Questão 2) que é necessária para a utilização da distância de Mahalanobis, a qual foi utilizada na hora de realizar classificação do conjunto de teste.

Programa Principal

Neste programa é realizado o cálculo de seus resultados após 50 rodadas de treinamento e teste para cada classificador. Depois de obter esses resultados é calculado suas taxas de acerto (i) média, (ii) mínima, (iii) máxima e (iv) taxas de acerto médias por classe. Para cada classificador são mostrados na tela os resultados junto com a matriz de confusão geral, matriz essa que é calculada com base nas matrizes retornadas pelos classificados a cada rodada de treinamento e teste.

O conjunto de dados utilizado para essa prática foi o "column_3C.dat"(Mesmo arquivo que o "coluna.dat", porém possui a especificação das classes). Abaixo é mostrado um exemplo do resultado da execução dos classificadores sobre o conjunto de dados. Onde a Figura 1, é o resultado da aplicação do conjunto de dados para o classificador DEM e a Figura 2 para o classificador DMM .

```
                              Classificador 01: Distância Euclidiana


Taxas de Acertos Geral:

Média: 0.077
Mínima: 0.0
Máxima: 0.2

Taxas de Acertos por Classe:

Hernia:
Média: 0.11
Mínima: 0.02
Máxima: 0.2

Spondylolisthesis:
Média: 0.0068
Mínima: 0.0
Máxima: 0.04

Normal:
Média: 0.1132
Mínima: 0.0
Máxima: 0.2
TESTE:  1.0

                                  Matriz de Confusão Geral: Classificador de Distância Euclidiana

+------------------+----------+--------------------+----------+
|                  |   Hernia |   Spondylolisthesis |   Normal |
+==================+==========+====================+==========+
| Hernia           |   0.0887 |             0.1839 |   0.1429 |
+------------------+----------+--------------------+----------+
| Spondylolisthesis |   0.0055 |             0.0713 |   0.01   |
+------------------+----------+--------------------+----------+
| Normal           |   0.0913 |             0.2342 |   0.1723 |
+------------------+----------+--------------------+----------+
```

Figura 1: Classificador de Distância Euclidiana Mínima

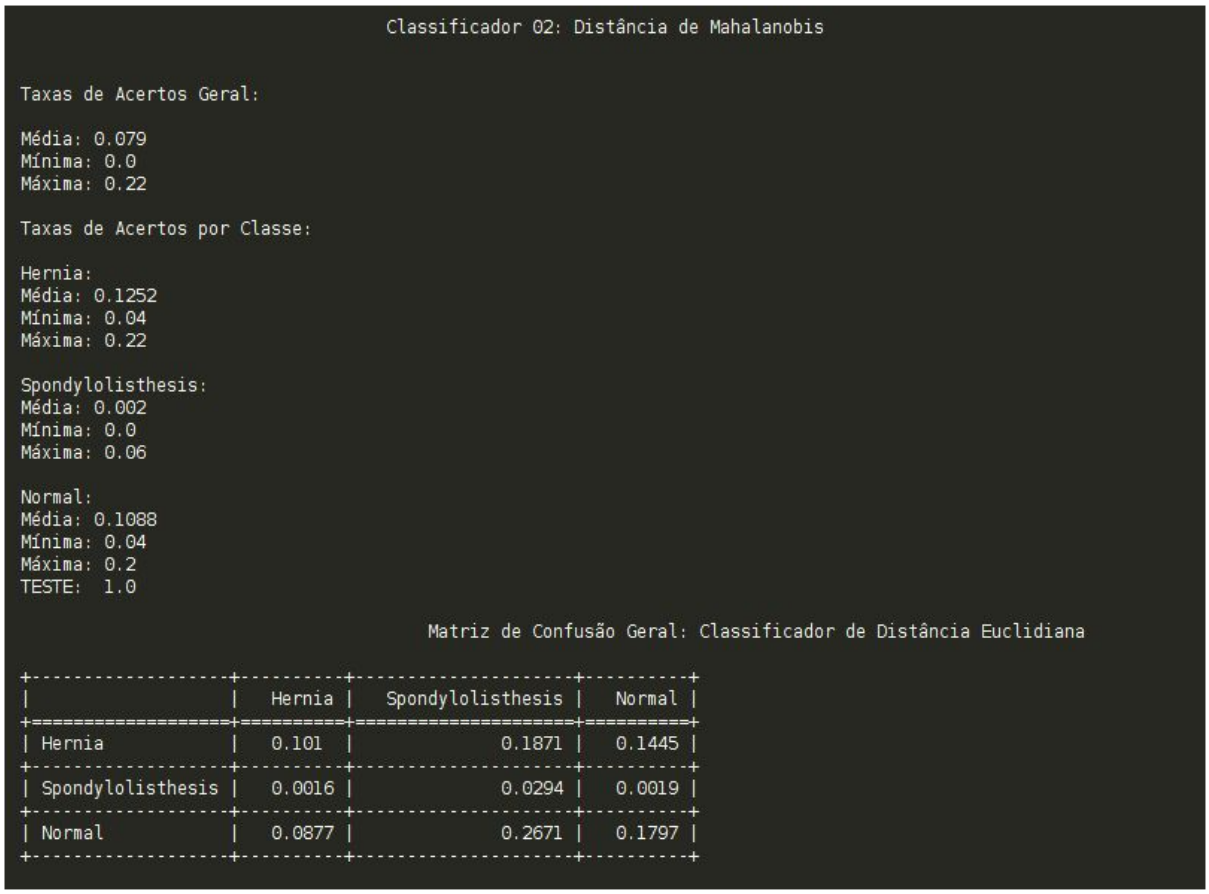

Figura 2: Classificador de Distância Mahalanobis Mínima

2 - Questão 02

No classificador de distância Euclidiana Mínima existe um conjunto de fatores que podem atrapalhar em seu desempenho sobre um conjunto de dados. Assim quando se utiliza este classificador pressupõe que não existe no conjunto de dados tais fatores. No entanto, somente após a classificação é que pode-se ter a certeza disso. Visto que, se o classificador é satisfatório, ou seja, possui um número elevado de acertos, não existem motivos para procurar um outro classificador mais complexos.

Contudo, quando o classificador não é satisfatório, ou seja, possui um número elevado de erros. Existem muitas razões para isso, dentre as mais comuns podem ser listadas as seguinte:

- As características podem ter sido inadequadamente escolhidas e, por isso não conseguem distinguir entre diversas classes.
- As características podem ser altamente correlacionadas.
- As superfícies de decisão devem ser não-lineares.
- Podem existir subclasses distintas embutidas nos dados.
- O espaço de padrões pode ser complexo demais.

Portanto, para a utilização de um classificador de distância Euclidiana mínima supõe-se inicialmente que o conjunto de dados a ser classificado não possua tais fatores, visto

que este tipo de classificador é limitado e não consegue tratar estas características. Logo, é recomendado utilizar um outro classificador mais complexo que consiga obter resultado satisfatório nestas condições.

No classificador de distância de Mahalanobis é utilizado uma forma de tornar esses fatores que atrapalham a classificação por meio da distância Euclidiana obsoletos, ou seja, essas condições não atrapalham sua classificação, visto que ele busca normalizar (O processo para a normalização das variáveis-características é diferente do anterior) a distância deixando insensível à escala das variáveis-características.

Para esta normalização ele utiliza o conceito de covariância, que basicamente é definida como o valor esperado do desvio de uma variável com o seu valor médio elevado ao quadrado. Assim a covariância de duas variáveis-características mede a tendência dessas variáveis covariar juntas, ou seja, de variar juntas. Também utiliza o conceito de matriz de covariância, a qual é uma matriz nxn (onde n é quantidade de atributos) e representa a co-variações de todas as variáveis para as demais. A matriz de covariação prover um modo de medir distâncias que é invariante a transformações lineares dos dados.

Contudo, o método usado pela distância de Mahalanobis permite que as unidades para cada uma das variáveis-características não afetarão os valores de distâncias resultantes, e por conseguinte não irá degradar a tarefa de classificação.

3 - Código Fonte

Classificador de Distância Euclidiana (DEM)

```
# -*- coding: utf-8 -*-
import numpy as np
import math

# Distância Euclediana
def euclidiana(valorTeste, centroide):
  distEucli = 0
  i = 0
  while i < len(valorTeste):
      distEucli += math.pow((np.abs(valorTeste[i] - centroide[i])), 2)
      i += 1

  return math.sqrt(distEucli)

# Quantidade de atributos classificadas corretamente
def positivo(classe, teste):
  cont = 0
  for valor in classe:
      if valor in teste:
          cont += 1
  return cont
```

```
def isOutra(classe1, classe2):
  cont = 0
  for valor in classe1:
      if (valor in classe2):
          cont += 1
  return cont

def classificadorDEM(data):

  # Normalizar os dados
  # Médias
  medias = []
  medias.append(np.mean([data[a][0] for a in range(len(data))], 0))
  medias.append(np.mean([data[a][1] for a in range(len(data))], 0))
  medias.append(np.mean([data[a][2] for a in range(len(data))], 0))
  medias.append(np.mean([data[a][3] for a in range(len(data))], 0))
  medias.append(np.mean([data[a][4] for a in range(len(data))], 0))
  medias.append(np.mean([data[a][5] for a in range(len(data))], 0))

  # Desvio Padrão
  desvioGeral = []
  desvioGeral.append(np.std([data[a][0] for a in range(len(data))], 0))
  desvioGeral.append(np.std([data[a][1] for a in range(len(data))], 0))
  desvioGeral.append(np.std([data[a][2] for a in range(len(data))], 0))
  desvioGeral.append(np.std([data[a][3] for a in range(len(data))], 0))
  desvioGeral.append(np.std([data[a][4] for a in range(len(data))], 0))
  desvioGeral.append(np.std([data[a][5] for a in range(len(data))], 0))

  # Normalizar, cada atributo recebe a diferença dele para a sua média sobre o desvio padrão
  for coluna in data:
      coluna[0] = np.abs((coluna[0] - medias[0]) / desvioGeral[0])
      coluna[1] = np.abs((coluna[1] - medias[1]) / desvioGeral[1])
      coluna[2] = np.abs((coluna[2] - medias[2]) / desvioGeral[2])
      coluna[3] = np.abs((coluna[3] - medias[3]) / desvioGeral[3])
      coluna[4] = np.abs((coluna[4] - medias[4]) / desvioGeral[4])
      coluna[5] = np.abs((coluna[5] - medias[5]) / desvioGeral[5])

  # Gerando posições aleatórias..
  posicoes = np.random.permutation(len(data))

  # Embaralhando os dados..
  data_alterado = np.zeros(data.shape)

  for i in xrange(0, len(posicoes)):
      data_alterado[i] = data[posicoes[i]]
```

```
# Separação entre treinamento e testes..
t_treinamento = int((0.8)*(len(data)))
t_testes = (len(data) - t_treinamento)

# Dados normalizados, encontrar classes
# Conjunto de treinamento
# treinamento = [data_alterado[i][0:6] for i in range(0, t_treinamento)]
treinamento = np.zeros((t_treinamento, 7))
for i in range(0, t_treinamento):
    linha = data_alterado[i].tolist()
    treinamento[i] = linha

classe1 = []  # Hernia
classe2 = []  # Spondylolisthesis
classe3 = []  # Normal
for linha in treinamento:
    if int(linha[-1]) == 1:
        classe1.append(linha[0:6])
    if int(linha[-1]) == 0:
        classe2.append(linha[0:6])
    if int(linha[-1]) == -1:
        classe3.append(linha[0:6])

# Calcular centroide das classes
centroide1 = np.mean(classe1, axis=0)
centroide2 = np.mean(classe2, axis=0)
centroide3 = np.mean(classe3, axis=0)

# Usar a distância euclidiana e classificar o conjunto de testes.
# Conjunto de teste.
teste = np.zeros((t_testes, 7))
cont = 0
for i in range(t_treinamento, (t_testes + t_treinamento)):
    linha = data_alterado[i].tolist()
    teste[cont] = linha  # [0:6]
    cont += 1

# Classificando
classe_1 = []
classe_2 = []
classe_3 = []
for linha in teste:
    d1 = euclidiana(linha[0:6], centroide1)
    d2 = euclidiana(linha[0:6], centroide2)
    d3 = euclidiana(linha[0:6], centroide3)
    if (d1 < d2) and (d1 < d3):
```

```
            classe_1.append(linha.tolist())
        if (d2 < d1) and (d2 < d3):
            classe_2.append(linha.tolist())
        if (d3 < d1) and (d3 < d2):
            classe_3.append(linha.tolist())  # Inserindo a linha completa, facilita o teste

    # Matriz de confusão
    t_c1 = []  # H
    t_c2 = []  # S
    t_c3 = []  # N
    # Colocando os atributos de teste em suas respectivas classes
    for linha in teste:
        if linha[-1] == 1:
            t_c1.append(linha.tolist())
        if linha[-1] == 0:
            t_c2.append(linha.tolist())
        if linha[-1] == -1:
            t_c3.append(linha.tolist())

    # Criando matriz de confusão
    matriz_cfs = []
    matriz_cfs.append([positivo(classe_1, t_c1), isOutra(classe_1, t_c2), isOutra(classe_1, t_c3)])
    matriz_cfs.append([isOutra(classe_2, t_c1), positivo(classe_2, t_c2), isOutra(classe_2, t_c3)])
    matriz_cfs.append([isOutra(classe_3, t_c1), isOutra(classe_3, t_c2), positivo(classe_3, t_c3)])
    return  matriz_cfs
```

Classificador de Distância de Mahalanobis(DMM)

```
# -*- coding: utf-8 -*-
import numpy as np

# Distância de Mahalanobis
def mahalanobis(valorTeste, centroide, matriz):
    normal = valorTeste
    transposto = np.transpose(valorTeste)
    v_normal = []
    v_transposto = []
    i = 0
    while i < len(valorTeste):
        v_normal.append(normal[i]-centroide[i])
        v_transposto.append(transposto[i]-centroide[i])
        i += 1

    intermediaria = np.dot(v_normal, matriz)
    d_Mahalanobis = np.dot(intermediaria, v_transposto)
    return d_Mahalanobis
```

```
# Quantidade de atributos classificadas corretamente
def positivo(classe, teste):
  cont = 0
  for valor in classe:
      if valor in teste:
          cont += 1
  return cont

def isOutra(classe1, classe2):
  cont = 0
  for valor in classe1:
      if (valor in classe2):
          cont += 1
  return cont


def classificadorDMM(data):
  # Gerando posições aleatórias..
  posicoes = np.random.permutation(len(data))

  # Embaralhando os dados..
  data_alterado = np.zeros(data.shape)

  for i in xrange(0, len(posicoes)):
      data_alterado[i] = data[posicoes[i]]

  # Separação entre treinamento e testes..
  t_treinamento = int((0.8) * (len(data)))
  t_testes = (len(data) - t_treinamento)

  # Conjunto de treinamento
  # treinamento = [data_alterado[i][0:6] for i in range(0, t_treinamento)]
  treinamento = np.zeros((t_treinamento, 7))
  for i in range(0, t_treinamento):
      linha = data_alterado[i].tolist()
      treinamento[i] = linha

  classe1 = []  # Hernia
  classe2 = []  # Spondylolisthesis
  classe3 = []  # Normal
  for linha in treinamento:
      if int(linha[-1]) == 1:
          classe1.append(linha[0:6])
      if int(linha[-1]) == 0:
          classe2.append(linha[0:6])
```

```
    if int(linha[-1]) == -1:
        classe3.append(linha[0:6])

# Calcular centroide das classes
centroide1 = np.mean(classe1, axis=0)
centroide2 = np.mean(classe2, axis=0)
centroide3 = np.mean(classe3, axis=0)

# Calculando a matriz de covariância
matriz = []
for linha in treinamento:
    matriz.append(linha[0:6])

matriz_covariancia = np.cov(matriz,None,0)

# Usar a distância mahalanobis e classificar o conjunto de testes.
# Conjunto de teste.
teste = np.zeros((t_testes, 7))
cont = 0
for i in range(t_treinamento, (t_testes + t_treinamento)):
    linha = data_alterado[i].tolist()
    teste[cont] = linha  # [0:6]
    cont += 1

# Classificando
classe_1 = []
classe_2 = []
classe_3 = []
for linha in teste:
    d1 = mahalanobis(linha[0:6], centroide1, matriz_covariancia)
    d2 = mahalanobis(linha[0:6], centroide2, matriz_covariancia)
    d3 = mahalanobis(linha[0:6], centroide3, matriz_covariancia)
    if (d1 < d2) and (d1 < d3):
        classe_1.append(linha.tolist())
    if (d2 < d1) and (d2 < d3):
        classe_2.append(linha.tolist())
    if (d3 < d1) and (d3 < d2):
        classe_3.append(linha.tolist())

# Matriz de confusão
t_c1 = []  # H
t_c2 = []  # S
t_c3 = []  # N
# Colocando os atributos de teste em suas respectivas classes
for linha in teste:
    if linha[-1] == 1:
```

```
            t_c1.append(linha.tolist())
        if linha[-1] == 0:
            t_c2.append(linha.tolist())
        if linha[-1] == -1:
            t_c3.append(linha.tolist())

    # Criando matriz de confusão
    matriz_cfs = []
    matriz_cfs.append([positivo(classe_1, t_c1), isOutra(classe_1, t_c2), isOutra(classe_1, t_c3)])
    matriz_cfs.append([isOutra(classe_2, t_c1), positivo(classe_2, t_c2), isOutra(classe_2, t_c3)])
    matriz_cfs.append([isOutra(classe_3, t_c1), isOutra(classe_3, t_c2), positivo(classe_3, t_c3)])
    return  matriz_cfs
```

Programa Principal

```
#-*- coding: utf-8 -*-
import numpy as np
import classificador_DEM as DEM
import classificador_DMM as DMM
import tabulate as tabela

def gerarRelatorio(data, tipo):
    # Quantidade de iterações
    quantidade = 50

    dem_geral = []
    dem_hernia = []
    dem_spondy = []
    dem_normal = []
    matriz_confusao = np.zeros((3, 3))  # Quantidade de classes
    matriz_confusao = matriz_confusao.tolist()

    j = 0
    while j < quantidade:
        # Resultado classifição
        if tipo == 1:
            matriz = DEM.classificadorDEM(data)
        else:
            matriz = DMM.classificadorDMM(data)
```

```
    # Distribuindo resultados..
    i = 0
    while i < len(matriz):
        if i == 0:
            dem_hernia.append(matriz[i][0]/float(quantidade))
            matriz_confusao[i][0] = matriz_confusao[i][0] + matriz[i][0]
            matriz_confusao[i][1] = matriz_confusao[i][1] + matriz[i][1]
            matriz_confusao[i][2] = matriz_confusao[i][2] + matriz[i][2]
        if i == 1:
            dem_spondy.append(matriz[i][0]/float(quantidade))
            matriz_confusao[i][0] = matriz_confusao[i][0] + matriz[i][0]
            matriz_confusao[i][1] = matriz_confusao[i][1] + matriz[i][1]
            matriz_confusao[i][2] = matriz_confusao[i][2] + matriz[i][2]
        if i == 2:
            dem_normal.append(matriz[i][0]/float(quantidade))
            matriz_confusao[i][0] = matriz_confusao[i][0] + matriz[i][0]
            matriz_confusao[i][1] = matriz_confusao[i][1] + matriz[i][1]
            matriz_confusao[i][2] = matriz_confusao[i][2] + matriz[i][2]
        dem_geral.append(matriz[i][0]/float(quantidade))
        i += 1
    j += 1

  print "\nTaxas de Acertos Geral: \n"
    print "Média: " + str(round(np.mean(dem_geral), 3)) + "\nMínima: " + str(min(dem_geral)) +
"\nMáxima: " + str(
    max(dem_geral))
  print "\nTaxas de Acertos por Classe: \n"
   print "Hernia: \nMédia: " + str(np.mean(dem_hernia)) + "\nMínima: " + str(min(dem_hernia)) +
"\nMáxima: " + str(
    max(dem_hernia))
  print "\nSpondylolisthesis: \nMédia: " + str(np.mean(dem_spondy)) + "\nMínima: " + str(
    min(dem_spondy)) + "\nMáxima: " + str(max(dem_spondy))
   print "\nNormal: \nMédia: " + str(np.mean(dem_normal)) + "\nMínima: " + str(min(dem_normal))
+ "\nMáxima: " + str(
    max(dem_normal))

  # Separação entre treinamento e testes..
  t_treinamento = int((0.8) * (len(data)))
  t_testes = (len(data) - t_treinamento)
  i = 0
  while i < len(matriz_confusao):
    j = 0
    while j < len(matriz_confusao):
        matriz_confusao[i][j] = round(matriz_confusao[i][j] / float(t_testes * 50), 4)
        j += 1
    i += 1
```

```
  matriz_cfs = [["Hernia"] + matriz_confusao[0], ["Spondylolisthesis"] + matriz_confusao[1],
                ["Normal"] + matriz_confusao[2]]

  print "\n                                Matriz de Confusão Geral: Classificador de Distância Euclidiana
\n"
  print(tabela.tabulate(matriz_cfs, ["Hernia", "Spondylolisthesis", "Normal"], tablefmt="grid"))

# Lendo o arquivo
# Mesmo conjunto de dados, porém este especifica as classes
data = np.genfromtxt("column_3C.dat", delimiter=",")

print "\n                                Classificador 01: Distância Euclidiana                \n"
gerarRelatorio(data, 1)

print "\n                                Classificador 02: Distância de Mahalanobis            \n"
gerarRelatorio(data, 2)
```