estudoFeatureSelection

Rodrigo

December 15, 2016

Proposta

Vamos utilizar o mesmo cenário de treino do relatório 1. Serão feitas as imputacoes pela média e removidas as linhas e colunas com mais de 10% de dados faltantes por serem consideradas muito esparsas. Em cima dos dados restantes será feita uma validacao cruzada para definir quais são os efeitos das duas técnicas de feature selecion selecionadas: selecao por baixa variancia e selecao por similaridade.

Remontando o cenário do relatório 1

Saccharomyces, a classe positiva

A levedura *Saccharomyces cerevisiae* é uma das leveduras mais importantes no processo industrial.Por esse motivo, ela é a que possui a maior quantidade de observações no banco de dados do NYCY[2]. Utilizaremos essa levedura como objeto de estudo sobre a eficácia de um modelo OCSVM para reconhecer uma levedura através de seus padrões fisiológicos.

Dados

Utilizamos scripts em Python e coletamos um grande número de observações entre 9 classes(espécies diferentes).

Temos portanto as seguintes dimensões de dados

```
dim(full_data)
```

```
## [1] 1069   53
```

O problema dos dados faltantes

Mesmo com um número tão expressivo de dados ainda temos o problema de dados faltantes. Os parâmetros utilizados dependem diretamente da facção ou não de experimentos de crescimento em determinadas condições, feitos em laboratório. Por isso, é comum encontrarmos observações muito completas, com quase todos os cenários testados, e encontrar outras muito incompletas e com quase todos os atributos faltantes.

Por isso resolvemos pegar um grande número de dados e remover a nosso critério aqueles com uma porcentagem muito grande de dados faltantes.

Imputacao pela moda

Para poder rodar os algoritmos supervisionados durante o estudo foi preciso adotar alguma abordagem de correção dos dados faltantes. Para resolver um atributo faltante em uma observação utilizamos como argumento que a moda da **classe** desse atributo seria uma boa abordagem, visto que acreditamos em um alto grau de similaridade as observações da mesma espécie. Além disso, mesmo que muitas das observações tenham que ser futuramente descartadas, pela grande porcentagem de atributos faltantes, não queremos

desperdiçar informação delas. Por isso, levamos em consideração **todo o conjunto original** para calcular a moda de cada atributo de cada classe.

Essa imputação foi feita através de um script em Python. Desse modo, temos dois arquivos CSV. Um original, com dados faltantes e um com todos os dados faltantes imputados. Carregando os dados imputados.

```
setwd('/home/rodrigo/Dropbox/UNICAMP/MC886/trabalho/ProjetoY/datasets')
imputed_full_data = read.csv("YPDS_merge_imputado.csv", header=T)
```

Removendo dados faltantes

Agora vamos retirar todos as observações com mais de 10% de dados faltantes.

```
pMiss = function(x){sum(is.na(x))/length(x)*100}

percent = 10

new_rows_idx = which(apply(x_data,1,pMiss) < 10)
new_rows_idx_length = length(new_rows_idx)
```

Vamos trabalhar então com 394 observações.

Agora que removemos o grosso das observações esparsas podemos remover as colunas com mais de 10% de dados faltantes.

```
pMiss = function(x){sum(is.na(x))/length(x)*100}

percent = 10

new_cols_idx = which(apply(x_data[new_rows_idx,],2,pMiss) < 10)
new_cols_idx_length = length(new_cols_idx)
```

Vamos trabalhar então com 47 atributos.

Então, vamos atualizar o nosso dataset para utilizar só observações e atributos com menos de 10% de dados faltantes.

```
# Selecionando o novo dataset no conjunto total de dados imputados
x_data = imputed_full_data[new_rows_idx,new_cols_idx]
y_data = imputed_full_data[new_rows_idx,1]
```

Analisando esse novo dataset...

```
length_x_data = dim(x_data)[1]
# 0 é o ID da sacchraromyces
length_sacchraromyces = length(which(y_data == 0))
```

Podemos ver que, dessas 394 observações, 333 são da espécie Saccharomyces, classe positiva.

Estudo sobre Feature Selection

No relatório passado foi constatado a necessidade de um estudo mais aprofundado da dimensão dos dados. Temos uma grande quantidade de atributos, 53 ao total, que consideramos para os treinamentos. Contudo, grande parte desses atributos são esparsos e/ou com pouca relevancia para uma predicao. De forma a dificultar um futuro levantamento de novos dados devido ao grande número de testes que deverão ser feitos para se obter todos esses atributos.

Faremos agora um estudo dos efeitos de duas técnicas de Feature Selection sobre o problema original. Utilizaremos a técnica de selecao através de baixa variancia e a tecnica de selecao através de colunas similares

Utilizaremos como métrica para os teste a acurácia. Utilizaremos uma cross-validacao 5-fold, sendo 3-folds usados para treino e 1-fold utilizado para teste. Contudo, a configuracao desse método não será ordinária. Como estamos fazendo um treinamento one-class, nós utilizaremos apenas exemplos da classe positiva como conjunto de treino. Ou seja, os exemplos de classe negativa permanecerão imutáveis no conjunto de testes. Dessa maneira, o conjunto de testes só muda em relacao aos exemplos da classe positiva nele presente.

Criando os Folds para o CV

```
# separando as obersevacoes positivas e negativas
x_positive_data = x_data[which(y_data == 0),]
x_negative_data = x_data[-which(y_data == 0),]


#Vetor de acumulacao de acuracia dos threshold's
# são 18 posicoes pois o range é de 0.1 até 0.95 com step de 0.05
accuracy_acu_vector = vector(mode="integer", length=18)
accuracy_acu_vector_var = vector(mode="integer", length=18)

# acumula a acurácia do treinamento sem feature selection
accuracy_acu_normal = 0

##Validacao externa
for(i in 0:4){

  # vetor das observacoes positivas para treino
  x_positive_train_out = x_positive_data[-((66*i+1):(66*i+1+66)),]

  ###################################
  # one class sem Feature selecion #
  ###################################
  #Loop interno para calculo de máxima acurácia
  max_accuracy = 0
  acu_accuracy = 0
  for(j in 0:2){
    #vetor das observacoes positivas para treino interno
    x_positive_train_in = x_positive_train_out[-((53*j+1):(53*j+1+53)),]

    #Auxiliares para montar o conjunto de treino
    positive_test_in = cbind(x_positive_train_out[(53*j+1):(53*j+1+53),],0)
    negative_test_in = cbind(x_negative_data, y_data[-which(y_data == 0)])
    names(negative_test_in)[ncol(negative_test_in)] = "0"
```

```
  #conjunto de treino (parte positiva + negativa)
  test_in = rbind(positive_test_in, negative_test_in)
  test_predicators = test_in[,-ncol(test_in)]
  test_labels = test_in[,ncol(test_in)]

  #Grid search para o hiperparametro nu
  for(nu_value in seq(from=0.01, to=1, by=0.05)){
      svm_model = svm(
                x = x_positive_train_in,
                y = NULL,
                scale = FALSE,
                kernel="radial",
                type="one-classification",
                nu=nu_value)


      svm.pred = predict(svm_model, test_predicators)

      #Calculo de acurácia
      TP = 0
      TN = 0
      FP = 0
      FN = 0
      for(k in 1:length(test_labels)){
        if(test_labels[k] == 0 & svm.pred[k] == T){
          TP = TP + 1;
        }else if (test_labels[k] == 0 & svm.pred[k] == F){
          FN = FN + 1
        }else if (test_labels[k] != 0 & svm.pred[k] == F){
          TN = TN + 1
        }else if (test_labels[k] != 0 & svm.pred[k] == T){
          FP = FP + 1
        }
      }

      accuracy = (TP + TN)/length(test_labels)
      max_accuracy = max(max_accuracy, accuracy)
  }
  acu_accuracy = acu_accuracy + max_accuracy
}
#Media de acurácia desse loop sera acumulada
accuracy_acu_normal = accuracy_acu_normal + acu_accuracy/3


###################################
# Feature Selection - Similaridade  #
###################################

#Como a estrutura dos é alterada, vamos manter em uma variável o valor original dos mesmos
original_x_positive_train_out = x_positive_train_out
original_x_negative_data = x_negative_data

#Loop para calculo de acurácia para cada um dos thresholds do range.
```

```
threshold_index = 1
for(threshold in seq(from=0.1, to=0.95, by=0.05)){


    # calculate correlation matrix
    correlationMatrix <- cor(x_data)

    # Indices que serão removidos por estarem com similaridade acima do threshold
    highly_correlated_index = findCorrelation(correlationMatrix, cutoff=threshold)

    # Removendo os indices apontados pelo metodo
    x_positive_train_out = original_x_positive_train_out[,-highly_correlated_index]
    x_negative_data = original_x_negative_data[,-highly_correlated_index]

    #Loop interno para calculo de maxima acurácia
    acu_accuracy = 0
    for(j in 0:2){
      #vetor das observacoes positivas para treino interno
      x_positive_train_in = x_positive_train_out[-((53*j+1):(53*j+1+53)),]

      #auxiliares para montar conjunto de testes
      positive_test_in = cbind(x_positive_train_out[(53*j+1):(53*j+1+53),],0)
      negative_test_in = cbind(x_negative_data, y_data[-which(y_data == 0)])
      names(negative_test_in)[ncol(negative_test_in)] = "0"

      #conjunto de testes (positivo + negativo)
      test_in = rbind(positive_test_in, negative_test_in)
      test_predicators = test_in[,-ncol(test_in)]
      test_labels = test_in[,ncol(test_in)]

      #Grid search para o hiperparametro nu
      max_accuracy = 0
      for(nu_value in seq(from=0.01, to=1, by=0.05)){
         svm_model = svm(
                   x = x_positive_train_in,
                   y = NULL,
                   scale = FALSE,
                   kernel="radial",
                   type="one-classification",
                   nu=nu_value)


        svm.pred = predict(svm_model, test_predicators)

        #Calculo de acurácia
        TP = 0
        TN = 0
        FP = 0
        FN = 0
        for(k in 1:length(test_labels)){
          if(test_labels[k] == 0 & svm.pred[k] == T){
            TP = TP + 1;
          }else if (test_labels[k] == 0 & svm.pred[k] == F){
```

```
              FN = FN + 1
            }else if (test_labels[k] != 0 & svm.pred[k] == F){
              TN = TN + 1
            }else if (test_labels[k] != 0 & svm.pred[k] == T){
              FP = FP + 1
            }
          }

          accuracy = (TP + TN)/length(test_labels)
          max_accuracy = max(max_accuracy, accuracy)

      }

      acu_accuracy = acu_accuracy + max_accuracy
    }
    #Acumulamos para cada threshold a acuracia encontrada
    accuracy_acu_vector[threshold_index] = accuracy_acu_vector[threshold_index] + acu_accuracy/3
    threshold_index = threshold_index + 1
}

#Retornamos o conjunto de dados as colunas originais
x_positive_train_out = original_x_positive_train_out
x_negative_data = original_x_negative_data

###################################
# Feature Selection - Variancia  #
###################################
#Como a estrutura dos é alterada, vamos manter em uma variável o valor original dos mesmos
original_x_positive_train_out = x_positive_train_out
original_x_negative_data = x_negative_data

#Loop para calculo de acurácia para cada um dos thresholds do range.
threshold_index = 1
for(threshold in seq(from=0.1, to=0.95, by=0.05)){

  #Calcula a correlacao entre as colunas
  metrics = nearZeroVar(original_x_positive_train_out, uniqueCut = threshold, saveMetrics = TRUE)

  #Remmove as colunas apontadas como tendo baixa variancia
  x_positive_train_out = original_x_positive_train_out[,-which(metrics['nzv'] == T)]
  x_negative_data = original_x_negative_data[,-which(metrics['nzv'] == T)]

  acu_accuracy = 0
  for(j in 0:2){
    #vetor das observacoes positivas para treino interno
    x_positive_train_in = x_positive_train_out[-((53*j+1):(53*j+1+53)),]

    #auxiliares para montar conjunto de testes
    positive_test_in = cbind(x_positive_train_out[(53*j+1):(53*j+1+53),],0)
    negative_test_in = cbind(x_negative_data, y_data[-which(y_data == 0)])
    names(negative_test_in)[ncol(negative_test_in)] = "0"

    #conjunto de testes(positivo + negativo)
```

```
      test_in = rbind(positive_test_in, negative_test_in)
      test_predicators = test_in[,-ncol(test_in)]
      test_labels = test_in[,ncol(test_in)]

      #Grid search para o hiperparametro nu
      max_accuracy = 0
      for(nu_value in seq(from=0.01, to=1, by=0.05)){
         svm_model = svm(
                   x = x_positive_train_in,
                   y = NULL,
                   scale = FALSE,
                   kernel="radial",
                   type="one-classification",
                   nu=nu_value)


          svm.pred = predict(svm_model, test_predicators)

          #calculo da acurácia
          TP = 0
          TN = 0
          FP = 0
          FN = 0
          for(k in 1:length(test_labels)){
            if(test_labels[k] == 0 & svm.pred[k] == T){
              TP = TP + 1;
            }else if (test_labels[k] == 0 & svm.pred[k] == F){
              FN = FN + 1
            }else if (test_labels[k] != 0 & svm.pred[k] == F){
              TN = TN + 1
            }else if (test_labels[k] != 0 & svm.pred[k] == T){
              FP = FP + 1
            }
          }

          accuracy = (TP + TN)/length(test_labels)
          max_accuracy = max(max_accuracy, accuracy)

      }

      acu_accuracy = acu_accuracy + max_accuracy
    }
      #Acumulamos para cada threshold a acuracia encontrada
      accuracy_acu_vector_var[threshold_index] = accuracy_acu_vector_var[threshold_index] + acu_accurac
      threshold_index = threshold_index + 1
  }

  #Retornamos o conjunto de dados as colunas originais
  x_positive_train_out = original_x_positive_train_out
  x_negative_data = original_x_negative_data
}
```

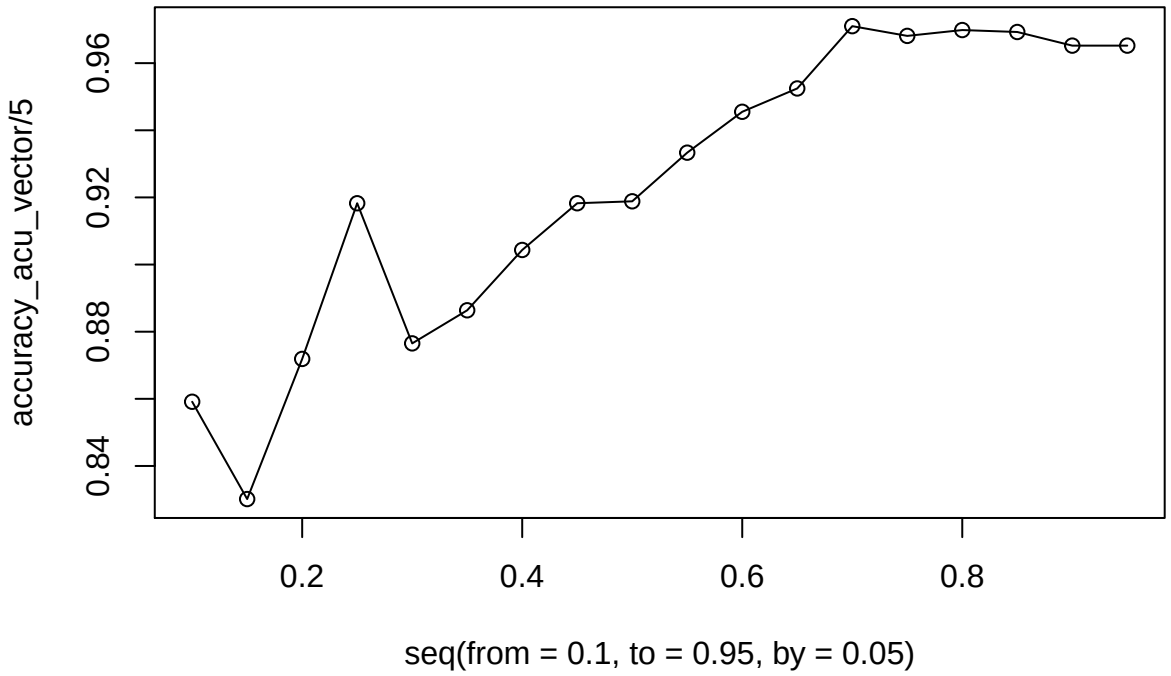
```r
#Ao final vamos printar a média da acurácia no cenário sem feature selection
print(accuracy_acu_normal/5)
```

```
## [1] 0.9849275
```

```r
#Printamos as médias de todos os thresholds e montamos um gráfico para analisar o seu comportamento
print(accuracy_acu_vector/5)
```

```
##  [1] 0.8591304 0.8301449 0.8718841 0.9182609 0.8765217 0.8863768 0.9043478
##  [8] 0.9182609 0.9188406 0.9333333 0.9455072 0.9524638 0.9710145 0.9681159
## [15] 0.9698551 0.9692754 0.9652174 0.9652174
```

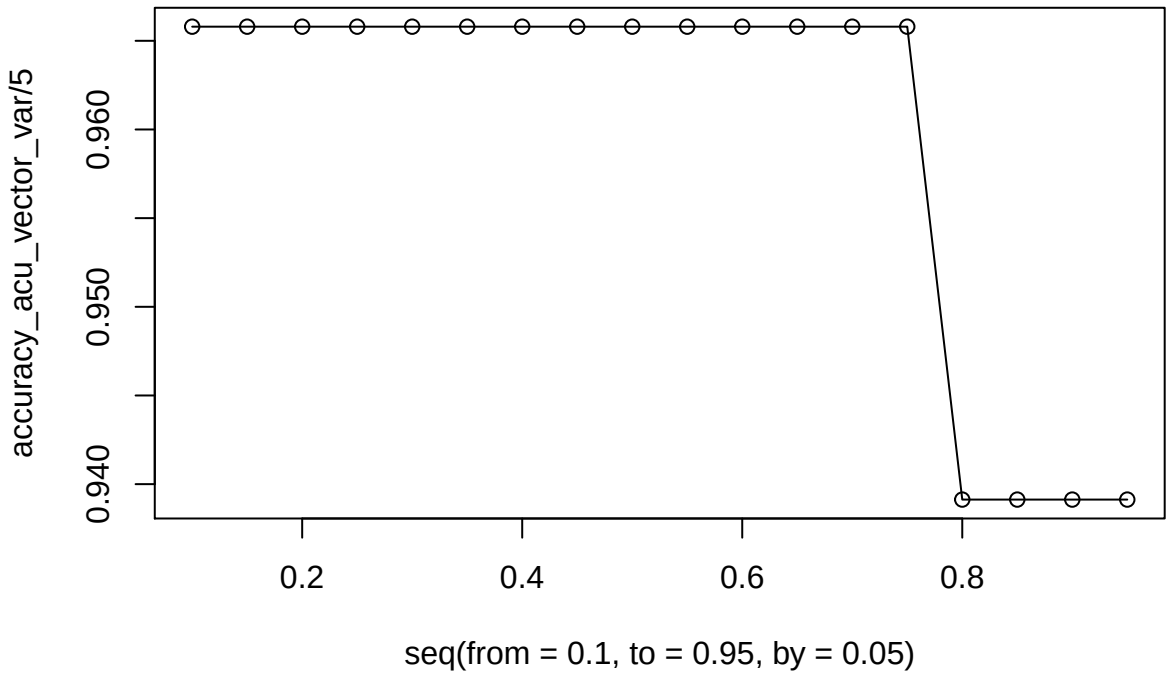
```r
plot(seq(from=0.1, to=0.95, by=0.05), accuracy_acu_vector/5)
lines(seq(from=0.1, to=0.95, by=0.05), accuracy_acu_vector/5)
```




```r
#Printamos as médias de todos os thresholds e montamos um gráfico para analisar o seu comportamento
print(accuracy_acu_vector_var/5)
```

```
##  [1] 0.9657971 0.9657971 0.9657971 0.9657971 0.9657971 0.9657971 0.9657971
##  [8] 0.9657971 0.9657971 0.9657971 0.9657971 0.9657971 0.9657971 0.9657971
## [15] 0.9391304 0.9391304 0.9391304 0.9391304
```

```r
plot(seq(from=0.1, to=0.95, by=0.05), accuracy_acu_vector_var/5)
lines(seq(from=0.1, to=0.95, by=0.05), accuracy_acu_vector_var/5)
```


accuracy_acu_vector_var/5
0.940
0.950
0.960
0.2
0.4
0.6
0.8
seq(from = 0.1, to = 0.95, by = 0.05)