Aprendizado de Máquina - Relatório

Prática 03 - Seleção de Atributos

1 - Seleção de Atributos

A seleção de atributos é uma etapa da fase de pré-processamento do conjunto de dados que será utilizado para o estudo, com o objetivo de encontrar um subconjunto de atributos que seja capaz de representar sem perda de informação o conjunto de dados original e assim conseguir reduzir a dimensão desse conjunto de dados, ou seja, diminuir a quantidade de atributos.

Assim ao efetuar uma seleção de atributos é possível evitar possíveis ruídos no resultado final de seu estudo, visto que com a retirada de atributos desnecessários (causadores do ruído) a chance disso ocorrer é reduzida consideravelmente. Tomar estas decisões com auxílio dos computadores é extremamente desejável, isto faz com que se impulsione o crescimento desta área ainda jovem cada vez mais, com o descobrimento de algoritmos novos e aprimoramento dos já usuais.

Algumas das técnicas mais conhecidas para a seleção de atributos são a Sequential Forward (Busca para frente) e a Sequential Backward (Busca para trás ou Eliminação Backward), nas seções seguintes é dado uma definição sobre a técnica e uma descrição de seu funcionamento.

2 - Seleção de Atributos Forward

Na seleção de atributos Forward a busca é iniciada sem atributos e os mesmos são adicionados um a um, cada atributo é adicionado isoladamente e o conjunto resultante é avaliado segundo um critério, o qual neste caso é a média geral das 50 (Cinquenta) iterações do classificador de distância Euclidiana Mínima para aquele atributo. Deste modo, o atributo que produz o melhor critério (melhor média) é incorporado ao conjunto de atributos.

Abaixo é mostrado um pseudocódigo para a seleção Forward:

Inicia com um conjunto vazio e adiciona sequencialmente o atributo que maximiza a média de classificação correta do classificador quando combinado com o conjunto de atributos que já foram selecionados.

1 - Inicie com um conjunto vazio.
2 - Selecione o próximo melhor atributo de modo que maximize a média geral de classificação.
3 - Adicione o atributo selecionado no passo 2 ao conjunto de atributos existentes.
4 - Volte ao passo 2.

1. Start with the empty set $Y_0 = \{\emptyset\}$
2. Select the next best feature $x^+ = \arg\max_{x \notin Y_k} J(Y_k + x)$
3. Update $Y_{k+1} = Y_k + x^+;\ k = k + 1$
4. Go to 2

Figura 1: Pseudocódigo para a Seleção Forward

Sendo assim, a implementação desta técnica deu-se da seguinte forma: Inicialmente foi selecionado o atributo com maior média de classificação do conjunto de dados. Após obter este atributo o mesmo foi adicionado ao conjunto solução e o processo foi repetido para o conjunto de dados restante até que a média desse conjunto solução combinado com este atributo não seja menor que do conjunto solução sem ele, ou seja, foi repetido para todos os atributos que restaram no conjuntos de dados que são diferentes do atributo escolhido, tal que a média de classificação deste novo conjunto solução seja melhor do que a do anterior.

Um detalhe observado sobre essa técnica é que ela depende também do conjunto do conjunto de dados, visto que se tiver um atributo que se destaca ele será escolhido como o único representante do conjunto de dados. Pois, isso ocorreu com o conjunto de dados utilizado.

3 - Eliminação Backward

A eliminação Backward é similar a seleção de atributos Forward, porém inicia com todo o conjunto de atributos, e vai eliminando um atributo a cada passo até que a média de classificação do conjunto de dados do próximo não seja pior que a do atual. Ou seja, a cada passo é solucionado um atributo tal que a média dele seja a menor entre os demais e então este atributo é removido do conjunto de dados, visto que sua remoção não causa perda de informação e assim não deixa o conjunto de dados menos representativo.

Abaixo é apresentado um pseudocódigo para a eliminação Backward.

1. Start with the full set $Y_0 = X$
2. Remove the worst feature $x^- = \arg\max_{x \in Y_k} J(Y_k - x)$
3. Update $Y_{k+1} = Y_k - x^-;\ k = k + 1$
4. Go to 2

Figura 2: Pseudocódigo Eliminação Backward

A implementação da técnica foi feita da seguinte forma, considerando todo o atributo de dados foi selecionado o atributo que possui menor média, ou seja, aquele ao qual sua remoção não interferi o conjunto de dados e então é o processo é repetido para o conjunto de dados resultante dessa remoção enquanto a média do conjunto resultante da remoção não for pior que o atual.

4 - Código Fonte

Seleção de atributos Forward:

```
#-*- coding: utf-8 -*-
import numpy as np
import classificador_q3 as DEM

# Efetua a soma de todos que ele classificou corretamente
def somaPrincipal(matriz):
  n = len(matriz)
  soma = 0
  i = 0
  while i < n:
     j = 0
     while j < n:
        if i == j:
           soma += matriz[i][j]
        j += 1
     i += 1
  return soma

# Auxiliar para o cálculo das médias
def gerarMatriz(base, atributo, dados):
  matriz = base.tolist()
  elementos = [dados[i][int(atributo)] for i in range(0, len(dados))]
  i = 0
  while i < len(elementos):
     matriz[i].append(elementos[i])
     i += 1
  return np.array(matriz)

# Seleciona inicialmente o melhor atributo do conjunto de dados
def selecionaAtributo(data,dimensoes):
  quantidade = 50
  dic_medias = {}
  dados = []
  # Inicializando dicionario de médias..
  k = 1
```

```
  while k < dimensoes[1]:
      dic_medias[k] = []
      k += 1
  # Classificando os dados de acordo com os atributos.
  i = 0
  while i < quantidade:
      k = 1
      while k < dimensoes[1]:
          classes = [data[j][0] for j in range(0, len(data))]
          dados = [[classes[j], data[j][k]] for j in range(0, len(data))]
          matriz = DEM.classificadorDEM(np.array(dados).reshape((len(dados), len(dados[0]))))
          dic_medias[k].append(somaPrincipal(matriz))
          k += 1
      i += 1
  # Calculando médias dos atributos de acordo com a classificação
  medias = []
  k = 1
  while k < dimensoes[1]:
      medias.append((k, np.mean(dic_medias[k], 0), np.std(dic_medias[k], 0)))
      k += 1
  medias_ordenadas = np.sort(medias, axis=0).tolist()
  return np.array(dados).reshape((len(dados), 2)), medias_ordenadas[-1]


# Calcula médias de cada subconjunto
def getMedias(dados):
  quantidade = 50
  medias = []
  # Classificando os dados de acordo com os atributos.
  i = 0
  while i < quantidade:
      matriz = DEM.classificadorDEM(np.array(dados))
      medias.append(somaPrincipal(matriz))
      i += 1
  # Calculando médias dos atributos de acordo com a classificação
  atributo = [np.mean(medias, 0), np.std(medias, 0)] # dados[0] é o atributo atual
  return atributo

#Inicia com nenhum atributo e inclui um atributo por vez no conjunto
def forward_selection(data):
  # Criando arquivos com os atributos
  dimensoes = data.shape
  # Melhor atributo inicial
  melhor_atributo, media = selecionaAtributo(data,dimensoes)
  # Atributo selecionado anteriormente
  selecionados = [int(media[0])]
```

```
  # quantidade de atributos após a escolha do primeiro
  i = 2
  while i < dimensoes[1]: # para cada atributo, efetuar a escolha do melhor daquele conjunto
      media_atual = media[1]
      medias = []
      j = 1
      while j < dimensoes[1]:
          if not j in selecionados:
              nv_matriz = gerarMatriz(melhor_atributo,j,data)
              media_proximo, desvio = getMedias(nv_matriz)
              if media_proximo > media_atual:
                  selecionados.append(j)
                  media_atual = media_proximo
                  medias.append((j, media_proximo))
          j += 1
      # adicionando
      if medias != []:
          medias = np.sort(medias, axis=-1)
          melhor_atributo = gerarMatriz(melhor_atributo, medias[-1][0], data)
      i += 1
  return melhor_atributo
```

Eliminação Backward

```
#-*- coding: utf-8 -*-
import numpy as np
import classificador_q3 as DEM

# Efetua a soma de todos que ele classificou corretamente
def somaPrincipal(matriz):
  n = len(matriz)
  soma = 0
  i = 0
  while i < n:
      j = 0
      while j < n:
          if i == j:
              soma += matriz[i][j]
          j += 1
      i += 1
  return soma

# Calcula médias de cada subconjunto
def getMedias(dados):
  quantidade = 50
  medias = []
```

```
    # Classificando os dados de acordo com os atributos.
    i = 0
    while i < quantidade:
        matriz = DEM.classificadorDEM(np.array(dados))
        medias.append(somaPrincipal(matriz))
        i += 1
    # Calculando médias dos atributos de acordo com a classificação
    atributo = [np.mean(medias, 0), np.std(medias, 0)] # dados[0] é o atributo atual
    return atributo

# Seleciona inicialmente o melhor atributo do conjunto de dados
def selecionaAtributo(data,dimensoes):
    quantidade = 50
    dic_medias = {}
    dados = []
    # Inicializando dicionario de médias..
    k = 1
    while k < dimensoes[1]:
        dic_medias[k] = []
        k += 1
    # Classificando os dados de acordo com os atributos.
    i = 0
    while i < quantidade:
        k = 1
        while k < dimensoes[1]:
            classes = [data[j][0] for j in range(0, len(data))]
            dados = [[classes[j], data[j][k]] for j in range(0, len(data))]
            matriz = DEM.classificadorDEM(np.array(dados).reshape((len(dados), len(dados[0]))))
            dic_medias[k].append(somaPrincipal(matriz))
            k += 1
        i += 1
    # Calculando médias dos atributos de acordo com a classificação
    medias = []
    k = 1
    while k < dimensoes[1]:
        medias.append((k, np.mean(dic_medias[k], 0), np.std(dic_medias[k], 0)))
        k += 1
    medias_ordenadas = np.sort(medias, axis=0).tolist()
    # Reduzindo o conjunto de dados
    atributo = medias_ordenadas[0]
    return np.delete(data, atributo[0], 1)

#Começa com todo o conjunto de atributos, eliminando um atributo a cada passo
def backward(data):
    # Melhor atributo para ser removido incialmente
    melhor_atributo = selecionaAtributo(data, data.shape)
```

```
  media_anterior = getMedias(melhor_atributo)
  proximo = selecionaAtributo(melhor_atributo, melhor_atributo.shape)
  media_proximo = getMedias(proximo)
  if media_proximo > media_anterior:
      return backward(proximo)
  else:
      return melhor_atributo
```

Classificador Utilizado

```
# -*- 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):

  #data = np.genfromtxt(nome_arquivo, delimiter=",")

  # Normalizar os dados
  # Médias
  dim = data.shape # dimensões
```

```
mediaGeral = []
i = 0
while i < dim[1]:
    mediaGeral.append(np.mean([data[a][i] for a in range(len(data))], 0))
    i += 1

# Desvio Padrão
desvioGeral = []
j = 0
while j < dim[1]:
    desvioGeral.append(np.std([data[a][j] for a in range(len(data))], 0))
    j += 1

# Normalizar, cada atributo recebe a diferença dele para a sua média sobre o desvio padrão
for coluna in data:
    k = 1
    while k < dim[1]:
        j = 0
        while j < len(mediaGeral):
            coluna[k] = np.abs((coluna[k] - mediaGeral[j]) / desvioGeral[j])
            j += 1
        k += 1

# 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, dim[1]))
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[0]) == 1:
        classe1.append(linha[1:])
    if int(linha[0]) == 2:
        classe2.append(linha[1:])
    if int(linha[0]) == 3:
        classe3.append(linha[1:])

# 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, dim[1]))
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[1:], centroide1)
    d2 = euclidiana(linha[1:], centroide2)
    d3 = euclidiana(linha[1:], 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[0] == 1:
        t_c1.append(linha.tolist())
```

```
    if linha[0] == 2:
        t_c2.append(linha.tolist())
    if linha[0] == 3:
        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 forward as F
import backward as B

# Lendo arquivo
data = np.genfromtxt("wine.data", delimiter=",")

print "Sequential Forward\n"
resultado_forward = F.forward_selection(data)

# Padronizando os valores para facilitar a visualização.
for linha in resultado_forward:
  linha = [round(linha[i],3) for i in range(0, len(linha))]
  print linha

print "\nSequential Backward\n"
resultado_backward = B.backward(data)

for linha in resultado_backward:
  linha = [round(linha[i],3) for i in range(0, len(linha))]
  print linha
```

5 - Conclusão

Contudo, o algoritmo principal foi executado diversas vezes com o objetivo de avaliar o comportamento de ambas as técnicas e foi possível perceber que ambas dependem diretamente do classificador e que o conjunto de atributos dado como resultado final varia entre elas, porém em alguns casos o resultado final foi o mesmo. A diminuição de dimensão

na maioria dos casos reduziu o conjunto de dados original quase pela metade e em outros quase não surtiu efeito.

Portanto, podemos concluir que quanto melhor for o classificador, ou seja, quanto maior for a qualidade do classificador, melhor será o resultado obtido pelas técnicas e assim será possível trabalhar com um conjunto de dados menor e sem perda de representatividade do dados originais.