Aprendizado de Máquina - MO444

Exercício 3

Aluno: Paulo Ricardo Finardi. RA: 144809

0 Preliminares

Foi utilizado `Python 3` na resolução dos exercícios. O código abaixo é o *header* de todos os itens.

```
import numpy as np
import pandas as pd
import scipy.stats as stats

from sklearn.preprocessing import scale
from sklearn.cross_validation import KFold
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier as MLP
from sklearn.ensemble import RandomForestClassifier as RF
from sklearn.ensemble import GradientBoostingClassifier as GBM
```

0.1 Enunciado / Resposta

Enunciado: *Usando um `5-fold` externo para calcular a acurácia, e um `3-fold` interno para a escolha dos hiperparâmetros, determine qual algoritimo entre `kNN`, `SVM` com kernel RBF, `redes neurais`, `Random Forest`, e `Gradient Boosting Machine` tem a maior acurácia.*
Resposta: Existiram dois métodos que obtiveram a mesma acurácia: `SVM` e `KNN`. Os melhores hiperparâmetros do `SVM` são: $C = 0.031250$ e $gamma = 0.000031$. Para o KNN (que foi realizado com dados transformados pelo PCA), o melhor hiperparâmetro é $k = 11$. A acurácia média desses dois métodos foi de:**93.36%**.

Os trechos restantes do enunciado estão postos acima de cada bloco de código.

1 Código

1.1 Pré-Processamento

Enunciado: *Preprocesse os dados do arquivo: Substitua os dados faltantes pela media da coluna (imputação pela média). Finalmente padronize as colunas para media 0 e desvio padrao 1.*

Resposta:

```
secom = pd.read_table('secom.data', header=None, delim_whitespace=True)
Y = pd.read_table('secom_labels.data', header=None, usecols=[0], \
                  squeeze=True, delim_whitespace=True)

# dados representados com numpy array
X = secom.as_matrix(columns=None)
labels = Y.as_matrix(columns=None)

# calculando a média das colunas
col_mean = stats.nanmean(X, axis=0)

#ind obtém os índices onde a coluna possui dados faltantes
ind = np.where(np.isnan(X))

# substitui os índices faltantes pela média da coluna
X[ind] = np.take(col_mean, ind[1])

# scale() fornece média zero das colunas e desvio padrão = 1
std_secom = scale(X)
```

1.2 PCA

Enunciado: *Para o kNN, faça um PCA que mantem 80% da variância.*

Resposta:

```
# função PCA com sklearn
def doPCA(x): # x é o argumento do número de componentes do PCA
```

```
    from sklearn.decomposition import PCA
    pca = PCA(n_components=x)
    pca.fit(std_secom)
    return pca

# executando o PCA com 590 componentes.
pca = doPCA(std_secom.shape[1])

# os elementos em _var contém a total variância.
_var = pca.explained_variance_ratio_

# armazena a variância.
requested_var = 0

# 80%  é a variância requisitada.
lim = 0.80

# iterações para encontrar a quantidade de dimensões que mantém
# 80% da variância
for i in range (len(_var)):
    requested_var += _var[i]
    if requested_var >= lim:
        element = i
        requested_var -= _var[i] # para ajuste do indice
        break

# repetindo o PCA com 80% da variância
pca = doPCA(element) # element == 89 componentes
_var = pca.explained_variance_ratio_

# dados transformados
secom_PCA_80 = pca.transform(std_secom)

# prints
print('Com %d elementos temos a variância requisitada.' % (element))
print('Formato dos dados com PCA (variância=80%): {}.'\
```

```
39        .format(secom_PCA_80.shape))
40  print('As %d componentes/colunas principais:' % (element))
41  np.set_printoptions(precision=3)
42  print(_var)
43  print('\nA soma dessas componentes/colunas: %.5f.' % np.sum(_var))
```

Os resultados das linhas 37 até 43 são:

```
Com 89 elementos temos a variância requisitada.
Formato dos dados com PCA (variância=80%): (1567, 89).
As 89 componentes/colunas principais:
[ 0.056  0.036  0.028  0.025  0.022  0.021  0.02   0.018  0.018  0.016
  0.015  0.013  0.013  0.013  0.013  0.012  0.011  0.011  0.011  0.011
  0.01   0.01   0.01   0.01   0.01   0.009  0.009  0.009  0.009  0.008
  0.008  0.008  0.008  0.008  0.008  0.008  0.008  0.007  0.007  0.007
  0.007  0.007  0.007  0.007  0.007  0.007  0.006  0.006  0.006  0.006
  0.006  0.006  0.006  0.006  0.006  0.006  0.006  0.005  0.005  0.005
  0.005  0.005  0.005  0.005  0.005  0.005  0.005  0.005  0.005  0.004
  0.004  0.004  0.004  0.004  0.004  0.004  0.004  0.004  0.004  0.004
  0.004  0.004  0.004  0.003  0.003  0.003  0.003  0.003  0.003]

A soma dessas componentes/colunas: 0.79580.
```

1.3 KNN

Enunciado: *Para o kNN, faça um PCA que mantem 80% da variância. Busque os valores do k entre os valores 1, 5, 11, 15, 21, 25.*

Resposta:

```
1   # acurácia inicial
2   acc_knn = 0
3
4   # separa os dados transformados  secom_PCA_80 em 5-fold
5   kf_5 = KFold(n = len(labels), n_folds=5)
6
7   #contador dos folds
8   count = 0
9
10  # para cada fold externo encontrar o melhor hiperparâmetro
11  for train, test in kf_5:
```

```
12      count += 1
13      kf_3 = KFold(n = len(labels[train]), n_folds=3)
14
15      # faz um grid search do melhor k em cada conjunto de treino do 5-fold
16      best_k, best_acc = 0,0
17
18      for k in [1, 5, 11, 15, 21, 25]:
19          new_acc = 0
20
21          for train_hp, test_hp in kf_3:
22              # define o knn
23              knn = KNN(n_neighbors=k)
24              # treina secom_PCA_80
25              knn.fit(secom_PCA_80[train][train_hp], labels[train][train_hp])
26              # avalia a acurária
27              new_acc += knn.score(secom_PCA_80[train][test_hp], labels[train][test_hp])
28
29          # acurácia média do hiperparâmetro k
30          new_acc = new_acc/3
31
32          # verifica se acurácia do hiperparametro k atual é a melhor
33          if (new_acc > best_acc):
34              best_acc = new_acc
35              best_k = k
36
37      # imprime a melhor acurácia obtida pelo fold interno
38      print('Em fold #{}: o melhor k é: {}'.format(count, best_k))
39
40      # ao fim do loop, calcula a acurácia para o melhor k
41      knn_out = KNN(n_neighbors=best_k)
42
43      # treina com os valores do 5-fold externo
44      knn_out.fit(secom_PCA_80[train], labels[train])
45
46      # obtém a acurácia usando o conjunto de teste do 5-fold externo
47      print('Acurácia do Fold = {}%'.format(100*knn_out.score(secom_PCA_80[test],\
```

```
48              labels[test])))
49
50      # acha a acurácia média do método
51      acc_knn += knn_out.score(secom_PCA_80[test], labels[test])
52
53  acc_knn /= 5
54  print('\nAcurácia do Método: {}%'.format(acc_knn*100))
```

O resultado da linha 54 é:

```
Em fold #1: o melhor k é: 11
Acurácia do Fold = 85.98726114649682%
Em fold #2: o melhor k é: 11
Acurácia do Fold = 93.31210191082803%
Em fold #3: o melhor k é: 11
Acurácia do Fold = 96.48562300319489%
Em fold #4: o melhor k é: 11
Acurácia do Fold = 96.48562300319489%
Em fold #5: o melhor k é: 11
Acurácia do Fold = 94.56869009584665%

Acurácia do Método: 93.36785983191224%
```

1.4 SVM

Enunciado: *Para o SVM RBF teste para C=2**(-5), 2**(0), 2**(5), 2**(10) e gamma= 2**(-15) 2**(-10) 2**(-5) 2**(0) 2**(5)*

Resposta:

```
1   # acurácia inicial
2   acc_svm = 0
3
4   # separa os dados transformados  std_secom em 5-fold
5   kf_5 = KFold(n = len(labels), n_folds=5)
6
7   #contador dos folds
8   count = 0
9
10  # para cada fold externo encontrar o melhor hiperparâmetro
11  for train, test in kf_5:
```

```
12      count += 1
13      kf_3 = KFold(n = len(labels[train]), n_folds=3)
14
15      best_c, best_gamma, best_acc = 0,0,0
16
17      # loop interno para determinar os hiperâmetros em 3-fold
18      for c in [2**-5, 2**0, 2**5, 2**10]:
19          for gamma in [2**-15, 2**-10, 2**-5, 2**0, 2**5]:
20              new_acc = 0
21              for train_hp, test_hp in kf_3:
22                  # define o SVM classificador svc
23                  svc = SVC(C=c, kernel='rbf', gamma=gamma)
24                  # treina std_secom
25                  svc.fit(std_secom[train][train_hp], labels[train][train_hp])
26                  # avalia a acurária
27                  new_acc += svc.score(std_secom[train][test_hp], \
28                                       labels[train][test_hp])
29
30              # acurácia média do hiperparâmetro
31              new_acc = new_acc/3
32
33              # verifica se acurácia do hiperparametro atual é a melhor
34              if (new_acc > best_acc):
35                  best_acc = new_acc
36                  best_c = c
37                  best_gamma = gamma
38
39      # imprime a melhor acurácia obtida pelo fold interno
40      print('Em fold #%d: o melhor c é: %.6f e gamma: %.6f'
41             % (count, best_c, best_gamma))
42
43      # ao fim do loop, calcula a acurácia para o melhor c e gamma
44      svc_out = SVC(C=best_c, kernel='rbf', gamma=best_gamma)
45
46      # treina com os valores do 5-fold externo
47      svc_out.fit(std_secom[train], labels[train])
```

```
48
49      # obtém a acurácia usando o conjunto de teste do 5-fold externo
50      print('Acurácia do Fold = {}%'.format(100*svc_out.score(std_secom[test],\
51             labels[test])))
52
53      # acha a acurácia média do método
54      acc_svm += svc_out.score(std_secom[test], labels[test])
55
56  acc_svm /= 5
57  print('\nAcurácia do Método: {}%'.format(acc_svm*100))
```

O resultado da linha 57 é:

```
Em fold #1: o melhor c é: 0.031250 e gamma: 0.000031
Acurácia do Fold = 85.98726114649682%
Em fold #2: o melhor c é: 0.031250 e gamma: 0.000031
Acurácia do Fold = 93.31210191082803%
Em fold #3: o melhor c é: 0.031250 e gamma: 0.000031
Acurácia do Fold = 96.48562300319489%
Em fold #4: o melhor c é: 0.031250 e gamma: 0.000031
Acurácia do Fold = 96.48562300319489%
Em fold #5: o melhor c é: 0.031250 e gamma: 0.000031
Acurácia do Fold = 94.56869009584665%

Acurácia do Método: 93.36785983191224%
```

1.5 Redes Neurais - MLP

Enunciado: *Para a rede neural, teste com 10, 20, 30 e 40 neuronios na camada escondida*

Resposta:

```
# acurácia inicial
acc = 0

# separa os dados transformados  std_secom em 5-fold
kf_5 = KFold(n = len(labels), n_folds=5)

#contador dos folds
count = 0

# para cada fold externo encontrar o melhor hiperparâmetro
for train, test in kf_5:
    count += 1
    kf_3 = KFold(n = len(labels[train]), n_folds=3)


    best_neur, best_acc = 0,0

    # loop interno para determinar os hiperparâmetros em 3-fold
    for neur in [10, 20, 30, 40]:
        new_acc = 0

        for train_hp, test_hp in kf_3:
            # define a rede neural
            nn = MLP(solver='lbfgs', alpha=1, hidden_layer_sizes=neur, random_state=1)
            # treina std_secom
            nn.fit(std_secom[train][train_hp], labels[train][train_hp])
            # avalia a acurária
            new_acc += nn.score(std_secom[train][test_hp], labels[train][test_hp])

        # acurácia média do hiperparâmetro
        new_acc = new_acc/3
```

```
32
33          # verifica se acurácia do hiperparametro atual é a melhor
34          if (new_acc > best_acc):
35              best_acc = new_acc
36              best_neur = neur
37
38      # imprime a melhor acurácia obtida pelo fold interno
39      print('Em fold #{}: o melhor número de neurônios na camada escondida é: {}'.format(cou
40
41      # ao fim do loop, calcula a acurácia para o melhor hiperpar.
42      nn_out = MLP(solver='lbfgs', alpha=1, hidden_layer_sizes=best_neur, random_state=1)
43
44      # treina com os valores do 5-fold externo
45      nn_out.fit(std_secom[train], labels[train])
46
47      # obtém a acurácia usando o conjunto de teste do 5-fold externo
48      print('Acurácia do Fold = {}%'.format(100*nn_out.score(std_secom[test],\
49             labels[test])))
50
51      # acha a acurácia média do método
52      acc += nn_out.score(std_secom[test], labels[test])
53
54  acc /= 5
55  print('\nAcurácia do Método: {}%'.format(acc*100))
```

O resultado da linha 54 é:

```
Em fold #1: o melhor número de neurônios na camada escondida é: 20
Acurácia do Fold = 76.75159235668791%
Em fold #2: o melhor número de neurônios na camada escondida é: 40
Acurácia do Fold = 90.76433121019109%
Em fold #3: o melhor número de neurônios na camada escondida é: 30
Acurácia do Fold = 94.56869009584665%
Em fold #4: o melhor número de neurônios na camada escondida é: 30
Acurácia do Fold = 94.56869009584665%
Em fold #5: o melhor número de neurônios na camada escondida é: 40
Acurácia do Fold = 93.9297124600639%

Acurácia do Método: 90.11660324372724%
```

1.6 Random Forest

Enunciado: *Para o RF, teste com n_featrues = 10, 15, 20, 25 e ntrees = 100, 200, 300 e 400.*

Resposta:

```
# acurácia inicial
acc = 0

# separa os dados transformados  std_secom em 5-fold
kf_5 = KFold(n = len(labels), n_folds=5)

#contador dos folds
count = 0

# para cada fold externo encontrar o melhor hiperparâmetro
for train, test in kf_5:
    count += 1
    kf_3 = KFold(n = len(labels[train]), n_folds=3)

    best_n_features, best_n_features, best_acc = 0,0,0

    # loop interno para determinar os hiperparâmentros em 3-fold
    for n_features in [10, 15, 20, 25]:
        for n_trees in [100, 200, 300, 400]:
            new_acc = 0
            for train_hp, test_hp in kf_3:
                # define o RF classificador rf
                rf = RF(max_depth=n_trees, n_estimators=n_features)
                # treina std_secom
                rf.fit(std_secom[train][train_hp], labels[train][train_hp])
                # avalia a acurária
                new_acc += rf.score(std_secom[train][test_hp], \
                                    labels[train][test_hp])

            # acurácia média do hiperparâmetro
            new_acc = new_acc/3
```

```
32
33              # verifica se acurácia do hiperparametro atual é a melhor
34              if (new_acc > best_acc):
35                  best_acc        = new_acc
36                  best_n_features = n_features
37                  best_n_trees    = n_trees
38
39      # imprime a melhor acurácia obtida pelo fold interno
40      print('Em fold #%d: o melhor n_feature é: %d e n_tree: %d'
41             % (count, best_n_features, best_n_trees))
42
43      # ao fim do loop, calcula a acurácia para os melhores hiperpar.
44      rf_out = RF(max_features=best_n_features, max_depth=best_n_trees)
45
46      # treina com os valores do 5-fold externo
47      rf_out.fit(std_secom[train], labels[train])
48
49      # obtém a acurácia usando o conjunto de teste do 5-fold externo
50      print('Acurácia do Fold = {}%'.format(100*rf_out.score(std_secom[test],\
51             labels[test])))
52
53      # acha a acurácia média do método
54      acc += rf_out.score(std_secom[test], labels[test])
55
56  acc /= 5
57  print('\nAcurácia do Método: {}%'.format(acc*100))
```

```
Em fold #1: o melhor n_feature é: 10 e n_tree: 400
Acurácia do Fold = 85.98726114649682%
Em fold #2: o melhor n_feature é: 25 e n_tree: 100
Acurácia do Fold = 93.31210191082803%
Em fold #3: o melhor n_feature é: 15 e n_tree: 200
Acurácia do Fold = 96.48562300319489%
Em fold #4: o melhor n_feature é: 10 e n_tree: 100
Acurácia do Fold = 96.1661341853035%
Em fold #5: o melhor n_feature é: 20 e n_tree: 100
Acurácia do Fold = 94.56869009584665%

Acurácia do Método: 93.30396206833397%
```

1.7 Gradient Boosting Machine

Enunciado: *Para o GBM teste para numero de arvores = 30, 70, e 100, com learning rate de 0.1 e 0.05, e profundidade da arvore=5.*

Resposta:

```
# acurácia inicial
acc = 0

# separa os dados transformados  std_secom em 5-fold
kf_5 = KFold(n = len(labels), n_folds=5)

#contador dos folds
count = 0

# para cada fold externo encontrar o melhor hiperparâmetro
for train, test in kf_5:
    count += 1
    kf_3 = KFold(n = len(labels[train]), n_folds=3)

    # best_learning_rate não pode iniciar com 0
    best_learning_rate, best_n_tree, best_acc = 0.1,0,0

    # loop interno para determinar os hiperparâmentros em 3-fold
    for ln in [0.1, 0.05]:
        for n_trees in [30, 70, 100]:
            new_acc = 0
            for train_hp, test_hp in kf_3:
                # define o GBM classificador
                gbm = GBM(n_estimators=n_trees, learning_rate=ln, max_depth=5)
                # treina std_secom
                rf.fit(std_secom[train][train_hp], labels[train][train_hp])
                # avalia a acurária
                new_acc += rf.score(std_secom[train][test_hp], \
                                    labels[train][test_hp])

```

```
31              # acurácia média
32              new_acc = new_acc/3
33
34              # verifica se acurácia do hiperparametro atual é a melhor
35              if (new_acc > best_acc):
36                  best_acc        = new_acc
37                  best_learning_rate = ln
38                  best_n_tree    = n_trees
39
40      # imprime a melhor acurácia obtida pelo fold interno
41      print('Em fold #%d: o melhor learning rate é: %0.2f e n_tree: %d'
42             % (count, best_learning_rate, best_n_tree))
43
44      # ao fim do loop, calcula a acurácia para os melhores hiperpar.
45      gbm_out = GBM(n_estimators=best_n_tree, learning_rate=best_learning_rate)
46
47      # treina com os valores do 5-fold externo
48      gbm_out.fit(std_secom[train], labels[train])
49
50      # obtém a acurácia usando o conjunto de teste do 5-fold externo
51      print('Acurácia do Fold = {}%'.format(100*gbm_out.score(std_secom[test],\
52             labels[test])))
53
54      # acha a acurácia média do método
55      acc += gbm_out.score(std_secom[test], labels[test])
56
57  acc /= 5
58  print('\nAcurácia do Método: {}%'.format(acc*100))
```

O resultado da linha 57 é:

```
Em fold #1: o melhor learning rate é: 0.10 e n_tree: 100
Acurácia do Fold = 71.97452229299363%
Em fold #2: o melhor learning rate é: 0.10 e n_tree: 70
Acurácia do Fold = 92.35668789808918%
Em fold #3: o melhor learning rate é: 0.10 e n_tree: 70
Acurácia do Fold = 94.88817891373802%
Em fold #4: o melhor learning rate é: 0.05 e n_tree: 30
Acurácia do Fold = 96.48562300319489%
Em fold #5: o melhor learning rate é: 0.10 e n_tree: 70
Acurácia do Fold = 93.9297124600639%

Acurácia do Método: 89.92694491361594%
```

2 Comentários

Durante a fase de pré-processamento, realizei um mini-teste para averiguar se a substituição dos valores faltantes pela média funcionava. Segue o código:

```
"""Mini teste com uma matriz pequena para visualizar
   se estou fazendo a substituição das colunas pela média"""

A = np.array([[1.0     , 2.0, np.nan],
              [np.nan, 3.0, np.nan],
              [3.0     , np.nan, 5.0]])
print('Matriz teste com colunas nan:\n', A)
col_mean = stats.nanmean(A, axis=0)
print()
for i in range(3):
    print('A média da %d coluna é: %s'% (i+1,col_mean[i]))
inds = np.where(np.isnan(A))
A[inds] = np.take(col_mean, inds[1])
print('\nMatriz substituída com as médias:\n', A)
```

```
Matriz teste com colunas nan:
 [[  1.   2.  nan]
 [ nan   3.  nan]
 [  3.  nan   5.]]

A média da 1 coluna é: 2.0
A média da 2 coluna é: 2.5
A média da 3 coluna é: 5.0

Matriz substituída com as médias:
 [[ 1.   2.   5. ]
 [ 2.   3.   5. ]
 [ 3.   2.5  5. ]]
```

A função `GridSearchCv` da biblioteca `sklearn`, não foi utilizada. Fiz explicitamente os *loops* no *grid search*.