Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas
Campus - Poços de Caldas / MG

3° Trabalho Prático - Algoritmos de Aprendizado de Máquina

Maria Clara Sanchez de Mira - 201911020026

1 Definição da técnica de validação a ser utilizada (cross-validation, hold-out, leave-one-out, etc);

A técnica utilizada na validação dos dados foi a **cross-validation**, poderia ter utilizado a leave-one-out, mas a escolha pela cross-validation se deu para reduzir o custo computacional. E o Cross-validation é uma técnica estatística muito conhecida por conta da sua eficiência em testes de desempenho de modelos de machine learning. E com isso vou usar ele para avaliar o desempenho dos meus algoritmos de AM.

No trabalho apliquei essa técnica para cada algoritmo de aprendizado de máquina, mas antes de aplicar foi preciso definir o número de folds que eu gostaria de ter, tal que seja possível manter cada divisão com o mesmo número de elementos. O código para essa divisão é:

```
n.folds <- 5
n.elements <- as.integer(nrow(base_treinamento) / n.folds)
folds <- seq(1, nrow(base_treinamento), by = n.elements)
```

Listing 1: Código fonte em R

2 Definição das métricas a serem utilizadas para avaliar os resultados preditivos dos modelos (acurácia, precisão, recall, matriz de confusão, etc);

A métrica a ser utilizada vai ser a **matriz de confusão, acurácia, precisão** e **recall**. A matriz de confusão é muito utilizada na resoluções de problemas de classificação, e como minha base é de classificação e tenho uma classificação binária, ela será utilizada, a matriz de confusão também é muito importante, pois a partir dela é possível calcular:

- **Acurácia:** Responsável por dizer o quanto o meu modelo acertou das previsões que tem.
- **Precisão:** Relacionada com os positivos, ou seja responde a pergunta o modelo realmente acertou? Nenhum exemplo negativo é incluído.

- **Recall:** Já analisa qual proporção de positivos foi identificados corretamente. Nenhum exemplo positivo é deixado de fora

Eu criei as funções para calcular as métricas e com o código fonte abaixo é possível visualizar elas:

```
1  ########################################
2  ##        MATRIZ DE CONFUSAO          ##
3  ########################################
4  matriz_confusao <- function(targetTest, predVal){
5    tp <- sum((targetTest == 1) & (predVal == 1))
6    fp <- sum((targetTest == 0) & (predVal == 1))
7    tn <- sum((targetTest == 0) & (predVal == 0))
8    fn <- sum((targetTest == 1) & (predVal == 0))
9    confusionMat <- matrix(c(tp, fp, fn, tn), nrow = 2, ncol = 2, dimnames = list(
       c("1","0"), c("1","0")))
10   return (confusionMat)
11 }
12
13 ########################################
14 ##             Acuracia               ##
15 ########################################
16
17 find_acuracia <- function(m_cf){
18   tp = m_cf[1,1]
19   tn = m_cf[2,2]
20
21   fn = m_cf[1,2]
22   fp = m_cf[2,1]
23
24   a = (tp + tn) / (tp + tn +  fn+ fp)
25   return (a)
26 }
27
28 ########################################
29 ##          Recall(TVP)                 ##
30 ########################################
31
32 find_recall <- function(m_cf){
33   tp = m_cf[1,1]
34   fn = m_cf[1,2]
35
36   r = tp / (tp+fn)
37   return (r)
38 }
39
40
```

```
41 ########################################
42 ##              Precisao              ##
43 ########################################
44
45 find_precisao <- function(m_cf){
46   tp = m_cf[1,1]
47   fp = m_cf[2,1]
48
49   r = tp / (tp+fp)
50   return (r)
51 }
```

Listing 2: Código fonte em R

3 Definição de um algoritmo base (baseline), que será utilizado como base para análise dos resultados – algoritmo classe majoritária;

O baseline utilizado foi o de classificação por frequência mais alta, ou seja classe majoritária, foi criado um vetor que recebe todos os tamanhos da base de teste, e classifica todos os elementos como sendo 1 - ou seja pacientes sem recorrência do câncer. E depois os valores são testando, gerando uma matriz de confusão e seus valores da acurácia, recall e precisão de cada modelo de aprendizado.

O código fonte da aplicação do algoritmo de baseline é :

```
1 ########################################
2 ##              Baseline              ##
3 ########################################
4 for (i in 1:nrow(base_teste)){
5   baseline[i]<-1
6 }
7
8 # KNN
9 B_KNN <- matriz_confusao(base_teste$class, baseline)
10 cat("Acuracia Baseline no KNN: " , find_acuracia(B_KNN), "\n")
11 cat("Recall Baseline no KNN: " , find_recall(B_KNN), "\n")
12 cat("Precisao Baseline no KNN: " , find_precisao(B_KNN), "\n")
13
14 #MLP
15 B_MLP <- matriz_confusao(base_teste$class, baseline)
16 cat("Acuracia Baseline MLP: " , find_acuracia(B_MLP), "\n")
17 cat("Recall Baseline MLP: " , find_recall(B_MLP), "\n")
18 cat("Precisao Baseline MLP: " , find_precisao(B_MLP), "\n")
```

```
19
20 # DECISION TREE
21 B_TREE <- matriz_confusao(base_teste$class, baseline)
22 cat("Acuracia Baseline Arvore de Decisao: ", find_acuracia(B_TREE), "\n")
23 cat("Recall Baseline Arvore de Decisao: ", find_recall(B_KNN), "\n")
24 cat("Precisao Baseline Arvore de Decisao: ", find_precisao(B_KNN), "\n")
```

Listing 3: Código fonte em R

4 Criação de modelo preditivo utilizando algoritmo de indução baseado nos vizinhos mais próximos e similaridade de dados (K-NN);

o K-NN é feito para classificar cada amostra de um conjunto de dados avaliando sua distância em relação aos vizinhos mais próximos. E é possível visualizar a implementação do K-NN abaixo:

```
1
2 ##########################################
3 ##                KNN                   ##
4 ##########################################
5
6 #soma matriz de confusao
7 soma_mcf_KNN = matrix(data = c(0,0,0,0) , nrow = 2, ncol = 2)
8
9 for (i in 1:(length(folds) - 1)) {
10
11
12   ########################################
13   ##        Cross-validation            ##
14   ########################################
15   temp.validation <- base_treinamento[folds[i]:folds[i+1],]
16   temp.train <- base_treinamento[-(folds[i]:folds[i+1]),]
17
18
19   model <- class::knn(train = temp.train[,-1], test = temp.validation[,-1],
20                       cl = temp.train$class, k = 105)
21
22   predsVal <- as.numeric(as.character(model))
23   predVal <- predsVal
24
25   matriz_cf <- matriz_confusao(temp.validation$class, predVal)
26
27   soma_mcf_KNN = soma_mcf_KNN + matriz_cf
```

```
28
29    print("Matriz_CF")
30    print(matriz_cf)
31    print("Soma Matriz")
32    print(soma_mcf_KNN)
33  }
```

Listing 4: Código fonte em R

5 Criação de modelo preditivo utilizando algoritmo de indução baseado em árvores de decisão (decision tree ou árvore C4.5);

O modelo utilizado foi a Árvore de Decisão, que é um tipo de AM Supervisionado, ou seja, foi preciso explica qual a entrada e qual é a saída correspondente nos dados de treinamento, onde os dados são continuamente divididos de acordo com um determinado parâmetro.

O código fonte abaixo é possível ver a aplicação desse modelo:

```
1  ##########################################
2  ##          DECISION TREE               ##
3  ##########################################
4
5  library(tree)
6  #soma matriz de confusao
7  soma_mcf_tree = matrix(data = c(0,0,0,0) , nrow = 2, ncol = 2)
8
9  for (i in 1:(length(folds) - 1)) {
10
11   ########################################
12   ##        Cross-validation            ##
13   ########################################
14   temp.validation <- base_treinamento[folds[i]:folds[i+1],]
15   temp.train <- base_treinamento[-(folds[i]:folds[i+1]),]
16
17   #Aplicacao modelo
18   model <- tree(temp.train$class ~ ., temp.train)
19   predsVal <- predict(model, temp.validation[,-1])
20   predVal <- ifelse (predsVal > 0.5, 1 , 0)
21
22   matriz_cf_tree <- matriz_confusao(temp.validation$class, predVal)
23
24   soma_mcf_tree = soma_mcf_tree + matriz_cf_tree
25
26   print("Matriz_CF")
```

```
27    print(matriz_cf_tree)
28    print("Soma Matriz")
29    print(soma_mcf_tree)
30  }
```

Listing 5: Código fonte em R

6 Criação de modelo preditivo utilizando algoritmo de indução redes neurais artificiais (MLP);

MLP (perceptron multicamadas) é uma rede neural MUITO semelhante à perceptron, mas com mais de uma camada de neurônios em alimentação direta. Com o código fonte abaixo é possível ver a aplicação do algorítimo de MLP:

```
1  ##########################################
2  ##                MLP                   ##
3  ##########################################
4
5  library(RSNNS)
6
7  #soma matriz de confusao
8  soma_mcf_mlp = matrix(data = c(0,0,0,0) , nrow = 2, ncol = 2)
9
10 for (i in 1:(length(folds) - 1)) {
11
12   ##########################################
13   ##        Cross-validation              ##
14   ##########################################
15   temp.validation <- base_treinamento[folds[i]:folds[i+1],]
16   temp.train <- base_treinamento[-(folds[i]:folds[i+1]),]
17
18
19   model <- mlp( x = temp.train,
20                 y = temp.train$class,
21                 size = 5,
22                 learnFuncParams = c(0.1),
23                 maxit = 40,
24                 inputsTest = temp.validation,
25                 targetsTest = temp.validation$class)
26   predsVal <- predict(model,temp.validation)
27   predVal <- ifelse (predsVal > 0.5, 1, 0)
28
29   # predValidas <- find_KNN(temp.train,temp.validation,n.folds)
30   matriz_cf_mlp <- matriz_confusao(temp.validation$class, predVal)
31
32   soma_mcf_mlp = soma_mcf_mlp + matriz_cf_mlp
```

```
33
34    print("Matriz_CF")
35    print(matriz_cf_mlp)
36    print("Soma Matriz")
37    print(soma_mcf_mlp)
38 }
```

Listing 6: Código fonte em R

7 Análise dos resultados do algoritmo baseline;

Para visualizar as analises dos resultados do algoritmo de baseline foi criada a seguinte tabela:

Algoritmo	Acurácia	Recall	Precisão
KNN	0.7058824	1	0.7058824
MLP	0.7058824	1	0.7058824
Árvore de Decisão	0.7058824	1	0.7058824

É possível notar que a classe acerta aproximadamente 70.3% que é o esperado, pois quando se utiliza o algoritmo de baseline pressupõe que tudo é saída da classe majoritária a que mais tem dados, ou seja o valor 1 de paciente sem recorrência do câncer é o que tem 70.3% dos dados, então os valores fazem sentindo, e o Recall chegar em 1, mostra que ele soube classificar de forma correta os valores não positivos, ou seja, os pacientes que tem a presença do cancêr.

8 Análise dos resultados dos três algoritmos de aprendizado de máquina supracitados;

O resultado com dos 3 algoritmos na validação foi:

Algoritmo	Acurácia	Recall	Precisão
KNN	0.702439	1	0.702439
MLP	0.8341463	1	0.8089888
Árvore de Decisão	0.502439	0.6388889	0.6478873

Ao realizar o método de cross-validation foi possível chegar aos valores acima, o único que não consegui bater o baseline foi infelizmente o da árvore de decisão. Já os outros com alguns ajustes no K no casso do algoritmo K-NN e o valor máximo de execuções na rede neural.

Após esses ajustes foi possível realizar novamente o treinamento e chegar em um modelo oficial agora, para poder testar da base testes, que até agora estava todo parado.

Logo após a validação e ter configurado os parâmetros e indo para base de teste, foi possível obter:

Algoritmo	Acurácia	Recall	Precisão
KNN	0.7058824	1	0.7058824
MLP	1	1	1
Árvore de Decisão	0.7058824	0.8666667	0.7536232

Com as bases de dados de treinamento e teste, foi possível depois de ter treinado o algoritmo e validado, obter um modelo muito bom, pois ele trouxe valores que foram muito satisfatórios e independentes do algoritmo todos conseguíram classificar os dados, porém o que me deixou intrigada foi o algoritmo da árvore de decisão, pois na validação ele não conseguiu bater o baseline, mas aqui ele conseguiu.

Portando, foi possível concluir, que para essa base de dados o modelo de aprendizado de máquina que mais teve resultados satisfatórios foi o de rede neurais, pois ele conseguiu aprender e classificar todos os dados de forma 100% correta e é possível ver isso analisando a acurácia, recall e a precisão.

Código fonte abaixo:

```
##########################################
##                Final                 ##
##########################################

# KNN
model_KNN <- class::knn(train = base_treinamento[,-1], test = base_teste[,-1],
                        cl = base_treinamento$class, k = 105)
predsVal_KNN <- as.numeric(as.character(model_KNN))
predVal_KNN <- predsVal_KNN
MC_KNK <- matriz_confusao(base_teste$class, predVal_KNN)

cat("Acuracia Final KNN: ", find_acuracia(MC_KNK), "\n")
cat("Recall Final KNN: ", find_recall(MC_KNK), "\n")
cat("Precisao Final KNN: ", find_precisao(MC_KNK), "\n")

#MLP
model_MLP <- mlp(x = base_treinamento,
                 y = base_treinamento$class,
                 size = 5,
                 learnFuncParams = c(0.1),
                 maxit = 20)

predsVal_MLP <- predict(model_MLP, base_teste)
predVal_MLP <- ifelse (predsVal_MLP > 0.5, 1, 0)


```

```
28 MC_MLP <- matriz_confusao(base_teste$class, predVal_MLP)
29
30 cat("Acuracia Final MLP: ", find_acuracia(MC_MLP), "\n")
31 cat("Recall Final MLP: ", find_recall(MC_MLP), "\n")
32 cat("Precisao Final MLP: ", find_precisao(MC_MLP), "\n")
33
34 # DECISION TREE
35
36 model_TREE <- tree(base_treinamento$class ~ ., base_treinamento)
37 predsVal_TREE <- predict(model_TREE, base_teste[,-1])
38 predVal_TREE <- ifelse (predsVal_TREE > 0.5, 1 , 0)
39
40 MC_TREE <- matriz_confusao(base_teste$class, predVal_TREE)
41
42 cat("Acuracia Final TREE: ", find_acuracia(MC_TREE), "\n")
43 cat("Recall Final TREE: ", find_recall(MC_TREE), "\n")
44 cat("Precisao Final TREE: ", find_precisao(MC_TREE), "\n")
```

Listing 7: Código fonte em R